Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
### Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>▼ SSL</th>
<th>[detail level 1 2 3 4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumeration/Typedefs</td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>▼ WLAN</td>
<td></td>
</tr>
<tr>
<td>► Defines</td>
<td></td>
</tr>
<tr>
<td>► Function</td>
<td></td>
</tr>
<tr>
<td>► DataTypes</td>
<td></td>
</tr>
<tr>
<td>▼ BSP</td>
<td></td>
</tr>
<tr>
<td>► Defines</td>
<td></td>
</tr>
<tr>
<td>► DataTypes</td>
<td></td>
</tr>
<tr>
<td>► Function</td>
<td></td>
</tr>
<tr>
<td>▼ Socket</td>
<td></td>
</tr>
<tr>
<td>► Defines</td>
<td></td>
</tr>
<tr>
<td>► Error Codes</td>
<td></td>
</tr>
<tr>
<td>► DataTypes</td>
<td></td>
</tr>
<tr>
<td>► Function</td>
<td></td>
</tr>
<tr>
<td>Modules</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>SSL</td>
<td></td>
</tr>
</tbody>
</table>

**WINC1500 IoT Software APIs 19.5.2**

WINC Software API Reference Manual
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumeration/Typedefs</td>
</tr>
<tr>
<td>Functions</td>
</tr>
</tbody>
</table>
Detailed Description

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
## Enumeration/Typedefs

### SSL
typedef void(*tpfAppSSLCb)(uint8 u8MsgType, void *pvMsg)
Detailed Description

Typedef Documentation
typedef void(* tpfAppSSLCb)(uint8 u8MsgType, void *pvMsg)
Functions

SSL
## Functions

<table>
<thead>
<tr>
<th>NMI_API sint8</th>
<th>m2m_ssl_init (tpfAppSSLCb pfAppSSLCb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8</td>
<td>m2m_ssl_handshake_rsp (tstrEccReqInfo *strECCResp, uint8 *pu8RspDataBuff, uint16 u16RspDataSz)</td>
</tr>
<tr>
<td>NMI_API sint8</td>
<td>m2m_ssl_send_certs_to_winc (uint8 *pu8Buffer, uint32 u32BuffSize)</td>
</tr>
<tr>
<td>NMI_API sint8</td>
<td>m2m_ssl_retrieve_cert (uint16 *pu16CurveType, uint8 *pu8Hash, uint8 *pu8Sig, tstrECPoint *pu8Key)</td>
</tr>
<tr>
<td>NMI_API sint8</td>
<td>m2m_ssl_retrieve_hash (uint8 *pu8Hash, uint16 u16HashSz)</td>
</tr>
<tr>
<td>NMI_API void</td>
<td>m2m_ssl_stop_processing_certs (void)</td>
</tr>
<tr>
<td>NMI_API void</td>
<td>m2m_ssl_ecc_process_done (void)</td>
</tr>
<tr>
<td>sint8</td>
<td>m2m_ssl_set_active_ciphersuites (uint32 u32SslCsBMP)</td>
</tr>
</tbody>
</table>
◆ m2m_ssl_init()

NMI_API sint8 m2m_ssl_init ( tpfAppSSLCb pfAppSSLCb )
◆ m2m_ssl_handshake_rsp()

```c
NMI_API sint8 m2m_ssl_handshake_rsp ( tstrEccReqInfo * strECCResp,
                                        uint8 * pu8RspDataBuff,
                                        uint16 u16RspDataSz )
```

Sends ECC responses to the WINC.

**Parameters**

- **[in]** `strECCResp` ECC Response struct.
- **[in]** `pu8RspDataBuff` Pointer of the response data to be sent.
- **[in]** `u16RspDataSz` Response data size.

**Returns**

The function SHALL return 0 for success and a negative value otherwise.
◆ m2m_ssl_send_certs_to_winc()

```c
NMI_API sint8 m2m_ssl_send_certs_to_winc ( uint8 * pu8Buffer,
                                          uint32 u32BufferSz
                                      )
```

Sends certificates to the WINC.

**Parameters**

- **[in] pu8Buffer**  
  Pointer to the certificates.
- **[in] u32BufferSz**  
  Size of the certificates.

**Returns**

The function SHALL return 0 for success and a negative value otherwise.
Retrieve the certificate to be verified from the WINC.

**Parameters**

- `[in] pu16CurveType` Pointer to the certificate curve type.
- `[in] pu8Hash` Pointer to the certificate hash.
- `[in] pu8Sig` Pointer to the certificate signature.
- `[in] pu8Key` Pointer to the certificate Key.

**Returns**

The function SHALL return 0 for success and a negative value otherwise.
m2m_ssl_retrieve_hash()

NMI_API sint8 m2m_ssl_retrieve_hash (uint8 * pu8Hash, 
                                  uint16 u16HashSz)

Retrieve the certificate hash.

Parameters
   [in] pu8Hash Pointer to the certificate hash.
   [in] u16HashSz Hash size.

Returns
   The function SHALL return 0 for success and a negative value otherwise.
◆ m2m_ssl_stop_processing_certs()

NMI_API void m2m_ssl_stop_processing_certs ( void )

Allow ssl driver to tidy up in case application does not read all available certificates.

Warning
This API must only be called if some certificates are left unread.

Returns
None.
m2m_ssl_ecc_process_done()

NMI_API void m2m_ssl_ecc_process_done ( void )

Allow ssl driver to tidy up after application has finished processing ecc message.

Warning
This API must be called after receiving a SSL callback with type M2M_SSL_REQ_ECC

Returns
None.
m2m_ssl_set_active_ciphersuites()

NMI_API sint8
m2m_ssl_set_active_ciphersuites ( uint32 u32SslCsBMP )

Override the default Active SSL ciphers in the SSL module with a certain combination selected by the caller in the form of a bitmap containing the required ciphers to be on. There is no need to call this function if the application will not change the default ciphersuites.

Parameters

[in] u32SslCsBMP Bitmap containing the desired ciphers to be enabled for the SSL module. The ciphersuites are defined in TLS Cipher Suite IDs. The default ciphersuites are all ciphersuites supported by the firmware with the exception of ECC ciphersuites. The caller can override the default with any desired combination, except for combinations involving both RSA and ECC; if any RSA ciphersuite is enabled, then firmware will disable all ECC ciphersuites. If u32SslCsBMP does not contain any ciphersuites supported by firmware, then the current active list will not be changed.

Returns

- SOCK_ERR_NO_ERROR
- SOCK_ERR_INVALID_ARG

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>DataTypes</td>
</tr>
</tbody>
</table>
### Defines

<p>| WLAN |</p>
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommonDefines</td>
</tr>
</tbody>
</table>
### Macros

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define M2M_MAJOR_SHIFT (8)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define M2M_MINOR_SHIFT (4)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define M2M_PATCH_SHIFT (0)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define M2M_DRV_VERSION_SHIFT (16)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define M2M_FW_VERSION_SHIFT (0)</code></td>
<td></td>
</tr>
</tbody>
</table>

```c
#define M2M_GET_MAJOR(ver_info_hword) ((uint8)((ver_info_hword) >> M2M_MAJOR_SHIFT))
#define M2M_GET_MINOR(ver_info_hword) ((uint8)((ver_info_hword) >> M2M_MINOR_SHIFT))
#define M2M_GET_PATCH(ver_info_hword) ((uint8)((ver_info_hword) >> M2M_PATCH_SHIFT))
#define M2M_GET_FW_VER(ver_info_word) ((uint16)((ver_info_word) >> M2M_FW_VERSION_SHIFT))
#define M2M_GET_DRV_VER(ver_info_word) ((uint16)((ver_info_word) >> M2M_DRV_VERSION_SHIFT))
#define M2M_MAKE_VERSION(major, minor, patch)                                    |
```
#define M2M_MAKE_VERSION_INFO(fw_major, fw_minor, fw_patch, drv_major, drv_minor, drv_patch)

#define REL_19_5_2_VER M2M_MAKE_VERSION_INFO(19,5,2,19,3)
#define REL_19_5_1_VER M2M_MAKE_VERSION_INFO(19,5,1,19,3)
#define REL_19_5_0_VER M2M_MAKE_VERSION_INFO(19,5,0,19,3)
#define REL_19_4_6_VER M2M_MAKE_VERSION_INFO(19,4,6,19,3)
#define REL_19_4_5_VER M2M_MAKE_VERSION_INFO(19,4,5,19,3)
#define REL_19_4_4_VER M2M_MAKE_VERSION_INFO(19,4,4,19,3)
#define REL_19_4_3_VER M2M_MAKE_VERSION_INFO(19,4,3,19,3)
#define REL_19_4_2_VER M2M_MAKE_VERSION_INFO(19,4,2,19,3)
#define REL_19_4_1_VER M2M_MAKE_VERSION_INFO(19,4,1,19,3)
#define REL_19_4_0_VER M2M_MAKE_VERSION_INFO(19,4,0,19,3)
#define REL_19_3_1_VER M2M_MAKE_VERSION_INFO(19,3,1,19,3)
#define REL_19_3_0_VER M2M_MAKE_VERSION_INFO(19,3,0,19,3)
#define REL_19_2_2_VER M2M_MAKE_VERSION_INFO(19,2,2,19,3)
#define REL_19_2_1_VER M2M_MAKE_VERSION_INFO(19,2,1,19,3)
#define REL_19_2_0_VER M2M_MAKE_VERSION_INFO(19,2,0,19,3)
#define REL_19_1_0_VER M2M_MAKE_VERSION_INFO(19,1,0,18,3)
#define REL_19_0_0_VER M2M_MAKE_VERSION_INFO(19,0,0,18,1)
#define M2M_RELEASE_VERSION_MAJOR_NO (19)
#define M2M_RELEASE_VERSION_MINOR_NO (5)
#define M2M_RELEASE_VERSION_PATCH_NO (2)
#define M2M_RELEASE_VERSION_SVN_VERSION (SVN_REVISION)
#define M2M_MIN_REQ_DRV_VERSION_MAJOR_NO (19)
#define M2M_MIN_REQ_DRV_VERSION_MINOR_NO (3)
#define M2M_MIN_REQ_DRV_VERSION_PATCH_NO (0)
#define M2M_MIN_REQ_DRV_SVN_VERSION (0)
#define M2M_BUFFER_MAX_SIZE (1600UL - 4)
#define M2M_MAC_ADDRES_LEN 6
#define M2M_ETHERNET_HDR_OFFSET 34
#define M2M_ETHERNET_HDR_LEN 14
#define M2M_MAX_SSID_LEN 33
#define M2M_MAX_PSK_LEN 65
#define M2M_MIN_PSK_LEN 9
#define M2M_DEVICE_NAME_MAX 48
#define M2M_LISTEN_INTERVAL 1
#define MAX_HIDDEN_SITES 4
#define M2M_1X_USR_NAME_MAX 21
#define M2M_1X_PWD_MAX 41

#define M2M_CUST_IE_LEN_MAX 252

#define PWR_DEFAULT PWR_HIGH

#define M2M_CONFIG_CMD_BASE 1

#define M2M_STA_CMD_BASE 40

#define M2M_AP_CMD_BASE 70

#define M2M_P2P_CMD_BASE 90

#define M2M_SERVER_CMD_BASE 100

#define M2M_OTA_CMD_BASE 100

#define M2M_CRYPTO_CMD_BASE 1

#define M2M_MAX_GRP_NUM_REQ (127)

#define WEP_40_KEY_STRING_SIZE ((uint8)10)

#define WEP_104_KEY_STRING_SIZE ((uint8)26)

#define WEP_KEY_MAX_INDEX ((uint8)4)

#define M2M_SHA256_CONTEXT_BUFF_LEN (128)

#define M2M_SCAN_DEFAULT_NUM_SLOTS (2)

#define M2M_SCAN_DEFAULT_SLOT_TIME (30)

#define M2M_SCAN_DEFAULT_NUM_PROBE (2)

#define TLS_FILE_NAME_MAX 48
#define TLS_SRV_SEC_MAX_FILES 8

#define TLS_SRV_SEC_START_PATTERN_LEN 8

#define OTA_STATUS_VALID (0x12526285)

#define OTA_STATUS_INVALID (0x23987718)

#define OTA_MAGIC_VALUE (0x1ABCDEF9)

#define M2M_MAGIC_APP (0xef52f61UL)

#define OTA_FORMAT_VER_0 (0) /* Till 19.2.2 format */

#define OTA_FORMAT_VER_1 (1) /* Starting from 19.3.0 CRC is used */

#define OTA_SHA256_DIGEST_SIZE (32)

#define TLS_CRL_DATA_MAX_LEN 64

#define TLS_CRL_MAX_ENTRIES 10

#define TLS_CRL_TYPE_NONE 0

#define TLS_CRL_TYPE_CERT_HASH 1
Enumerations

tenuM2mDefaultConnErrcode {
M2M_DEFAULT_CONN_INPROGRESS = ((sint8)-23),
M2M_DEFAULT_CONN_FAIL,
M2M_DEFAULT_CONN_SCAN_MISMATCH,
M2M_DEFAULT_CONN_EMPTY_LIST }

Detailed Description

Macro Definition Documentation
◆ M2M_MAJOR_SHIFT

#define M2M_MAJOR_SHIFT (8)
M2M_MINOR_SHIFT

#define M2M_MINOR_SHIFT (4)
#define M2M_PATCH_SHIFT (0)
#define M2M_DRV_VERSION_SHIFT (16)
M2M_FW_VERSION_SHIFT

#define M2M_FW_VERSION_SHIFT (0)
#define M2M_GET_MAJOR ((uint8)((ver_info_hword) >> M2M_MAJOR_SHIFT) & 0xff)
#define M2M_GET_MINOR ((uint8)((ver_info_hword) >> M2M_MINOR_SHIFT) & 0x0f)
#define M2M_GET_PATCH ((uint8)((ver_info_hword) >> M2M_PATCH_SHIFT) & 0x0f)
M2M_GET_FW_VER

```c
#define M2M_GET_FW_VER((uint16)((ver_info_word) >>= M2M_FW_VERSION_SHIFT))
```
#define M2M_GET_DRV_VER (((uint16)((ver_info_word) >> M2M_DRV_VERSION_SHIFT)) M2M_DRV_VERSION_SHIFT)
#define M2M_GET_DRV_MAJOR (ver_info_word)
M2M_GET_MAJOR(M2M_GET_DRV_MAJOR(M2M_GET_DRV_MAJOR_MAJ))
M2M_GET_DRV_MINOR

#define M2M_GET_DRV_MINOR (ver_info_word) M2M_GET_MINOR(M2M_
◆ M2M_GET_DRV_PATCH

```c
#define M2M_GET_DRV_PATCH ( ver_info_word ) M2M_GET_PATCH( M2M_...
```
#define M2M_GET_FW_MAJOR ( ver_info_word ) M2M_GET_MAJOR( M2M_
#define M2M_GET_FW_MINOR
M2M_GET_FW_MINOR ( ver_info_word )
M2M_GET_MINOR(M2M_0
◆ M2M_GET_FW_PATCH

```c
#define M2M_GET_FW_PATCH ( ver_info_word ) M2M_GET_PATCH(M2M_GET_FW_VER)
```
```c
#define M2M_MAKE_VERSION(major, minor, patch)

Value:

```
\((\text{uint16})(\text{major} \& 0xff) \ll \text{M2M_MAJOR_SHIFT}) \\
| \text{\ll} \\
(\text{uint16})(\text{minor} \& 0x0f) \ll \text{M2M_MINOR_SHIFT}) \\
| \text{\ll} \\
(\text{uint16})(\text{patch} \& 0x0f) \ll \text{M2M_PATCH_SHIFT})```
```
#define M2M_MAKE_VERSION_INFO(fw_major, fw_minor, fw_patch, drv_major, drv_minor, drv_patch)

Value:

\[
( ((\text{uint32})\text{M2M\_MAKE\_VERSION}((fw\text{\_major}), (fw\text{\_minor}), (fw\text{\_patch}))) \ll \text{M2M\_FW\_VERSION\_SHIFT}) | \ll
( ((\text{uint32})\text{M2M\_MAKE\_VERSION}((drv\text{\_major}), (drv\text{\_minor}), (drv\text{\_patch}))) \ll \text{M2M\_DRV\_VERSION\_SHIFT}))
\]
#define REL_19_5_2_VER  M2M_MAKE_VERSION_INFO(19,5,2,19,3,0)
```c
#define REL_19_5_1_VER M2M_MAKE_VERSION_INFO(19,5,1,19,3,0)
```
#define REL_19_5_0_VER M2M_MAKE_VERSION_INFO(19,5,0,19,3,0)
#define REL_19_4_6_VER M2M_MAKE_VERSION_INFO(19,4,6,19,3,0)
#define REL_19_4_5_VER M2M_MAKE_VERSION_INFO(19,4,5,19,3,0)
#define REL_19_4_4_VER M2M_MAKE_VERSION_INFO(19,4,4,19,3,0)
#define REL_19_4_3_VER M2M_MAKE_VERSION_INFO(19,4,3,19,3,0)
#define REL_19_4_2_VER M2M_MAKE_VERSION_INFO(19,4,2,19,3,0)
#define REL_19_4_1_VER M2M_MAKE_VERSION_INFO(19,4,1,19,3,0)
#define REL_19_4_0_VER M2M_MAKE_VERSION_INFO(19,4,0,19,3,0)
#define REL_19_3_1_VER M2M_MAKE_VERSION_INFO(19,3,1,19,3,0)
#define REL_19_3_0_VER M2M_MAKE_VERSION_INFO(19,3,0,19,3,0)
<table>
<thead>
<tr>
<th>REL_19_2_2_VER</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define REL_19_2_2_VER M2M_MAKE_VERSION_INFO(19,2,2,19,2,0)</td>
</tr>
</tbody>
</table>
◆ REL_19_2_1_VER

```c
#define REL_19_2_1_VER M2M_MAKE_VERSION_INFO(19,2,1,19,2,0)
```
#define REL_19_2_0_VER M2M_MAKE_VERSION_INFO(19,2,0,19,2,0)
#define REL_19_1_0_VER M2M_MAKE_VERSION_INFO(19,1,0,18,2,0)
#define REL_19_0_0_VER M2M_MAKE_VERSION_INFO(19,0,0,18,1,1)
#define M2M_RELEASE_VERSION_MAJOR_NO (19)

Firmware Major release version number.
M2M_RELEASE_VERSION_MINOR_NO

#define M2M_RELEASE_VERSION_MINOR_NO (5)

Firmware Minor release version number.
M2M_RELEASE_VERSION_PATCH_NO

#define M2M_RELEASE_VERSION_PATCH_NO (2)

Firmware patch release version number.
#define M2M_RELEASE_VERSION_SVN_VERSION (SVN_REVISION)

Firmware SVN release version number.
#define M2M_MIN_REQ_DRV_VERSION_MAJOR_NO  (19)

Driver Major release version number.
M2M_MIN_REQ_DRV_VERSION_MINOR_NO

#define M2M_MIN_REQ_DRV_VERSION_MINOR_NO (3)

Driver Minor release version number.
◆ M2M_MIN_REQ_DRV_VERSION_PATCH_NO

#define M2M_MIN_REQ_DRV_VERSION_PATCH_NO (0)

Driver patch release version number.
M2M_MIN_REQ_DRV_SVN_VERSION

#define M2M_MIN_REQ_DRV_SVN_VERSION (0)

Driver svn version.
**M2M_BUFFER_MAX_SIZE**

```c
#define M2M_BUFFER_MAX_SIZE (1600UL - 4)
```

Maximum size for the shared packet buffer.
#define M2M_MAC_ADDRES_LEN 6

The size for 802 MAC address.
M2M_ETHERNET_HDR_OFFSET

#define M2M_ETHERNET_HDR_OFFSET 34

The offset of the Ethernet header within the WLAN Tx Buffer.
M2M_Ethernet_HDR_LEN

#define M2M_Ethernet_HDR_LEN 14

Length of the Etherenet header in bytes.
◆ M2M_MAX_SSID_LEN

#define M2M_MAX_SSID_LEN 33

Maximum size for the Wi-Fi SSID including the NULL termination.
**M2M_MAX_PSK_LEN**

```c
#define M2M_MAX_PSK_LEN 65
```

Maximum size for the WPA PSK including the NULL termination.
M2M_MIN_PSK_LEN

#define M2M_MIN_PSK_LEN 9

Maximum size for the WPA PSK including the NULL termination.
M2M_DEVICE_NAME_MAX

#define M2M_DEVICE_NAME_MAX 48

Maximum Size for the device name including the NULL termination.
M2M_LISTEN_INTERVAL

#define M2M_LISTEN_INTERVAL 1

The STA uses the Listen Interval parameter to indicate to the AP how many beacon intervals it shall sleep before it retrieves the queued frames from the AP.
MAX_HIDDEN_SITES

#define MAX_HIDDEN_SITES 4

max number of hidden SSID supported by scan request
M2M_1X_USR_NAME_MAX

#define M2M_1X_USR_NAME_MAX 21

The maximum size of the user name including the NULL termination. It is used for RADIUS authentication in case of connecting the device to an AP secured with WPA-Enterprise.
◆ M2M_1X_PWD_MAX

#define M2M_1X_PWD_MAX 41

The maximum size of the password including the NULL termination. It is used for RADIUS authentication in case of connecting the device to an AP secured with WPA-Enterprise.
**M2M_CUST_IE_LEN_MAX**

```c
#define M2M_CUST_IE_LEN_MAX 252
```

The maximum size of IE (Information Element).
PWR_DEFAULT

#define PWR_DEFAULT PWR_HIGH
M2M_CONFIG_CMD_BASE

#define M2M_CONFIG_CMD_BASE 1

The base value of all the host configuration commands opcodes.
M2M_STA_CMD_BASE

#define M2M_STA_CMD_BASE  40

The base value of all the station mode host commands opcodes.
M2M_AP_CMD_BASE

#define M2M_AP_CMD_BASE 70

The base value of all the Access Point mode host commands opcodes.
**M2M_P2P_CMD_BASE**

```c
#define M2M_P2P_CMD_BASE 90
```

The base value of all the P2P mode host commands opcodes.
**M2M_SERVER_CMD_BASE**

```c
#define M2M_SERVER_CMD_BASE 100
```

The base value of all the power save mode host commands codes.
#define M2M_OTA_CMD_BASE  100

The base value of all the OTA mode host commands opcodes. The OTA have special group so can extended from 1-M2M_MAX_GRP_NUM_REQ
**M2M_CRYPTO_CMD_BASE**

```c
#define M2M_CRYPTO_CMD_BASE 1
```

The base value of all the crypto mode host commands opcodes. The crypto have special group so can extended from 1-M2M_MAX_GRP_NUM_REQ
<table>
<thead>
<tr>
<th>M2M_MAX_GRP_NUM_REQ</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define M2M_MAX_GRP_NUM_REQ  (127)</code></td>
</tr>
</tbody>
</table>

max number of request in one group equal to 127 as the last bit reserved for config or data pkt
```c
#define WEP_40_KEY_STRING_SIZE ((uint8)10)
```

Indicate the wep key size in bytes for 40 bit string passphrase.
# WEP_104_KEY_STRING_SIZE

```c
#define WEP_104_KEY_STRING_SIZE 
	(uint8)26
```

Indicate the wep key size in bytes for 104 bit string passphrase.
◆ WEP_KEY_MAX_INDEX

```c
#define WEP_KEY_MAX_INDEX ((uint8)4)
```

Indicate the max key index value for WEP authentication
◆ M2M_SHA256_CONTEXT_BUFF_LEN

#define M2M_SHA256_CONTEXT_BUFF_LEN (128)

sha256 context size
M2M_SCAN_DEFAULT_NUM_SLOTS

#define M2M_SCAN_DEFAULT_NUM_SLOTS (2)

The default number of scan slots performed by the WINC board.
#define M2M_SCAN_DEFAULT SLOT_TIME (30)

The default duration in milliseconds of a scan slots performed by the WINC board.
#define M2M_SCAN_DEFAULT_NUM_PROBE (2)

The default number of scan slots performed by the WINC board.
**TLS_FILE_NAME_MAX**

```
#define TLS_FILE_NAME_MAX 48
```

Maximum length for each TLS certificate file name including null terminator.
**TLS_SRV_SEC_MAX_FILES**

```c
#define TLS_SRV_SEC_MAX_FILES 8
```

Maximum number of certificates allowed in TLS_SRV section.
#define TLS_SRV_SEC_START_PATTERN_LEN 8

Length of certificate struct start pattern.
OTA_STATUS_VALID

#define OTA_STATUS_VALID  (0x12526285)

Magic value updated in the Control structure in case of ROLLACK image Valid
OTA_STATUS_INVALID

#define OTA_STATUS_INVALID (0x23987718)

Magic value updated in the Control structure in case of ROLLACK image Invalid
OTA_MAGIC_VALUE

#define OTA_MAGIC_VALUE  (0x1ABCDEF9)

Magic value set at the beginning of the OTA image header
Define M2M_MAGIC_APP (0xef522f61UL)

Magic value set at the beginning of the Cortus OTA image header
OTA_FORMAT_VER_0

#define OTA_FORMAT_VER_0 (0) /* Till 19.2.2 format*/
OTA_FORMAT_VER_1

#define OTA_FORMAT_VER_1 (1) /*starting from 19.3.0 CRC is used and sequence number is used*/

Control structure format version
OTA_SHA256_DIGEST_SIZE

#define OTA_SHA256_DIGEST_SIZE (32)

Sha256 digest size in the OTA image, the sha256 digest is set at the beginning of image before the OTA header
#define TLS_CRL_DATA_MAX_LEN 64
#define TLS_CRL_MAX_ENTRIES 10
#define TLS_CRL_TYPE_NONE 0
#define TLS_CRL_TYPE_CERT_HASH 1
Enumeration Type Documentation
# tenuM2mDefaultConnErrcode

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_DEFAULT_CONN_INPROGRESS</td>
<td>A failure that indicates that a default connection or forced connection is in progress.</td>
</tr>
<tr>
<td>M2M_DEFAULT_CONN_FAIL</td>
<td>A failure response that indicates that the winc failed to connect to the cached network.</td>
</tr>
<tr>
<td>M2M_DEFAULT_CONN_SCAN_MISMATCH</td>
<td>A failure response that indicates that no one of the cached networks was found in the scan results, as a result to the function call m2m_default_connect.</td>
</tr>
<tr>
<td>M2M_DEFAULT_CONN_EMPTY_LIST</td>
<td>A failure response that indicates an empty network list as a result to the function call m2m_default_connect.</td>
</tr>
</tbody>
</table>
CommonDefines
WLAN » Defines
### Macros

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_TIME_OUT_DELAY</td>
<td>10000</td>
</tr>
<tr>
<td>M2M_SUCCESS</td>
<td>(sint8)0</td>
</tr>
<tr>
<td>M2M_ERR_SEND</td>
<td>(sint8)-1</td>
</tr>
<tr>
<td>M2M_ERR_RCV</td>
<td>(sint8)-2</td>
</tr>
<tr>
<td>M2M_ERR_MEM_ALLOC</td>
<td>(sint8)-3</td>
</tr>
<tr>
<td>M2M_ERR_TIME_OUT</td>
<td>(sint8)-4</td>
</tr>
<tr>
<td>M2M_ERR_INIT</td>
<td>(sint8)-5</td>
</tr>
<tr>
<td>M2M_ERR_BUS_FAIL</td>
<td>(sint8)-6</td>
</tr>
<tr>
<td>M2M_NOT_YET</td>
<td>(sint8)-7</td>
</tr>
<tr>
<td>M2M_ERR_FIRMWARE</td>
<td>(sint8)-8</td>
</tr>
<tr>
<td>M2M_SPI_FAIL</td>
<td>(sint8)-9</td>
</tr>
<tr>
<td>M2M_ERR_FIRMWARE_bURN</td>
<td>(sint8)-10</td>
</tr>
<tr>
<td>M2M_ACK</td>
<td>(sint8)-11</td>
</tr>
<tr>
<td>M2M_ERR_FAIL</td>
<td>(sint8)-12</td>
</tr>
<tr>
<td>M2M_ERR_FW_VER_MISMATCH</td>
<td>(sint8)-13</td>
</tr>
<tr>
<td>M2M_ERR_SCAN_IN_PROGRESS</td>
<td>(sint8)-14</td>
</tr>
<tr>
<td>M2M_ERR_INVALID_ARG</td>
<td>(sint8)-15</td>
</tr>
</tbody>
</table>
#define M2M_ERR_INVALID ((sint8)-16)

#define I2C_ERR_LARGE_ADDRESS 0xE1UL /*the address exceed the max addressing mode in i2c flash*/

#define I2C_ERR_TX_ABRT 0xE2UL /*NO ACK from slave*/

#define I2C_ERR_OVER_SIZE 0xE3UL

#define ERR_PREFIX_NMIS 0xE4UL /*wrong first four byte in flash NMIS*/

#define ERR_FIRMWARE_EXCEED_SIZE 0xE5UL /*Total size of firmware exceed the max size 256k*/

#define PROGRAM_START 0x26961735UL

#define BOOT_SUCCESS 0x10add09eUL

#define BOOT_START 0x12345678UL

#define NBIT31 (0x80000000)

#define NBIT30 (0x40000000)

#define NBIT29 (0x20000000)

#define NBIT28 (0x10000000)

#define NBIT27 (0x08000000)

#define NBIT26 (0x04000000)

#define NBIT25 (0x02000000)

#define NBIT24 (0x01000000)
#define NBIT23 (0x00800000)
#define NBIT22 (0x00400000)
#define NBIT21 (0x00200000)
#define NBIT20 (0x00100000)
#define NBIT19 (0x00080000)
#define NBIT18 (0x00040000)
#define NBIT17 (0x00020000)
#define NBIT16 (0x00010000)
#define NBIT15 (0x00008000)
#define NBIT14 (0x00004000)
#define NBIT13 (0x00002000)
#define NBIT12 (0x00001000)
#define NBIT11 (0x00000800)
#define NBIT10 (0x00000400)
#define NBIT9  (0x00000200)
#define NBIT8  (0x00000100)
#define NBIT7  (0x00000080)
#define NBIT6  (0x00000040)
#define NBIT5  (0x00000020)
```c
#define NBIT4  (0x00000010)
#define NBIT3  (0x00000008)
#define NBIT2  (0x00000004)
#define NBIT1  (0x00000002)
#define NBIT0  (0x00000001)

#define M2M_MAX(A, B)  ((A) > (B) ? (A) : (B))
#define M2M_SEL(x, m1, m2, m3)  ((x>1)?((x>2)?(m3):(m2)):(m1))
#define WORD_ALIGN(val)  (((val) & 0x03) ? ((val) + 4 - ((val) & 0x03)) : (val))

#define DATA_PKT_OFFSET  4
#define BYTE_0(word)  (((uint8)(((word) >> 0) & 0x000000FFUL))
#define BYTE_1(word)  (((uint8)(((word) >> 8) & 0x000000FFUL))
#define BYTE_2(word)  (((uint8)(((word) >> 16) & 0x000000FFUL))
#define BYTE_3(word)  (((uint8)(((word) >> 24) & 0x000000FFUL))
```
#define M2M_TIME_OUT_DELAY 10000
M2M_SUCCESS

```c
#define M2M_SUCCESS ((sint8)0)
```
#define M2M_ERR_SEND ((sint8)-1)
#define M2M_ERR_RCV ((sint8)-2)
M2M_ERR_MEM_ALLOC

```c
#define M2M_ERR_MEM_ALLOC ((sint8)-3)
```
◆ M2M_ERR_TIME_OUT

#define M2M_ERR_TIME_OUT ((sint8)-4)
#define M2M_ERR_INIT ((sint8)-5)
◆M2M_ERR_BUS_FAIL

```c
#define M2M_ERR_BUS_FAIL ((sint8)-6)
```
◆ M2M_NOT_YET

```c
#define M2M_NOT_YET ((sint8)-7)
```
#define M2M_ERR_FIRMWARE ((sint8)-8)
◆ M2M_SPI_FAIL

#define M2M_SPI_FAIL ((sint8)-9)
◆ M2M_ERR_FIRMWARE burned

#define M2M_ERR_FIRMWARE_bURN ((sint8)-10)
M2M_ACK

#define M2M_ACK ((sint8)-11)
#define M2M_ERR_FAIL ((sint8)-12)
#define M2M_ERR_FW_VER_MISMATCH ((sint8)-13)
M2M_ERRSCANINPROGRESS

#define M2M_ERR_SCAN_IN_PROGRESS ((sint8)-14)
M2M_ERR_INVALID_ARG

#define M2M_ERR_INVALID_ARG ((sint8)-15)
◆ M2M_ERR_INVALID

#define M2M_ERR_INVALID ((sint8)-16)
I2C_ERR_LARGE_ADDRESS

#define I2C_ERR_LARGE_ADDRESS 0xE1UL /*the address exceed the max addressing mode in i2c flash*/
I2C_ERR_TX_ABRT

#define I2C_ERR_TX_ABRT 0xE2UL /* NO ACK from slave */
I2C_ERR_OVER_SIZE

#define I2C_ERR_OVER_SIZE 0xE3UL
ADDR

#define ERR_PREFIX_NMIS	0xE4UL /*wrong first four byte in flash NMIS*/
#define ERR_FIRMEWARE_EXCEED_SIZE 0xE5UL /*Total size of firmware exceed the max size 256k*/
#define PROGRAM_START 0x26961735UL
#define BOOT_SUCCESS 0x10add09eUL
#define BOOT_START 0x12345678UL
#define NBIT31 (0x80000000)
NBIT30

#define NBIT30 (0x40000000)
#define NBIT29 (0x20000000)
#define NBIT28 (0x10000000)
#define NBIT27 (0x08000000)
#define NBIT26 (0x04000000)
#define NBIT25 (0x02000000)
<table>
<thead>
<tr>
<th>NBIT24</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define NBIT24 (0x01000000)</code></td>
</tr>
</tbody>
</table>
#define NBIT23   (0x00800000)
#define NBIT22  (0x00400000)
#define NBIT21 (0x00200000)
#define NBIT20 (0x00100000)
◆ NBIT19

#define NBIT19   (0x00080000)
#define NBIT18 (0x00040000)
◆ NBIT17

```c
#define NBIT17 (0x00020000)
```
NBIT16

#define NBIT16  (0x00010000)
#define NBIT15 (0x00008000)
◆ NBIT14

#define NBIT14  (0x00004000)
NBIT13

#define NBIT13  (0x00002000)
#define NBIT12  (0x00001000)
#define NBIT11 (0x00000800)
◆ NBIT10

```c
#define NBIT10 (0x00000400)
```
#define NBIT9   (0x00000200)
NBIT8

#define NBIT8 (0x00000100)
#define NBIT7  (0x00000080)
◆ NBIT6

#define NBIT6  (0x00000040)
#define NBIT5 (0x00000020)
◆ NBIT4

#define NBIT4 (0x00000010)
#define NBIT3 (0x00000008)
#define NBIT2 (0x00000004)
◆ NBIT1

#define NBIT1 (0x00000002)
#define NBIT0 (0x00000001)
#define M2M_MAX ( A, B ) ((A) > (B) ? (A) : (B))
#define M2M_SEL ( x, m1, m2, m3 ) ((x>1)?((x>2)?(m3):(m2)):(m1))
```c
#define WORD_ALIGN
WORD_ALIGN (val) : ((val) & 0x03) ? ((val) + 4 - ((val) & 0x03)) : (val)
```
DATA_PKT_OFFSET

#define DATA_PKT_OFFSET 4
◆ BYTE_0

```c
#define BYTE_0 ( word ) ((uint8)(((word) >> 0) & 0x000000FFUL))
```
#define BYTE_1 ( word ) ((uint8)(((word) >> 8 ) & 0x000000FFUL))
```c
#define BYTE_2 ( word ) ((uint8)(((word) >> 16) & 0x000000FFUL))
```
#define BYTE_3 (word) ((uint8)(((word) >> 24) & 0x000000FFUL))
<table>
<thead>
<tr>
<th>Function</th>
<th>WLAN</th>
</tr>
</thead>
</table>

WINC1500
IoT Software
APIs  19.5.2
WINC Software API Reference Manual
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_ota_init</td>
</tr>
<tr>
<td>m2m_ota_notif_set_url</td>
</tr>
<tr>
<td>m2m_ota_notif_check_for_update</td>
</tr>
<tr>
<td>m2m_ota_notif_sched</td>
</tr>
<tr>
<td>m2m_ota_start_update</td>
</tr>
<tr>
<td>m2m_ota_rollback</td>
</tr>
<tr>
<td>m2m_ota_abort</td>
</tr>
<tr>
<td>m2m_ota_switch_firmware</td>
</tr>
<tr>
<td>m2m_wifi_download_mode</td>
</tr>
<tr>
<td>m2m_wifi_init</td>
</tr>
<tr>
<td>m2m_wifi_deinit</td>
</tr>
<tr>
<td>m2m_wifi_handle_events</td>
</tr>
<tr>
<td>m2m_wifi_send_crl</td>
</tr>
<tr>
<td>m2m_wifi_default_connect</td>
</tr>
<tr>
<td>m2m_wifi_connect</td>
</tr>
<tr>
<td>m2m_wifi_disconnect</td>
</tr>
<tr>
<td>m2m_wifi_start_provision_mode</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>m2m_wifi_stop_provision_mode</td>
</tr>
<tr>
<td>m2m_wifi_get_connection_info</td>
</tr>
<tr>
<td>m2m_wifi_set_mac_address</td>
</tr>
<tr>
<td>m2m_wifi_wps</td>
</tr>
<tr>
<td>m2m_wifi_wps_disable</td>
</tr>
<tr>
<td>m2m_wifi_p2p</td>
</tr>
<tr>
<td>m2m_wifi_p2p_disconnect</td>
</tr>
<tr>
<td>m2m_wifi_enable_ap</td>
</tr>
<tr>
<td>m2m_wifi_disable_ap</td>
</tr>
<tr>
<td>m2m_wifi_set_static_ip</td>
</tr>
<tr>
<td>m2m_wifi_request_dhcp_client</td>
</tr>
<tr>
<td>m2m_wifi_request_dhcp_server</td>
</tr>
<tr>
<td>m2m_wifi_enable_dhcp</td>
</tr>
<tr>
<td>m2m_wifi_set_scan_options</td>
</tr>
<tr>
<td>m2m_wifi_set_scan_region</td>
</tr>
<tr>
<td>m2m_wifi_request_scan</td>
</tr>
<tr>
<td>m2m_wifi_get_num_ap_found</td>
</tr>
<tr>
<td>m2m_wifi_req_scan_result</td>
</tr>
<tr>
<td>m2m_wifi_req_curr_rssi</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td><code>m2m_wifi_get_otp_mac_address</code></td>
</tr>
<tr>
<td><code>m2m_wifi_get_mac_address</code></td>
</tr>
<tr>
<td><code>m2m_wifi_set_sleep_mode</code></td>
</tr>
<tr>
<td><code>m2m_wifi_request_sleep</code></td>
</tr>
<tr>
<td><code>m2m_wifi_get_sleep_mode</code></td>
</tr>
<tr>
<td><code>m2m_wifi_req_client_ctrl</code></td>
</tr>
<tr>
<td><code>m2m_wifi_req_server_init</code></td>
</tr>
<tr>
<td><code>m2m_wifi_set_device_name</code></td>
</tr>
<tr>
<td><code>m2m_wifi_set_lsn_int</code></td>
</tr>
<tr>
<td><code>m2m_wifi_enable_monitoring_mode</code></td>
</tr>
<tr>
<td><code>m2m_wifi_disable_monitoring_mode</code></td>
</tr>
<tr>
<td><code>m2m_wifi_send_wlan_pkt</code></td>
</tr>
<tr>
<td><code>m2m_wifi_send_ethernet_pkt</code></td>
</tr>
<tr>
<td><code>m2m_wifi_enable_sntp</code></td>
</tr>
<tr>
<td><code>m2m_wifi_set_system_time</code></td>
</tr>
<tr>
<td><code>m2m_wifi_get_system_time</code></td>
</tr>
<tr>
<td><code>m2m_wifi_set_cust_InfoElement</code></td>
</tr>
<tr>
<td><code>m2m_wifi_set_power_profile</code></td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>m2m_wifi_set_tx_power</td>
</tr>
<tr>
<td>m2m_wifi_enable_firmware_logs</td>
</tr>
<tr>
<td>m2m_wifi_set_battery_voltage</td>
</tr>
<tr>
<td>m2m_wifi_set_gains</td>
</tr>
<tr>
<td>m2m_wifi_get_firmware_version</td>
</tr>
<tr>
<td>m2m_wifi_prng_get_random_bytes</td>
</tr>
</tbody>
</table>
### Functions

| NMI_API  | sint8 | m2m_ota_init (tpfOtaUpdateCb pfOtaUpdateCb, tpfOtaNotifCb pfOtaNotifCb) |
Detailed Description

Synchronous initialization function for the OTA layer by registering the update callback. The notification callback is not supported at the current version. Calling this API is a MUST for all the OTA API's.
**m2m_ota_init()**

```c
NMI_API sint8 m2m_ota_init (tpfOtaUpdateCb pfOtaUpdateCb, tpfOtaNotifCb pfOtaNotifCb)
```

**Parameters**

- `[in] pfOtaUpdateCb` OTA Update callback function
- `[in] pfOtaNotifCb` OTA notify callback function

**Returns**

The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m OTA notification set URL</td>
</tr>
</tbody>
</table>

WLAN » Function
Functions

NMI_API sint8 m2m_ota_notif_set_url (uint8 *u8Url)
Detailed Description

Set the OTA notification server URL, the functions need to be called before any check for update
Function Documentation
**m2m_ota_notif_set_url()**

```c
NMI_API sint8 m2m_ota_notif_set_url ( uint8 * u8Url )
```

**Parameters**

[in] **u8Url**
Set the OTA notification server URL, the functions need to be called before any check for update.

**Warning**
Calling m2m_ota_init is required Notification Server is not supported in the current version (function is not implemented)

**See also**
m2m_ota_init

**Returns**
The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
m2m_ota_notif_check_for_update

WLAN » Function
### Functions

| NMI_API sint8 m2m_ota_notif_check_for_update (void) |  |
Detailed Description

Synchronous function to check for the OTA update using the Notification Server URL. Function is not implemented (not supported at the current version)
Function Documentation
◆ m2m_ota_notif_check_for_update()

NMI_API sint8 m2m_ota_notif_check_for_update ( void )

Warning
Function is not implemented (not supported at the current version)

See also
m2m_ota_init m2m_ota_notif_set_url

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_ota_notif_sched

WLAN » Function
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8  m2m OTA notif sched (uint32 u32Period)</td>
</tr>
</tbody>
</table>
Detailed Description

Schedule OTA notification Server check for update request after specific number of days
Function Documentation
**m2m_ota_notif_sched()**

```c
NMI_API sint8 m2m_ota_notif_sched ( uint32 u32Period )
```

**Parameters**

[in] `u32Period` Period in days

**See also**

- `m2m_ota_init`
- `m2m_ota_notif_check_for_update`
- `m2m_ota_notif_set_url`

**Returns**

The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by `doxygen` 1.8.13
m2m_ota_start_update

WLAN » Function
Functions

NMI_API sint8 m2m_ota_start_update (uint8 *u8DownloadUrl)

NMI_API sint8 m2m_ota_start_update_crt (uint8 *u8DownloadUrl)
Detailed Description

Request OTA start update using the downloaded URL, the OTA module will download the OTA image and ensure integrity of the image, and update the validity of the image in control structure. Switching to that image requires calling `m2m_ota_switch_firmware` API. As a prerequisite `m2m_ota_init` should be called before using `m2m_ota_start()`.  

Request OTA start for cortus application image using the downloaded URL, the OTA module will download the OTA image and ensure integrity of the image, and update the validity of the image in control structure. Switching to that image requires calling `m2m_ota_switch_crt` API. As a prerequisite `m2m_ota_init` should be called before using `m2m_ota_start_update_crt()`.
Function Documentation
m2m_ota_start_update()

NMI_API sint8 m2m_ota_start_update ( uint8 * u8DownloadUrl )

Parameters

[in] u8DownloadUrl The download firmware URL, you get it from device info according to the application server

Warning

Calling this API does not guarantee OTA WINC image update. It depends on the connection with the download server and the validity of the image. If the API response is failure this may invalidate the roll-back image if it was previously valid, since the WINC does not have any internal memory except the flash roll-back image location to validate the downloaded image from

See also

m2m_ota_init tpfOtaUpdateCb

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
Example

The example shows an example of how the OTA image update is carried out.

```c
static void OtaUpdateCb(uint8 u8OtaUpdateStatusType,
                        uint8 u8OtaUpdateStatus)
{
    if(u8OtaUpdateStatusType == DL_STATUS) {
        if(u8OtaUpdateStatus == OTA_STATUS_SUCSESS) {
            //switch to the upgraded firmware
            m2m_ota_switch_firmware();
        }
    }
    else if(u8OtaUpdateStatusType == SW_STATUS) {
        if(u8OtaUpdateStatus == OTA_STATUS_SUCSESS) {
            M2M_INFO("Now OTA successfully done");
            //start the host SW upgrade then system reset is
            required (Reinitialize the driver)
        }
    }
}

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg) {
    case M2M_WIFI_REQ_DHCP_CONF:
    {
        //after successfully connection, start the over
        air upgrade
        m2m_ota_start_update(OTA_URL);
    }
    break;
    default:
    break;
}
int main (void)
```
{ 
  tstrWifiInitParam param;
  tstr1xAuthCredentials gstrCred1x = AUTH_CREDENTIALS;
  nm_bsp_init();

  m2m_memset((uint8*) &param, 0, sizeof(param));
  param.pfAppWifiCb = wifi_event_cb;

  //Initialize the WINC Driver
  ret = m2m_wifi_init(&param);
  if (M2M_SUCCESS != ret)
  {
    M2M_ERR("Driver Init Failed <\%d>\n", ret);
    while(1);
  }

  //Initialize the OTA module
  m2m_ota_init(OtaUpdateCb, NULL);
  //connect to AP that provide connection to the OTA server
  m2m_wifi_default_connect();

  while(1)
  {

    //Handle the app state machine plus the WINC event handler
    while(m2m_wifi_handle_events(NULL) != M2M_SUCCESS)
    {
    }
  }
}
m2m_ota_start_update_crt()

NMI_API sint8 m2m_ota_start_update_crt (uint8 * u8DownloadUrl)

Parameters

[in] u8DownloadUrl The cortus application image url.

Warning

Calling this API does not guarantee cortus application image update, It depends on the connection with the download server and the validity of the image. If the API response is failure this may invalidate the roll-back image if it was previously valid, since the WINC does not have any internal memory except the flash roll-back image location to validate the downloaded image from

See also

m2m_ota_init tpfOtaUpdateCb

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_ota_rollback

WLAN » Function
## Functions

<table>
<thead>
<tr>
<th>NMI_API sint8</th>
<th>m2m_ota_rollback</th>
<th>(void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8</td>
<td>m2m_ota_rollback_crt</td>
<td>(void)</td>
</tr>
</tbody>
</table>
Detailed Description

Request OTA Roll-back to the old (other) WINC image, the WINC firmware will check the validation of the Roll-back image and switch to it if it is valid. If the API response is success, system restart is required (re-initialize the driver with hardware rest) update the host driver version may be required if it is did not match the minimum version supported by the WINC firmware.

Request Cortus application OTA Roll-back to the old (other) cortus application image, the WINC firmware will check the validation of the Roll-back image and switch to it if it is valid. If the API response is success, system restart is required (re-initialize the driver with hardware rest) update the host driver version may be required.
Function Documentation
◆ m2m_ota_rollback()

<table>
<thead>
<tr>
<th>NMI_API</th>
<th>sint8</th>
<th>m2m_ota_rollback ( void )</th>
</tr>
</thead>
</table>

See also
m2m_ota_init m2m_ota_start_update

Returns
The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
**m2m_ota_rollback_crt()**

```c
NMI_API sint8 m2m_ota_rollback_crt ( void )
```

See also

- `m2m_ota_init`
- `m2m_ota_start_update_crt`

**Returns**

The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_ota_abort</td>
</tr>
</tbody>
</table>

WLAN » Function
Functions

<table>
<thead>
<tr>
<th>NMI_API</th>
<th>sint8</th>
<th>m2m_ota_abort (void)</th>
</tr>
</thead>
</table>

Detailed Description

Request abort of current OTA download. The WINC firmware will terminate the OTA download if one is in progress. If no download is in progress, the API will respond with failure.
◆ m2m_ota_abort()

NMI_API sint8 m2m_ota_abort ( void )

Returns
The function returns **M2M_SUCCESS** for successful operation and a negative value otherwise.
m2m OTA switch firmware

WLAN » Function
<table>
<thead>
<tr>
<th>NMI_API sint8</th>
<th>m2m_ota_switch_firmware (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8</td>
<td>m2m_ota_switch_crt (void)</td>
</tr>
<tr>
<td>NMI_API sint8</td>
<td>m2m_ota_get_firmware_version (tstrM2mRev *pstrRev)</td>
</tr>
</tbody>
</table>
Detailed Description

Switch to the upgraded Firmware, that API will update the control structure working image to the upgraded image take effect will be on the next system restart

Switch to the upgraded cortus application, that API will update the control structure working image to the upgraded image take effect will be on the next system restart
Function Documentation
◆ m2m_ota_switch_firmware()

NMI_API sint8 m2m_ota_switch_firmware ( void )

Warning
It is important to note that if the API succeeds, system restart is required (re-initializing the driver with hardware reset) updating the host driver version may be required if it does not match the minimum driver version supported by the WINC's firmware.

See also
m2m_ota_init m2m_ota_start_update

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.

Warning
It is important to note that if the API succeeds, system restart is required (re-initializing the driver with hardware reset) updating the host driver version may be required if it does not match the minimum driver version supported by the WINC's firmware.

See also
m2m_ota_init m2m_ota_start_update_crt

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_ota_switch_crt()

NMI_API sint8 m2m_ota_switch_crt ( void )
m2m_ota_get_firmware_version()

NMI_API sint8
m2m_ota_get_firmware_version ( tstrM2mRev * pstrRev )

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen
1.8.13
m2m_wifi_download_mode

WLAN » Function
### Functions

<table>
<thead>
<tr>
<th>NMI_API sint8  m2m_wifi_download_mode (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Synchronous download mode entry function that prepares the WINC board to enter the download mode, ready for the firmware or certificate download. The WINC board is prepared for download, through initializations for the WINC driver including bus initializations and interrupt enabling, it also halts the chip, to allow for the firmware downloads. Firmware can be downloaded through a number of interfaces, UART, I2C and SPI.
# m2m_wifi_download_mode()

**NMI_API**

```c
void m2m_wifi_download_mode()
```

Prepares the WINC board before downloading any data (Firmware, Certificates .. etc)

This function should be called before starting to download any data to the WINC board. The WINC board is prepared for download, through initializations for the WINC driver including bus initializations and interrupt enabling, it also halts the chip, to allow for the firmware downloads. Firmware can be downloaded through a number of interfaces, UART, I2C and SPI.

**Returns**

The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_init</td>
</tr>
</tbody>
</table>

**WLAN » Function**
Functions

NMI_API sint8 m2m_wifi_init (tstrWifiInitParam *pWifiInitParam)
Detailed Description

Synchronous initialization function for the WINC driver. This function initializes the driver by, registering the call back function for M2M_WIFI layer(also the call back function for bypass mode/monitoring mode if defined), initializing the host interface layer and the bus interfaces. Wi-Fi callback registering is essential to allow the handling of the events received, in response to the asynchronous Wi-Fi operations.

Following are the possible Wi-Fi events that are expected to be received through the call back function(provided by the application) to the M2M_WIFI layer are :

```
@ref M2M_WIFI_RESP_CON_STATE_CHANGED 
@ref M2M_WIFI_RESP_CONN_INFO 
@ref M2M_WIFI_REQ_DHCP_CONF 
@ref M2M_WIFI_REQ_WPS 
@ref M2M_WIFI_RESP_IP_CONFLICT 
@ref M2M_WIFI_RESP_SCAN_DONE 
@ref M2M_WIFI_RESP_SCAN_RESULT 
@ref M2M_WIFI_RESP_CURRENT_RSSI 
@ref M2M_WIFI_RESP_CLIENT_INFO 
@ref M2M_WIFI_RESP_PROVISION_INFO 
@ref M2M_WIFI_RESP_DEFAULT_CONNECT 
```

Example: 

In case Bypass mode is defined :
```
@ref M2M_WIFI_RESP_ETHERNET_RX_PACKET
```

In case Monitoring mode is used:
```
@ref M2M_WIFI_RESP_WIFI_RX_PACKET
```

Any application using the WINC driver must call this function at the start of its main function.
m2m_wifi_init()

NMI_API sint8 m2m_wifi_init ( tstrWifiInitParam * pWifiInitParam )

Initialize the WINC host driver. This function initializes the driver by, registering the call back function for M2M_WIFI layer(also the call back function for bypass mode/monitoring mode if defined), initializing the host interface layer and the bus interfaces.

Parameters

[ in ] pWifiInitParam This is a pointer to the tstrWifiInitParam structure which holds the pointer to the application WIFI layer call back function, monitoring mode call back and tstrEthInitParam structure containing bypass mode parameters.

Precondition

Prior to this function call, The application should initialize the BSP using "nm_bsp_init". Also, application users must provide a call back function responsible for receiving all the WI-FI events that are received on the M2M_WIFI layer.

Warning

Failure to successfully complete function indicates that the driver couldn't be initialized and a fatal error will prevent the application from proceeding.

See also

nm_bsp_init m2m_wifi_deinit tenuM2mStaCmd

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_wifi_deinit

WLAN » Function
Functions

| NMI_API sint8  m2m_wifi_deinit | (void *arg) |
Detailed Description

Synchronous de-initialization function to the WINC1500 driver. De-initializes the host interface and frees any resources used by the M2M_WIFI layer. This function must be called in the application closing phase to ensure that all resources have been correctly released. No arguments are expected to be passed in.
Function Documentation
◆ m2m_wifi_deinit()

```c
NMI_API sint8 m2m_wifi_deinit ( void * arg )
```

Deinitialize the WINC driver and host interface. This function must be called at the De-initialization stage of the application. Generally, this function should be the last function before switching off the chip and it should be followed only by "nm_bsp_deinit" function call. Every function call of "nm_wifi_init" should be matched with a call to nm_wifi_deinit.

**Parameters**


**See also**

- nm_bsp_deinit nm_wifi_init

**Returns**

The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
m2m_wifi_handle_events

WLAN » Function
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8 m2m_wifi_handle_events (void *arg)</td>
</tr>
</tbody>
</table>
Detailed Description

Synchronous M2M event handler function, responsible for handling interrupts received from the WINC firmware. Application developers should call this function periodically in-order to receive the events that are to be handled by the callback functions implemented by the application.
Function Documentation
m2m_wifi_handle_events()

NMI_API sint8 m2m_wifi_handle_events ( void * arg )

Handle the various events received from the WINC board. Whenever an event happen in the WINC board (e.g. Connection, Disconnection, DHCP .. etc), WINC will interrupt the host to let it know that a new event has occurred. The host driver will attempt to handle these events whenever the host driver decides to do that by calling the "m2m_wifi_handle_events" function. It's mandatory to call this function periodically and independently of any other condition. It's ideal to include this function in the main and the most frequent loop of the host application.

Precondition
Prior to receiving events, the WINC driver should have been successfully initialized by calling the m2m_wifi_init function.

Warning
Failure to successfully complete this function indicates bus errors and hence a fatal error that will prevent the application from proceeding.

Returns
The function returns M2M_SUCCESS for successful interrupt handling and a negative value otherwise.
m2m_wifi_send_crl

WLAN » Function
Functions

sint8  m2m_wifi_send_crl (tstrTlsCrlInfo *pCRL)
Detailed Description

Asynchronous API that notifies the WINC with the Certificate Revocation List to be used for TLS.
**m2m_wifi_send_crl()**

```c
sint8 m2m_wifi_send_crl ( tstrTlsCrlInfo * pCRL )
```

Asynchronous API that notifies the WINC with the Certificate Revocation List.

**Parameters**

- `[in] pCRL` Pointer to the structure containing certificate revocation list details.

**Returns**

The function returns **M2M_SUCCESS** if the command has been successfully queued to the WINC, and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_default_connect</td>
</tr>
</tbody>
</table>

WLAN » Function
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8  m2m_wifi_default_connect (void)</td>
</tr>
</tbody>
</table>
Detailed Description

Asynchronous Wi-Fi connection function. An application calling this function will cause the firmware to correspondingly connect to the last successfully connected AP from the cached connections. A failure to connect will result in a response of **M2M_WIFI_RESP_DEFAULT_CONNECT** indicating the connection error as defined in the structure `tstrM2MDefaultConnResp`. Possible errors are: The connection list is empty **M2M_DEFAULT_CONN_EMPTY_LIST** or a mismatch for the saved AP name **M2M_DEFAULT_CONN_SCAN_MISMATCH**. Only difference between this function and `m2m_wifi_connect`, is the connection parameters. Connection using this function is expected to connect using cached connection parameters.
Function Documentation
◆ **m2m_wifi_default_connect()**

```c
NMI_API sint8 m2m_wifi_default_connect ( void )
```

Connect to the last successfully connected AP from the cached connections.

**Precondition**
Prior to connecting, the WINC driver should have been successfully initialized by calling the `m2m_wifi_init` function.

**Warning**
This function must be called in station mode only. It's important to note that successful completion of a call to `m2m_wifi_default_connect()` does not guarantee success of the WIFI connection, and a negative return value indicates only locally-detected errors.

**See also**
`m2m_wifi_connect`

**Returns**
The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
m2m_wifi_connect

WLAN » Function
<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NMI_API sint8</strong></td>
<td><strong>m2m_wifi_connect</strong> (char *pcSsid, <strong>uint8</strong> u8SsidLen, <strong>uint8</strong> u8SecType, void *pvAuthInfo, <strong>uint16</strong> u16Ch)</td>
</tr>
<tr>
<td><strong>NMI_API sint8</strong></td>
<td><strong>m2m_wifi_connect_sc</strong> (char *pcSsid, <strong>uint8</strong> u8SsidLen, <strong>uint8</strong> u8SecType, void *pvAuthInfo, <strong>uint16</strong> u16Ch, <strong>uint8</strong> u8SaveCred)</td>
</tr>
</tbody>
</table>
Asynchronous wi-fi connection function to a specific AP. Prior to a successful connection, the application must define the SSID of the AP, the security type, the authentication information parameters and the channel number to which the connection will be established. The connection status is known when a response of `M2M_WIFI_RESP_CON_STATE_CHANGED` is received based on the states defined in `tenuM2mConnState`, successful connection is defined by `M2M_WIFI_CONNECTED`.

The only difference between this function and `m2m_wifi_default_connect`, is the connection parameters. Connection using this function is expected to be made to a specific AP and to a specified channel.

Asynchronous wi-fi connection function to a specific AP. Prior to a successful connection, the application developers must know the SSID of the AP, the security type, the authentication information parameters and the channel number to which the connection will be established. This API allows the user to choose whether to The connection status is known when a response of `M2M_WIFI_RESP_CON_STATE_CHANGED` is received based on the states defined in `tenuM2mConnState`, successful connection is defined by `M2M_WIFI_CONNECTED`. The only difference between this function and `m2m_wifi_connect`, is the option to save the access point info (SSID, password...etc) or not. Connection using this function is expected to be made to a specific AP and to a specified channel.
◆ m2m_wifi_connect()

NMI_API sint8 m2m_wifi_connect ( char * pcSsid,
uint8 u8SsidLen,
uint8 u8SecType,
void * pvAuthInfo,
uint16 u16Ch )

Parameters

[in] **pcSsid**  A buffer holding the SSID corresponding to the requested AP.

[in] **u8SsidLen**  Length of the given SSID (not including the NULL termination). A length less than ZERO or greater than the maximum defined SSID M2M_MAX_SSID_LEN will result in a negative error M2M_ERR_FAIL.

[in] **u8SecType**  Wi-Fi security type security for the network. It can be one of the following types:
- M2M_WIFI_SEC_OPEN
- M2M_WIFI_SEC_WEP
- M2M_WIFI_SEC_WPA_PSK
- M2M_WIFI_SEC_802_1X  A value outside these possible values will result in a negative return error M2M_ERR_FAIL.

[in] **pvAuthInfo**  Authentication parameters required for completing the connection. It is type is based on the Security type. If the authentication parameters are NULL or are greater than the maximum length of the authentication parameters length as defined by M2M_MAX_PSK_LEN a negative error will return M2M_ERR_FAIL(-12) indicating connection failure.

[in] **u16Ch**  Wi-Fi channel number as defined in tenuM2mScanCh enumeration. Channel
number greater than `M2M_WIFI_CH_14` returns a negative error `M2M_ERR_FAIL`(-12). Except if the value is `M2M_WIFI_CH_ALL(255)`, since this indicates that the firmware should scan all channels to find the SSID requested to connect to. Failure to find the connection match will return a negative error `M2M_DEFAULT_CONN_SCAN_MISMATCH`.

**Precondition**
Prior to a successful connection request, the Wi-Fi driver must have been successfully initialized through the call of the function

**See also**
- `tuniM2MWifiAuth`
- `tstr1xAuthCredentials`
- `tstrM2mWifiWepParams`

**Warning**
- This function must be called in station mode only.
- Successful completion of this function does not guarantee success of the WIFI connection, and a negative return value indicates only locally-detected errors.

**Returns**
The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
◆ **m2m_wifi_connect_sc()**

```c
NMI_API sint8 m2m_wifi_connect_sc( char * pcSsid,
    uint8 u8SsidLen,
    uint8 u8SecType,
    void * pvAuthInfo,
    uint16 u16Ch,
    uint8 u8SaveCred )
```

**Parameters**

- **[in] pcSsid**
  A buffer holding the SSID corresponding to the requested AP.

- **[in] u8SsidLen**
  Length of the given SSID (not including the NULL termination). A length less than ZERO or greater than the maximum defined SSID M2M_MAX_SSID_LEN will result in a negative error M2M_ERR_FAIL.

- **[in] u8SecType**
  Wi-Fi security type security for the network. It can be one of the following types:
  - M2M_WIFI_SEC_OPEN
  - M2M_WIFI_SEC_WEP
  - M2M_WIFI_SEC_WPA_PSK
  - M2M_WIFI_SEC_802_1X
  A value outside these possible values will result in a negative return error M2M_ERR_FAIL.

- **[in] pvAuthInfo**
  Authentication parameters required for completing the connection. It is type is based on the Security type. If the authentication parameters are NULL or are greater than the maximum length of the authentication parameters length as defined by M2M_MAX_PSK_LEN a negative error will return M2M_ERR_FAIL(-12) indicating connection failure.

- **[in] u16Ch**
  Wi-Fi channel number as defined in
tenuM2mScanCh enumeration. Channel number greater than M2M_WIFI_CH_14 returns a negative error M2M_ERR_FAIL(-12). Except if the value is M2M_WIFI_CH_ALL(255), since this indicates that the firmware should scan all channels to find the SSID requested to connect to. Failure to find the connection match will return a negative error M2M_DEFAULT_CONN_SCAN_MISMATCH.

[in] u8NoSaveCred Option to store the access point SSID and password into the WINC flash memory or not

Precondition
Prior to a successful connection request, the wi-fi driver must have been successfully initialized through the call of the function.

See also
tuniM2MWifiAuth tstr1xAuthCredentials tstrM2mWifiWepParam

Warning
-This function must be called in station mode only. -Successful completion of this function does not guarantee success of the WIFI connection, and a negative return value indicates only locally-detected errors.

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_disconnect</td>
</tr>
</tbody>
</table>

WLAN » Function
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NMI_API sint8 m2m_wifi_disconnect (void)</strong></td>
</tr>
</tbody>
</table>
Detailed Description

Synchronous wi-fi disconnection function, requesting a Wi-Fi disconnection from the currently connected AP.
Function Documentation
◆ m2m_wifi_disconnect()

NMI_API sint8 m2m_wifi_disconnect ( void )

Request a Wi-Fi disconnect from the currently connected AP. After the Disconnect is complete the driver should receive a response of M2M_WIFI_RESP_CON_STATE_CHANGED based on the states defined in tenuM2mConnState, successful disconnection is defined by M2M_WIFI_DISCONNECTED.

Precondition
Disconnection request must be made to a successfully connected AP. If the WINC is not in the connected state, a call to this function will hold insignificant.

Warning
This function must be called in station mode only.

See also
m2m_wifi_connect m2m_wifi_default_connect

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_wifi_start_provision_mode
<table>
<thead>
<tr>
<th>NMI_API</th>
<th>sint8</th>
<th>m2m_wifi_start_provision_mode (tstrM2MAPConfig *pstrAPConfig, char *pcHttpServerDomainName, uint8 bEnableHttpRedirect)</th>
</tr>
</thead>
</table>
Detailed Description

Asynchronous Wi-Fi provisioning function, which starts the WINC HTTP PROVISIONING mode. The function triggers the WINC to activate the Wi-Fi AP (HOTSPOT) mode with the passed configuration parameters and then starts the HTTP Provision WEB Server. The provisioning status is returned in an event M2M_WIFI_RESP_PROVISION_INFO
Function Documentation
m2m_wifi_start_provision_mode()

Parameters

[in] **pstrAPConfig**  
AP configuration parameters as defined in tstrM2MAPConfig configuration structure. A NULL value passed in will result in a negative error M2M_ERR_FAIL.

[in] **pcHttpServerDomainName**  
Domain name of the HTTP Provision WEB server which others will use to load the provisioning Home page. The domain name can have one of the following 3 forms: 1- "wincprov.com" 2- "http://wincprov.com" 3- "https://wincprov.com" The forms 1 and 2 are equivalent, they both will start a plain http server, while form 3 will start a secure HTTP provisioning Session (HTTP over SSL connection).

[in] **bEnableHttpRedirect**  
A flag to enable/disable the HTTP Redirect feature. If Secure provisioning is enabled (i.e. the server domain name uses the https prefix) this flag is ignored (no meaning for redirect in HTTPS). Possible values:

- **ZERO** DO NOT Use HTTP Redirect.
  In this case the associated device could open the provisioning page ONLY when the HTTP Provision URL of the WINC HTTP Server is correctly written on the browser.
- Non-Zero value Use HTTP Redirect.
In this case, all http traffic (http://URL) from the associated device (Phone, PC, ...etc) will be redirected to the WINC HTTP Provisioning Home page.

**Precondition**
- A Wi-Fi notification callback of type tpfAppWifiCb MUST be implemented and registered at startup. Registering the callback is done through passing it to the initialization `m2m_wifi_init` function.
- The event `M2M_WIFI_RESP_CONN_INFO` must be handled in the callback to receive the requested connection info.

**See also**
- `tpfAppWifiCb`
- `m2m_wifi_init`
- `M2M_WIFI_RESP_PROVISION_INFO`
- `m2m_wifi_stop_provision_mode`
- `tstrM2MAPConfig`

**Warning**
DO Not use ".local" in the pcHttpServerDomainName.

**Returns**
The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
Example

The example demonstrates a code snippet for how provisioning is triggered and the response event received accordingly.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    switch(u8WiFiEvent)
    {
    case M2M_WIFI_RESP_PROVISION_INFO:
    {
        tstrM2MProvisionInfo *pstrProvInfo = (tstrM2MProvisionInfo*)pvMsg;
        if(pstrProvInfo->u8Status == M2M_SUCCESS)
        {
            m2m_wifi_connect((char*)pstrProvInfo->au8SSID,
                            (uint8)strlen(pstrProvInfo->au8SSID),
                            pstrProvInfo->u8SecType,
                            pstrProvInfo->au8Password,
                            M2M_WIFI_CH_ALL);

            printf("PROV SSID : \n",pstrProvInfo->au8SSID);
            printf("PROV PSK : \n",pstrProvInfo->au8Password);
        }
        else
        {
            printf("(ERR) Provisioning Failed\n");
        }
    }
```
break;

default:
    break;
}

int main()
{
    tstrWifiInitParam param;

    param.pfAppWifiCb = wifi_event_cb;
    if (!m2m_wifi_init(&param)) {
        tstrM2MAPConfig apConfig;
        uint8 bEnableRedirect = 1;

        strcpy(apConfig.au8SSID, "WINC_SSID");
        apConfig.u8ListenChannel = 1;
        apConfig.u8SecType = M2M_WIFI_SEC_OPEN;
        apConfig.u8SsidHide = 0;

        // IP Address
        apConfig.au8DHCPServerIP[0] = 192;
        apConfig.au8DHCPServerIP[1] = 168;
        apConfig.au8DHCPServerIP[2] = 1;
        apConfig.au8DHCPServerIP[0] = 1;

        m2m_wifi_start_provision_mode(&apConfig,
                                      "atmelwinccconf.com", bEnableRedirect);

        while (1)
        {
            m2m_wifi_handle_events(NULL);
        }
    }
}
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_stop_provision_mode</td>
<td>WLAN » Function</td>
</tr>
</tbody>
</table>
### Functions

| NMI_API sint8       | m2m_wifi_stop_provision_mode (void) |
Detailed Description

Synchronous provision termination function which stops the provision mode if it is active.
Function Documentation
◆ `m2m_wifi_stop_provision_mode()`

`sint8 m2m_wifi_stop_provision_mode ( void )`

**Precondition**
An active provisioning session must be active before it is terminated through this function.

**See also**
`m2m_wifi_start_provision_mode`

**Returns**
The function returns ZERO for success and a negative value otherwise.
m2m_wifi_get_connection_info

WLAN » Function
## Functions

| NMI_API sint8 m2m_wifi_get_connection_info (void) |  |
Detailed Description

Asynchronous connection status retrieval function, retrieves the status information of the currently connected AP. The result is passed to the Wi-Fi notification callback through the event M2M_WIFI_RESP_CONN_INFO. Connection information is retrieved from the structure tstrM2MConnInfo.
Function Documentation
- **m2m_wifi_get_connection_info()**

  ```c
  sint8 m2m_wifi_get_connection_info ( void )
  ```

  Retrieve the current Connection information. The result is passed to the Wi-Fi notification callback with `M2M_WIFI_RESP_CONN_INFO`.

**Precondition**
- A Wi-Fi notification callback of type tpfAppWifiCb MUST be implemented at startup. Registering the callback is done through passing it to the `m2m_wifi_init` function.
- The event `M2M_WIFI_RESP_CONN_INFO` must be handled in the requested connection info.

Connection Information retrieved:

- Connection Security
- Connection RSSI
- Remote MAC address
- Remote IP address

and in case of WINC station mode the SSID of the AP is also retrieved.

**Warning**
- In case of WINC AP mode or P2P mode, ignore the SSID field (NULL string).

**See also**
- `M2M_WIFI_RESP_CONN_INFO`, `tstrM2MConnInfo`

**Returns**
- The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
Example

The code snippet shows an example of how wi-fi connection information is retrieved.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    switch(u8WiFiEvent)
    {
    case M2M_WIFI_RESP_CONN_INFO:
        
        tstrM2MConnInfo *pstrConnInfo = (tstrM2MConnInfo*)pvMsg;

        printf("CONNECTED AP INFO\n");
        printf("SSID      : %s\n", pstrConnInfo->acSSID);
        printf("SEC TYPE  : %d\n", pstrConnInfo->u8SecType);
        printf("Signal Strength : %d\n", pstrConnInfo->s8RSSI);
        printf("Local IP Address : %d.%d.%d.%d\n",
                pstrConnInfo->au8IPAddr[0],
                pstrConnInfo->au8IPAddr[1],
                pstrConnInfo->au8IPAddr[2],
                pstrConnInfo->au8IPAddr[3]);
        break;

    case M2M_WIFI_REQ_DHCP_CONF:
        
        break;
    }
```
/ Get the current AP information.
m2m_wifi_get_connection_info();
  }
basic;
default:
basic;
}

int main()
{
tstrWifiInitParam    param;
    param.pfAppWifiCb  = wifi_event_cb;
if(!m2m_wifi_init(&param))
  {
    // connect to the default AP
    m2m_wifi_default_connect();

    while(1)
      {
        m2m_wifi_handle_events(NULL);
      }
  }
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>m2m_wifi_set_mac_address</strong></td>
</tr>
<tr>
<td>WLAN » Function</td>
</tr>
</tbody>
</table>
Functions

| NMI_API sint8 | m2m_wifi_set_mac_address (uint8 au8MacAddress[6]) |
Detailed Description

Synchronous MAC address assigning to the NMC1500. It is used for non-production SW. Assign MAC address to the WINC device.
Function Documentation
m2m_wifi_set_mac_address()  

NMI_API sint8  
m2m_wifi_set_mac_address ( uint8 au8MacAddress[6] )  

Assign a MAC address to the WINC board. This function override the already assigned MAC address of the WINC board with a user provided one. This is for experimental use only and should never be used in the production SW.

Parameters  

[in] **au8MacAddress** MAC Address to be set to the WINC.

Returns  
The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
m2m_wifi_wps

WLAN » Function
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8</td>
<td><code>m2m_wifi_wps</code> *(uint8 u8TriggerType, const char <em>pcPinNumber)</em></td>
</tr>
</tbody>
</table>
Asynchronous WPS triggering function. This function is called for the WINC to enter the WPS (Wi-Fi Protected Setup) mode. The result is passed to the Wi-Fi notification callback with the event \texttt{M2M\_WiFi\_REQ\_WPS}.
m2m_wifi_wps()

```c
NMI_API sint8 m2m_wifi_wps ( uint8 u8TriggerType, const char * pcPinNumber )
```

**Parameters**

- [in] **u8TriggerType** WPS Trigger method. Could be:
  - `WPS_PIN_TRIGGER` Push button method
  - `WPS_PBC_TRIGGER` Pin method
- [in] **pcPinNumber** PIN number for WPS PIN method. It is not used if the trigger type is `WPS_PBC_TRIGGER`. It must follow the rules stated by the WPS standard.

**Warning**

This function is not allowed in AP or P2P modes.

**Precondition**

- A Wi-Fi notification callback of type (tpfAppWifiCb MUST be implemented and registered at startup. Registering the callback is done through passing it to the m2m_wifi_init.
- The event `M2M_WIFI_REQ_WPS` must be handled in the callback to receive the WPS status.
- The WINC device MUST be in IDLE or STA mode. If AP or P2P mode is active, the WPS will not be performed.
- The `m2m_wifi_handle_events` MUST be called periodically to receive the responses in the callback.

**See also**

tpfAppWifiCb m2m_wifi_init M2M_WIFI_REQ_WPS
tenuWPSTrigger tstrM2MWPSInfo

**Returns**

The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
Example

The code snippet shows an example of how Wi-Fi WPS is triggered.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    switch(u8WiFiEvent)
    {
    case M2M_WIFI_REQ_WPS:
        {
            tstrM2MWPSInfo *pstrWPS = (tstrM2MWPSInfo*)pvMsg;
            if(pstrWPS->u8AuthType != 0)
            {
                printf("WPS SSID : ");
                printf("%s\n",pstrWPS->au8SSID);
                printf("WPS PSK : ");
                printf("%s\n",pstrWPS->au8PSK);
                printf("WPS SSID Auth Type : ");
                printf("%s\n",pstrWPS->u8AuthType == M2M_WIFI_SEC_OPEN ? "OPEN" : "WPA/WPA2");
                printf("WPS Channel : ");
                printf("%d\n",pstrWPS->u8Ch + 1);

                // establish Wi-Fi connection
                m2m_wifi_connect((char*)pstrWPS->au8SSID,
                                 (uint8)m2m_strlen(pstrWPS->au8SSID),
                                 pstrWPS->u8AuthType, pstrWPS->au8PSK, pstrWPS->u8Ch);
            }
            else
            {
                printf("(ERR) WPS Is not enabled OR ");
            }
        }
    }
```
Timed out

break;

default:
break;
}

int main()
{
tstrWifiInitParam param;

param.pfAppWifiCb = wifi_event_cb;
if(!m2m_wifi_init(&param))
{
// Trigger WPS in Push button mode.
m2m_wifi_wps(WPS_PBC_TRIGGER, NULL);

while(1)
{
m2m_wifi_handle_events(NULL);
}
}
m2m_wifi_wps_disable
WLAN » Function
Functions

NMI_API sint8 m2m_wifi_wps_disable (void)
Disable the WINC1500 WPS operation.
Function Documentation
**m2m_wifi_wps_disable()**

NMI_API sint8 m2m_wifi_wps_disable ( void )

Stops the WPS ongoing session.

**Precondition**
WINC should be already in WPS mode using `m2m_wifi_wps`

**See also**
`m2m_wifi_wps`

**Returns**
The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
m2m_wifi_p2p
WLAN » Function
Functions

```c
NMI_API sint8 m2m_wifi_p2p (uint8 u8Channel)
```
Detailed Description

Asynchronous Wi-Fi direct (P2P) enabling mode function. The WINC supports P2P in device listening mode ONLY (intent is ZERO). The WINC P2P implementation does not support P2P GO (Group Owner) mode. Active P2P devices (e.g. phones) could find the WINC in the search list. When a device is connected to WINC, a Wi-Fi notification event M2M_WIFI_RESP_CON_STATE_CHANGED is triggered. After a short while, the DHCP IP Address is obtained and an event M2M_WIFI_REQ_DHCP_CONF is triggered. Refer to the code examples for a more illustrative example.
Function Documentation
### m2m_wifi_p2p()

**NMI_API** `sint8 m2m_wifi_p2p ( uint8 u8Channel )`

**Parameters**

- `[in] u8Channel` P2P Listen RF channel. According to the P2P standard it must hold only one of the following values 1, 6 or 11.

**Precondition**

- A Wi-Fi notification callback of type `tpfAppWifiCb` MUST be implemented and registered at initialization. Registering the callback is done through passing it to the `m2m_wifi_init`.
- The events `M2M_WIFI_RESP_CON_STATE_CHANGED` and `M2M_WIFI_REQ_DHCP_CONF` must be handled in the callback.
- The `m2m_wifi_handle_events` MUST be called to receive the responses in the callback.

**Warning**

This function is not allowed in AP or STA modes.

**See also**

- `tpfAppWifiCb`
- `m2m_wifi_init`
- `M2M_WIFI_RESP_CON_STATE_CHANGED`
- `M2M_WIFI_REQ_DHCP_CONF`
- `tstrM2mWifiStateChanged`

**Returns**

The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
Example

The code snippet shown an example of how the p2p mode operates.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    switch(u8WiFiEvent)
    {
    case M2M_WIFI_RESP_CON_STATE_CHANGED:
    {
        tstrM2mWifiStateChanged *pstrWifiState = (tstrM2mWifiStateChanged*)pvMsg;
        M2M_INFO("Wifi State :: %s :: ErrCode %d
             "CONNECTED":"DISCONNECTED",pstrWifiState->u8ErrCode

        // Do something
    }
    break;

    case M2M_WIFI_REQ_DHCP_CONF:
    {
        uint8  *pu8IPAddress = (uint8*)pvMsg;

        printf("P2P IP Address \
            "%u.%u.%u.%u"
            ,pu8IPAddress[0],pu8IPAddress[1]
        }
    break;

    default:
    break;
    }
}
```
int main()
{
    tstrWifiInitParam param;

    param.pfAppWifiCb = wifi_event_cb;
    if(!m2m_wifi_init(&param))
    {
        // Trigger P2P
        m2m_wifi_p2p(M2M_WIFI_CH_1);

        while(1)
        {
            m2m_wifi_handle_events(NULL);
        }
    }
}
m2m_wifi_p2p_disconnect
WLAN » Function
Functions

NMI_API sint8 m2m_wifi_p2p_disconnect (void)
Disable the WINC1500 device Wi-Fi direct mode (P2P).
Function Documentation
m2m_wifi_p2p_disconnect()

NMI_API sint8 m2m_wifi_p2p_disconnect ( void )

Precondition
The p2p mode must have be enabled and active before a disconnect can be called.

See also
m2m_wifi_p2p

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_enable_ap</td>
</tr>
</tbody>
</table>

WLAN » Function
| NMI_API sint8 | m2m_wifi_enable_ap (CONST tstrM2MAPConfig *pstrM2MAPConfig) |
Detailed Description

Asynchronous Wi-Fi hot-spot enabling function. The WINC supports AP mode operation with the following limitations:

- Only 1 STA could be associated at a time.
- Open and WEP are the only supported security types
◆ m2m_wifi_enable_ap()

NMI_API sint8
m2m_wifi_enable_ap(CONST tstrM2MAPConfig * pstrM2MAPConfig)

Parameters
[in] pstrM2MAPConfig A structure holding the AP configurations.

Warning
This function is not allowed in P2P or STA modes.

Precondition
- A Wi-Fi notification callback of type tpfAppWifiCb MUST be implemented and registered at initialization. Registering the callback is done through passing it to the m2m_wifi_init.
- The event M2M_WIFI_REQ_DHCP_CONF must be handled in the callback.
- The m2m_wifi_handle_events MUST be called to receive the responses in the callback.

See also
tpfAppWifiCb tenuM2mSecType m2m_wifi_init
M2M_WIFI_REQ_DHCP_CONF tstrM2mWifiStateChanged
tstrM2MAPConfig

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
Example

The code snippet demonstrates how the AP mode is enabled after the driver is initialized in the application's main function and the handling of the event `M2M_WIFI_REQ_DHCP_CONF`, to indicate successful connection.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    switch(u8WiFiEvent)
    {
    case M2M_WIFI_REQ_DHCP_CONF:
        {
            uint8 *pu8IPAddress = (uint8*)pvMsg;

            printf("Associated STA has IP Address \\
                    %u.%u.%u.%u\n",pu8IPAddress[0],pu8IPAddress[1],pu8IPAddress[2],pu8IPAddress[3]);
        }
        break;
    default:
        break;
    }
}

int main()
{
    tstrWifiInitParam     param;

    param.pfAppWifiCb    = wifi_event_cb;
    if(!m2m_wifi_init(&param))
    {
        tstrM2MAPConfig    apConfig;
```
strcpy(apConfig.au8SSID, "WINC_SSID");
apConfig.u8ListenChannel = 1;
apConfig.u8SecType = M2M_WIFI_SEC_OP;
apConfig.u8SsidHide = 0;

// IP Address
apConfig.au8DHCPServerIP[0] = 192;
apConfig.au8DHCPServerIP[1] = 168;
apConfig.au8DHCPServerIP[2] = 1;
apConfig.au8DHCPServerIP[0] = 1;

// Trigger AP
m2m_wifi_enable_ap(&apConfig);

while(1)
{
    m2m_wifi_handle_events(NULL);
}
}
m2m_wifi_disable_ap

WLAN » Function
## Functions

<table>
<thead>
<tr>
<th>NMI_API sint8</th>
<th>m2m_wifi_disable_ap (void)</th>
</tr>
</thead>
</table>
Detailed Description

Synchronous Wi-Fi hot-spot disabling function. Must be called only when the AP is enabled through the `m2m_wifi_enable_ap` function. Otherwise the call to this function will not be useful.
Function Documentation
m2m_wifi_disable_ap()

NMI_API sint8 m2m_wifi_disable_ap ( void )

See also
m2m_wifi_enable_ap

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_wifi_set_static_ip

WLAN » Function
## Functions

| NMI_API sint8  | m2m_wifi_set_static_ip (tstrM2MIPConfig *pstrStaticIPConf) |
Detailed Description

Synchronous static IP Address configuration function.
m2m_wifi_set_static_ip()

NMI_API sint8 m2m_wifi_set_static_ip ( tstrM2MIPConfig * pstrStaticIPConf )

Assign a static IP address to the WINC board. This function assigns a static IP address in case the AP doesn't have a DHCP server or in case the application wants to assign a predefined known IP address. The user must take in mind that assigning a static IP address might result in an IP address conflict. In case of an IP address conflict observed by the WINC board the user will get a response of M2M_WIFI_RESP_IP_CONFLICT in the wifi callback. The application is then responsible to either solve the conflict or assign another IP address.

Parameters
   [in] pstrStaticIPConf Pointer to a structure holding the static IP Configurations (IP, Gateway, subnet mask and DNS address).

Precondition
   The application must disable auto DHCP using m2m_wifi_enable_dhcp before assigning a static IP address.

Warning
   Normally this function normally should not be used. DHCP configuration is requested automatically after successful Wi-Fi connection is established.

See also
   tstrM2MIPConfig

Returns
   The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_wifi_request_dhcp_client
WLAN » Function
## Functions

| NMI_API sint8  m2m_wifi_request_dhcp_client (void) |  |
Detailed Description

Starts the DHCP client operation (DHCP requested by the firmware automatically in STA/AP/P2P mode).
Function Documentation
m2m_wifi_request_dhcp_client()

```
NMI_API sint8 m2m_wifi_request_dhcp_client ( void )
```

**Warning**
This function is legacy and exists only for compatibility with older applications. DHCP configuration is requested automatically after successful Wi-Fi connection is established.

**Returns**
The function returns `M2M_SUCCESS` always.

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by [Doxygen](https://www.doxygen.org/) 1.8.13
m2m_wifi_request_dhcp_server
Functions

NMI_API sint8 m2m_wifi_request_dhcp_server (uint8 *addr)
Detailed Description

Dhcp requested by the firmware automatically in STA/AP/P2P mode).
Function Documentation
**m2m_wifi_request_dhcp_server()**

```
NMI_API sint8 m2m_wifi_request_dhcp_server ( uint8 * addr )
```

**Warning**
This function is legacy and exists only for compatibility with older applications. DHCP server is started automatically when enabling the AP mode.

**Returns**
The function returns **M2M_SUCCESS** always.
m2m_wifi_enable_dhcp

WLAN » Function
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NMI_API sint8  m2m_wifi_enable_dhcp</strong> (uint8 u8DhcpEn)</td>
</tr>
</tbody>
</table>
Detailed Description

Synchronous Wi-Fi DHCP enable function. This function Enable/Disable DHCP protocol.
Function Documentation
◆ m2m_wifi_enable_dhcp()

NMI_API sint8 m2m_wifi_enable_dhcp ( uint8 u8DhcpEn )

Enable/Disable the DHCP client after connection.

Parameters

[in] u8DhcpEn Possible values: 1: Enable DHCP client after connection. 0: Disable DHCP client after connection. -DHCP client is enabled by default -

This Function should be called before using m2m_wifi_set_static_ip()

See also

m2m_wifi_set_static_ip()

Returns

The function SHALL return M2M_SUCCESS for successful operation and a negative value otherwise.
m2m_wifi_set_scan_options
WLAN » Function

Synchronous Wi-Fi scan settings function. This function sets the time configuration parameters for the scan operation.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>m2m_wifi_set_scan_region</strong></td>
</tr>
<tr>
<td>WLAN » Function</td>
</tr>
</tbody>
</table>
### Functions

| NMI_API sint8 m2m_wifi_set_scan_region (uint16 ScanRegion) |  |
Detailed Description

Synchronous wi-fi scan region setting function. This function sets the scan region, which will affect the range of possible scan channels. For 2.5GHz supported in the current release, the requested scan region can't exceed the maximum number of channels (14).
m2m_wifi_set_scan_region()

```c
sint8 m2m_wifi_set_scan_region ( uint16  ScanRegion )
```

Parameters

`[in] ScanRegion; ASIA NORTH_AMERICA`

See also

`tenuM2mScanCh m2m_wifi_request_scan`

Returns

The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
m2m_wifi_request_scan

WLAN » Function
### Functions

<table>
<thead>
<tr>
<th>NMI_API sint8</th>
<th>m2m_wifi_request_scan (uint8 ch)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8</td>
<td>m2m_wifi_request_scan_passive (uint8 ch, uint16 scan_time)</td>
</tr>
<tr>
<td>NMI_API sint8</td>
<td>m2m_wifi_request_scan_ssid_list (uint8 ch, uint8 *u8Ssidlist)</td>
</tr>
</tbody>
</table>
Detailed Description

Asynchronous Wi-Fi scan request on the given channel. The scan status is delivered in the wifi event callback and then the application is supposed to read the scan results sequentially. The number of APs found (N) is returned in event `M2M_WIFI_RESP_SCAN_DONE` with the number of found APs. The application reads the list of APs by calling the function `m2m_wifi_req_scan_result` N times.

Same as `m2m_wifi_request_scan` but perform passive scanning while the other one perform active scanning.

Asynchronous wi-fi scan request on the given channel and the hidden scan list. The scan status is delivered in the wi-fi event callback and then the application is to read the scan results sequentially. The number of APs found (N) is returned in event `M2M_WIFI_RESP_SCAN_DONE` with the number of found APs. The application could read the list of APs by calling the function `m2m_wifi_req_scan_result` N times.
Function Documentation
◆ m2m_wifi_request_scan()

NMI_API sint8 m2m_wifi_request_scan ( uint8 ch )

Parameters
[in] ch RF Channel ID for SCAN operation. It should be set according to tenuM2mScanCh. With a value of M2M_WIFI_CH_ALL(255)), means to scan all channels.

Warning
This function is not allowed in P2P or AP modes. It works only for STA mode (both connected or disconnected states).

Precondition
- A Wi-Fi notification callback of type tpfAppWifiCb MUST be implemented and registered at initialization. Registering the callback is done through passing it to the m2m_wifi_init.
- The events M2M_WIFI_RESP_SCAN_DONE and M2M_WIFI_RESP_SCAN_RESULT must be handled in the callback.
- The m2m_wifi_handle_events function MUST be called to receive the responses in the callback.

See also
M2M_WIFI_RESP_SCAN_DONE
M2M_WIFI_RESP_SCAN_RESULT tpfAppWifiCb
tstrM2mWifiscanResult tenuM2mScanCh m2m_wifi_init
m2m_wifi_handle_events m2m_wifi_req_scan_result

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
Example

The code snippet demonstrates an example of how the scan request is called from the application's main function and the handling of the events received in response.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg) {
    static uint8 u8ScanResultIdx = 0;

    switch(u8WiFiEvent) {
    case M2M_WIFI_RESP_SCAN_DONE:
        tstrM2mScanDone *pstrInfo = (tstrM2mScanDone*)pvMsg;

        printf("Num of AP found %d\n",pstrInfo->u8NumofCh);
        if(pstrInfo->s8ScanState == M2M_SUCCESS) {
            u8ScanResultIdx = 0;
            if(pstrInfo->u8NumofCh >= 1) {
                m2m_wifi_req_scan_result(u8ScanResultIdx);
                u8ScanResultIdx ++;
            }
        } else {
            printf("No AP Found Rescan\n");
            m2m_wifi_request_scan(M2M_WIFI_CH_ALL);
        }
    }
```
else
{
    printf("(ERR) Scan fail with error <\%d>\n", pstrInfo->s8ScanState);
}
break;

case M2M_WIFI_RESP_SCAN_RESULT:
{
tstrM2mWifiscanResult *pstrScanResult =
(tstrM2mWifiscanResult*)pvMsg;
uint8 u8NumFoundAPs =
m2m_wifi_get_num_ap_found();

    printf(">>\%02d RI %d SEC %s CH \%02d BSSID \%02X:%02X:%02X:%02X:%02X:%02X SSID %s\n",
            pstrScanResult->u8index,
pstrScanResult->s8rssi,
pstrScanResult->u8AuthType,
pstrScanResult->u8ch,
pstrScanResult->au8BSSID[0],
pstrScanResult->au8BSSID[1], pstrScanResult->au8BSSID[2],
pstrScanResult->au8BSSID[3], pstrScanResult->au8BSSID[4], pstrScanResult->au8BSSID[5],
pstrScanResult->au8SSID);

    if(u8ScanResultIdx < u8NumFoundAPs)
    {
        // Read the next scan result
        m2m_wifi_req_scan_result(index);
        u8ScanResultIdx ++;
    }
```c
int main()
{
    tstrWifiInitParam   param;
    
    param.pfAppWifiCb   = wifi_event_cb;
    if(!m2m_wifi_init(&param))
    {
        // Scan all channels
        m2m_wifi_request_scan(M2M_WIFI_CH_ALL);

        while(1)
        {
            m2m_wifi_handle_events(NULL);
        }
    }
}
```
m2m_wifi_request_scan_passive()

NMI_API sint8 m2m_wifi_request_scan_passive ( uint8 ch, uint16 scan_time )

Parameters

[in] ch RF Channel ID for SCAN operation. It should be set according to tenuM2mScanCh. With a value of M2M_WIFI_CH_ALL(255)), means to scan all channels.

[in] scan_time The time in ms that passive scan is listening to beacons on each channel per one slot, enter 0 for default setting.

Warning

This function is not allowed in P2P or AP modes. It works only for STA mode (both connected or disconnected states).

Precondition

- A Wi-Fi notification callback of type tpfAppWifiCb MUST be implemented and registered at initialization. Registering the callback is done through passing it to the m2m_wifi_init.
- The events M2M_WIFI_RESP_SCAN_DONE and M2M_WIFI_RESP_SCAN_RESULT must be handled in the callback.
- The m2m_wifi_handle_events function MUST be called to receive the responses in the callback.

See also

m2m_wifi_request_scan M2M_WIFI_RESP_SCAN_DONE M2M_WIFI_RESP_SCAN_RESULT tpfAppWifiCb tstrM2mWifiscanResult tenuM2mScanCh m2m_wifi_init m2m_wifi_handle_events m2m_wifi_req_scan_result

Returns

The function returns M2M_SUCCESS for successful operations
and a negative value otherwise.
**m2m_wifi_request_scan_ssid_list()**

**NMI_API**

{sint8} m2m_wifi_request_scan_ssid_list (uint8 ch, uint8 *u8SsidList)

**Parameters**

**[in] ch**
RF Channel ID for SCAN operation. It should be set according to tenuM2mScanCh. With a value of M2M_WIFI_CH_ALL(255)), means to scan all channels.

**[in] u8SsidList**
u8SsidList is a buffer containing a list of hidden SSIDs to include during the scan. The first byte in the buffer, u8SsidList[0], is the number of SSIDs encoded in the string. The number of hidden SSIDs cannot exceed MAX_HIDDEN_SITES. All SSIDs are concatenated in the following bytes and each SSID is prefixed with a one-byte header containing its length. The total number of bytes in u8SsidList buffer, including length byte, cannot exceed 133 bytes (MAX_HIDDEN_SITES SSIDs x 32 bytes each, which is max SSID length). For instance, encoding the two hidden SSIDs "DEMO_AP" and "TEST" results in the following buffer content:

```c
uint8 u8SsidList[14];
u8SsidList[0] = 2; // Number of SSIDs is 2
u8SsidList[1] = 7; // Length of the string "DEMO_AP" without NULL termination
memcpy(&u8SsidList[2], "DEMO_AP", 7); // Bytes index 2-9 containing the string DEMO_AP
```
u8SsidList[9] = 4; // Length of the string "TEST" without NULL termination
memcpy(&u8SsidList[10], "TEST", 4); // Bytes index 10-13 containing the string TEST

**Warning**
This function is not allowed in P2P. It works only for STA/AP mode (connected or disconnected).

**Precondition**
- A Wi-Fi notification callback of type tpfAppWifiCb MUST be implemented and registered at initialization. Registering the callback is done through passing it to the m2m_wifi_init.
- The events M2M_WIFI_RESP_SCAN_DONE and M2M_WIFI_RESP_SCAN_RESULT must be handled in the callback.
- The m2m_wifi_handle_events function MUST be called to receive the responses in the callback.

**See also**
M2M_WIFI_RESP_SCAN_DONE
M2M_WIFI_RESP_SCAN_RESULT tpfAppWifiCb
tstrM2mWifiscanResult tenuM2mScanCh m2m_wifi_init
m2m_wifi_handle_events m2m_wifi_req_scan_result

**Returns**
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
Example

The code snippet demonstrates an example of how the scan request is called from the application's main function and the handling of the events received in response.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

static void request_scan_hidden_demo_ap(void);

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    static uint8 u8ScanResultIdx = 0;

    switch(u8WiFiEvent)
    {
    case M2M_WIFI_RESP_SCAN_DONE:
    {
        tstrM2mScanDone *pstrInfo =
            (tstrM2mScanDone*)pvMsg;

        printf("Num of AP found %d\n",pstrInfo->u8NumofCh);
        if(pstrInfo->s8ScanState == M2M_SUCCESS)
        {
            u8ScanResultIdx = 0;
            if(pstrInfo->u8NumofCh >= 1)
            {
                m2m_wifi_req_scan_result(u8ScanResultIdx);
                u8ScanResultIdx ++;
            }
        }
        else
        {
            printf("No AP Found Rescan\n");
        }
    }
    ```
request_scan_hidden_demo_ap();

else
{
    printf("(ERR) Scan fail with error <\%d>\n", pstrInfo->s8ScanState);
}
break;

case M2M_WIFI_RESP_SCAN_RESULT:
{
tstrM2mWifiscanResult *pstrScanResult =
(tstrM2mWifiscanResult*)pvMsg;
uint8 u8NumFoundAPs =
m2m_wifi_get_num_ap_found();

    printf(">%02d RI %d SEC %s CH %02d BSSID %02X:%02X:%02X:%02X:%02X:%02X SSID %s\n",
           pstrScanResult->u8index,
pstrScanResult->s8rssi,
pstrScanResult->u8AuthType,
pstrScanResult->u8ch,
pstrScanResult->au8BSSID[0],
pstrScanResult->au8BSSID[1],
pstrScanResult->au8BSSID[2],
pstrScanResult->au8BSSID[3],
pstrScanResult->au8BSSID[4],
pstrScanResult->au8BSSID[5],
pstrScanResult->au8SSID);

    if(u8ScanResultIdx < u8NumFoundAPs)
    {
        // Read the next scan result
        m2m_wifi_req_scan_result(index);
    }
u8ScanResultIdx ++;
}
break;
default:
break;
}

static void request_scan_hidden_demo_ap(void)
{
    uint8 list[9];
    char ssid[] = "DEMO_AP";
    uint8 len = (uint8)(sizeof(ssid)-1);

    list[0] = 1;
    list[1] = len;
    memcpy(&list[2], ssid, len); // copy 7 bytes
    // Scan all channels
    m2m_wifi_request_scan_ssid_list(M2M_WIFI_CH_ALL,
        list);
}

int main()
{
    tstrWifiInitParam param;

    param.pfAppWifiCb = wifi_event_cb;
    if(!m2m_wifi_init(&param))
    {
        request_scan_hidden_demo_ap();
        while(1)
        {
            m2m_wifi_handle_events(NULL);
        }
    }
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_get_num_ap_found</td>
<td>WLAN » Function</td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>NMI_API</th>
<th>uint8</th>
<th>m2m_wifi_get_num_ap_found (void)</th>
</tr>
</thead>
</table>
Detailed Description

Synchronous function to retrieve the number of AP's found in the last scan request. The function reads the number of APs from global variable which was updated in the Wi-Fi callback function through the M2M_WIFI_RESP_SCAN_DONE event. Function used only in STA mode only.
Function Documentation
**m2m_wifi_get_num_ap_found()**

```c
NMI_API uint8 m2m_wifi_get_num_ap_found ( void )
```

See also
- `m2m_wifi_request_scan`
- `M2M_WIFI_RESP_SCAN_DONE`
- `M2M_WIFI_RESP_SCAN_RESULT`

**Precondition**
- `m2m_wifi_request_scan` need to be called first
  - A Wi-Fi notification callback of type `tpfAppWifiCb` MUST be implemented and registered at initialization. Registering the callback is done through passing it to the `m2m_wifi_init`
  - The event `M2M_WIFI_RESP_SCAN_DONE` must be handled in the callback to receive the requested scan information.

**Warning**
- This function must be called only in the wi-fi callback function when the events `M2M_WIFI_RESP_SCAN_DONE` or `M2M_WIFI_RESP_SCAN_RESULT` are received. Calling this function in any other place will result in undefined/outdated numbers.

**Returns**
- Return the number of AP's found in the last Scan Request.
Example

The code snippet demonstrates an example of how the scan request is called from the application's main function and the handling of the events received in response.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    static uint8 u8ScanResultIdx = 0;

    switch(u8WiFiEvent)
    {
    case M2M_WIFI_RESP_SCAN_DONE:
        {
            tstrM2mScanDone *pstrInfo = (tstrM2mScanDone*)pvMsg;

            printf("Num of AP found %d\n",pstrInfo->u8NumofCh);
            if(pstrInfo->s8ScanState == M2M_SUCCESS)
            {
                u8ScanResultIdx = 0;
                if(pstrInfo->u8NumofCh >= 1)
                {
                    m2m_wifi_req_scan_result(u8ScanResultIdx);
                    u8ScanResultIdx ++;
                }
            }
            else
            {
                printf("No AP Found Rescan\n");
                m2m_wifi_request_scan(M2M_WIFI_CH_ALL);
            }
        } break;
    ...
```
else
{
    printf("(ERR) Scan fail with error <\%d>\n",pstrInfo->s8ScanState);
}
break;

case M2M_WIFI_RESP_SCAN_RESULT:
{
tstrM2mWifiscanResult *pstrScanResult =
    (tstrM2mWifiscanResult*)pvMsg;
uint8 u8NumFoundAPs =
    m2m_wifi_get_num_ap_found();

    printf(">>\%02d RI %d SEC %s CH %02d BSSID %02X:%02X:%02X:%02X:%02X:%02X SSID %s
",%

    pstrScanResult->u8index,pstrScanResult->s8rssi,
pstrScanResult->u8AuthType,
pstrScanResult->u8ch,
pstrScanResult->au8BSSID[0],
pstrScanResult->au8BSSID[1], pstrScanResult-
    ->au8BSSID[2],
pstrScanResult->au8BSSID[3],
pstrScanResult->au8BSSID[4], pstrScanResult-
    ->au8BSSID[5],
pstrScanResult->au8SSID);

if(u8ScanResultIdx < u8NumFoundAPs)
{
    // Read the next scan result
    m2m_wifi_req_scan_result(index);
    u8ScanResultIdx ++;
}
```c
break;
default:
break;
}

int main()
{
tstrWifiInitParam  param;
    param.pfAppWifiCb  = wifi_event_cb;
if(!m2m_wifi_init(&param))
{
    // Scan all channels
m2m_wifi_request_scan(M2M_WIFI_CH_ALL);

while(1)
{
    m2m_wifi_handle_events(NULL);
}
}
```
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m2m_wifi_req_scan_result</code></td>
<td>WLAN » Function</td>
</tr>
</tbody>
</table>
Functions

NMI_API sint8  m2m_wifi_req_scan_result (uint8 index)
Detailed Description

Synchronous call to read the AP information from the SCAN Result list with the given index. This function is expected to be called when the response events M2M_WIFI_RESP_SCAN_RESULT or M2M_WIFI_RESP_SCAN_DONE are received in the wi-fi callback function. The response information received can be obtained through the casting to the tstrM2mWifiscanResult structure.
Function Documentation
m2m_wifi_req_scan_result()

NMI_API sint8 m2m_wifi_req_scan_result ( uint8  index )

Parameters

[in] index Index for the requested result, the index range start from 0 till number of AP's found

See also

tstrM2mWifiscanResult m2m_wifi_get_num_ap_found
m2m_wifi_request_scan

Precondition

m2m_wifi_request_scan needs to be called first, then m2m_wifi_get_num_ap_found to get the number of AP's found

- A Wi-Fi notification callback of type tpfAppWifiCb MUST be implemented and registered at startup. Registering the callback is done through passing it to the m2m_wifi_init function.
- The event M2M_WIFI_RESP_SCAN_RESULT must be handled in the callback to receive the requested scan information.

Warning

Function used in STA mode only. the scan results are updated only if the scan request is called. Calling this function only without a scan request will lead to firmware errors. Refrain from introducing a large delay between the scan request and the scan result request, to prevent errors occurring.

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
Example

The code snippet demonstrates an example of how the scan request is called from the application's main function and the handling of the events received in response.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    static uint8 u8ScanResultIdx = 0;

    switch(u8WiFiEvent)
    {
    case M2M_WIFI_RESP_SCAN_DONE:
    {
        tstrM2mScanDone *pstrInfo = (tstrM2mScanDone*)pvMsg;

        printf("Num of AP found %d\n",pstrInfo->u8NumofCh);
        if(pstrInfo->s8ScanState == M2M_SUCCESS)
        {
            u8ScanResultIdx = 0;
            if(pstrInfo->u8NumofCh >= 1)
            {
                m2m_wifi_req_scan_result(u8ScanResultIdx);
                u8ScanResultIdx ++;
            }
            else
            {
                printf("No AP Found Rescan\n");
                m2m_wifi_request_scan(M2M_WIFI_CH_ALL);
            }
        }
    }
```
} else {
    printf("(ERR) Scan fail with error <\%d\n", pstrInfo->s8ScanState);
}
break;

case M2M_WIFI_RESP_SCAN_RESULT:
{
tstrM2mWifiscanResult *pstrScanResult =
    (tstrM2mWifiscanResult*)pvMsg;
uint8 u8NumFoundAPs =
    m2m_wifi_get_num_ap_found();

    printf(">>%02d RI %d SEC %s CH %02d BSSID %02X:%02X:%02X:%02X:%02X:%02X SSID %s\n",
            pstrScanResult->u8index, pstrScanResult->s8rssi,
            pstrScanResult->u8AuthType,
            pstrScanResult->u8ch,
            pstrScanResult->au8BSSID[0],
            pstrScanResult->au8BSSID[1], pstrScanResult->au8BSSID[2],
            pstrScanResult->au8BSSID[3],
            pstrScanResult->au8BSSID[4], pstrScanResult->au8BSSID[5],
            pstrScanResult->au8SSID);

    if(u8ScanResultIdx < u8NumFoundAPs)
    {
        // Read the next scan result
        m2m_wifi_req_scan_result(index);
        u8ScanResultIdx ++;
    }

{
    break;
    default:
    break;
}

int main()
{
tstrWifiInitParam  param;

    param.pfAppWifiCb  = wifi_event_cb;
if(!m2m_wifi_init(&param))
    {
        // Scan all channels
        m2m_wifi_request_scan(M2M_WIFI_CH_ALL);

        while(1)
            {
            m2m_wifi_handle_events(NULL);
                }
    }
}
m2m_wifi_req_curr_rssi
WLAN » Function
Functions

<table>
<thead>
<tr>
<th>NMI_API sint8</th>
<th>m2m_wifi_req_curr_rssi (void)</th>
</tr>
</thead>
</table>


Detailed Description

Asynchronous request for the current RSSI of the connected AP. The response received in through the `M2M_WIFI_RESP_CURRENT_RSSI` event.
Function Documentation
◆ m2m_wifi_req_curr_rssi()

**NMI_API** `sint8 m2m_wifi_req_curr_rssi (void)`

**Precondition**
- A Wi-Fi notification callback of type `tpfAppWifiCb` MUST be implemented and registered before initialization. Registering the callback is done through passing it to the `m2m_wifi_init` through the `tstrWifInitParam` initialization structure.
  - The event `M2M_WIFI_RESP_CURRENT_RSSI` must be handled in the callback to receive the requested Rssi information.

**Returns**
- The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
The code snippet demonstrates how the RSSI request is called in the application's main function and the handling of the event received in the callback.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

void wifi_event_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    static uint8 u8ScanResultIdx = 0;

    switch(u8WiFiEvent)
    {
    case M2M_WIFI_RESP_CURRENT_RSSI:
    {
        sint8 *rssi = (sint8*)pvMsg;
        M2M_INFO("ch rssi %d\n",*rssi);
    }
    break;
    default:
    break;
}
}

int main()
{
    tstrWifiInitParam param;

    param.pfAppWifiCb = wifi_event_cb;
    if(!m2m_wifi_init(&param))
    {
        // Scan all channels
        m2m_wifi_req_curr_rssi();
    }
```
while(1)
{
    m2m_wifi_handle_events(NULL);
}
}
m2m_wifi_get_otp_mac_address

WLAN » Function
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API  sint8 m2m_wifi_get_otp_mac_address (uint8 *pu8MacAddr, uint8 *pu8IsValid)</td>
</tr>
</tbody>
</table>
Detailed Description

Request the MAC address stored on the One Time Programmable (OTP) memory of the device. The function is blocking until the response is received.
Function Documentation
◆ m2m_wifi_get_otp_mac_address()

```c
NMI_API sint8 m2m_wifi_get_otp_mac_address ( uint8 * pu8MacAddr,
                             uint8 * pu8IsValid
)
```

**Parameters**

- `[out] pu8MacAddr` Output MAC address buffer of 6 bytes size. Valid only if *pu8Valid=1.
- `[out] pu8IsValid` Output boolean value to indicate the validity of pu8MacAddr in OTP. Output zero if the OTP memory is not programmed, non-zero otherwise.

**Precondition**

m2m_wifi_init required to be called before any WIFI/socket function

**See also**

m2m_wifi_get_mac_address

**Returns**

The function returns **M2M_SUCCESS** for success and a negative value otherwise.
m2m_wifi_get_mac_address

WLAN » Function
Functions

NMI_API sint8 m2m_wifi_get_mac_address (uint8 *pu8MacAddr)
Detailed Description

Function to retrieve the current MAC address. The function is blocking until the response is received.
Function Documentation
m2m_wifi_get_mac_address()

NMI_API sint8 m2m_wifi_get_mac_address ( uint8 * pu8MacAddr )

Parameters
   [out] pu8MacAddr Output MAC address buffer of 6 bytes size.

Precondition
   m2m_wifi_init required to be called before any WIFI/socket function

See also
   m2m_wifi_get_otp_mac_address

Returns
   The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_wifi_set_sleep_mode

WLAN » Function
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NMI_API</strong> sint8</td>
</tr>
</tbody>
</table>
Detailed Description

This is one of the two synchronous power-save setting functions that allow the host MCU application to tweak the system power consumption. Such tweaking can be done through one of two ways: 1) Changing the power save mode, to one of the allowed power save modes `tenuPowerSaveModes`. This is done by setting the first parameter 2) Configuring DTIM monitoring: Configuring beacon monitoring parameters by enabling or disabling the reception of broadcast/multicast data. This is done by setting the second parameter.
Function Documentation
**m2m_wifi_set_sleep_mode()**

```c
NMI_API sint8 m2m_wifi_set_sleep_mode (uint8 PsTyp,
                                   uint8 BcastEn)
```

**Parameters**

- **[in] PsTyp** Desired power saving mode. Supported types are enumerated in `tenuPowerSaveModes`.
- **[in] BcastEn** Broadcast reception enable flag. If it is 1, the WINC1500 will be awake each DTIM beacon for receiving broadcast traffic. If it is 0, the WINC1500: disable broadcast traffic. Through this flag the WINC1500 will not wakeup at the DTIM beacon, but it will wakeup depends only on the the configured Listen Interval.

**Warning**

The function called once after initialization.

**See also**

- `tenuPowerSaveModes`
- `m2m_wifi_get_sleep_mode`
- `m2m_wifi_set_lsn_int`

**Returns**

The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
m2m_wifi_request_sleep

WLAN » Function
Functions

NMI_API sint8 m2m_wifi_request_sleep (uint32 u32SlpReqTime)
Detailed Description

Synchronous power-save sleep request function, which requests from the WINC1500 device to sleep in the currently configured power save mode as defined by the `m2m_wifi_set_sleep_mode`, for a specific time as defined by the passed in parameter. This function should be used in the `M2M_PS_MANUAL` power save mode only. A wake up request is automatically performed by the WINC1500 device when any host driver API function, e.g. Wi-Fi or socket operation is called.
Function Documentation
m2m_wifi_request_sleep()

NMI_API sint8 m2m_wifi_request_sleep ( uint32 u32SlpReqTime )

Parameters

[in] u32SlpReqTime Request sleep time in ms The best recommended sleep duration is left to be determined by the application. Taking into account that if the application sends notifications very rarely, sleeping for a long time can be a power-efficient decision. In contrast applications that are sensitive for long periods of absence can experience performance degradation in the connection if long sleeping times are used.

Warning

The function should be called in M2M_PS_MANUAL power save mode only. As enumerated in tenuPowerSaveModes It's also important to note that during the sleeping time while in the M2M_PS_MANUAL mode, AP beacon monitoring is bypassed and the wifi-connection may drop if the sleep period is elongated.

See also

tenuPowerSaveModes m2m_wifi_set_sleep_mode

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_wifi_get_sleep_mode
WLAN » Function
## Functions

| NMI_API uint8 m2m_wifi_get_sleep_mode (void) |
Detailed Description

Synchronous power save mode retrieval function.
Function Documentation
◆ m2m_wifi_get_sleep_mode()

```
NMI_API uint8 m2m_wifi_get_sleep_mode ( void )
```

See also
- tenuPowerSaveModes m2m_wifi_set_sleep_mode

Returns
The current operating power saving mode based on the enumerated sleep modes `tenuPowerSaveModes`.
m2m_wifi_req_client_ctrl

WLAN » Function
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NMI_API</strong> sint8 m2m_wifi_req_client_ctrl (uint8 cmd)</td>
</tr>
</tbody>
</table>


Detailed Description

Asynchronous command sending function to the PS Client (An WINC1500 board running the ps_firmware) if the PS client send any command it will be received through the M2M_WIFI_RESP_CLIENT_INFO event
Function Documentation
m2m_wifi_req_client.ctrl()

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>Control command sent from PS Server to PS Client (command values defined by the application)</td>
</tr>
</tbody>
</table>

**Precondition**

m2m_wifi_req_server_init should be called first

**Warning**

This mode is not supported in the current release.

**See also**

m2m_wifi_req_server_init M2M_WIFI_RESP_CLIENT_INFO

**Returns**

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_req_server_init</td>
</tr>
<tr>
<td>Functions</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td><strong>NMI_API sint8 m2m_wifi_req_server_init (uint8 ch)</strong></td>
</tr>
</tbody>
</table>
Detailed Description

Synchronous function to initialize the PS Server. The WINC1500 supports non secure communication with another WINC1500, (SERVER/CLIENT) through one byte command (probe request and probe response) without any connection setup. The server mode can't be used with any other modes (STA/P2P/AP)
Function Documentation
**m2m_wifi_req_server_init()**

```
NMI_API sint8 m2m_wifi_req_server_init ( uint8 ch )
```

**Parameters**

[in] **ch** Server listening channel

**See also**

m2m_wifi_req_client_ctrl

**Warning**

This mode is not supported in the current release.

**Returns**

The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_set_device_name</td>
</tr>
</tbody>
</table>

WLAN » Function
### Functions

| NMI_API sint8 m2m_wifi_set_device_name (uint8 *pu8DeviceName, uint8 u8DeviceNameLength) |
Detailed Description

Sets the WINC device name. The name string is used as a device name in both (P2P) WiFi-Direct mode as well as DHCP hostname (option 12). For P2P devices to communicate a device name must be present. If it is not set through this function a default name is assigned. The default name is WINC-XX-YY, where XX and YY are the last 2 octets of the OTP MAC address. If OTP (eFuse) is programmed, then the default name is WINC-00-00.
Function Documentation
◆ m2m_wifi_set_device_name()

NMI_API sint8 m2m_wifi_set_device_name ( uint8 * pu8DeviceName,  
                                           uint8 u8DeviceNameLength )

Parameters

[in] pu8DeviceName A Buffer holding the device name. Device name is a null terminated C string.

[in] u8DeviceNameLength The length of the device name. Should not exceed the maximum device name's length M2M_DEVICE_NAME_MAX (including null character).

Warning

The function called once after initialization. Used for the Wi-Fi Direct (P2P) as well as DHCP client hostname option (12).
Device name shall contain only characters allowed in valid internet host name as defined in RFC 952 and 1123.

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
### m2m_wifi_set_lsn_int

#### WLAN » Function

**WINC1500 IoT Software APIs 19.5.2**

**WINC Software API Reference Manual**
## Functions

| NMI_API sint8 m2m_wifi_set_lsn_int (tstrM2mLsnInt *pstrM2mLsnInt) |  |
Detailed Description

This is one of the two synchronous power-save setting functions that allow the host MCU application to tweak the system power consumption. Such tweaking can be done by modifying the Wi-Fi listen interval. The listen interval is how many beacon periods the station can sleep before it wakes up to receive data buffer in AP. It is represented in units of AP beacon periods (100ms).
Function Documentation
m2m_wifi_set_lsn_int()

NMI_API sint8 m2m_wifi_set_lsn_int ( tstrM2mLsnInt * pstrM2mLsnInt )

Parameters
[in] pstrM2mLsnInt Structure holding the listen interval configurations.

Precondition
Function shall be called first, to set the power saving mode required.

Warning
The function should be called once after initialization.

See also
tstrM2mLsnInt m2m_wifi_set_sleep_mode

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_enable_monitoring_mode</td>
</tr>
</tbody>
</table>

WLAN » Function
Functions

<table>
<thead>
<tr>
<th>NMI_API</th>
<th>sint8</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_enable_monitoring_mode (tstrM2MWifiMonitorModeCtrl *pstrMtrCtrl, uint8 *pu8PayloadBuffer, uint16 u16BufferSize, uint16 u16DataOffset)</td>
<td></td>
</tr>
</tbody>
</table>
Asynchronous Wi-Fi monitoring mode (Promiscuous mode) enabling function. This function enables the monitoring mode, thus allowing two operations to be performed: 1) Transmission of manually configured frames, through using the `m2m_wifi_send_wlan_pkt` function. 2) Reception of frames based on a defined filtering criteria. When the monitoring mode is enabled, reception of all frames that satisfy the filter criteria passed in as a parameter is allowed, on the current wireless channel.

All packets that meet the filtering criteria are passed to the application layer, to be handled by the assigned monitoring callback function.

The monitoring callback function must be implemented before starting the monitoring mode, in-order to handle the packets received.

Registering of the implemented callback function is through the callback pointer `tpfAppMonCb` in the `tstrWifiInitParam` structure.

Passed to `m2m_wifi_init` function at initialization.
m2m_wifi_enable_monitoring_mode()

NMI_API sint8
m2m_wifi_enable_monitoring_mode ( tstrM2MWifiMonitorModeCtrl * pstrMtrCtrl, uint8 * pu8PayloadBuffer, uint16 u16BufferSize, uint16 u16DataOffset )

Parameters

[in] pstrMtrCtrl Pointer to tstrM2MWifiMonitorModeCtrl holding the filtering parameters.

[in] pu8PayloadBuffer Pointer to a buffer allocated by the application. It SHALL hold the Data field of the WIFI RX packet (Or a part from it). If it is set to NULL, the WIFI data payload will be discarded by the monitoring driver.

[in] u16BufferSize The total size of the pu8PayloadBuffer in bytes.

[in] u16DataOffset Starting offset in the DATA FIELD of the received WIFI packet. The application may be interested in reading specific information from the received packet. It must assign the offset to the starting position of DATA payload start.

Example, if the SSID is needed to be read from a REQ packet, the u16Offset MUST be set to 0.

Warning
When this mode is enabled, you cannot be connected in any mode (Station, Access Point, or P2P).

See also
tstrM2MWifiMonitorModeCtrl tstrM2MWifiRxPacketInfo tstrWifiInitParam tenuM2mScanCh m2m_wifi_disable_monitoring_mode m2m_wifi_send_wlan_pkt m2m_wifi_send_ethernet_pkt

Returns
The function returns M2M_SUCCESS for successful operations and
value otherwise.
Example

The example demonstrates the main function where-by the monitoring enable function is called after the initialization of the driver and the packets are handled in the callback function.

```c
#include "m2m_wifi.h"
#include "m2m_types.h"

//Declare receive buffer
uint8 gmgmt[1600];

//Callback functions
void wifi_cb(uint8 u8WiFiEvent, void * pvMsg)
{
    //
}
void wifi_monitoring_cb(tstrM2MWifiRxPacketInfo *pstrWifiRxPacket, uint8 *pu8Payload, uint16 u16PayloadSize)
{
    if((NULL != pstrWifiRxPacket) && (0 != u16PayloadSize)) {
        if(MANAGEMENT == pstrWifiRxPacket->u8FrameType) {
            M2M_INFO("### MGMT PACKET ###\n");
        } else if(DATA_BASICTYPE == pstrWifiRxPacket->u8FrameType) {
            M2M_INFO("### DATA PACKET ###\n");
        } else if(CONTROL == pstrWifiRxPacket->u8FrameType) {
            M2M_INFO("### CONTROL PACKET ###\n");
        }
    }
}
```
```c
int main()
{
    //Register wifi_monitoring_cb
    tstrWifiInitParam param;
    param.pfAppWifiCb = wifi_cb;
    param.pfAppMonCb = wifi_monitoring_cb;

    nm_bsp_init();

    if(!m2m_wifi_init(&param)) {
        //Enable Monitor Mode with filter to receive all data frames on channel 1
        tstrM2MWifiMonitorModeCtrl strMonitorCtrl = {0};
        strMonitorCtrl.u8ChannelID = M2M_WIFI_CH_1;
        strMonitorCtrl.u8FrameType = DATA_BASICTYPE;
        strMonitorCtrl.u8FrameSubtype = M2M_WIFI_FRAME_SUB_TYPE_ANY; //Receive any subtype of data frame
        m2m_wifi_enable_monitoring_mode(&strMonitorCtrl, gmgmt, sizeof(gmgmt), 0);

        while(1) {
            m2m_wifi_handle_events(NULL);
        }
    }
    return 0;
}
```
m2m_wifi_disable_monitoring_mode

WLAN » Function
Functions

| NMI_API sint8  | m2m_wifi_disable_monitoring_mode (void) |
Detailed Description

Synchronous function to disable Wi-Fi monitoring mode (Promiscuous mode). Expected to be called, if the enable monitoring mode is set, but if it was called without enabling no negative impact will reside.
Function Documentation
m2m_wifi_disable_monitoring_mode()

NMI_API sint8 m2m_wifi_disable_monitoring_mode ( void )

See also

m2m_wifi_enable_monitoring_mode

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_send_wlan_pkt</td>
</tr>
</tbody>
</table>

WLAN » Function
## Functions

| NMI_API sint8 | m2m_wifi_send_wlan_pkt (uint8 *pu8WlanPacket, uint16 u16WlanHeaderLength, uint16 u16WlanPktSize) |
Detailed Description

Synchronous function to transmit a WIFI RAW packet while the implementation of this packet is left to the application developer.
m2m_wifi_send_wlan_pkt()

NMI_API sint8 m2m_wifi_send_wlan_pkt ( uint8 * pu8WlanPacket, uint16 u16WlanHeaderLength, uint16 u16WlanPktSize )

Parameters

[in] pu8WlanPacket Pointer to a buffer holding the whole WIFI frame.

[in] u16WlanHeaderLength The size of the WIFI packet header ONLY.

[in] u16WlanPktSize The size of the whole bytes in packet.

See also
m2m_wifi_enable_monitoring_mode
m2m_wifi_disable_monitoring_mode

Precondition
Enable Monitoring mode first using m2m_wifi_enable_monitoring_mode

Warning
This function available in monitoring mode ONLY.

Note
Packets are user's responsibility.

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>m2m_wifi_send_ethernet_pkt</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN » Function</td>
</tr>
</tbody>
</table>
### Functions

| NMI_API sint8 | m2m_wifi_send_ethernet_pkt (uint8 *pu8Packet, uint16 u16PacketSize) |
Detailed Description

Synchronous function to transmit an Ethernet packet. Transmit a packet directly in ETHERNET/bypass mode where the TCP/IP stack is disabled and the implementation of this packet is left to the application developer. The Ethernet packet composition is left to the application developer.
Function Documentation
m2m_wifi_send_ethernet_pkt()

NMI_API sint8 m2m_wifi_send_ethernet_pkt ( uint8 * pu8Packet, uint16 u16PacketSize )

Parameters
- [in] pu8Packet Pointer to a buffer holding the whole Ethernet frame.
- [in] u16PacketSize The size of the whole bytes in packet.

Warning
This function available in ETHERNET/Bypass mode ONLY. Make sure that application defines ETH_MODE.

Note
Packets are the user's responsibility.

See also
m2m_wifi_enable_mac_mcast, m2m_wifi_set_receive_buffer

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
m2m_wifi_enable_sntp

WLAN » Function
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NMI_API sint8 m2m_wifi_enable_sntp (uint8 bEnable)</strong></td>
</tr>
</tbody>
</table>
Detailed Description

Synchronous function to enable/disable the native Simple Network Time Protocol (SNTP) client in the WINC1500 firmware. The SNTP is enabled by default at start-up. The SNTP client at firmware is used to synchronize the system clock to the UTC time from the well-known time servers (e.g. "time-c.nist.gov"). The SNTP client uses a default update cycle of 1 day. The UTC is important for checking the expiration date of X509 certificates used while establishing TLS (Transport Layer Security) connections. It is highly recommended to use it if there is no other means to get the UTC time. If there is a RTC on the host MCU, the SNTP could be disabled and the host should set the system time to the firmware using the m2m_wifi_set_system_time function.
Function Documentation
◆ m2m_wifi_enable_sntp()

NMI_API sint8 m2m_wifi_enable_sntp ( uint8 bEnable )

Parameters

[in] bEnable Enabling/Disabling flag '0': disable SNTP '1': enable SNTP

See also
m2m_wifi_set_system_time

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_set_system_time</td>
</tr>
</tbody>
</table>

WLAN » Function
Functions

NMI_API sint8 m2m_wifi_set_system_time (uint32 u32UTCSeconds)
Detailed Description

Synchronous function for setting the system time in time/date format (\texttt{uint32}). The \texttt{tstrSystemTime} structure can be used as a reference to the time values that should be set and pass its value as \texttt{uint32}.
Function Documentation
**m2m_wifi_set_system_time()**

```c
NMI_API sint8 m2m_wifi_set_system_time ( uint32 u32UTCSeconds )
```

**Parameters**

[in] **u32UTCSeconds** Seconds elapsed since January 1, 1900 (NTP Timestamp).

**See also**

m2m_wifi_enable_sntp tstrSystemTime

**Note**

If there is an RTC on the host MCU, the SNTP could be disabled and the host should set the system time to the firmware using the API `m2m_wifi_set_system_time`.

**Returns**

The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
m2m_wifi_get_system_time

WLAN » Function
| NMI_API sint8 m2m_wifi_get_system_time (void) |
Asynchronous function used to retrieve the system time through the use of the response `M2M_WIFI_RESP_GET_SYS_TIME`. Response time retrieved is parsed into the members defined in the structure `tstrSystemTime`.
Function Documentation
**m2m_wifi_get_system_time()**

```c
NMI_API sint8 m2m_wifi_get_system_time ( void )
```

See also

- `m2m_wifi_enable_snpt tstrSystemTime`

**Note**

Get the system time from the SNTP client using the API `m2m_wifi_get_system_time`.

**Returns**

The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>m2m_wifi_set_cust_InfoElement</strong></td>
</tr>
<tr>
<td>WLAN » Function</td>
</tr>
</tbody>
</table>
## Functions

| NMI_API | sint8 m2m_wifi_set_cust_InfoElement (uint8 *pau8M2mCustInfoElement) |
Detailed Description

Synchronous function to Add/Remove user-defined Information Element to the WIFIBeacon and Probe Response frames while chip mode is Access Point Mode.

According to the information element layout shown bellow, if it is required to set new data for the information elements, pass in the buffer with the information according to the sizes and ordering defined bellow. However, if it's required to delete these IEs, fill the buffer with zeros.
Function Documentation
m2m_wifi_set_cust_InfoElement()

**NMI_API sint8 m2m_wifi_set_cust_InfoElement (uint8 * pau8M2mCustInfoElement)**

**Parameters**

[in] **pau8M2mCustInfoElement** Pointer to Buffer containing the IE element's ordering passed in.

**Warning**

- Size of All elements combined must not exceed 255 byte.
  - Used in Access Point Mode

**Note**

IEs Format will be follow the following layout:

<table>
<thead>
<tr>
<th># of all Bytes</th>
<th>IE1 ID</th>
<th>Length1</th>
</tr>
</thead>
<tbody>
<tr>
<td># of all Bytes</td>
<td>IE1 ID</td>
<td>Length1</td>
</tr>
</tbody>
</table>

See also

m2m_wifi_enable_sntp tstrSystemTime

**Returns**

The function returns **M2M_SUCCESS** for successful operations and
Example

The example demonstrates how the information elements are set using this function.

```c
char elementData[21];
static char state = 0; // To Add, Append, and Delete
if(0 == state) { //Add 3 IEs
    state = 1;
    //Total Number of Bytes
    elementData[0]=12;
    //First IE
    elementData[3]='A';
    //Second IE
    //Third IE
    elementData[12]='F';
} else if(1 == state) {
    //Append 2 IEs to others, Notice that we keep old data in array starting with\n    //element 13 and total number of bytes increased to 20
    state = 2;
    //Total Number of Bytes
    elementData[0]=20;
    //Fourth IE
    elementData[15]='G';
    //Fifth IE
    elementData[16]=204; elementData[17]=3;
```
elementData[18] = 'X'; elementData[19] = 5;
    elementData[20] = 'Z';
} else if (2 == state) {
    // Delete All IEs
    state = 0;
    // Total Number of Bytes
    elementData[0] = 0;
}

m2m_wifi_set_cust_InfoElement(elementData);
m2m_wifi_set_power_profile

WLAN » Function
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sint8 m2m_wifi_set_power_profile (uint8 u8PwrMode)</code></td>
</tr>
</tbody>
</table>
Detailed Description

Change the power profile mode
◆ m2m_wifi_set_power_profile()

NMI_API sint8 m2m_wifi_set_power_profile ( uint8 u8PwrMode )

Parameters

[in] u8PwrMode Change the WINC1500 power profile to different mode based on the enumeration tenuM2mPwrMode

Precondition

Must be called after the initializations and before any connection request and can't be changed in run time.

See also

tenuM2mPwrMode m2m_wifi_init

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
# Functions

## m2m_wifi_set_tx_power

### WLAN » Function
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sint8 m2m_wifi_set_tx_power (uint8 u8TxPwrLevel)</td>
<td>Set the transmitter power level.</td>
</tr>
</tbody>
</table>
Detailed Description

Set the TX power tenuM2mTxPwrLevel
Function Documentation
m2m_wifi_set_tx_power()

NMI_API sint8 m2m_wifi_set_tx_power ( uint8 u8TxPwrLevel )

Parameters

[in] 

u8TxPwrLevel change the TX power based on the enumeration tenuM2mTxPwrLevel

Precondition

Must be called after the initialization and before any connection request and can't be changed in runtime.

See also

tenuM2mTxPwrLevel m2m_wifi_init

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_enable_firmware_logs</td>
</tr>
<tr>
<td>WLAN » Function</td>
</tr>
</tbody>
</table>
Functions

```c
sint8 m2m_wifi_enable_firmware_logs (uint8 u8Enable)
```
Enable or Disable logs in run time (Disabling Firmware logs will enhance the firmware start-up time and performance)
m2m_wifi_enable_firmware_logs()

NMI_API sint8 m2m_wifi_enable_firmware_logs ( uint8 u8Enable )

Parameters
[ in ] u8Enable Set 1 to enable the logs, 0 for disable

Precondition
Must be called after initialization through the following function
m2m_wifi_init

See also
DISABLE_FIRMWARE_LOGS (build option to disable logs from initializations) m2m_wifi_init

Returns
The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_set_battery_voltage</td>
</tr>
</tbody>
</table>

WLAN » Function
## Functions

```
sint8  m2m_wifi_set_battery_voltage (uint16 u16BattVoltx100)
```
Detailed Description

Set the battery voltage to update the firmware calculations
m2m_wifi_set_battery_voltage()

sint8 m2m_wifi_set_battery_voltage ( uint16 u16BattVoltx100 )
### m2m_wifi_set_gains

WLAN » Function
Functions

`sint8 m2m_wifi_set_gains (tstrM2mWifiGainsParams *pstrM2mGain)`
Detailed Description

Set the chip gains mainly (PPA for 11b/11gn)
◆ m2m_wifi_set_gains()

```c
sint8 m2m_wifi_set_gains ( tstrM2mWifiGainsParams * pstrM2mGain )
```

Set the chip PPA gain for 11b/11gn.

**Parameters**

- `[in] pstrM2mGain tstrM2mWifiGainsParams` contain gain parameters as implemented in rf document

**Precondition**

Must be called after initialization through the following function `m2m_wifi_init`

**See also**

- `m2m_wifi_init`

**Returns**

The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
### m2m_wifi_get_firmware_version

**WLAN » Function**

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_get_firmware_version</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>sint8  m2m_wifi_get_firmware_version (tstrM2mRev *pstrRev)</td>
</tr>
</tbody>
</table>
Detailed Description

Get Firmware version info as defined in the structure tstrM2mRev.
Function Documentation
◆ m2m_wifi_get_firmware_version()

m2m_wifi_get_firmware_version ( tstrM2mRev * pstrRev )

Parameters

[out] M2mRev Pointer to the structure tstrM2mRev that contains the firmware version parameters

Precondition

Must be called after intialization through the following function

m2m_wifi_init

See also

m2m_wifi_init

Returns

The function returns M2M_SUCCESS for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2m_wifi_prng_get_random_bytes</td>
<td>WLAN Function</td>
</tr>
</tbody>
</table>
**Functions**

```c
sint8 m2m_wifi_prng_get_random_bytes (uint8 *pu8PrngBuff, uint16 u16PrngSize)
```
Detailed Description

Asynchronous function for retrieving from the firmware a pseudo-random set of bytes as specified in the size passed in as a parameter. The registered wifi-cb function retrieves the random bytes through the response M2M_WIFI_RESP_GET_PRNG
Function Documentation
◆ m2m_wifi_prng_get_random_bytes()

```c
sint8 m2m_wifi_prng_get_random_bytes ( uint8 * pu8PRNGBuff,
                                        uint16 u16PRNGSize )
```

**Parameters**
- **[out]** `pu8PrngBuff` Pointer to a buffer to receive data.
- **[in]** `u16PrngSize` Request size in bytes

**Warning**
- Size greater than the maximum specified
  (`M2M_BUFFER_MAX_SIZE - sizeof(tstrPrng)`) causes a negative error `M2M_ERR_FAIL`.

**See also**
- `tstrPrng`

**Returns**
- The function returns `M2M_SUCCESS` for successful operations
  and a negative value otherwise.
DataTypes

WLAN
<table>
<thead>
<tr>
<th>Data Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct tstrM2mPwrMode</td>
</tr>
<tr>
<td>struct tstrM2mTxPwrLevel</td>
</tr>
<tr>
<td>struct tstrM2mEnableLogs</td>
</tr>
<tr>
<td>struct tstrM2mBatteryVoltage</td>
</tr>
<tr>
<td>struct tstrM2mWifiGainsParams</td>
</tr>
<tr>
<td>struct tstrM2mWifiWepParams</td>
</tr>
<tr>
<td>struct tstr1xAuthCredentials</td>
</tr>
<tr>
<td>union tuniM2MWifiAuth</td>
</tr>
<tr>
<td>struct tstrM2MWifiSecInfo</td>
</tr>
<tr>
<td>struct tstrM2mWifiConnect</td>
</tr>
<tr>
<td>struct tstrM2MWPSConnect</td>
</tr>
<tr>
<td>struct tstrM2MWPSInfo</td>
</tr>
<tr>
<td>struct tstrM2MDefaultConnResp</td>
</tr>
<tr>
<td>struct tstrM2MScanOption</td>
</tr>
<tr>
<td>struct tstrM2MScanRegion</td>
</tr>
<tr>
<td>struct tstrM2MScan</td>
</tr>
<tr>
<td>struct tstrCryptoResp</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>struct</td>
</tr>
<tr>
<td>Struct</td>
</tr>
<tr>
<td>-------------------------------</td>
</tr>
<tr>
<td>tstrM2MProvisionModeConfig</td>
</tr>
<tr>
<td>tstrM2MProvisionInfo</td>
</tr>
<tr>
<td>tstrM2MConnInfo</td>
</tr>
<tr>
<td>tstrOtaInitHdr</td>
</tr>
<tr>
<td>tstrOtaControlSec</td>
</tr>
<tr>
<td>tstrOtaUpdateStatusResp</td>
</tr>
<tr>
<td>tstrOtaUpdateInfo</td>
</tr>
<tr>
<td>tstrSystemTime</td>
</tr>
<tr>
<td>tstrM2MMulticastMac</td>
</tr>
<tr>
<td>tstrPrng</td>
</tr>
<tr>
<td>tstrTlsCrlEntry</td>
</tr>
<tr>
<td>tstrTlsCrlInfo</td>
</tr>
<tr>
<td>tstrTlsSrvSecFileEntry</td>
</tr>
<tr>
<td>tstrTlsSrvSecHdr</td>
</tr>
<tr>
<td>tstrSslSetActiveCsList</td>
</tr>
<tr>
<td>tstrM2mPwrState</td>
</tr>
</tbody>
</table>
**Typedefs**

typedef void(* tpfOtaNotifCb) (tstrOtaUpdateInfo *pstrOtaUpdateInfo)

typedef void(* tpfOtaUpdateCb) (uint8 u8OtaUpdateStatusType, uint8 u8OtaUpdateStatus)
Enumerations

tenuOtaError {
    OTA_SUCCESS = (0),
    OTA_ERR_WORKING_IMAGE_LOAD_FAIL = ((sint8) -1),
    OTA_ERR_INVAILD_CONTROL_SEC = ((sint8) -2),
    M2M_ERR_OTA_SWITCH_FAIL = ((sint8) -3),
    M2M_ERR_OTA_START_UPDATE_FAIL = ((sint8) -4),
    M2M_ERR_OTA_ROLLBACK_FAIL = ((sint8) -5),
    M2M_ERR_OTA_INVAILD_FLASH_SIZE = ((sint8) -6),
    M2M_ERR_OTA_INVAILD_ARG = ((sint8) -7),
    M2M_ERR_OTA_INPROGRESS = ((sint8) -8)
}

tenuM2mConnChangedErrcode {
    M2M_ERR_SCAN_FAIL = ((uint8)1), M2M_ERR_JOIN_FAIL,
    M2M_ERR_AUTH_FAIL, M2M_ERR_ASSOC_FAIL,
    M2M_ERR_CONN_INPROGRESS
}

tenuM2mWepKeyIndex { M2M_WIFI_WEP_KEY_INDEX_1 =
    ((uint8) 1), M2M_WIFI_WEP_KEY_INDEX_2,
    M2M_WIFI_WEP_KEY_INDEX_3,
    M2M_WIFI_WEP_KEY_INDEX_4 }

tenuM2mPwrMode { PWR_AUTO = ((uint8) 1), PWR_LOW1,
    PWR_LOW2, PWR_HIGH }

tenuM2mTxPwrLevel { TX_PWR_HIGH = ((uint8) 1),
    TX_PWR_MED, TX_PWR_LOW }

tenuM2mReqGroup {
    M2M_REQ_GROUP_MAIN = 0, M2M_REQ_GROUP_WIFI,
    M2M_REQ_GROUP_IP, M2M_REQ_GROUP_HIF,
    M2M_REQ_GROUP_OTA, M2M_REQ_GROUP_SSL,
    M2M_REQ_GROUP_CRYPTO, M2M_REQ_GROUP_SIGMA
}
enum tenuM2mReqpkt { M2M_REQ_CONFIG_PKT, M2M_REQ_DATA_PKT = 0x80 }

tenuM2mConfigCmd {
    M2M_WIFI_REQ_RESTART = M2M_CONFIG_CMD_BASE,
    M2M_WIFI_REQ_SET_MAC_ADDRESS,
    M2M_WIFI_REQ_CURRENT_RSSI,
    M2M_WIFI_RESP_CURRENT_RSSI,
    M2M_WIFI_REQ_GET_CONN_INFO,
    M2M_WIFI_RESP_CONN_INFO,
    M2M_WIFI_REQ_SET_DEVICE_NAME,
    M2M_WIFI_REQ_START_PROVISION_MODE,
    M2M_WIFI_RESP_PROVISION_INFO,
    M2M_WIFI_REQ_STOP_PROVISION_MODE,
    M2M_WIFI_REQ_SET_SYS_TIME,
    M2M_WIFI_REQ_ENABLE_SNTP_CLIENT,
    M2M_WIFI_REQ_DISABLE_SNTP_CLIENT,
    M2M_WIFI_RESP_MEMORY_RECOVER,
    M2M_WIFI_REQ_CUST_INFO_ELEMENT,
    M2M_WIFI_REQ_SCAN,
    M2M_WIFI_RESP_SCAN_DONE,
    M2M_WIFI_REQ_SCAN_RESULT,
    M2M_WIFI_RESP_SCAN_RESULT,
    M2M_WIFI_REQ_SET_SCAN_OPTION,
    M2M_WIFI_REQ_SET_SCAN_REGION,
    M2M_WIFI_REQ_SET_POWER_PROFILE,
    M2M_WIFI_REQ_SET_TX_POWER,
    M2M_WIFI_REQ_SET_BATTERY_VOLTAGE,
    M2M_WIFI_REQ_SET_ENABLE_LOGS,
    M2M_WIFI_REQ_GET_SYS_TIME,
    M2M_WIFI_RESP_GET_SYS_TIME,
    M2M_WIFI_REQ_SEND_ETHERNET_PACKET,
    M2M_WIFI_RESP_ETHERNET_RX_PACKET,
    M2M_WIFI_REQ_SET_MAC_MCAST,
    M2M_WIFI_REQ_GET_PRNG,
    M2M_WIFI_REQ_SET_PRNG,
    M2M_WIFI_REQ_SCAN_SSID_LIST,
    M2M_WIFI_REQ_SET_GAINS,
    M2M_WIFI_REQ_PASSIVE_SCAN,
    M2M_WIFI_MAX_CONFIG_ALL
}
enum tenuM2mStaCmd {
    M2M_WIFI_REQ_CONNECT = M2M_STA_CMD_BASE,
    M2M_WIFI_REQ_DEFAULT_CONNECT,
    M2M_WIFI_RESP_DEFAULT_CONNECT,
    M2M_WIFI_REQ_DISCONNECT,
    M2M_WIFI_RESP_CON_STATE_CHANGED,
    M2M_WIFI_REQ_SLEEP, M2M_WIFI_REQ_WPS_SCAN,
    M2M_WIFI_REQ_WPS,
    M2M_WIFI_REQ_START_WPS,
    M2M_WIFI_REQ_DISABLE_WPS,
    M2M_WIFI_REQ_DHCP_CONF,
    M2M_WIFI_RESP_IP_CONFIGURED,
    M2M_WIFI_RESP_IP_CONFLICT,
    M2M_WIFI_REQ_ENABLE_MONITORING,
    M2M_WIFI_REQ_DISABLE_MONITORING,
    M2M_WIFI_RESP_WIFI_RX_PACKET,
    M2M_WIFI_REQ_SEND_WIFI_PACKET,
    M2M_WIFI_REQ_LSN_INT, M2M_WIFI_REQ_DOZE,
    M2M_WIFI_MAX_STA_ALL
}

enum tenuM2mApCmd { M2M_WIFI_REQ_ENABLE_AP =
    M2M_AP_CMD_BASE, M2M_WIFI_REQ_DISABLE_AP,
    M2M_WIFI_REQ_RESTART_AP, M2M_WIFI_MAX_AP_ALL }
<table>
<thead>
<tr>
<th>Enum</th>
<th>tenuM2mOtaCmd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M2M_OTA_REQ_NOTIF_SET_URL = M2M_OTA_CMD_BASE,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_NOTIF_CHECK_FOR_UPDATE,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_NOTIF_SCHED,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_START_FW_UPDATE,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_SWITCH_FIRMWARE,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_ROLLBACK_FW,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_RESP_NOTIF_UPDATE_INFO,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_RESP_UPDATE_STATUS,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_TEST,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_START_CRT_UPDATE,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_SWITCH_CRT_IMG,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_ROLLBACK_CRT,</td>
</tr>
<tr>
<td></td>
<td>M2M_OTA_REQ_ABORT, M2M_OTA_MAX_ALL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enum</th>
<th>tenuM2mCryptoCmd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M2M_CRYPTO_REQ_SHA256_INIT = M2M_CRYPTO_CMD_BASE,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_RESP_SHA256_INIT,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_REQ_SHA256_UPDATE,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_RESP_SHA256_UPDATE,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_REQ_SHA256_FINSIH,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_RESP_SHA256_FINSIH,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_REQ_RSA_SIGN_GEN,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_RESP_RSA_SIGN_GEN,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_REQ_RSA_SIGN_VERIFY,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_RESP_RSA_SIGN_VERIFY,</td>
</tr>
<tr>
<td></td>
<td>M2M_CRYPTO_MAX_ALL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enum</th>
<th>tenuM2mIpCmd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M2M_IP_REQ_STATIC_IP_CONF = ((uint8) 10), M2M_IP_REQ_ENABLE_DHCP,</td>
</tr>
<tr>
<td></td>
<td>M2M_IP_REQ_DISABLE_DHCP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enum</th>
<th>tenuM2mSigmaCmd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M2M_SIGMA_ENABLE = ((uint8) 3),</td>
</tr>
</tbody>
</table>
|      | M2M_SIGMA_TA_START, M2M_SIGMA_TA_STATS,
enum M2M_SIGMA_TA_RECEIVE_STOP,
    M2M_SIGMA_ICMP_ARP, M2M_SIGMA_ICMP_RX,
    M2M_SIGMA_ICMP_TX, M2M_SIGMA_UDP_TX,
    M2M_SIGMA_UDP_TX_DEFER,
    M2M_SIGMA_SECURITY_POLICY,
    M2M_SIGMA_SET_SYSTIME
}

enum tenuM2mSslCmd {
    M2M_SSL_REQ_CERT_VERIF, M2M_SSL_REQ_ECC,
    M2M_SSL_RESP_ECC, M2M_SSL_IND_CRL,
    M2M_SSL_IND_CERTS_ECC,
    M2M_SSL_REQ_SET_CS_LIST,
    M2M_SSL_RESP_SET_CS_LIST
}

enum tenuM2mConnState {
    M2M_WIFI_DISCONNECTED = 0,
    M2M_WIFI_CONNECTED,
    M2M_WIFI_UNDEF = 0xff
}

enum tenuM2mSecType {
    M2M_WIFI_SEC_INVALID = 0,
    M2M_WIFI_SEC_OPEN,
    M2M_WIFI_SEC_WPA_PSK, M2M_WIFI_SEC_WEP,
    M2M_WIFI_SEC_802_1X
}

enum tenuM2mSsidMode {
    SSID_MODE_VISIBLE = 0,
    SSID_MODE_HIDDEN
}

enum tenuM2mScanCh {
    M2M_WIFI_CH_1 = ((uint8) 1),
    M2M_WIFI_CH_2,
    M2M_WIFI_CH_3, M2M_WIFI_CH_4,
    M2M_WIFI_CH_5, M2M_WIFI_CH_6, M2M_WIFI_CH_7,
    M2M_WIFI_CH_8,
    M2M_WIFI_CH_9, M2M_WIFI_CH_10, M2M_WIFI_CH_11,
    M2M_WIFI_CH_12,
    M2M_WIFI_CH_13, M2M_WIFI_CH_14, M2M_WIFI_CH_ALL
    = ((uint8) 255)
}
enum tenuM2mScanRegion {
    REG_CH_1 = ((uint16) 1 << 0),
    REG_CH_2 = ((uint16) 1 << 1),
    REG_CH_3 = ((uint16) 1 << 2),
    REG_CH_4 = ((uint16) 1 << 3),
    REG_CH_5 = ((uint16) 1 << 4),
    REG_CH_6 = ((uint16) 1 << 5),
    REG_CH_7 = ((uint16) 1 << 6),
    REG_CH_8 = ((uint16) 1 << 7),
    REG_CH_9 = ((uint16) 1 << 8),
    REG_CH_10 = ((uint16) 1 << 9),
    REG_CH_11 = ((uint16) 1 << 10),
    REG_CH_12 = ((uint16) 1 << 11),
    REG_CH_13 = ((uint16) 1 << 12),
    REG_CH_14 = ((uint16) 1 << 13),
    REG_CH_ALL = ((uint16) 0xFFFF),
    NORTH_AMERICA = ((uint16) 0x7FF),
    EUROPE = ((uint16) 0x1FFF),
    ASIA = ((uint16) 0x3FFF)
}

enum tenuPowerSaveModes {
    M2M_NO_PS, M2M_PS_AUTOMATIC,
    M2M_PS_H_AUTOMATIC, M2M_PS_DEEP_AUTOMATIC, M2M_PS_MANUAL
}

enum tenuM2mWifiMode {
    M2M_WIFI_MODE_NORMAL = ((uint8) 1),
    M2M_WIFI_MODE_AТЕ_HIGH,
    M2M_WIFI_MODE_AТЕ_LOW,
    M2M_WIFI_MODE_ETHERNET,
    M2M_WIFI_MODE_MAX
}

enum tenuWPSTrigger {
    WPS_PIN_TRIGGER = 0,
    WPS_PBC_TRIGGER = 4
}

enum tenuOtaUpdateStatus {
    OTA_STATUS_SUCSESS = 0,
    OTA_STATUS_FAIL = 1,
    OTA_STATUS_INVAILD_ARG = 2,
    OTA_STATUS_INVAILD_RB_IMAGE = 3,
    OTA_STATUS_INVAILD_FLASH_SIZE = 4,
    OTA_STATUS_ALREADY_ENABLED = 5,
    OTA_STATUS_UPDATE_INPROGRESS = 6,
    OTA_STATUS_IMAGE_VERIF_FAILED = 7,
    OTA_STATUS_CONNECTION_ERROR = 8,
OTA_STATUS_SERVER_ERROR = 9,
OTA_STATUS_ABORTED = 10
}

tenuOtaUpdateStatusType { DL_STATUS = 1, SW_STATUS = 2, RB_STATUS = 3, AB_STATUS = 4 }

tenuSslCertExpSettings {
SSL_CERT_EXP_CHECK_DISABLE,
SSL_CERT_EXP_CHECK_ENABLE,
SSL_CERT_EXP_CHECK_EN_IF_SYS_TIME }

Detailed Description

Typedef Documentation
**tpfOtaNotifCb**

```c
void(* tpfOtaNotifCb)(tstrOtaUpdateInfo *)
```

A callback to get notification about a potential OTA update.

**Parameters**

- **[in]** `pstrOtaUpdateInfo` A structure to provide notification payload.

**See also**

`tstrOtaUpdateInfo`

**Warning**

The notification is not supported (Not implemented yet)
◆ tpfOtaUpdateCb

```c
void(* tpfOtaUpdateCb)(uint8 u8OtaUpdateStatusType, uint8 u8OtaUpdateStatus)
```

A callback to get OTA status update, the callback provide the status type and its status. The OTA callback provides the download status, the switch to the downloaded firmware status and roll-back status.

**Parameters**

- **[in]** `u8OtaUpdateStatusType` Possible values are listed in `tenuOtaUpdateStatusType`
- **[in]** `u8OtaUpdateStatus` Possible values are listed as enumerated by `tenuOtaUpdateStatus`

**See also**

`tenuOtaUpdateStatusType` `tenuOtaUpdateStatus`
Enumeration Type Documentation
### tenuOtaError

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTA_SUCCESS</td>
<td>OTA Success status</td>
</tr>
<tr>
<td>OTA_ERR_WORKING_IMAGE_LOAD_FAIL</td>
<td>Failure to load the firmware image</td>
</tr>
<tr>
<td>OTA_ERR_INVAILD_CONTROL_SEC</td>
<td>Control structure is being corrupted</td>
</tr>
<tr>
<td>M2M_ERR_OTA_SWITCH_FAIL</td>
<td>Switching to the updated image failed as may be the image is invalid</td>
</tr>
<tr>
<td>M2M_ERR_OTA_START_UPDATE_FAIL</td>
<td>OTA update fail due to multiple reasons</td>
</tr>
<tr>
<td>M2M_ERR_OTA_ROLLBACK_FAIL</td>
<td>Roll-back failed due to Roll-back image is not valid</td>
</tr>
<tr>
<td>M2M_ERR_OTA_INVAILD_FLASH_SIZE</td>
<td>The OTA Support at least 4MB flash size, if the above error will appear if the current flash is less than 4M</td>
</tr>
<tr>
<td>M2M_ERR_OTA_INVAILD_ARG</td>
<td>Ota still in progress</td>
</tr>
<tr>
<td>M2M_ERR_OTA_INPROGRESS</td>
<td>Invalid argument in any OTA Function</td>
</tr>
</tbody>
</table>
### enum tenuM2mConnChangedErrcode

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_ERR_SCAN_FAIL</td>
<td>Indicate that the WINC board has failed to perform the scan operation.</td>
</tr>
<tr>
<td>M2M_ERR_JOIN_FAIL</td>
<td>Indicate that the WINC board has failed to join the BSS.</td>
</tr>
<tr>
<td>M2M_ERR_AUTH_FAIL</td>
<td>Indicate that the WINC board has failed to authenticate with the AP.</td>
</tr>
<tr>
<td>M2M_ERR_ASSOC_FAIL</td>
<td>Indicate that the WINC board has failed to associate with the AP.</td>
</tr>
<tr>
<td>M2M_ERR_CONN_INPROGRESS</td>
<td>Indicate that the WINC board has another connection request in progress.</td>
</tr>
</tbody>
</table>
enum tenuM2mWepKeyIndex

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Index for WEP key Authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_WEP_KEY_INDEX_1</td>
<td></td>
</tr>
<tr>
<td>M2M_WIFI_WEP_KEY_INDEX_2</td>
<td></td>
</tr>
<tr>
<td>M2M_WIFI_WEP_KEY_INDEX_3</td>
<td></td>
</tr>
<tr>
<td>M2M_WIFI_WEP_KEY_INDEX_4</td>
<td></td>
</tr>
</tbody>
</table>
**tenuM2mPwrMode**

```plaintext
enum tenuM2mPwrMode
{
    PWR_AUTO,  // FW will decide the best power mode to use internally.
    PWR_LOW1,
    PWR_LOW2,
    PWR_HIGH,
}
```
**tenuM2mTxPwrLevel**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>PPA Gain 6dbm PA Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX_PWR_HIGH</td>
<td>18dbm</td>
</tr>
<tr>
<td>TX_PWR_MED</td>
<td>12dbm</td>
</tr>
<tr>
<td>TX_PWR_LOW</td>
<td>6dbm</td>
</tr>
</tbody>
</table>
## enum tenuM2mReqGroup

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_REQ_GROUP_MAIN</td>
</tr>
<tr>
<td>M2M_REQ_GROUP_WIFI</td>
</tr>
<tr>
<td>M2M_REQ_GROUP_IP</td>
</tr>
<tr>
<td>M2M_REQ_GROUP_HIF</td>
</tr>
<tr>
<td>M2M_REQ_GROUP_OTA</td>
</tr>
<tr>
<td>M2M_REQ_GROUP_SSL</td>
</tr>
<tr>
<td>M2M_REQ_GROUP_CRYPTO</td>
</tr>
<tr>
<td>M2M_REQ_GROUP_SIGMA</td>
</tr>
</tbody>
</table>
enum `tenuM2mReqpkt`{
    M2M_REQ_CONFIG_PKT,
    M2M_REQ_DATA_PKT
}
**tenuM2mConfigCmd**

This enum contains all the host commands used to configure the WINC board.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_REQ_RESTART</td>
<td>Restart the WINC MAC layer, it's doesn't restart the IP layer.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_MAC_ADDRESS</td>
<td>Set the WINC mac address (not possible for production boards).</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_CURRENT_RSSI</td>
<td>Request the current connected AP RSSI.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_CURRENT_RSSI</td>
<td>Response to M2M_WIFI_REQ_CURRENT_RSSI with the RSSI value.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_GET_CONN_INFO</td>
<td>Request connection information command.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_CONN_INFO</td>
<td>Connect with default AP response.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_DEVICE_NAME</td>
<td>Set the WINC device name property.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_START_PROVISION_MODE</td>
<td>Start the provisioning mode for the M2M Device.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_PROVISION_INFO</td>
<td>Send the provisioning information to the host.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_STOP_PROVISION_MODE</td>
<td>Stop the current running provisioning mode.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_SYS_TIME</td>
<td>Set time of day from host.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_ENABLE_SNTP_CLIENT</td>
<td>Enable the simple network time protocol to get the time from Internet. this is required for security purposes.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DISABLE_SNTP_CLIENT</td>
<td>Disable the simple network time protocol for applications that do not need it.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_MEMORY_RECOVER</td>
<td>Reserved for debugging</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_CUST_INFO_ELEMENT</td>
<td>Add Custom Element to Beacon Management Frame.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SCAN</td>
<td>Request scan command</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_SCAN_DONE</td>
<td>Scan complete notification response.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SCAN_RESULT</td>
<td>Request Scan results command</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_SCAN_RESULT</td>
<td>Request Scan results command</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_SCAN_OPTION</td>
<td>Set Scan options &quot;slot time, slot number .. etc&quot;.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_SCAN_REGION</td>
<td>Set scan region.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_POWER_PROFILE</td>
<td>The API shall set power mode to one of 3 modes</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_TX_POWER</td>
<td>API to set TX power.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_BATTERY_VOLTAGE</td>
<td>API to set Battery Voltage.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_ENABLE_LOGS</td>
<td>API to set Battery Voltage.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_GET_SYS_TIME</td>
<td>REQ GET time of day from WINC.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_GET_SYS_TIME</td>
<td>RESP time of day from host.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SEND_Ethernet_PACKET</td>
<td>Send Ethernet packet in bypass mode.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_Ethernet_RX_PACKET</td>
<td>Receive Ethernet packet in bypass mode.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_MAC_MCAST</td>
<td>Set the WINC multicast.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_GET_PRNG</td>
<td>Request PRNG.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_GET_PRNG</td>
<td>Response for PRNG</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SCAN_SSID_LIST</td>
<td>Request scan with list of hidden SSID plus the broadcast SSID.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SET_GAINS</td>
<td>Request set the PPA</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_PASSIVE_SCAN</td>
<td>Request a passive scan</td>
</tr>
</tbody>
</table>
M2M_WIFI_MAX_CONFIG_ALL
## tenuM2mStaCmd

### Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_REQ_CONNECT</td>
<td>Connect with AP command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DEFAULT_CONNECT</td>
<td>Connect with default AP command.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_DEFAULT_CONNECT</td>
<td>Request connection information response.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DISCONNECT</td>
<td>Request to disconnect from AP command.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_CON_STATE_CHANGED</td>
<td>Connection state changed response.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SLEEP</td>
<td>Set PS mode command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_WPS_SCAN</td>
<td>Request WPS scan command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_WPS</td>
<td>Request WPS start command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_START_WPS</td>
<td>This command is for internal use by the WINC and should not be used by the host driver.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DISABLE_WPS</td>
<td>Request to disable WPS command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DHCP_CONF</td>
<td>Response indicating</td>
</tr>
</tbody>
</table>

This enum contains all the WINC commands while in Station mode.
<table>
<thead>
<tr>
<th>Message Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_RESP_IP_CONFIGURED</td>
<td>This command is for internal use by the WINC and should not be used by the host driver.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_IP_CONFLICT</td>
<td>Response indicating a conflict in obtained IP address. The user should reattempt the DHCP request.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_ENABLE_MONITORING</td>
<td>Request to enable monitor mode command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DISABLE_MONITORING</td>
<td>Request to disable monitor mode command.</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_WIFI_RX_PACKET</td>
<td>Indicate that a packet was received in monitor mode.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SEND_WIFI_PACKET</td>
<td>Send packet in monitor mode.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_LSN_INT</td>
<td>Set WiFi listen interval.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DOZE</td>
<td>Used to force the WINC to sleep in manual PS mode.</td>
</tr>
<tr>
<td>M2M_WIFI_MAX_STA_ALL</td>
<td></td>
</tr>
</tbody>
</table>
enum tenuM2mApCmd

This enum contains all the WINC commands while in AP mode.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_REQ_ENABLE_AP</td>
<td>Enable AP mode command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DISABLE_AP</td>
<td>Disable AP mode command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_RESTART_AP</td>
<td></td>
</tr>
<tr>
<td>M2M_WIFI_MAX_AP_ALL</td>
<td></td>
</tr>
</tbody>
</table>
**tenuM2mP2pCmd**

This enum contains all the WINC commands while in P2P mode.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_REQ_P2P_INT_CONNECT</td>
<td>This command is for internal use by the WINC and should not be used by the host driver.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_ENABLE_P2P</td>
<td>Enable P2P mode command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_DISABLE_P2P</td>
<td>Disable P2P mode command.</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_P2P_REPOST</td>
<td>This command is for internal use by the WINC and should not be used by the host driver.</td>
</tr>
<tr>
<td>M2M_WIFI_MAX_P2P_ALL</td>
<td></td>
</tr>
</tbody>
</table>
enum tenuM2mServerCmd

This enum contains all the WINC commands while in PS mode. These command are currently not supported.

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_REQ_CLIENT_CTRL</td>
</tr>
<tr>
<td>M2M_WIFI_RESP_CLIENT_INFO</td>
</tr>
<tr>
<td>M2M_WIFI_REQ_SERVER_INIT</td>
</tr>
<tr>
<td>M2M_WIFI_MAX_SERVER_ALL</td>
</tr>
<tr>
<td>Enumerator</td>
</tr>
<tr>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>M2M_OTA_REQ_NOTIF_SET_URL</td>
</tr>
<tr>
<td>M2M_OTA_REQ_NOTIF_CHECK_FOR_UPDATE</td>
</tr>
<tr>
<td>M2M_OTA_REQ_NOTIF_SCHED</td>
</tr>
<tr>
<td>M2M_OTA_REQ_START_FW_UPDATE</td>
</tr>
<tr>
<td>M2M_OTA_REQ_SWITCH_FIRMWARE</td>
</tr>
<tr>
<td>M2M_OTA_REQ_ROLLBACK_FW</td>
</tr>
<tr>
<td>M2M_OTA_RESP_NOTIF_UPDATE_INFO</td>
</tr>
<tr>
<td>M2M_OTA_RESP_UPDATE_STATUS</td>
</tr>
<tr>
<td>M2M_OTA_REQ_TEST</td>
</tr>
<tr>
<td>M2M_OTA_REQ_START_CRT_UPDATE</td>
</tr>
<tr>
<td>M2M_OTA_REQ_SWITCH_CRT_IMG</td>
</tr>
<tr>
<td>M2M_OTA_REQ_ROLLBACK_CRT</td>
</tr>
<tr>
<td>M2M_OTA_REQ_ABORT</td>
</tr>
<tr>
<td>M2M_OTA_MAX_ALL</td>
</tr>
</tbody>
</table>
enum tenuM2mCryptoCmd

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_CRYPTO_REQ_SHA256_INIT</td>
</tr>
<tr>
<td>M2M_CRYPTO_RESP_SHA256_INIT</td>
</tr>
<tr>
<td>M2M_CRYPTO_REQ_SHA256_UPDATE</td>
</tr>
<tr>
<td>M2M_CRYPTO_RESP_SHA256_UPDATE</td>
</tr>
<tr>
<td>M2M_CRYPTO_REQ_SHA256_FINSIH</td>
</tr>
<tr>
<td>M2M_CRYPTO_RESP_SHA256_FINSIH</td>
</tr>
<tr>
<td>M2M_CRYPTO_REQ_RSA_SIGN_GEN</td>
</tr>
<tr>
<td>M2M_CRYPTO_RESP_RSA_SIGN_GEN</td>
</tr>
<tr>
<td>M2M_CRYPTO_REQ_RSA_SIGN_VERIFY</td>
</tr>
<tr>
<td>M2M_CRYPTO_RESP_RSA_SIGN_VERIFY</td>
</tr>
<tr>
<td>M2M_CRYPTO_MAX_ALL</td>
</tr>
</tbody>
</table>
```c
tenM2mIpCmd

define enum tenM2mIpCmd

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_IP_REQ_STATIC_IP_CONF</td>
</tr>
<tr>
<td>M2M_IP_REQ_ENABLE_DHCP</td>
</tr>
<tr>
<td>M2M_IP_REQ_DISABLE_DHCP</td>
</tr>
</tbody>
</table>
```
### enum tenuM2mSigmaCmd

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_SIGMA_ENABLE</td>
</tr>
<tr>
<td>M2M_SIGMA_TA_START</td>
</tr>
<tr>
<td>M2M_SIGMA_TA_STATS</td>
</tr>
<tr>
<td>M2M_SIGMA_TA_RECEIVE_STOP</td>
</tr>
<tr>
<td>M2M_SIGMA_ICMP_ARP</td>
</tr>
<tr>
<td>M2M_SIGMA_ICMP_RX</td>
</tr>
<tr>
<td>M2M_SIGMA_ICMP_TX</td>
</tr>
<tr>
<td>M2M_SIGMA_UDP_TX</td>
</tr>
<tr>
<td>M2M_SIGMA_UDP_TX_DEFER</td>
</tr>
<tr>
<td>M2M_SIGMA_SECURITY_POLICY</td>
</tr>
<tr>
<td>M2M_SIGMA_SET_SYSTIME</td>
</tr>
</tbody>
</table>
enum tenuM2mSslCmd

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_SSL_REQ_CERT_VERIF</td>
</tr>
<tr>
<td>M2M_SSL_REQ_ECC</td>
</tr>
<tr>
<td>M2M_SSL_RESP_ECC</td>
</tr>
<tr>
<td>M2M_SSL_IND_CRL</td>
</tr>
<tr>
<td>M2M_SSL_IND_CERTS_ECC</td>
</tr>
<tr>
<td>M2M_SSL_REQ_SET_CS_LIST</td>
</tr>
<tr>
<td>M2M_SSL_RESP_SET_CS_LIST</td>
</tr>
</tbody>
</table>
**tenuM2mConnState**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_DISCONNECTED</td>
<td>Wi-Fi state is disconnected.</td>
</tr>
<tr>
<td>M2M_WIFI_CONNECTED</td>
<td>Wi-Fi state is connected.</td>
</tr>
<tr>
<td>M2M_WIFI_UNDEF</td>
<td>Undefined Wi-Fi State.</td>
</tr>
</tbody>
</table>
enum tenuM2mSecType

Wi-Fi Supported Security types.

Wi-Fi Supported SSID types.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_SEC_INVALID</td>
<td>Invalid security type.</td>
</tr>
<tr>
<td>M2M_WIFI_SEC_OPEN</td>
<td>Wi-Fi network is not secured.</td>
</tr>
<tr>
<td>M2M_WIFI_SEC_WPA_PSK</td>
<td>Wi-Fi network is secured with WPA/WPA2 personal(PSK).</td>
</tr>
<tr>
<td>M2M_WIFI_SEC_WEP</td>
<td>Security type WEP (40 or 104) OPEN OR SHARED.</td>
</tr>
<tr>
<td>M2M_WIFI_SEC_802_1X</td>
<td>Wi-Fi network is secured with WPA/WPA2 Enterprise.IEEE802.1x user-name/password authentication.</td>
</tr>
</tbody>
</table>
```plaintext
enum tenuM2mSsidMode

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSID_MODE_VISIBLE</td>
<td>SSID is visible to others.</td>
</tr>
<tr>
<td>SSID_MODE_HIDDEN</td>
<td>SSID is hidden.</td>
</tr>
</tbody>
</table>
```
**tenuM2mScanCh**

enum *tenuM2mScanCh*

Wi-Fi RF Channels.

**See also**
*tstrM2MScan tstrM2MScanOption*

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_CH_1</td>
</tr>
<tr>
<td>M2M_WIFI_CH_2</td>
</tr>
<tr>
<td>M2M_WIFI_CH_3</td>
</tr>
<tr>
<td>M2M_WIFI_CH_4</td>
</tr>
<tr>
<td>M2M_WIFI_CH_5</td>
</tr>
<tr>
<td>M2M_WIFI_CH_6</td>
</tr>
<tr>
<td>M2M_WIFI_CH_7</td>
</tr>
<tr>
<td>M2M_WIFI_CH_8</td>
</tr>
<tr>
<td>M2M_WIFI_CH_9</td>
</tr>
<tr>
<td>M2M_WIFI_CH_10</td>
</tr>
<tr>
<td>M2M_WIFI_CH_11</td>
</tr>
<tr>
<td>M2M_WIFI_CH_12</td>
</tr>
<tr>
<td>M2M_WIFI_CH_13</td>
</tr>
<tr>
<td>M2M_WIFI_CH_14</td>
</tr>
<tr>
<td>M2M_WIFI_CH_ALL</td>
</tr>
</tbody>
</table>
`enum tenuM2mScanRegion`

Wi-Fi RF Channels.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>REG_CH_1</td>
<td></td>
</tr>
<tr>
<td>REG_CH_2</td>
<td></td>
</tr>
<tr>
<td>REG_CH_3</td>
<td></td>
</tr>
<tr>
<td>REG_CH_4</td>
<td></td>
</tr>
<tr>
<td>REG_CH_5</td>
<td></td>
</tr>
<tr>
<td>REG_CH_6</td>
<td></td>
</tr>
<tr>
<td>REG_CH_7</td>
<td></td>
</tr>
<tr>
<td>REG_CH_8</td>
<td></td>
</tr>
<tr>
<td>REG_CH_9</td>
<td></td>
</tr>
<tr>
<td>REG_CH_10</td>
<td></td>
</tr>
<tr>
<td>REG_CH_11</td>
<td></td>
</tr>
<tr>
<td>REG_CH_12</td>
<td></td>
</tr>
<tr>
<td>REG_CH_13</td>
<td></td>
</tr>
<tr>
<td>REG_CH_14</td>
<td></td>
</tr>
<tr>
<td>REG_CH_ALL</td>
<td></td>
</tr>
<tr>
<td>NORTH_AMERICA</td>
<td>11 channel</td>
</tr>
<tr>
<td>EUROPE</td>
<td>13 channel</td>
</tr>
<tr>
<td>ASIA</td>
<td></td>
</tr>
</tbody>
</table>
enum tenuPowerSaveModes

Power Save Modes.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_NO_PS</td>
<td>Power save is disabled.</td>
</tr>
<tr>
<td>M2M_PS_AUTOMATIC</td>
<td>Power save is done automatically by the WINC. This mode doesn't disable all of the WINC modules and use higher amount of power than the H_AUTOMATIC and the DEEP_AUTOMATIC modes.</td>
</tr>
<tr>
<td>M2M_PS_H_AUTOMATIC</td>
<td>Power save is done automatically by the WINC. Achieve higher power save than the AUTOMATIC mode by shutting down more parts of the WINC board.</td>
</tr>
<tr>
<td>M2M_PS_DEEP_AUTOMATIC</td>
<td>Power save is done automatically by the WINC. Achieve the highest possible power save.</td>
</tr>
<tr>
<td>M2M_PS_MANUAL</td>
<td>Power save is done manually by the user.</td>
</tr>
</tbody>
</table>
**tenuM2mWifiMode**

enum **tenuM2mWifiMode**

Wi-Fi Operation Mode.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M_WIFI_MODE_NORMAL</td>
<td>Normal Mode means to run customer firmware version.</td>
</tr>
<tr>
<td>M2M_WIFI_MODE_ATE_HIGH</td>
<td>Config Mode in HIGH POWER means to run production test firmware version which is known as ATE (Burst) firmware.</td>
</tr>
<tr>
<td>M2M_WIFI_MODE_ATE_LOW</td>
<td>Config Mode in LOW POWER means to run production test firmware version which is known as ATE (Burst) firmware.</td>
</tr>
<tr>
<td>M2M_WIFI_MODEETHERNET</td>
<td>etherent Mode</td>
</tr>
<tr>
<td>M2M_WIFI_MODE_MAX</td>
<td></td>
</tr>
</tbody>
</table>
enum tenuWPSTrigger

WPS Triggering Methods.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WPS_PIN_TRIGGER</td>
<td>WPS is triggered in PIN method.</td>
</tr>
<tr>
<td>WPS_PBC_TRIGGER</td>
<td>WPS is triggered via push button.</td>
</tr>
</tbody>
</table>
enum tenuOtaUpdateStatus

OTA return status.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTA_STATUS_SUCSESS</td>
<td>OTA Success with not errors.</td>
</tr>
<tr>
<td>OTA_STATUS_FAIL</td>
<td>OTA generic fail.</td>
</tr>
<tr>
<td>OTA_STATUS_INVAILD_ARG</td>
<td>Invalid or malformed download URL.</td>
</tr>
<tr>
<td>OTA_STATUS_INVAILD_RB_IMAGE</td>
<td>Invalid rollback image.</td>
</tr>
<tr>
<td>OTA_STATUS_INVAILD_FLASH_SIZE</td>
<td>Flash size on device is not enough for OTA.</td>
</tr>
<tr>
<td>OTA_STATUS_AIRREADY_ENABLED</td>
<td>An OTA operation is already enabled.</td>
</tr>
<tr>
<td>OTA_STATUS_UPDATE_INPROGRESS</td>
<td>An OTA operation update is in progress</td>
</tr>
<tr>
<td>OTA_STATUS_IMAGE_VERIF_FAILED</td>
<td>OTA Verification failed.</td>
</tr>
<tr>
<td>OTA_STATUS_CONNECTION_ERROR</td>
<td>OTA connection error.</td>
</tr>
<tr>
<td>OTA_STATUS_SERVER_ERROR</td>
<td>OTA server Error (file not found or else ...)</td>
</tr>
<tr>
<td>OTA_STATUS_ABORTED</td>
<td>OTA download has been aborted by the application.</td>
</tr>
</tbody>
</table>
enum tenuOtaUpdateStatusType

OTA update Status type.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_STATUS</td>
<td>Download OTA file status</td>
</tr>
<tr>
<td>SW_STATUS</td>
<td>Switching to the upgrade firmware status</td>
</tr>
<tr>
<td>RB_STATUS</td>
<td>Roll-back status</td>
</tr>
<tr>
<td>AB_STATUS</td>
<td>Abort status</td>
</tr>
</tbody>
</table>
**tenuSslCertExpSettings**

enum **tenuSslCertExpSettings**

SSL Certificate Expiry Validation Options.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_CERT_EXP_CHECK_DISABLE</td>
<td>ALWAYS OFF. Ignore certificate expiration date validation. If a certificate is expired or there is no configured system time, the SSL connection SUCCEEDs.</td>
</tr>
<tr>
<td>SSL_CERT_EXP_CHECK_ENABLE</td>
<td>ALWAYS ON. Validate certificate expiration date. If a certificate is expired or there is no configured system time, the SSL connection FAILs.</td>
</tr>
<tr>
<td>SSL_CERT_EXP_CHECK_EN_IF_SYS_TIME</td>
<td>CONDITIONAL VALIDATION (Default setting at startup). Validate the certificate expiration date only if there is a configured system time. If there is no configured system time, the certificate...</td>
</tr>
</tbody>
</table>
expiration is bypassed and the SSL connection SUCCEEDs.
tstrM2mPwrMode
Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8PwrMode</td>
</tr>
</tbody>
</table>

uint8  __PAD24__ [3]
<table>
<thead>
<tr>
<th><strong>u8PwrMode</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8 u8PwrMode</code></td>
</tr>
</tbody>
</table>

**power Save Mode**
__PAD24__

**uint8 __PAD24__[3]**

Padding bytes for forcing 4-byte alignment
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>tstrM2mTxPwrLevel</td>
</tr>
</tbody>
</table>

Struct Reference

WLAN » DataTypes
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8  u8TxPwrLevel</code></td>
</tr>
<tr>
<td><code>uint8  __PAD24__ [3]</code></td>
</tr>
</tbody>
</table>
Detailed Description

Tx power level.
Field Documentation
◆ u8TxPwrLevel

**uint8** u8TxPwrLevel

Tx power level
__PAD24__

`uint8 __PAD24__[3]`

Padding bytes for forcing 4-byte alignment
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>tstrM2mEnableLogs</td>
</tr>
<tr>
<td>Struct Reference</td>
</tr>
</tbody>
</table>
### Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8Enable</td>
</tr>
<tr>
<td></td>
<td><strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
Detailed Description

Enable Firmware logs.
Field Documentation
◆ u8Enable

**uint8** u8Enable

Enable/Disable firmware logs
◆ __PAD24__

```
uint8 __PAD24__[3]
```

Padding bytes for forcing 4-byte alignment
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>tstrM2mBatteryVoltage Struct Reference</td>
</tr>
</tbody>
</table>

WLAN » DataTypes
Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>u16BattVolt</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD16</strong> [2]</td>
</tr>
</tbody>
</table>
Detailed Description

Battery Voltage.
Field Documentation
**u16BattVolt**

```
uint16 u16BattVolt
```

Battery Voltage
uint8 __PAD16__[2]

Padding bytes for forcing 4-byte alignment
tstrM2mWifiGainsParams Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>u8PPAGFor11B</td>
</tr>
<tr>
<td>uint16</td>
<td>u8PPAGFor11GN</td>
</tr>
</tbody>
</table>
Detailed Description

Gain Values.
Field Documentation
◆ u8PPAGFor11B

`uint16 u8PPAGFor11B`

PPA gain for 11B (as the RF document representation) PPA_AG<0:2>
Every bit have 3dB gain control each. For example: 1 ->3db 3 ->6db 7 ->9db
◆ u8PPAGFor11GN

```c
uint16 u8PPAGFor11GN
```

PPA gain for 11GN (as the RF document represented) PPA_AGC<0:2>
Every bit have 3dB gain control each. for example: 1 ->3db 3 ->6db 7 ->9db
tstrM2mWifiWepParams Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8KeyIndx</td>
</tr>
<tr>
<td>uint8</td>
<td>u8KeySz</td>
</tr>
<tr>
<td>uint8</td>
<td>au8WepKey [WEP_104_KEY_STRING_SIZE+1]</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
Detailed Description

WEP security key parameters.
<table>
<thead>
<tr>
<th><strong>u8KeyIndx</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8 u8KeyIndx</code></td>
</tr>
</tbody>
</table>

Wep key Index.
**u8KeySz**

```c
uint8 u8KeySz
```

Wep key Size.
<table>
<thead>
<tr>
<th><code>au8WepKey</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8 au8WepKey[WEP_104_KEY_STRING_SIZE+1]</code></td>
</tr>
</tbody>
</table>

WEP Key represented as a NULL terminated ASCII string.
<table>
<thead>
<tr>
<th><strong>PAD24</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 <strong>PAD24</strong>[3]</td>
</tr>
</tbody>
</table>

Padding bytes to keep the structure word aligned.
<table>
<thead>
<tr>
<th>tstr1xAuthCredentials Struct Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN » DataTypes</td>
</tr>
</tbody>
</table>
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>au8UserName</td>
<td>[M2M_1X_USR_NAME_MAX]</td>
</tr>
<tr>
<td>uint8</td>
<td>au8Passwd</td>
<td>[M2M_1X_PWD_MAX]</td>
</tr>
</tbody>
</table>
Detailed Description

Credentials for the user to authenticate with the AAA server (WPA-Enterprise Mode IEEE802.1x).
Field Documentation
◆ au8UserName

```
uint8 au8UserName[M2M_1X_USR_NAME_MAX]
```

User Name. It must be Null terminated string.
Password corresponding to the user name. It must be Null terminated string.
<table>
<thead>
<tr>
<th>Data Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 au8PSK [M2M_MAX_PSK_LEN]</td>
<td></td>
</tr>
<tr>
<td>tstr1xAuthCredentials strCred1x</td>
<td></td>
</tr>
<tr>
<td>tstrM2mWifiWepParams strWepInfo</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi Security Parameters for all supported security modes.
**au8PSK**

```c
uint8 au8PSK[M2M_MAX_PSK_LEN]
```

Pre-Shared Key in case of WPA-Personal security.
Credentials for RADIUS server authentication in case of WPA-Enterprise security.
WEP key parameters in case of WEP security.
Data Fields

WINC1500
IoT Software
APIs 19.5.2
WINC Software API Reference Manual

**tstrM2MWifiSecInfo**
Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>tuniM2MWifiAuth</td>
<td>uniAuth</td>
</tr>
<tr>
<td></td>
<td>uint8</td>
</tr>
<tr>
<td>u8SecType</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8</td>
</tr>
<tr>
<td><strong>PAD</strong></td>
<td>[PAD]</td>
</tr>
</tbody>
</table>
Detailed Description

Authentication credentials to connect to a Wi-Fi network.
Field Documentation
uniAuth

tuniM2MWifiAuth uniAuth

Union holding all possible authentication parameters corresponding to the current security types.
**u8SecType**

`uint8 u8SecType`

Wi-Fi network security type. See `tenuM2mSecType` for supported security types.
uint8 __PAD__[__PADDING__]

Padding bytes for forcing 4-byte alignment
### tstrM2mWifiConnect

**Struct Reference**

**WLAN » DataTypes**

<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINC1500 IoT Software APIs 19.5.2</td>
</tr>
<tr>
<td>WINC Software API Reference Manual</td>
</tr>
<tr>
<td>Data Fields</td>
</tr>
<tr>
<td>------------------------------------------------</td>
</tr>
<tr>
<td><code>tstrM2MWifiSecInfo</code> strSec</td>
</tr>
<tr>
<td><code>uint16</code> u16Ch</td>
</tr>
<tr>
<td><code>uint8</code> au8SSID [M2M_MAX_SSID_LEN]</td>
</tr>
<tr>
<td><code>uint8</code> u8NoSaveCred</td>
</tr>
<tr>
<td><code>uint8 __PAD__ [__CONN_PAD_SIZE__]</code></td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi Connect Request.
Security parameters for authenticating with the AP.
◆ u16Ch

`uint16 u16Ch`

RF Channel for the target SSID.
◆ au8SSID

```c
uint8 au8SSID[M2M_MAX_SSID_LEN]
```

SSID of the desired AP. It must be NULL terminated string.
◆ u8NoSaveCred

**uint8** u8NoSaveCred
__PAD__

**uint8** __PAD__[__CONN_PAD_SIZE__]

Padding bytes for forcing 4-byte alignment
WINC1500
IoT Software
APIs 19.5.2
WINC Software API Reference Manual

tstrM2MWPSCConnect
Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8</code></td>
<td><code>u8TriggerType</code></td>
<td></td>
</tr>
<tr>
<td><code>char</code></td>
<td><code>acPinNumber</code></td>
<td>[8]</td>
</tr>
<tr>
<td><code>uint8</code></td>
<td><code>__PAD24__</code></td>
<td>[3]</td>
</tr>
</tbody>
</table>
Detailed Description

WPS Configuration parameters.

See also
tenuWPSTrigger
Field Documentation
◆ u8TriggerType

```
uint8 u8TriggerType
```

WPS triggering method (Push button or PIN)
**acPinNumber**

char acPinNumber[8]

WPS PIN No (for PIN method)
uint8 __PAD24__[3]

Padding bytes for forcing 4-byte alignment
<table>
<thead>
<tr>
<th>tstrM2MWPSInfo Struct Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN » DataTypes</td>
</tr>
</tbody>
</table>

### Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Data Fields

- **uint8 u8AuthType**
- **uint8 u8Ch**
- **uint8 au8SSID [M2M_MAX_SSID_LEN]**
- **uint8 au8PSK [M2M_MAX_PSK_LEN]**
Detailed Description

WPS Result.

This structure is passed to the application in response to a WPS request. If the WPS session is completed successfully, the structure will have Non-ZERO authentication type. If the WPS Session fails (due to error or timeout) the authentication type is set to ZERO.

See also

`tenuM2mSecType`
Field Documentation
**u8AuthType**

<table>
<thead>
<tr>
<th><code>uint8</code> u8AuthType</th>
</tr>
</thead>
</table>

Network authentication type.
<table>
<thead>
<tr>
<th>u8Ch</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 u8Ch</td>
</tr>
</tbody>
</table>

RF Channel for the AP.
**au8SSID**

`uint8 au8SSID[M2M_MAX_SSID_LEN]`

SSID obtained from WPS.
`au8PSK`

`uint8 au8PSK[M2M_MAX_PSK_LEN]`

PSK for the network obtained from WPS.
tstrM2MDefaultConnResp Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sint8</td>
<td>s8ErrorCode</td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong></td>
<td>[3]</td>
</tr>
</tbody>
</table>
Detailed Description

Response error of the m2m_default_connect.

See also

M2M_DEFAULT_CONN_SCAN_MISMATCH
M2M_DEFAULT_CONN_EMPTY_LIST
◆ s8ErrorCode

```c
sint8 s8ErrorCode
```

Default connect error code. Possible values are:

- M2M_DEFAULT_CONN_EMPTY_LIST
- M2M_DEFAULT_CONN_SCAN_MISMATCH
__PAD24__

uint8 __PAD24__[3]
tstrM2MScanOption
Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8NumOfSlot</td>
</tr>
<tr>
<td>uint8</td>
<td>u8SlotTime</td>
</tr>
<tr>
<td>uint8</td>
<td>u8ProbesPerSlot</td>
</tr>
<tr>
<td>sint8</td>
<td>s8RssiThresh</td>
</tr>
</tbody>
</table>
Detailed Description

Scan options and configurations.

See also
  tenuM2mScanCh tstrM2MScan
◆ u8NumOfSlot

**uint8 u8NumOfSlot**
• u8SlotTime

```c
uint8 u8SlotTime
```
◆ u8ProbesPerSlot

`uint8 u8ProbesPerSlot`

Number of probe requests to be sent per channel scan slot.
s8RssiThresh

sint8 s8RssiThresh
tstrM2MScanRegion
Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>u16ScanRegion</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD16</strong> [2]</td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi channel regulation region information.

See also
tenuM2mScanRegion
Field Documentation
◆ u16ScanRegion

**uint16** u16ScanRegion
| uint8 __PAD16__[2] |

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by **doxygen** 1.8.13
tstrM2MScan Struct Reference

WLAN » DataTypes
Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8ChNum</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>RSVD8</strong> [1]</td>
</tr>
<tr>
<td>uint16</td>
<td>u16PassiveScanTime</td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi Scan Request.

See also
tenuM2mScanCh tstrM2MScanOption
◆ u8ChNum

table

<table>
<thead>
<tr>
<th>uint8 u8ChNum</th>
</tr>
</thead>
</table>

The Wi-Fi RF Channel number
<table>
<thead>
<tr>
<th><strong>RSVD8</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 <strong>RSVD8</strong>[1]</td>
</tr>
</tbody>
</table>

Reserved for future use.
**u16PassiveScanTime**

<table>
<thead>
<tr>
<th>uint16 u16PassiveScanTime</th>
</tr>
</thead>
</table>

Passive Scan Timeout in ms. The field is ignored for active scan.
tstrCryptoResp Struct Reference

WLAN » DataTypes
<table>
<thead>
<tr>
<th>Data Fields</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sint8</td>
<td>s8Resp</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
crypto response
Field Documentation
- **s8Resp**

| sint8 | s8Resp |
uint8 __PAD24__[3]
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tstrM2mScanDone</strong></td>
</tr>
<tr>
<td><strong>Struct Reference</strong></td>
</tr>
<tr>
<td><strong>WLAN » DataTypes</strong></td>
</tr>
</tbody>
</table>
### Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8NumofCh</td>
</tr>
<tr>
<td>sint8</td>
<td>s8ScanState</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD16</strong> [2]</td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi Scan Result.
u8NumofCh

uint8 u8NumofCh

Number of found APs
s8ScanState

sint8 s8ScanState

Scan status
<table>
<thead>
<tr>
<th><code>__PAD16__</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8 __PAD16__[2]</code></td>
</tr>
</tbody>
</table>

Padding bytes for forcing 4-byte alignment

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
tstrM2mReqScanResult Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8Index</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
Detailed Description

Scan Result Request.

The Wi-Fi Scan results list is stored in Firmware. The application can request a certain scan result by its index.
Field Documentation
**u8Index**

<table>
<thead>
<tr>
<th><code>uint8 u8Index</code></th>
</tr>
</thead>
</table>

Index of the desired scan result
uint8 __PAD24__[3]

Padding bytes for forcing 4-byte alignment
tstrM2mWifiscanResult Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 u8index</td>
<td></td>
</tr>
<tr>
<td>sint8 s8rssi</td>
<td></td>
</tr>
<tr>
<td>uint8 u8AuthType</td>
<td></td>
</tr>
<tr>
<td>uint8 u8ch</td>
<td></td>
</tr>
<tr>
<td>uint8 au8BSSID [6]</td>
<td></td>
</tr>
<tr>
<td>uint8 au8SSID [M2M_MAX_SSID_LEN]</td>
<td></td>
</tr>
<tr>
<td>uint8 <em>PAD8</em></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi Scan Result.

Information corresponding to an AP in the Scan Result list identified by its order (index) in the list.
Field Documentation
AP index in the scan result list.


**s8rssi**

| **sint8** | **s8rssi** |

AP signal strength.
- **u8AuthType**

```c
uint8 u8AuthType
```

AP authentication type.
u8ch

uint8 u8ch

AP RF channel.
- au8BSSID

**uint8** au8BSSID[6]

BSSID of the AP.
**au8SSID**

```c
uint8 au8SSID[M2M_MAX_SSID_LEN]
```

AP ssid.
Padding bytes for forcing 4-byte alignment

uint8 _PAD8_

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
tstrM2mWifiStateChanged Struct Reference

WLAN » DataTypes
**Data Fields**

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8CurrState</td>
</tr>
<tr>
<td>uint8</td>
<td>u8ErrCode</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD16</strong>   [2]</td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi Connection State.

See also

M2M_WIFI_DISCONNECTED, M2M_WIFI_CONNECTED,
M2M_WIFI_REQ_CON_STATE_CHANGED, tenuM2mConnChange
<table>
<thead>
<tr>
<th><strong>u8CurrState</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8 u8CurrState</code></td>
</tr>
</tbody>
</table>

Current Wi-Fi connection state
<table>
<thead>
<tr>
<th>u8ErrCode</th>
</tr>
</thead>
</table>

```c
uint8 u8ErrCode
```

Error type review tenuM2mConnChangedErrcode
__PAD16__

```c
uint8 __PAD16__[2]
```

Padding bytes for forcing 4-byte alignment
<table>
<thead>
<tr>
<th>Data Fields</th>
<th>tstrM2mPsType Struct Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN » DataTypes</td>
<td></td>
</tr>
</tbody>
</table>
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8PsType</td>
</tr>
<tr>
<td>uint8</td>
<td>u8BcastEn</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD16</strong> [2]</td>
</tr>
</tbody>
</table>
Detailed Description

Power Save Configuration.

See also
tenuPowerSaveModes
<table>
<thead>
<tr>
<th></th>
<th>u8PsType</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8PsType</td>
</tr>
</tbody>
</table>

Power save operating mode
u8BcastEn

uint8 u8BcastEn
Padding bytes for forcing 4-byte alignment
tstrM2mSlpReqTime
Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>u32SleepTime</td>
</tr>
</tbody>
</table>
Detailed Description

Manual power save request sleep time.
Field Documentation
◆ u32SleepTime

**uint32** u32SleepTime

< Sleep time in ms
### tstrM2mLsnInt Struct Reference

**WLAN » DataTypes**

<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINC1500 IoT Software APIs 19.5.2</td>
</tr>
<tr>
<td>WINC Software API Reference Manual</td>
</tr>
</tbody>
</table>
## Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>u16LsnInt</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD16</strong> [2]</td>
</tr>
</tbody>
</table>
Detailed Description

Listen interval.

It is the value of the Wi-Fi STA listen interval for power saving. It is given in units of Beacon period. Periodically after the listen interval fires, the WINC is wake up and listen to the beacon and check for any buffered frames for it from the AP.
Field Documentation
uint16 u16LsnInt

Listen interval in Beacon period count.
**uint8 __PAD16__[2]**

Padding bytes for forcing 4-byte alignment
### Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8ChannelID</td>
</tr>
<tr>
<td>uint8</td>
<td>u8FrameType</td>
</tr>
<tr>
<td>uint8</td>
<td>u8FrameSubtype</td>
</tr>
<tr>
<td>uint8</td>
<td>au8SrcMacAddress [6]</td>
</tr>
<tr>
<td>uint8</td>
<td>au8DstMacAddress [6]</td>
</tr>
<tr>
<td>uint8</td>
<td>au8BSSID [6]</td>
</tr>
<tr>
<td>uint8</td>
<td>u8EnRecvHdr</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD16</strong> [2]</td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi Monitor Mode Filter.

This structure sets the filtering criteria for WLAN packets when monitoring mode is enable. The received packets matching the filtering parameters, are passed directly to the application.
Field Documentation
◆ u8ChannelID

uint8 u8ChannelID
◆ u8FrameType

```c
uint8 u8FrameType
```

It must use values from tenuWifiFrameType.
◆ u8FrameSubtype

uint8 u8FrameSubtype

It must use values from tenuSubTypes.
◆ au8SrcMacAddress

\textbf{uint8} au8SrcMacAddress[6]
<table>
<thead>
<tr>
<th>au8DstMacAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 au8DstMacAddress[6]</td>
</tr>
<tr>
<td>au8BSSID</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>uint8   au8BSSID[6]</td>
</tr>
</tbody>
</table>
\texttt{\textbf{uint8}} \texttt{u8EnRecvHdr}
__PAD16__

uint8 __PAD16__[2]

Padding bytes for forcing 4-byte alignment
tstrM2MWifiRxPacketInfo Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8FrameType</td>
</tr>
<tr>
<td>uint8</td>
<td>u8FrameSubtype</td>
</tr>
<tr>
<td>uint8</td>
<td>u8ServiceClass</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Priority</td>
</tr>
<tr>
<td>uint8</td>
<td>u8HeaderLength</td>
</tr>
<tr>
<td>uint8</td>
<td>u8CipherType</td>
</tr>
<tr>
<td>uint8</td>
<td>au8SrcMacAddress [6]</td>
</tr>
<tr>
<td>uint8</td>
<td>au8DstMacAddress [6]</td>
</tr>
<tr>
<td>uint8</td>
<td>au8BSSID [6]</td>
</tr>
<tr>
<td>uint16</td>
<td>u16DataLength</td>
</tr>
<tr>
<td>uint16</td>
<td>u16FrameLength</td>
</tr>
<tr>
<td>uint32</td>
<td>u32DataRateKbps</td>
</tr>
<tr>
<td>sint8</td>
<td>s8RSSI</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi RX Frame Header.

The M2M application has the ability to allow Wi-Fi monitoring mode for receiving all Wi-Fi Raw frames matching a well defined filtering criteria. When a target Wi-Fi packet is received, the header information are extracted and assigned in this structure.
Field Documentation
◆ u8FrameType

```c
uint8 u8FrameType
```

It must use values from tenuWifiFrameType.
◆ u8FrameSubtype

```
uint8 u8FrameSubtype
```

It must use values from tenuSubTypes.
◆ u8ServiceClass

```c
uint8 u8ServiceClass
```

Service class from Wi-Fi header.
<table>
<thead>
<tr>
<th></th>
<th>u8Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint8</strong> u8Priority</td>
<td></td>
</tr>
</tbody>
</table>

Priority from Wi-Fi header.
## u8HeaderLength

```c
uint8 u8HeaderLength
```

Frame Header length.
◆ u8CipherType

uint8 u8CipherType

Encryption type for the rx packet.
◆ au8SrcMacAddress

```c
uint8 au8SrcMacAddress[6]
```
au8DstMacAddress

uint8 au8DstMacAddress[6]
◆ au8BSSID

```
uint8 au8BSSID[6]
```
u16DataLength

**uint16 u16DataLength**

Data payload length (Header excluded).
◆ u16FrameLength

\texttt{uint16} u16FrameLength

Total frame length (Header + Data).
**u32DataRateKbps**

**uint32** u32DataRateKbps

Data Rate in Kbps.
s8RSSI

sint8 s8RSSI

RSSI.
**uint8 __PAD24__[3]**

Padding bytes for forcing 4-byte alignment
tstrM2MWifiTxPacketInfo Struct Reference

WLAN » DataTypes
<table>
<thead>
<tr>
<th>Data Fields</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16 u16PacketSize</td>
<td></td>
</tr>
<tr>
<td>uint16 u16HeaderLength</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Wi-Fi TX Packet Info.

The M2M Application has the ability to compose a RAW Wi-Fi frames (under the application responsibility). When transmitting a Wi-Fi packet, the application must supply the firmware with this structure for sending the target frame.
Field Documentation
u16PacketSize

uint16 u16PacketSize

Wlan frame length.
◆ u16HeaderLength

```c
uint16 u16HeaderLength
```

Wlan frame header length.
WINC1500 IoT Software APIs 19.5.2
WINC Software API Reference Manual

tstrM2MP2PConnect Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8ListenChannel</td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong> [3]</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Set the device to operate in the Wi-Fi Direct (P2P) mode.
◆ u8ListenChannel

uint8 u8ListenChannel

P2P Listen Channel (1, 6 or 11)
___PAD24___

**uint8 ___PAD24___[3]**

Padding bytes for forcing 4-byte alignment
<table>
<thead>
<tr>
<th>Data Fields</th>
<th>WLAN » DataTypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>tstrM2MAPConfig</td>
<td></td>
</tr>
<tr>
<td>Struct Reference</td>
<td></td>
</tr>
</tbody>
</table>

WINC1500
IoT Software APIs 19.5.2
WINC Software API Reference Manual
## Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 au8SSID</td>
<td>[M2M_MAX_SSID_LEN]</td>
</tr>
<tr>
<td>uint8 u8ListenChannel</td>
<td></td>
</tr>
<tr>
<td>uint8 u8KeyIndx</td>
<td></td>
</tr>
<tr>
<td>uint8 u8KeySz</td>
<td></td>
</tr>
<tr>
<td>uint8 au8WepKey</td>
<td>[WEP_104_KEY_STRING_SIZE+1]</td>
</tr>
<tr>
<td>uint8 u8SecType</td>
<td></td>
</tr>
<tr>
<td>uint8 u8SsidHide</td>
<td></td>
</tr>
<tr>
<td>uint8 au8DHCPServerIP</td>
<td>[4]</td>
</tr>
<tr>
<td>uint8 au8Key</td>
<td>[M2M_MAX_PSK_LEN]</td>
</tr>
<tr>
<td>uint8 <strong>PAD24</strong></td>
<td>[2]</td>
</tr>
</tbody>
</table>
Detailed Description

AP Configuration.

This structure holds the configuration parameters for the M2M AP mode. It should be set by the application when it requests to enable the M2M AP operation mode. The M2M AP mode currently supports only WEP security (with the NO Security option available of course).
Field Documentation
**au8SSID**

```
uint8 au8SSID[M2M_MAX_SSID_LEN]
```

< Configuration parameters for the WiFi AP AP SSID
<table>
<thead>
<tr>
<th><strong>u8ListenChannel</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8 u8ListenChannel</code></td>
</tr>
</tbody>
</table>

Wi-Fi RF Channel which the AP will operate on
◆ u8KeyIndx

`uint8 u8KeyIndx`

Wep key Index
u8KeySz

uint8 u8KeySz

Wep/WPA key Size
◆ au8WepKey

uint8 au8WepKey[WEP_104_KEY_STRING_SIZE+1]

Wep key
◆ u8SecType

uint8 u8SecType

Security type: Open or WEP or WPA in the current implementation
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint8 u8SsidHide</strong></td>
<td></td>
</tr>
</tbody>
</table>

SSID Status "Hidden(1)/Visible(0)"
◆ au8DHCPPserverIP

uint8 au8DHCPPserverIP[4]

Ap IP server address
<table>
<thead>
<tr>
<th>au8Key</th>
</tr>
</thead>
</table>

`uint8 au8Key[M2M_MAX_PSK_LEN]`  

WPA key
**uint8 **PAD24**[2]**

Padding bytes for forcing alignment
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 u8Channel</td>
</tr>
<tr>
<td>uint8 <strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
PS Server initialization.
Field Documentation
◆ u8Channel

```c
uint8 u8Channel
```

Server Listen channel
__PAD24__

uint8 __PAD24__[3]

Padding bytes for forcing 4-byte alignment
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>tstrM2mClientState</td>
</tr>
<tr>
<td>Struct Reference</td>
</tr>
</tbody>
</table>

WLAN » DataTypes
<table>
<thead>
<tr>
<th>Data Fields</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8State</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
Detailed Description

PS Client State.
Field Documentation
u8State

uint8 u8State

PS Client State
Padding bytes for forcing 4-byte alignment
tstrM2Mservercmd
Struct Reference

WLAN » DataTypes
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8  u8cmd</td>
</tr>
<tr>
<td>uint8  <strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
Detailed Description

PS Server CMD.
<table>
<thead>
<tr>
<th>u8cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 u8cmd</td>
</tr>
</tbody>
</table>

PS Server Cmd
.uint8 __PAD24__[3]

Padding bytes for forcing 4-byte alignment
tstrM2mSetMacAddress Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 au8Mac</td>
<td>[6]</td>
</tr>
<tr>
<td>uint8 <strong>PAD16</strong></td>
<td>[2]</td>
</tr>
</tbody>
</table>

Detailed Description

Sets the MAC address from application. The WINC load the mac address from the effuse by default to the WINC configuration memory, but that function is used to let the application overwrite the configuration memory with the mac address from the host.

Note
It's recommended to call this only once before calling connect request and after the m2m_wifi_init
<table>
<thead>
<tr>
<th>au8Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8  au8Mac[6]</td>
</tr>
</tbody>
</table>

MAC address array
__PAD16__

`uint8 __PAD16__[2]`

Padding bytes for forcing 4-byte alignment
tstrM2MDeviceNameConfig Struct Reference

WLAN » DataTypes
Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>au8DeviceName</td>
<td>[M2M_DEVICE_NAME_MAX]</td>
</tr>
</tbody>
</table>
Detailed Description

Device name.

It is assigned by the application. It is used mainly for Wi-Fi Direct device discovery and WPS device information.
Field Documentation
◆ au8DeviceName

uint8 au8DeviceName[M2M_DEVICE_NAME_MAX]

NULL terminated device name
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>tstrM2MIPConfig</em></td>
</tr>
<tr>
<td><strong>Struct Reference</strong></td>
</tr>
<tr>
<td><strong>WLAN » DataTypes</strong></td>
</tr>
</tbody>
</table>
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>u32StaticIP</td>
</tr>
<tr>
<td>uint32</td>
<td>u32Gateway</td>
</tr>
<tr>
<td>uint32</td>
<td>u32DNS</td>
</tr>
<tr>
<td>uint32</td>
<td>u32SubnetMask</td>
</tr>
<tr>
<td>uint32</td>
<td>u32DhcpLeaseTime</td>
</tr>
</tbody>
</table>
Detailed Description

Static IP configuration.

Note
All member IP addresses are expressed in Network Byte Order (eg. "192.168.10.1" will be expressed as 0x010AA8C0).
**u32StaticIP**

`uint32 u32StaticIP`

The static IP assigned to the device.
◆ u32Gateway

**uint32** u32Gateway

IP of the Default internet gateway.
◆ u32DNS

**uint32** u32DNS

IP for the DNS server.
◆ u32SubnetMask

```markdown
uint32 u32SubnetMask
```

Subnet mask for the local area network.
◆ u32DhcpLeaseTime

```c
uint32 u32DhcpLeaseTime
```

Dhcp Lease Time in sec

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
<table>
<thead>
<tr>
<th><strong>tstrM2mIpRsvdPkt</strong> Struct Reference</th>
</tr>
</thead>
</table>

*WLAN* » *DataTypes*
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>u16PktSz</td>
</tr>
<tr>
<td>uint16</td>
<td>u16PktOffset</td>
</tr>
</tbody>
</table>
Detailed Description

Received Packet Size and Data Offset.
Field Documentation
<table>
<thead>
<tr>
<th><strong>u16PktSz</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint16</code> u16PktSz</td>
</tr>
<tr>
<td>uint16 u16PktOffset</td>
</tr>
</tbody>
</table>

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
tstrM2MProvisionModeConfig Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>tstrM2MAPConfig</td>
<td>strApConfig</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>acHttpServerDomainName</td>
<td>[64]</td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td>u8EnableRedirect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong></td>
<td>[3]</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

M2M Provisioning Mode Configuration.
<table>
<thead>
<tr>
<th>tstrM2MAPConfig</th>
<th>strApConfig</th>
</tr>
</thead>
</table>

Configuration parameters for the WiFi AP.
<table>
<thead>
<tr>
<th>acHttpServerDomainName</th>
</tr>
</thead>
<tbody>
<tr>
<td>char acHttpServerDomainName[64]</td>
</tr>
</tbody>
</table>

The device domain name for HTTP provisioning.
**u8EnableRedirect**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8EnableRedirect</td>
<td>A flag to enable/disable HTTP redirect feature for the HTTP Provisioning server. If the Redirect is enabled, all HTTP traffic (<a href="http://URL">http://URL</a>) from the device associated with WINC AP will be redirected to the HTTP Provisioning Web page.</td>
</tr>
</tbody>
</table>

- 0 : Disable HTTP Redirect.
- 1 : Enable HTTP Redirect.
uint8 __PAD24__[3]
tstrM2MProvisionInfo
Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>au8SSID</td>
<td>[M2M_MAX_SSID_LEN]</td>
</tr>
<tr>
<td>uint8</td>
<td>au8Password</td>
<td>[M2M_MAX_PSK_LEN]</td>
</tr>
<tr>
<td>uint8</td>
<td>u8SecType</td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td>u8Status</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

M2M Provisioning Information obtained from the HTTP Provisioning server.
Field Documentation
**au8SSID**

```c
uint8 au8SSID[M2M_MAX_SSID_LEN]
```

Provisioned SSID.
◆ au8Password

```
uint8 au8Password[M2M_MAX_PSK_LEN]
```

Provisioned Password.
<table>
<thead>
<tr>
<th><strong>u8SecType</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint8 u8SecType</strong></td>
</tr>
</tbody>
</table>

Wifi Security type.
**u8Status**

**uint8 u8Status**

Provisioning status. It must be checked before reading the provisioning information. It may be

- M2M_SUCCESS : Provision successful.
- M2M_FAIL : Provision Failed.
tstrM2MConnInfo Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>acSSID [M2M_MAX_SSID_LEN]</td>
<td>String of SSID</td>
</tr>
<tr>
<td>uint8</td>
<td>u8SecType</td>
<td>Security type</td>
</tr>
<tr>
<td>uint8</td>
<td>au8IPAddr [4]</td>
<td>IP Address</td>
</tr>
<tr>
<td>uint8</td>
<td>au8MACAddress [6]</td>
<td>MAC Address</td>
</tr>
<tr>
<td>sint8</td>
<td>s8RSSI</td>
<td>RSSI</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD24</strong> [3]</td>
<td>Padding</td>
</tr>
</tbody>
</table>
Detailed Description

M2M Provisioning Information obtained from the HTTP Provisioning server.
Field Documentation
acSSID

char acSSID[M2M_MAX_SSID_LEN]

AP connection SSID name
- `u8SecType`

<table>
<thead>
<tr>
<th><code>uint8 u8SecType</code></th>
</tr>
</thead>
</table>

Security type
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8</code></td>
<td><code>au8IPAddr[4]</code></td>
</tr>
</tbody>
</table>

Connection IP address
◆ au8MACAddress

```plaintext
uint8 au8MACAddress[6]
```

MAC address of the peer Wi-Fi station
<table>
<thead>
<tr>
<th>sint8</th>
<th>s8RSSI</th>
</tr>
</thead>
</table>

Connection RSSI signal
__PAD24__

uint8 __PAD24__[3]

Padding bytes for forcing 4-byte alignment
## tstrOtaInitHdr Struct Reference

### WLAN » DataTypes

<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINC1500 IoT Software APIs 19.5.2</td>
</tr>
<tr>
<td>WINC Software API Reference Manual</td>
</tr>
<tr>
<td>Data Fields</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>uint32   u32OtaMagicValue</td>
</tr>
<tr>
<td>uint32   u32OtaPayloadSzie</td>
</tr>
</tbody>
</table>
Detailed Description

OTA Image Header.
u32OtaMagicValue

uint32 u32OtaMagicValue

Magic value kept in the OTA image after the sha256 Digest buffer to define the Start of OTA Header
◆ u32OtaPayloadSzie

```c
uint32 u32OtaPayloadSzie
```

The Total OTA image payload size, include the sha256 key size

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by `doxygen` 1.8.13
tstrOtaControlSec
Struct Reference

WLAN » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>u32OtaMagicValue</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaFormatVersion</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaSequenceNumber</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaLastCheckTime</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaCurrentworkingImageOffset</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaCurrentworkingImageFirmwareVer</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaRollbackImageOffset</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaRollbackImageValidStatus</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaRollbackImageFirmwareVer</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaCortusAppWorkingOffset</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaCortusAppWorkingValidSts</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaCortusAppWorkingVer</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaCortusAppRollbackOffset</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaCortusAppRollbackValidSts</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaCortusAppRollbackVer</td>
</tr>
<tr>
<td>uint32</td>
<td>u32OtaControlSecCrc</td>
</tr>
</tbody>
</table>
**Detailed Description**

Control section structure is used to define the working image and the validity of the roll-back image and its offset, also both firmware versions is kept in that structure.
Field Documentation
<table>
<thead>
<tr>
<th>u32OtaMagicValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32 u32OtaMagicValue</td>
</tr>
</tbody>
</table>

Magic value used to ensure the structure is valid or not
◆ u32OtaFormatVersion

**uint32** u32OtaFormatVersion

NA NA NA Flash version cs struct version 00 00 00 00 00 Control
structure format version, the value will be incremented in case of
structure changed or updated
◆ u32OtaSequenceNumber

```
uint32 u32OtaSequenceNumber
```

Sequence number is used while update the control structure to keep track of how many times that section updated
◆ u32OtaLastCheckTime

```c
uint32 u32OtaLastCheckTime
```

Last time OTA check for update
u32OtaCurrentworkingImagOffset

uint32 u32OtaCurrentworkingImagOffset

Current working offset in flash
◆ u32OtaCurrentworkingImagFirmwareVer

uint32 u32OtaCurrentworkingImagFirmwareVer

current working image version ex 18.0.1
◆ u32OtaRollbackImageOffset

**uint32** u32OtaRollbackImageOffset

Roll-back image offset in flash
◆ u32OtaRollbackImageValidStatus

<table>
<thead>
<tr>
<th>uint32</th>
<th>u32OtaRollbackImageValidStatus</th>
</tr>
</thead>
</table>

roll-back image valid status
◆ u32OtaRollbackImgFirmwareVer

**uint32** u32OtaRollbackImgFirmwareVer

Roll-back image version (ex 18.0.3)
u32OtaCortusAppWorkingOffset

uint32 u32OtaCortusAppWorkingOffset

cortus app working offset in flash
u32OtaCortusAppWorkingValidSts

uint32 u32OtaCortusAppWorkingValidSts

Working Cortus app valid status
<table>
<thead>
<tr>
<th>u32OtaCortusAppWorkingVer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint32</strong> u32OtaCortusAppWorkingVer</td>
</tr>
</tbody>
</table>

Working cortus app version (ex 18.0.3)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u32OtaCortusAppRollbackOffset</td>
<td><code>uint32 u32OtaCortusAppRollbackOffset</code>&lt;br&gt;cortus app rollback offset in flash</td>
</tr>
</tbody>
</table>
◆ u32OtaCortusAppRollbackValidSts

**uint32 u32OtaCortusAppRollbackValidSts**

roll-back cortus app valid status
◆ u32OtaCortusAppRollbackVer

```
uint32 u32OtaCortusAppRollbackVer
```

Roll-back cortus app version (ex 18.0.3)
◆ u32OtaControlSecCrc

**uint32 u32OtaControlSecCrc**

CRC for the control structure to ensure validity
tstrOtaUpdateStatusResp Struct Reference

WLAN » DataTypes
Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8OtaUpdateStatusType</td>
</tr>
<tr>
<td>uint8</td>
<td>u8OtaUpdateStatus</td>
</tr>
<tr>
<td>uint8</td>
<td><em>PAD16</em> [2]</td>
</tr>
</tbody>
</table>
Detailed Description

OTA Update Information.

See also
tenuWPSTrigger
u8OtaUpdateStatusType

```
uint8 u8OtaUpdateStatusType
```

Status type tenuOtaUpdateStatusType
```plaintext
◆ u8OtaUpdateStatus

<table>
<thead>
<tr>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTA_SUCCESS</td>
</tr>
<tr>
<td>OTA_ERR_WORKING_IMAGE_LOAD_FAIL</td>
</tr>
<tr>
<td>OTA_ERR_INVAILD_CONTROL_SEC</td>
</tr>
<tr>
<td>M2M_ERR_OTA_SWITCH_FAIL</td>
</tr>
<tr>
<td>M2M_ERR_OTA_START_UPDATE_FAIL</td>
</tr>
<tr>
<td>M2M_ERR_OTA_ROLLBACK_FAIL</td>
</tr>
<tr>
<td>M2M_ERR_OTA_INVAILD_FLASH_SIZE</td>
</tr>
<tr>
<td>M2M_ERR_OTA_INVAILD_ARG</td>
</tr>
</tbody>
</table>
```

**uint8 u8OtaUpdateStatus**
uint8 _PAD16_[2]
tstrOtaUpdateInfo
Struct Reference
WLAN » DataTypes
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint32  u8NcfUpgradeVersion</code></td>
</tr>
<tr>
<td><code>uint32  u8NcfCurrentVersion</code></td>
</tr>
<tr>
<td><code>uint32  u8NcdUpgradeVersion</code></td>
</tr>
<tr>
<td><code>uint8  u8NcdRequiredUpgrade</code></td>
</tr>
<tr>
<td><code>uint8  u8DownloadUrlOffset</code></td>
</tr>
<tr>
<td><code>uint8  u8DownloadUrlSize</code></td>
</tr>
<tr>
<td><code>uint8  __PAD8__</code></td>
</tr>
</tbody>
</table>
Detailed Description

OTA Update Information.

See also
tenuWPSTrigger
Field Documentation
◆ u8NcfUpgradeVersion

uint32 u8NcfUpgradeVersion

NCF OTA Upgrade Version
u8NcfCurrentVersion

uint32 u8NcfCurrentVersion

NCF OTA Current firmware version
- **u8NcdUpgradeVersion**

```c
uint32 u8NcdUpgradeVersion
```

NCD (host) upgraded version (if the u8NcdRequiredUpgrade == true)
◆ u8NcdRequiredUpgrade

uint8 u8NcdRequiredUpgrade

NCD Required upgrade to the above version
u8DownloadUrlOffset

uint8 u8DownloadUrlOffset

Download URL offset in the received packet
u8DownloadUrlSize

uint8 u8DownloadUrlSize

Download URL size in the received packet
Padding bytes for forcing 4-byte alignment

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by **doxygen** 1.8.13
### tstrSystemTime Struct Reference

**WLAN » DataTypes**

<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINC1500 IoT Software APIs 19.5.2</td>
</tr>
<tr>
<td>WINC Software API Reference Manual</td>
</tr>
</tbody>
</table>

Atmel®
<table>
<thead>
<tr>
<th>Data Fields</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>u16Year</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Month</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Day</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Hour</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Minute</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Second</td>
</tr>
<tr>
<td>uint8</td>
<td><strong>PAD8</strong></td>
</tr>
</tbody>
</table>
Detailed Description

Used for time storage.
<table>
<thead>
<tr>
<th>u16Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16  u16Year</td>
</tr>
</tbody>
</table>
◆ u8Month

uint8 u8Month
u8Day

| uint8 u8Day |
◆ u8Hour

`uint8 u8Hour`
◆ u8Minute

uint8 u8Minute
◆ u8Second

uint8 u8Second
<table>
<thead>
<tr>
<th>tstrM2MMulticastMac Struct Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN » DataTypes</td>
</tr>
</tbody>
</table>
## Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8 au8macaddress</td>
<td>[M2M_MAC_ADDRES_LEN]</td>
</tr>
<tr>
<td>uint8 u8AddRemove</td>
<td></td>
</tr>
<tr>
<td>uint8 <strong>PAD8</strong></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

M2M add/remove multi-cast mac address.
Field Documentation
uint8 au8macaddress[M2M_MAC_ADDRES_LEN]

Mac address needed to be added or removed from filter.
◆ u8AddRemove

**uint8 u8AddRemove**

set by 1 to add or 0 to remove from filter.
uint8 __PAD8__

Padding bytes for forcing 4-byte alignment
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tstrPrng Struct Reference</strong></td>
</tr>
<tr>
<td>WLAN » DataTypes</td>
</tr>
<tr>
<td>Data Fields</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>uint8 * pu8RngBuff</td>
</tr>
<tr>
<td>uint16 u16PrngSize</td>
</tr>
<tr>
<td>uint8 <strong>PAD16</strong> [2]</td>
</tr>
</tbody>
</table>
Detailed Description

M2M Request PRNG.
Field Documentation
<table>
<thead>
<tr>
<th><strong>pu8RngBuff</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8* pu8RngBuff</code></td>
</tr>
</tbody>
</table>

< return buffer address PRNG size requested
◆ u16PrngSize

```c
uint16 u16PrngSize
```

PRNG pads
**__PAD16__**

```c
uint8 __PAD16__[2]
```

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by **doxygen** 1.8.13
tstrTlsCrlEntry Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8</code></td>
<td>u8DataLen</td>
</tr>
<tr>
<td><code>uint8</code></td>
<td>au8Data [TLS_CRL_DATA_MAX_LEN]</td>
</tr>
<tr>
<td><code>uint8</code></td>
<td><strong>PAD24</strong> [3]</td>
</tr>
</tbody>
</table>
Detailed Description

Certificate data for inclusion in a revocation list (CRL)
Field Documentation
u8DataLen

uint8 u8DataLen

Length of certificate data (maximum possible is TLS_CRL_DATA_MAX_LEN)
- au8Data

```c
uint8 au8Data[TLS_CRL_DATA_MAX_LEN]
```

Certificate data
**PAD24**

| uint8 | PAD24 | [3] |

Padding bytes for forcing 4-byte alignment
### tstrTlsCrlInfo Struct

**WLAN » DataTypes**

<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINC1500 IoT Software APIs 19.5.2</td>
</tr>
</tbody>
</table>
### Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>u8CrlType</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Rsv1</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Rsv2</td>
</tr>
<tr>
<td>uint8</td>
<td>u8Rsv3</td>
</tr>
<tr>
<td>tstrTlsCrlEntry</td>
<td>astrTlsCrl [TLS_CRL_MAX_ENTRIES]</td>
</tr>
</tbody>
</table>
Certificate revocation list details.
Field Documentation
◆ u8CrlType

**uint8 u8CrlType**

Type of certificate data contained in list
<table>
<thead>
<tr>
<th><strong>u8Rsv1</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint8</strong></td>
</tr>
</tbody>
</table>

Reserved for future use
<table>
<thead>
<tr>
<th><strong>u8Rsv2</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint8 u8Rsv2</strong></td>
</tr>
</tbody>
</table>

Reserved for future use
u8Rsv3

uint8 u8Rsv3

Reserved for future use
astrTlsCrl

<table>
<thead>
<tr>
<th>tstrTlsCrlEntry</th>
<th>astrTlsCrl[TLS_CRL_MAX_ENTRIES]</th>
</tr>
</thead>
</table>

List entries

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
# tstrTlsSrvSecFileEntry

**Struct Reference**

**WLAN » DataTypes**

<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINC1500 IoT Software APIs 19.5.2</td>
</tr>
<tr>
<td>Atmel WINC Software API Reference Manual</td>
</tr>
</tbody>
</table>
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>acFileName [TLS_FILE_NAME_MAX]</td>
</tr>
<tr>
<td>uint32</td>
<td>u32FileSize</td>
</tr>
<tr>
<td>uint32</td>
<td>u32FileAddr</td>
</tr>
</tbody>
</table>
Detailed Description

This struct contains a TLS certificate.
acFileName

char acFileName[TLS_FILE_NAME_MAX]

Name of the certificate.
<table>
<thead>
<tr>
<th>u32FileSize</th>
</tr>
</thead>
</table>

```c
uint32 u32FileSize
```

Size of the certificate.
◆ u32FileAddr

uint32 u32FileAddr

Error Code.
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tstrTlsSrvSecHdr</strong></td>
</tr>
</tbody>
</table>

Struct Reference

WLAN » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>au8SecStartPattern</td>
<td>[TLS_SRV_SEC_START_PATTERN_LEN]</td>
</tr>
<tr>
<td>uint32</td>
<td>u32nEntries</td>
<td></td>
</tr>
<tr>
<td>uint32</td>
<td>u32NextWriteAddr</td>
<td></td>
</tr>
<tr>
<td>tstrTlsSrvSecFileEntry</td>
<td>astrEntries</td>
<td>[TLS_SRV_SEC_MAX_FILES]</td>
</tr>
</tbody>
</table>
Detailed Description

This struct contains a set of TLS certificates.
- **au8SecStartPattern**

  ```c
  uint8 au8SecStartPattern[TLS_SRV_SEC_START_PATTERN_LEN]
  ```

  Start pattern.
◆ u32nEntries

uint32 u32nEntries

Number of certificates stored in the struct.
u32NextWriteAddr

uint32 u32NextWriteAddr

TLS Certificates.
<table>
<thead>
<tr>
<th><strong>astrEntries</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tstrTlsSrvSecFileEntry</strong> astrEntries[TLS_SRV_SEC_MAX_FILES]</td>
</tr>
</tbody>
</table>

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by [doxygen](https://www.doxygen.org) 1.8.13
### tstrSslSetActiveCsList

#### Struct Reference

**WLAN » DataTypes**

<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
</table>
# Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>u32CsBMP</td>
</tr>
</tbody>
</table>
Field Documentation
u32CsBMP

uint32 u32CsBMP

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
Detailed Description

Power Mode.
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines</td>
</tr>
<tr>
<td>DataTypes</td>
</tr>
<tr>
<td>Function</td>
</tr>
</tbody>
</table>
Defines
BSP
#define NMI_API
#define CONST const
#define NULL ((void*)0)
#define BSP_MIN(x, y) ((x)>(y)?(y):(x))
◆ NMI_API

#define NMI_API

Attribute used to define memory section to map Functions in host memory.
CONST

#define CONST const

Used for code portability.
◆ NULL

```c
#define NULL ((void*)0)
```

Void Pointer to '0' in case of NULL is not defined.
◆ BSP_MIN

```c
#define BSP_MIN ( x, y ) ((x)>(y)?(y):(x))
```

Computes the minimum of `x` and `y`.
## DataTypes

<table>
<thead>
<tr>
<th>BSP</th>
<th></th>
</tr>
</thead>
</table>
### Typedefs

<table>
<thead>
<tr>
<th>Typedef Definition</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef void(*  <code>tpfNmBspIsr</code>) (void)</td>
<td></td>
</tr>
<tr>
<td>typedef unsigned char</td>
<td><code>uint8</code></td>
</tr>
<tr>
<td>typedef unsigned short</td>
<td><code>uint16</code></td>
</tr>
<tr>
<td>typedef unsigned long</td>
<td><code>uint32</code></td>
</tr>
<tr>
<td>typedef signed char</td>
<td><code>sint8</code></td>
</tr>
<tr>
<td>typedef signed short</td>
<td><code>sint16</code></td>
</tr>
<tr>
<td>typedef signed long</td>
<td><code>sint32</code></td>
</tr>
</tbody>
</table>
**tpfNmBspIsr**

`void(* tpfNmBspIsr)(void)`

Pointer to function.
Used as a data type of ISR function registered by `nm_bsp_register_isr`.

**Returns**

None
◆ uint8

unsigned char uint8

Range of values between 0 to 255.
**uint16**

Unsigned short **uint16**

Range of values between 0 to 65535.
**uint32**

unsigned long **uint32**

Range of values between 0 to 4294967295.
◆ **sint8**

| signed char sint8 |

Range of values between -128 to 127.
◆ **sint16**

<table>
<thead>
<tr>
<th>signed short <strong>sint16</strong></th>
</tr>
</thead>
</table>

Range of values between -32768 to 32767.
signed long **sint32**

Range of values between -2147483648 to 2147483647.
## Function

<table>
<thead>
<tr>
<th>BSP</th>
<th>Modules</th>
</tr>
</thead>
</table>

**WINC1500 IoT Software APIs 19.5.2**

WINC Software API Reference Manual
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>nm_bsp_init</td>
</tr>
<tr>
<td>nm_bsp_deinit</td>
</tr>
<tr>
<td>nm_bsp_reset</td>
</tr>
<tr>
<td>nm_bsp_sleep</td>
</tr>
<tr>
<td>nm_bsp_register_isr</td>
</tr>
<tr>
<td>nm_bsp_interrupt_ctrl</td>
</tr>
</tbody>
</table>
nm_bsp_init

BSP » Function
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sint8 nm_bsp_init (void)</code></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Initialization for BSP (Board Support Package) such as Reset and Chip Enable Pins for WINC, delays, register ISR, enable/disable IRQ for WINC, ...etc. You must use this function in the head of your application to enable WINC and Host Driver to communicate with each other.
◆ nm_bsp_init()

`sint8 nm_bsp_init ( void )`

This function is used to initialize the **Board Support Package (BSP)** in order to prepare the WINC before it starts working.

The `nm_bsp_init` function is the first function that should be called at the beginning of every application to initialize the BSP and the WINC board. Otherwise, the rest of the BSP function calls will return with failure. This function should also be called after the WINC has been switched off with a successful call to "nm_bsp_deinit" in order to reinitialize the BSP before the user can use any of the WINC API functions again. After the function initializes the WINC, hard reset must be applied to start the WINC board.

**Note**  
Implementation of this function is host dependent.

**Warning**  
inappropriate use of this function will lead to unavailability of host-chip communication.

**See also**  
`nm_bsp_deinit, nm_bsp_reset`

**Returns**  
The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
<table>
<thead>
<tr>
<th>nm_bsp_deinit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP » Function</td>
</tr>
</tbody>
</table>
Functions

sint8 nm_bsp_deinit (void)
Detailed Description

De-initialization for BSP ((Board Support Package)). This function should be called only after a successful call to nm_bsp_init.
◆ nm_bsp_deinit()

```c
sint8 nm_bsp_deinit ( void )
```

This function is used to de-initialize the BSP and turn off the WINC board.

**Precondition**
The BSP should be initialized through `nm_bsp_init` first. The `nm_bsp_deinit` is the last function that should be called after the application has finished and before the WINC is switched off. The function call turns off the WINC board by setting CHIP_EN and RESET_N signals low. Every function call of "nm_bsp_init" should be matched with a call to `nm_bsp_deinit`. Failure to do so may result in the WINC consuming higher power than expected.

**Note**
Implementation of this function is host dependent.

**Warning**
misuse may lead to unknown behavior in case of soft reset.

**See also**
`nm_bsp_init`

**Returns**
The function returns `M2M_SUCCESS` for successful operations and a negative value otherwise.
nm_bsp_reset

BSP » Function
Functions

void nm_bsp_reset (void)
Detailed Description

Resetting WINC1500 SoC by setting CHIP_EN and RESET_N signals low, then after specific delay the function will put CHIP_EN high then RESET_N high, for the timing between signals please review the WINC data-sheet
Function Documentation
◆ nm_bsp_reset()

```c
void nm_bsp_reset ( void )
```

Applies a hardware reset to the WINC board. The "nm_bsp_reset" is used to apply a hard reset to the WINC board by setting CHIP_EN and RESET_N signals low, then after specific delay the function will put CHIP_EN high then RESET_N high, for the detailed timing between signals please review the WINC data-sheet. After a successful call, the WINC board firmware will kick off to load and kick off the WINC firmware. This function should be called to reset the WINC firmware after the BSP is initialized and before the start of any communication with WINC board. Calling this function at any other time will result in losing the state and connections saved in the WINC board and starting again from the initial state. The host driver will need to be de-initialized before calling nm_bsp_reset and initialized again after it using the "m2m_wifi_(de)init".

**Parameters**

- **[in]** None

**Precondition**

- Initialize nm_bsp_init first

**Note**

Implementation of this function is host dependent and called by HIF layer.

**Warning**

- Calling this function will drop any connection and internal state saved on the WINC firmware.

**See also**

- nm_bsp_init, m2m_wifi_init, m2m_wifi_deinit

**Returns**

- None
nm_bsp_sleep
BSP » Function
### Functions

| void nm_bsp_sleep (uint32 u32TimeMsec) |  |
Detailed Description

Sleep in units of milliseconds. This function used by HIF Layer according to different situations.
Function Documentation
**nm_bsp_sleep()**

```c
void nm_bsp_sleep ( uint32 u32TimeMsec )
```

Used to put the host to sleep for the specified duration. Forcing the host to sleep for extended period may lead to host not being able to respond to WINC board events. It's important to be considerate while choosing the sleep period.

**Parameters**

- `[in] u32TimeMsec` Time unit in milliseconds

**Precondition**

Initialize `nm_bsp_init` first

**Warning**

Maximum value must nor exceed 4294967295 milliseconds which is equal to 4294967.295 seconds.

**Note**

Implementation of this function is host dependent.

**See also**

`nm_bsp_init`

**Returns**

None
nm_bsp_register_isr

BSP » Function
Functions

```c
void nm_bsp_register_isr (tpfNmBsplsr pIsr)
```
Detailed Description

Register ISR (Interrupt Service Routine) in the initialization of HIF (Host Interface) Layer. When the interrupt trigger the BSP layer should call the pfisr function once inside the interrupt.
Function Documentation
**nm_bsp_register_isr()**

```c
void nm_bsp_register_isr ( tpfNmBspIsr pfIsr )
```

Register the host interface interrupt service routine. WINC board utilize SPI interface to communicate with the host. This function register the SPI interrupt the notify the host whenever there is an outstanding message from the WINC board. The function should be called during the initialization of the host interface. It an internal driver function and shouldn’t be called by the application.

**Parameters**

- `[in]` `tpfNmBspIsr pfIsr` Pointer to ISR handler in HIF

**Warning**

Make sure that ISR for IRQ pin for WINC is disabled by default in your implementation.

**Note**

Implementation of this function is host dependent and called by HIF layer.

**See also**

`tpfNmBspIsr`

**Returns**

None
nm_bsp_interrupt_ctrl

BSP » Function
### Functions

```c
void nm_bsp_interrupt_ctrl (uint8 u8Enable)
```
Detailed Description

Synchronous enable/disable interrupts function
Function Documentation
**nm_bsp_interrupt_ctrl()**

```c
void nm_bsp_interrupt_ctrl ( uint8 u8Enable )
```

Enable/Disable interrupts This function can be used to enable/disable the WINC to host interrupt as the depending on how the driver is implemented. It an internal driver function and shouldn't be called by the application.

**Precondition**

The interrupt must be registered using `nm_bsp_register_isr` first.

**Parameters**

- **[in]** `u8Enable` '0' disable interrupts. '1' enable interrupts

**See also**

`tpfNmBspIsr, nm_bsp_register_isr`

**Note**

Implementation of this function is host dependent and called by HIF layer.

**Returns**

None
## Modules

<table>
<thead>
<tr>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Codes</td>
</tr>
<tr>
<td>DataTypes</td>
</tr>
<tr>
<td>Function</td>
</tr>
</tbody>
</table>
Detailed Description

BSD compatible socket interface between the host layer and the network protocol stacks in the firmware. These functions are used by the host application to send or receive packets and to do other socket operations.
<table>
<thead>
<tr>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
</tr>
<tr>
<td>Modules</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>TCP/IP Defines</td>
</tr>
<tr>
<td>TLS Defines</td>
</tr>
</tbody>
</table>
TCP/IP Defines

Socket » Defines
### Macros

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>HOSTNAME_MAX_SIZE</code></td>
<td>64</td>
</tr>
<tr>
<td><code>SOCKET_BUFFER_MAX_LENGTH</code></td>
<td>1400</td>
</tr>
<tr>
<td><code>AF_INET</code></td>
<td>2</td>
</tr>
<tr>
<td><code>SOCK_STREAM</code></td>
<td>1</td>
</tr>
<tr>
<td><code>SOCK_DGRAM</code></td>
<td>2</td>
</tr>
<tr>
<td><code>SOCKET_FLAGS_SSL</code></td>
<td>0x01</td>
</tr>
<tr>
<td><code>TCP_SOCK_MAX</code></td>
<td>(7)</td>
</tr>
<tr>
<td><code>UDP_SOCK_MAX</code></td>
<td>4</td>
</tr>
<tr>
<td><code>MAX_SOCKET</code></td>
<td>(TCP_SOCK_MAX + UDP_SOCK_MAX)</td>
</tr>
<tr>
<td><code>SOL_SOCKET</code></td>
<td>1</td>
</tr>
<tr>
<td><code>SOL_SSL_SOCKET</code></td>
<td>2</td>
</tr>
<tr>
<td><code>SO_SET_UDP_SEND_CALLBACK</code></td>
<td>0x00</td>
</tr>
<tr>
<td><code>IP_ADD_MEMBERSHIP</code></td>
<td>0x01</td>
</tr>
<tr>
<td><code>IP_DROP_MEMBERSHIP</code></td>
<td>0x02</td>
</tr>
</tbody>
</table>
Detailed Description

The following list of macros are used to define constants used throughout the socket layer.
Macro Definition Documentation
HOSTNAME_MAX_SIZE

#define HOSTNAME_MAX_SIZE 64

Maximum allowed size for a host domain name passed to the function `gethostbyname`. Used with the `setsockopt` function.
SOCKET_BUFFER_MAX_LENGTH

#define SOCKET_BUFFER_MAX_LENGTH 1400

Maximum allowed size for a socket data buffer. Used with `send` socket function to ensure that the buffer sent is within the allowed range.
AF_INET

#define AF_INET 2

The AF_INET is the address family used for IPv4. An IPv4 transport address is specified with the sockaddr_in structure. (It is the only supported type for the current implementation.)
◆ SOCK_STREAM

#define SOCK_STREAM 1

One of the IPv4 supported socket types for reliable connection-oriented stream connection. Passed to the socket function for the socket creation operation.
◆ SOCK_DGRAM

#define SOCK_DGRAM 2

One of the IPv4 supported socket types for unreliable connectionless datagram connection. Passed to the `socket` function for the socket creation operation.
◆ SOCKET_FLAGS_SSL

#define SOCKET_FLAGS_SSL 0x01

This flag shall be passed to the socket API for SSL session.
TCP_SOCK_MAX

#define TCP_SOCK_MAX (7)

Maximum number of simultaneous TCP sockets.
**UDP_SOCK_MAX**

```
#define UDP_SOCK_MAX 4
```

Maximum number of simultaneous UDP sockets.
MAX_SOCKET

#define MAX_SOCKET (TCP_SOCK_MAX + UDP_SOCK_MAX)

Maximum number of Sockets.
## SOL_SOCKET

```c
#define SOL_SOCKET 1
```

Socket option. Used with the `setsockopt` function
◆ **SOL_SSL_SOCKET**

```c
#define SOL_SSL_SOCKET 2
```

SSL Socket option level. Used with the `setsockopt` function.
**SO_SET_UDP_SEND_CALLBACK**

```c
#define SO_SET_UDP_SEND_CALLBACK 0x00
```

Socket option used by the application to enable/disable the use of UDP send callbacks. Used with the `setsockopt` function.
◆ IP_ADD_MEMBERSHIP

```c
#define IP_ADD_MEMBERSHIP 0x01
```

Set Socket Option Add Membership command value (to join a multicast group). Used with the `setsockopt` function.
#define IP_DROP_MEMBERSHIP 0x02

Set Socket Option Drop Membership command value (to leave a multicast group). Used with the `setsockopt` function.
TLS Defines

Socket » Defines
## Modules

<table>
<thead>
<tr>
<th>TLS Socket Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy names for TLS Cipher Suite IDs</td>
</tr>
<tr>
<td>TLS Cipher Suite IDs</td>
</tr>
</tbody>
</table>
## Macros

```
#define SO_SSL_BYPASS_X509_VERIF 0x01
#define SO_SSL_SNI 0x02
#define SO_SSL_ENABLE_SESSION_CACHING 0x03
#define SO_SSL_ENABLE_SNI_VALIDATION 0x04
```
Detailed Description

The following list of macros are used to define SSL Socket options.

See also

setsockopt
Macro Definition Documentation
**SO_SSL_BYPASS_X509_VERIF**

```c
#define SO_SSL_BYPASS_X509_VERIF 0x01
```

Allow an opened SSL socket to bypass the X509 certificate verification process. It is highly required NOT to use this socket option in production software applications. It is supported for debugging and testing purposes. The option value should be casted to int type and it is handled as a boolean flag.
Set the Server Name Indicator (SNI) for an SSL socket. The SNI is a NULL terminated string containing the server name associated with the connection. It must not exceed the size of HOSTNAME_MAX_SIZE.
**SO_SSL_ENABLE_SESSION_CACHING**

```c
#define SO_SSL_ENABLE_SESSION_CACHING 0x03
```

This option allows the TLS to cache session information for fast TLS session establishment in future connections using the TLS Protocol session resume features.


SO_SSL_ENABLE_SNI_VALIDATION

#define SO_SSL_ENABLE_SNI_VALIDATION 0x04

Enable SNI validation against the server's certificate subject common name. If there is no SNI provided (via the SO_SSL_SNI option), setting this option does nothing.
Legacy names for TLS Cipher Suite IDs

Socket » Defines » TLS Defines
### Macros

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_ENABLE_RSA_SHA_SUITES</td>
<td>0x01</td>
</tr>
<tr>
<td>SSL_ENABLE_RSA_SHA256_SUITES</td>
<td>0x02</td>
</tr>
<tr>
<td>SSL_ENABLE_DHE_SHA_SUITES</td>
<td>0x04</td>
</tr>
<tr>
<td>SSL_ENABLE_DHE_SHA256_SUITES</td>
<td>0x08</td>
</tr>
<tr>
<td>SSL_ENABLE_RSA_GCM_SUITES</td>
<td>0x10</td>
</tr>
<tr>
<td>SSL_ENABLE_DHE_GCM_SUITES</td>
<td>0x20</td>
</tr>
<tr>
<td>SSL_ENABLE_ALL_SUITES</td>
<td>0x0000003F</td>
</tr>
</tbody>
</table>
Detailed Description

The following list of macros MUST NOT be used. Instead use the new names under SSLCipherSuiteID

See also
   sslSetActiveCipherSuites
Macro Definition Documentation
# SSL_ENABLE_RSA_SHA_SUITES

```c
#define SSL_ENABLE_RSA_SHA_SUITES 0x01
```

Enable RSA Hmac_SHA based Cipher suites. For example, `TLS_RSA_WITH_AES_128_CBC_SHA`
SSL_ENABLE_RSA_SHA256_SUITES

#define SSL_ENABLE_RSA_SHA256_SUITES  0x02

Enable RSA Hmac_SHA256 based Cipher suites. For example, TLS_RSA_WITH_AES_128_CBC_SHA256
SSL_ENABLE_DHE_SHA_SUITES

#define SSL_ENABLE_DHE_SHA_SUITES 0x04

Enable DHE Hmac_SHA based Cipher suites. For example, TLS_DHE_RSA_WITH_AES_128_CBC_SHA
SSL_ENABLE_DHE_SHA256_SUITES

#define SSL_ENABLE_DHE_SHA256_SUITES 0x08

Enable DHE Hmac_SHA256 based Cipher suites. For example, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
SSL_ENABLE_RSA_GCM_SUITES

#define SSL_ENABLE_RSA_GCM_SUITES 0x10

Enable RSA AEAD based Cipher suites. For example, TLS_RSA_WITH_AES_128_GCM_SHA256
Enable DHE AEAD based Cipher suites. For example, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
```c
#define SSL_ENABLE_ALL_SUITES 0x0000003F
```

Enable all possible supported cipher suites.
# TLS Cipher Suite IDs

Socket » Defines » TLS Defines
### Macros

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA</code></td>
<td>NBIT0</td>
</tr>
<tr>
<td><code>SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA256</code></td>
<td>NBIT1</td>
</tr>
<tr>
<td><code>SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA</code></td>
<td>NBIT2</td>
</tr>
<tr>
<td><code>SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA256</code></td>
<td>NBIT3</td>
</tr>
<tr>
<td><code>SSL_CIPHER_RSA_WITH_AES_128_GCM_SHA256</code></td>
<td>NBIT4</td>
</tr>
<tr>
<td><code>SSL_CIPHER_DHE_RSA_WITH_AES_128_GCM_SHA256</code></td>
<td>NBIT5</td>
</tr>
<tr>
<td><code>SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA</code></td>
<td>NBIT6</td>
</tr>
<tr>
<td><code>SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA256</code></td>
<td>NBIT7</td>
</tr>
<tr>
<td><code>SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA</code></td>
<td>NBIT8</td>
</tr>
<tr>
<td><code>SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA256</code></td>
<td>NBIT9</td>
</tr>
<tr>
<td><code>SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA</code></td>
<td>NBIT10</td>
</tr>
<tr>
<td><code>SSL_CIPHER_ECDHE_RSA_WITH_AES_256_CBC_SHA256</code></td>
<td>NBIT11</td>
</tr>
<tr>
<td><code>SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA256</code></td>
<td>NBIT12</td>
</tr>
<tr>
<td><code>SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256</code></td>
<td>NBIT13</td>
</tr>
<tr>
<td><code>SSL_CIPHER_ECDHE_RSA_WITH_AES_128_GCM_SHA256</code></td>
<td>NBIT14</td>
</tr>
<tr>
<td><code>SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</code></td>
<td>NBIT15</td>
</tr>
<tr>
<td><code>SSL_ECC_ONLY_CIPHERS</code></td>
<td></td>
</tr>
</tbody>
</table>
#define SSL_ECC_ALL_CIPHERS

#define SSL_NON_ECC_CIPHERS_AES_128

#define SSL_ECC_CIPHERS_AES_256

#define SSL_NON_ECC_CIPHERS_AES_256

#define SSL_CIPHER_ALL
Detailed Description

The following list of macros defined the list of supported TLS Cipher suites. Each MACRO defines a single Cipher suite.

See also

m2m_ssl_set_active_ciphersuites
SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA

#define SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA NBIT0
#define SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA256 NBIT1
#define SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA NBIT2
SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA256

#define SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA256 NBIT3
SSL_CIPHER_RSA_WITH_AES_128_GCM_SHA256

#define SSL_CIPHER_RSA_WITH_AES_128_GCM_SHA256 NBIT4
#define SSL_CIPHER_DHE_RSA_WITH_AES_128_GCM_SHA256 NBIT5
SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA

#define SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA NBIT6
SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA256

#define SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA256 NBIT7
SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA

#define
SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA NBIT8
#define SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA256 NBIT9
SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA

#define SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA NBIT10
#define
SSL_CIPHER_ECDHE_RSA_WITH_AES_256_CBC_SHA NBIT11
SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA256

#define

SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA256 NBIT12
#define SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 NBIT
```c
#define SSL_CIPHER_ECDHE_RSA_WITH_AES_128_GCM_SHA256 NBIT14
```
SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

#define SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 NBIT15
#define SSL_ECC_ONLY_CIPHERS

Value:

```c
( 
  SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
  |  
  SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
 )
```

All ciphers that use ECC crypto only. This excludes ciphers that use RSA. They use ECDSA instead. These ciphers are turned off by default at startup. The application may enable them if it has an ECC math engine (like ATECC508).
#define SSL_ECC_ALL_CIPHERS

Value:

```
( 
SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA
| \
SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA256
| \
SSL_CIPHER_ECDHE_RSA_WITH_AES_128_GCM_SHA256
| \
SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
| \
SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
\ 
)
```

All supported ECC Ciphers including those ciphers that depend on RSA and ECC. These ciphers are turned off by default at startup. The application may enable them if it has an ECC math engine (like ATECC508).
SSL_NON_ECC_CIPHERS_AES_128

#define SSL_NON_ECC_CIPHERS_AES_128

Value:

( 
  SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA 
  | 
  SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA256 
  | 
  SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA 
  | 
  SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA256 
  | 
  SSL_CIPHER_RSA_WITH_AES_128_GCM_SHA256 
  | 
  SSL_CIPHER_DHE_RSA_WITH_AES_128_GCM_SHA256 
)

All supported AES-128 Ciphers (ECC ciphers are not counted). This is the default active group after startup.
SSL_ECC_CIPHERS_AES_256

#define SSL_ECC_CIPHERS_AES_256

Value:

(\nSSL_CIPHER_ECDHE_RSA_WITH_AES_256_CBC_SHA \n)

ECC AES-256 supported ciphers.
#define SSL_NON_ECC_CIPHERS_AES_256

Value:

```
( \
  SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA | \n  SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA256 | \n  SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA | \n  SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA256 \n)
```

AES-256 Ciphers. This group is disabled by default at startup because the WINC1500 HW Accelerator supports only AES-128. If the application needs to force AES-256 cipher support, it could enable them (or any of them) explicitly by calling sslSetActiveCipherSuites.
#define SSL_CIPHER_ALL

Value:

(\n  SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA
  | \n  SSL_CIPHER_RSA_WITH_AES_128_CBC_SHA256
  | \n  SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA
  | \n  SSL_CIPHER_DHE_RSA_WITH_AES_128_CBC_SHA256
  | \n  SSL_CIPHER_RSA_WITH_AES_128_GCM_SHA256
  | \n  SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA
  | \n  SSL_CIPHER_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  | \n  SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA
  | \n  SSL_CIPHER_RSA_WITH_AES_256_CBC_SHA256
  | \n  SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA
  | \n  SSL_CIPHER_DHE_RSA_WITH_AES_256_CBC_SHA256
  | \n  SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA
  | \n  SSL_CIPHER_ECDHE_RSA_WITH_AES_128_CBC_SHA256
  | \n  SSL_CIPHER_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  | \n  SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
  | \n  SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  | \n  SSL_CIPHER_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
)
SSL_CIPHER_ECDHE_RSA_WITH_AES_256_CBC_SHA

Turn On All TLS Ciphers.
## Error Codes

<table>
<thead>
<tr>
<th>Socket</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

**Macros**
### Macros

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCK_ERR_NO_ERROR</td>
<td>0</td>
</tr>
<tr>
<td>SOCK_ERR_INVALID_ADDRESS</td>
<td>-1</td>
</tr>
<tr>
<td>SOCK_ERR_ADDR_ALREADY_IN_USE</td>
<td>-2</td>
</tr>
<tr>
<td>SOCK_ERR_MAX_TCP_SOCK</td>
<td>-3</td>
</tr>
<tr>
<td>SOCK_ERR_MAX_UDP_SOCK</td>
<td>-4</td>
</tr>
<tr>
<td>SOCK_ERR_INVALID_ARG</td>
<td>-6</td>
</tr>
<tr>
<td>SOCK_ERR_MAX_LISTEN_SOCK</td>
<td>-7</td>
</tr>
<tr>
<td>SOCK_ERR_INVALID</td>
<td>-9</td>
</tr>
<tr>
<td>SOCK_ERR_ADDR_IS_REQUIRED</td>
<td>-11</td>
</tr>
<tr>
<td>SOCK_ERR_CONN_ABORTED</td>
<td>-12</td>
</tr>
<tr>
<td>SOCK_ERR_TIMEOUT</td>
<td>-13</td>
</tr>
<tr>
<td>SOCK_ERR_BUFFER_FULL</td>
<td>-14</td>
</tr>
</tbody>
</table>

```c
#define _htonl(m) ((uint32)(((uint32)(m << 24)) | ((uint32)((m & 0x0000FF00) << 8)) | ((uint32)((m & 0x00FF0000) >> 8)) | ((uint32)(m >> 24)))
#define _htons(A) (((uint16)((uint16) (A)) << 8) | (((uint16) (A)) >> 8))
#define _ntohl _htonl
#define _ntohs _htons
```
Detailed Description

The following list of macros are used to define the possible error codes returned as a result of a call to a socket function. Errors are listed in numerical order with the error macro name.
Macro Definition Documentation
◆ SOCK_ERR_NO_ERROR

#define SOCK_ERR_NO_ERROR 0

Successful socket operation
Socket address is invalid. The socket operation cannot be completed successfully without specifying a specific address For example: bind is called without specifying a port number
Socket operation cannot bind on the given address. With socket operations, only one IP address per socket is permitted. Any attempt for a new socket to bind with an IP address already bound to another open socket, will return the following error code. States that bind operation failed.
<table>
<thead>
<tr>
<th><strong>SOCK_ERR_MAX_TCP_SOCK</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define SOCK_ERR_MAX_TCP_SOCK -3</code></td>
</tr>
</tbody>
</table>

Exceeded the maximum number of TCP sockets. A maximum number of TCP sockets opened simultaneously is defined through `TCP_SOCK_MAX`. It is not permitted to exceed that number at socket creation. Identifies that **socket** operation failed.
**SOCK_ERR_MAX_UDP_SOCK**

```c
#define SOCK_ERR_MAX_UDP_SOCK  -4
```

Exceeded the maximum number of UDP sockets. A maximum number of UDP sockets opened simultaneously is defined through `UDP_SOCK_MAX`. It is not permitted to exceed that number at socket creation. Identifies that **socket** operation failed.
SOCK_ERR_INVALID_ARG

#define SOCK_ERR_INVALID_ARG -6

An invalid argument is passed to a function.
**SOCK_ERR_MAX_LISTEN_SOCK**

```c
#define SOCK_ERR_MAX_LISTEN_SOCK  -7
```

Exceeded the maximum number of TCP passive listening sockets. Identifies that `listen` operation failed.
◆ SOCK_ERR_INVALID

#define SOCK_ERR_INVALID -9

The requested socket operation is not valid in the current socket state. For example: accept is called on a TCP socket before bind or listen.
◆ SOCK_ERR_ADDR_IS_REQUIRED

#define SOCK_ERR_ADDR_IS_REQUIRED  -11

Destination address is required. Failure to provide the socket address required for the socket operation to be completed. It is generated as an error to the sendto function when the address required to send the data to is not known.
**SOCK_ERR_CONN_ABORTED**

```c
#define SOCK_ERR_CONN_ABORTED -12
```

The socket is closed by the peer. The local socket is closed also.
#define SOCK_ERR_TIMEOUT -13

The socket pending operation has Timedout.
**SOCK_ERR_BUFFER_FULL**

```c
#define SOCK_ERR_BUFFER_FULL -14
```

No buffer space available to be used for the requested socket operation.
◆ _htonl

```
#define _htonl (m) (((uint32)(m << 24)) | (((uint32)((m & 0x0000FF00) << 8)) | (((uint32)((m & 0x00FF0000) >> 8)) | ((uint32)(m >> 24))))
```

Convert a 4-byte integer from the host representation to the Network byte order representation.
◆ _htons

```c
#define _htons (uint16)(((uint16) (A)) << 8) | (((uint16) (A)) >> (A * 8))
```

Convert a 2-byte integer (short) from the host representation to the Network byte order representation.
◆ _ntohl

```c
#define _ntohl _htonl
```

Convert a 4-byte integer from the Network byte order representation to the host representation.
◆ _nths

#define _nths _htons

Convert a 2-byte integer from the Network byte order representation to the host representation.
 DataTypes

Socket
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous Events</td>
</tr>
</tbody>
</table>
### Data Structures

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct in_addr</td>
<td></td>
</tr>
<tr>
<td>struct sockaddr</td>
<td></td>
</tr>
<tr>
<td>struct sockaddr_in</td>
<td></td>
</tr>
</tbody>
</table>
## Typedefs

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef sint8</td>
<td>SOCKET</td>
</tr>
</tbody>
</table>
Detailed Description

Specific Enumeration-typedefs used for socket operations
typedef Documentation
Definition for socket handler data type. Socket ID, used with all socket operations to uniquely identify the socket handler. Such an ID is uniquely assigned at socket creation when calling socket operation.
Asynchronous Events

Socket » DataTypes
**Data Structures**

<table>
<thead>
<tr>
<th>struct</th>
<th>tstrSocketBindMsg</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>tstrSocketListenMsg</td>
</tr>
<tr>
<td>struct</td>
<td>tstrSocketAcceptMsg</td>
</tr>
<tr>
<td>struct</td>
<td>tstrSocketConnectMsg</td>
</tr>
<tr>
<td>struct</td>
<td>tstrSocketRecvMsg</td>
</tr>
</tbody>
</table>
### Typedefs

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef void(*) tpfAppSocketCb (SOCKET sock, uint8 u8Msg, void *pvMsg)</code></td>
<td>Function for socket-related callbacks.</td>
</tr>
<tr>
<td><code>typedef void(*) tpfAppResolveCb (uint8 *pu8DomainName, uint32 u32ServerIP)</code></td>
<td>Function for resolving domain names.</td>
</tr>
<tr>
<td><code>typedef void(*) tpfPingCb (uint32 u32IPAddr, uint32 u32RTT, uint8 u8ErrorCode)</code></td>
<td>Function for ping-related callbacks.</td>
</tr>
</tbody>
</table>
Enumerations

```c
enum tenuSocketCallbackMsgType {
    SOCKET_MSG_BIND = 1, SOCKET_MSG_LISTEN,
    SOCKET_MSG_DNS_RESOLVE, SOCKET_MSG_ACCEPT,
    SOCKET_MSG_CONNECT, SOCKET_MSG_RECV,
    SOCKET_MSG_SEND, SOCKET_MSG_SENDTO,
    SOCKET_MSG_RECVFROM
}
```
Detailed Description

Specific Enumeration used for asynchronous operations
Typedef Documentation
tpfAppSocketCb

The main socket application callback function. Applications register their main socket application callback through this function by calling registerSocketCallback. In response to events received, the following callback function is called to handle the corresponding asynchronous function called. Example: bind, connect,...etc.

Parameters

[in] sock Socket ID for the callback.

The socket callback function is called whenever a new event is received in response to socket operations.

Parameters

[in] u8Msg Socket event type. Possible values are:
- SOCKET_MSG_BIND
- SOCKET_MSG_LISTEN
- SOCKET_MSG_ACCEPT
- SOCKET_MSG_CONNECT
- SOCKET_MSG_RECV
- SOCKET_MSG_SEND
- SOCKET_MSG_SENDTO
- SOCKET_MSG_RECVFROM

[in] pvMsg Pointer to message structure. Existing types are:
- tstrSocketBindMsg
- tstrSocketListenMsg
- tstrSocketAcceptMsg
- tstrSocketConnectMsg
- tstrSocketRecvMsg

See also
tenuSocketCallbackMsgType tstrSocketRecvMsg
tstrSocketConnectMsg tstrSocketAcceptMsg
tstrSocketListenMsg tstrSocketBindMsg
tpfAppResolveCb

Applications requiring DNS resolution should register their callback through this function by calling `registerSocketCallback`. The following callback is triggered in response to asynchronous call to the `gethostbyname` function (DNS Resolution callback).

**Parameters**

- **[in]** `pu8DomainName` Domain name of the host.
- **[in]** `u32ServerIP` Server IPv4 address encoded in NW byte order format. If it is Zero, then the DNS resolution failed.
tpfPingCb

PING Callback.

The function delivers the ping statistics for the sent ping triggered by calling m2m_ping_req.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>u32IPAddr</td>
<td>Destination IP.</td>
</tr>
<tr>
<td>[in]</td>
<td>u8ErrorCode</td>
<td>Ping error code. It may be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PING_ERR_SUCCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PING_ERR_DEST_UNREACH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PING_ERR_TIMEOUT</td>
</tr>
</tbody>
</table>
Enumeration Type Documentation
Asynchronous APIs, make use of callback functions, in-order to return back the results once the corresponding socket operation is completed. Hence resuming the normal execution of the application code while the socket operation returns the results. Callback functions expect event messages to be passed in, in-order to identify the operation they're returning the results for. The following enum identifies the type of events that are received in the callback function.

Application Use: In order for application developers to handle the pending events from the network controller through the callback functions. A function call must be made to the function `m2m_wifi_handle_events` at least once for each socket operation.

**See also**
- `bind`
- `listen`
- `accept`
- `connect`
- `send`
- `recv`

### Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCKET_MSG_BIND</td>
<td>Bind socket event.</td>
</tr>
<tr>
<td>SOCKET_MSG_LISTEN</td>
<td>Listen socket event.</td>
</tr>
<tr>
<td>SOCKET_MSG_DNS_RESOLVE</td>
<td>DNS Resolution event.</td>
</tr>
<tr>
<td>SOCKET_MSG_ACCEPT</td>
<td>Accept socket event.</td>
</tr>
<tr>
<td>SOCKET_MSG_CONNECT</td>
<td>Connect socket event.</td>
</tr>
<tr>
<td>SOCKET_MSG_RECV</td>
<td>Receive socket event.</td>
</tr>
<tr>
<td>SOCKET_MSG_SEND</td>
<td>Send socket event.</td>
</tr>
<tr>
<td>SOCKET_MSG_SENDTO</td>
<td>Sendto socket event.</td>
</tr>
<tr>
<td>SOCKET_MSG_RECVFROM</td>
<td>Recvfrom socket event.</td>
</tr>
</tbody>
</table>
tstrSocketBindMsg
Struct Reference
Socket » DataTypes » Asynchronous Events
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>sint8</td>
</tr>
</tbody>
</table>
Detailed Description

Socket bind status.

An asynchronous call to the `bind` socket operation, returns information through this structure in response. This structure together with the event `SOCKET_MSG_BIND` are passed in parameters to the callback function.

See also
`bind`
Field Documentation
**status**

**sint8 status**

The result of the bind operation. Holding a value of ZERO for a successful bind or otherwise a negative error code corresponding to the type of error.
tstrSocketListenMsg
Struct Reference

Socket » DataTypes » Asynchronous Events
<table>
<thead>
<tr>
<th>Data Fields</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sint8</td>
<td>status</td>
</tr>
</tbody>
</table>
Detailed Description

Socket listen status.

Socket listen information is returned through this structure in response to the asynchronous call to the listen function. This structure together with the event SOCKET_MSG_LISTEN are passed-in parameters to the callback function.

See also
listen
Field Documentation
status

sint8 status

Holding a value of ZERO for a successful listen or otherwise a negative error code corresponding to the type of error.
tstrSocketAcceptMsg
Struct Reference
Socket » DataTypes » Asynchronous Events
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCKET</td>
<td>sock</td>
<td></td>
</tr>
<tr>
<td>struct</td>
<td>sockaddr_in</td>
<td>strAddr</td>
</tr>
</tbody>
</table>
Detailed Description

Socket accept status.

Socket accept information is returned through this structure in response to the asynchronous call to the `accept` function. This structure together with the event `SOCKET_MSG_ACCEPT` are passed-in parameters to the callback function.
Field Documentation
On a successful `accept` operation, the return information is the socket ID for the accepted connection with the remote peer. Otherwise a negative error code is returned to indicate failure of the accept operation.
◆ strAddr

```c
struct sockaddr_in strAddr
```

Socket address structure for the remote peer.
tstrSocketConnectMsg

Struct Reference

Socket » DataTypes » Asynchronous Events
### Data Fields

<table>
<thead>
<tr>
<th>SOCKET</th>
<th>sock</th>
</tr>
</thead>
<tbody>
<tr>
<td>sint8</td>
<td>s8Error</td>
</tr>
</tbody>
</table>
Detailed Description

Socket connect status.

Socket connect information is returned through this structure in response to the asynchronous call to the connect socket function. This structure together with the event SOCKET_MSG_CONNECT are passed-in parameters to the callback function.
Field Documentation
● sock

**SOCKET sock**

Socket ID referring to the socket passed to the connect function call.
s8Error

**sint8 s8Error**

Connect error code. Holding a value of ZERO for a successful connect or otherwise a negative error code corresponding to the type of error.
tstrSocketRecvMsg
Struct Reference

Socket » DataTypes » Asynchronous Events
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>pu8Buffer</td>
</tr>
<tr>
<td>sint16</td>
<td>s16BufferSize</td>
</tr>
<tr>
<td>uint16</td>
<td>u16RemainingSize</td>
</tr>
<tr>
<td>struct</td>
<td>sockaddr_in</td>
</tr>
</tbody>
</table>
Detailed Description

Socket recv status.

Socket receive information is returned through this structure in response to the asynchronous call to the recv or recvfrom socket functions. This structure together with the events SOCKET_MSG_RECV or SOCKET_MSG_RECVFROM are passed-in parameters to the callback function.

Remarks

In case the received data from the remote peer is larger than the USER buffer size defined during the asynchronous call to the recv function, the data is delivered to the user in a number of consecutive chunks according to the USER Buffer size. A negative or zero buffer size indicates an error with the following code:

- SOCK_ERR_NO_ERROR: Socket connection closed
- SOCK_ERR_CONN_ABORTED: Socket connection aborted
- Socket receive timed out
uint8* pu8Buffer

Pointer to the USER buffer (passed to `recv` and `recvfrom` function) containing the received data chunk.
<table>
<thead>
<tr>
<th>s16BufferSize</th>
</tr>
</thead>
</table>

The received data chunk size. Holds a negative value if there is a receive error or ZERO on success upon reception of close socket message.
<table>
<thead>
<tr>
<th><strong>u16RemainingSize</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint16 u16RemainingSize</code></td>
</tr>
</tbody>
</table>

The number of bytes remaining in the current `recv` operation.
## strRemoteAddr

```c
struct sockaddr_in strRemoteAddr
```

Socket address structure for the remote peer. It is valid for `SOCKET_MSG_RECVFROM` event.
## Data Fields

<table>
<thead>
<tr>
<th>Data Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_addr Struct</td>
<td>Reference</td>
</tr>
</tbody>
</table>

**Socket » DataTypes**
# Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32</td>
<td>s_addr</td>
</tr>
</tbody>
</table>
Detailed Description

IPv4 address representation.

This structure is used as a placeholder for IPV4 address in other structures.

See also
sockaddr_in
s_addr

uint32 s_addr

Network Byte Order representation of the IPv4 address. For example, the address "192.168.0.10" is represented as 0xA00A8C0.
sockaddr Struct Reference
Socket » DataTypes
### Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>sa_family</td>
<td></td>
</tr>
<tr>
<td>uint8</td>
<td>sa_data</td>
<td>[14]</td>
</tr>
</tbody>
</table>
Detailed Description

Generic socket address structure.

See also
sockaddr_in
**sa_family**

```c
uint16 sa_family
```

Socket address family.
◆ sa_data

**uint8 sa_data[14]**

Maximum size of all the different socket address structures.
sockaddr_in Struct Reference

Socket » DataTypes
## Data Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>sin_family</td>
</tr>
<tr>
<td>uint16</td>
<td>sin_port</td>
</tr>
<tr>
<td>in_addr</td>
<td>sin_addr</td>
</tr>
<tr>
<td>uint8</td>
<td>sin_zero  [8]</td>
</tr>
</tbody>
</table>
Detailed Description

Socket address structure for IPV4 addresses. Used to specify socket address information to which to connect to. Can be cast to sockaddr structure.
Field Documentation
sin_family

uint16 sin_family

Specifies the address family (AF). Members of AF_INET address family are IPv4 addresses. Hence, the only supported value for this is AF_INET.
**sin_port**

`uint16 sin_port`

Port number of the socket. Network sockets are identified by a pair of IP addresses and port number. It must be set in the Network Byte Order format, `_htons` (e.g. `_htons(80)`). Can NOT have zero value.
◆ **sin_addr**

<table>
<thead>
<tr>
<th>in_addr</th>
<th>sin_addr</th>
</tr>
</thead>
</table>

IP Address of the socket. The IP address is of type `in_addr` structure. Can be set to "0" to accept any IP address for server operation. non zero otherwise.
◆ sin_zero

```c
union
tuint8 sin_zero[8]
```

Padding to make structure the same size as sockaddr.
Function
Socket
<table>
<thead>
<tr>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>socketInit</td>
</tr>
<tr>
<td>registerSocketCallback</td>
</tr>
<tr>
<td>socket</td>
</tr>
<tr>
<td>bind</td>
</tr>
<tr>
<td>listen</td>
</tr>
<tr>
<td>accept</td>
</tr>
<tr>
<td>connect</td>
</tr>
<tr>
<td>recv</td>
</tr>
<tr>
<td>recvfrom</td>
</tr>
<tr>
<td>send</td>
</tr>
<tr>
<td>sendto</td>
</tr>
<tr>
<td>close</td>
</tr>
<tr>
<td>nmi_inet_addr</td>
</tr>
<tr>
<td>gethostbyname</td>
</tr>
<tr>
<td>sslEnableCertExpirationCheck</td>
</tr>
<tr>
<td>setsockopt</td>
</tr>
<tr>
<td>getsockopt</td>
</tr>
</tbody>
</table>
socketInit
Socket » Function
## Functions

<table>
<thead>
<tr>
<th>NMI_API void socketInit (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API void socketDeinit (void)</td>
</tr>
</tbody>
</table>
Detailed Description

The function performs the necessary initializations for the socket library through the following steps:

- A check made by the global variable gbSocketInit, ensuring that initialization for sockets is performed only once, in-order to prevent resetting the socket instances already created in the global socket array (gastrSockets).
- Zero initializations to the global socket array (gastrSockets), which holds the list of TCP sockets.
- Registers the socket (Host Interface)hif callback function through the call to the hif_register_cb function. This facilitates handling all of the socket related functions received through interrupts from the firmware.
Function Documentation
socketInit()

NMI_API void socketInit ( void )

Parameters
  [in] void

Returns
  void

Remarks
  This initialization function must be invoked before any socket operation is performed. No error codes from this initialization function since the socket array is statically allocated based in the maximum number of sockets MAX_SOCKET based on the systems capability.
Example

This example demonstrates the use of the socketinit for socket initialization for an mqtt chat application.

```c
// Initialize the board.
system_init();

// Initialize the UART console.
configure_console();

// Initialize the BSP.
nm_bsp_init();

----------

// Initialize socket interface.
socketInit();
registerSocketCallback(socket_event_handler, socket_resolve_handler);

// Connect to router.
m2m_wifi_connect((char *)MAIN_WLAN_SSID,
    sizeof(MAIN_WLAN_SSID),
    MAIN_WLAN_AUTH, (char *)MAIN_WLAN_PSK,
    M2M_WIFI_CH_ALL);
```
socketDeinit()

**NMI_API** void socketDeinit ( void )

Socket Layer De-initialization.

The function performs the necessary cleanup for the socket library static data. It must be invoked as the last any socket operation is performed on any active sockets.
<table>
<thead>
<tr>
<th>registerSocketCallback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket » Function</td>
</tr>
</tbody>
</table>
## Functions

| NMI_API void registerSocketCallback (tpfAppSocketCb socket_cb, tpfAppResolveCb resolve_cb) |   |
Detailed Description

Register two callback functions one for asynchronous socket events and the other one for DNS callback registering function. The registered callback functions are used to retrieve information in response to the asynchronous socket functions called.
Function Documentation
◆ registerSocketCallback()

```c
NMI_API void registerSocketCallback ( tpfAppSocketCb socket_cb,
                                 tpfAppResolveCb resolve_cb )
```

**Parameters**

[in] `tpfAppSocketCb` Assignment of callback function to the global callback `tpfAppSocketCb gpfAppSocketCb`. Delivers socket messages to the host application. In response to the asynchronous function calls, such as `bind listen accept connect`

[in] `tpfAppResolveCb` Assignment of callback function to the global callback `tpfAppResolveCb gpfAppResolveCb`. Used for DNS resolving functionalities. The DNS resolving technique is determined by the application registering the callback. NULL is assigned when, DNS resolution is not required.

**Returns**

`void`

**Remarks**

If any of the socket functionalities is not to be used, NULL is passed in as a parameter. It must be invoked after `socketinit` and before other socket layer operations.
Example

This example demonstrates the use of the registerSocketCallback to register a socket callback function with DNS resolution CB set to null for a simple UDP server example.

```c

tstrWifiInitParam param;
int8_t ret;
struct sockaddr_in addr;

// Initialize the board
system_init();

// Initialize the UART console.
configure_console();

// Initialize the BSP.
nm_bsp_init();

// Initialize socket address structure.
addr.sin_family = AF_INET;
addr.sin_port = _htons(MAIN_WIFI_M2M_SERVER_PORT);
addr.sin_addr.s_addr = _htonl(MAIN_WIFI_M2M_SERVER_IP);

// Initialize Wi-Fi parameters structure.
memset((uint8_t *)&param, 0,
       sizeof(tstrWifiInitParam));

// Initialize Wi-Fi driver with data and status callbacks.
param.pfAppWifiCb = wifi_cb;
ret = m2m_wifi_init(&param);
if (M2M_SUCCESS != ret) {
    printf("main: m2m_wifi_init call error!")
}
```
while (1) {
    
}

// Initialize socket module
socketInit();
registerSocketCallback(socket_cb, NULL);

// Connect to router.
m2m_wifi_connect((char *)MAIN_WLAN_SSID,
    sizeof(MAIN_WLAN_SSID), MAIN_WLAN_AUTH, (char
    *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);
<table>
<thead>
<tr>
<th>Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>socket</td>
<td></td>
</tr>
</tbody>
</table>

Socket » Function
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API SOCKET socket</td>
<td>Function to create a socket</td>
</tr>
<tr>
<td>(uint16 u16Domain, uint8 u8Type, uint8 u8Flags)</td>
<td>Arguments required to create a socket</td>
</tr>
</tbody>
</table>
Detailed Description

Synchronous socket allocation function based on the specified socket type. Created sockets are non-blocking and their possible types are either TCP or a UDP sockets. The maximum allowed number of TCP sockets is `TCP_SOCK_MAX` sockets while the maximum number of UDP sockets that can be created simultaneously is `UDP_SOCK_MAX` sockets.
Function Documentation
socket()

NMI_API SOCKET socket ( uint16 u16Domain,
        uint8  u8Type,
        uint8  u8Flags
    )

Parameters

    [in] u16Domain Socket family. The only allowed value is AF_INET (IPv4.0) for TCP/UDP sockets.
    [in] u8Type Socket type. Allowed values are:
          • SOCK_STREAM
          • SOCK_DGRAM
    [in] u8Flags Used to specify the socket creation flags. It shall be set to zero for normal TCP/UDP sockets. It could be SOCKET_FLAGS_SSL if the socket is used for SSL session. The use of the flag SOCKET_FLAGS_SSL has no meaning in case of UDP sockets.

Precondition

    The socketInit function must be called once at the beginning of the application to initialize the socket handler. before any call to the socket function can be made.

See also

    connect bind listen accept recv recvfrom send sendto close setsockopt getsockopt

Returns

    On successful socket creation, a non-blocking socket type is created and a socket ID is returned. In case of failure the function returns a negative value, identifying one of the socket error codes defined. For example: SOCK_ERR_INVALID for invalid argument or SOCK_ERR_MAX_TCP.SOCK if the number of TCP allocated sockets exceeds the number of available sockets.
Remarks

The socket function must be called a priori to any other related socket functions "e.g. send, recv, close ..etc"
Example

This example demonstrates the use of the socket function to allocate the socket, returning the socket handler to be used for other socket operations. Socket creation is dependent on the socket type.
UDP example

```c
SOCKET UdpServerSocket = -1;
UdpServerSocket = socket(AF_INET, SOCK_DGRAM, 0);
```
TCP example

```c
static SOCKET tcp_client_socket = -1;
tcp_client_socket = socket(AF_INET, SOCK_STREAM, 0));
```
SSL example

```c
static SOCKET ssl_socket = -1;

ssl_socket = socket(AF_INET, SOCK_STREAM, SOCK_FLAGS_SSL));
```
bind
Socket » Function
| Functions          | NMI_API sint8 bind (SOCKET sock, struct sockaddr *pstrAddr, uint8 u8AddrLen) |
Detailed Description

Asynchronous bind function associates the provided address and local port to the socket. The function can be used with both TCP and UDP sockets it's mandatory to call the bind function before starting any UDP or TCP server operation. Upon socket bind completion, the application will receive a SOCKET_MSG_BIND message in the socket callback.
**bind()**

```c
NMI_API sint8 bind ( SOCKET sock,
                    struct sockaddr * pstrAddr,
                    uint8 u8AddrLen )
```

**Parameters**

- **[in]** `sock`  
  Socket ID, must hold a non negative value. A negative value will return a socket error `SOCK_ERR_INVALID_ARG`. Indicating that an invalid argument is passed in.

- **[in]** `pstrAddr`  
  Pointer to socket address structure "sockaddr_in" `sockaddr_in`

- **[in]** `u8AddrLen`  
  Size of the given socket address structure in bytes.

**Precondition**

The socket function must be called to allocate a socket before passing the socket ID to the bind function.

**See also**

`socket connect listen accept recv recvfrom send sendto`

**Returns**

The function returns ZERO for successful operations and a negative value otherwise. The possible error values are:

- **`SOCK_ERR_NO_ERROR`** Indicating that the operation was successful.
- **`SOCK_ERR_INVALID_ARG`** Indicating passing invalid arguments such as negative socket ID or NULL socket address structure.
- **`SOCK_ERR_INVALID`** Indicate socket bind failure.
Example

This example demonstrates the call of the bind socket operation after a successful socket operation.

```c
struct sockaddr_in addr;
SOCKET udpServerSocket = -1;
int ret = -1;

if(udpServerSocket == -1)
{
    udpServerSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if(udpServerSocket >= 0)
    {
        addr.sin_family = AF_INET;
        addr.sin_port = _htons(1234);
        addr.sin_addr.s_addr = 0;
        ret = bind(udpServerSocket,(struct sockaddr*)&addr,sizeof(addr));

        if(ret != 0)
        {
            printf("Bind Failed. Error code = %d\n",ret);
            close(udpServerSocket);
        }
    }
    else
    {
        printf("UDP Server Socket Creation Failed\n");
        return;
    }
}
```
listen
Socket » Function
Functions

NMI_API sint8  listen (SOCKET sock, uint8 backlog)
Detailed Description

After successful socket binding to an IP address and port on the system, start listening on a passive socket for incoming connections. The socket must be bound on a local port or the listen operation fails. Upon the call to the asynchronous listen function, response is received through the event `SOCKET_MSG_BIND` in the socket callback. A successful listen means the TCP server operation is active. If a connection is accepted, then the application socket callback function is notified with the new connected socket through the event `SOCKET_MSG_ACCEPT`. Hence there is no need to call the `accept` function after calling `listen`.

After a connection is accepted, the user is then required to call the `recv` to receive any packets transmitted by the remote host or to receive notification of socket connection termination.
Function Documentation
listen()

```c
NMI_API sint8 listen ( SOCKET sock,
                        uint8 backlog
)
```

**Parameters**
- `[in] sock` Socket ID, must hold a non negative value. A negative value will return a socket error `SOCK_ERR_INVALID_ARG`. Indicating that an invalid argument is passed in.
- `[in] backlog` Not used by the current implementation.

**Precondition**
The bind function must be called to assign the port number and IP address to the socket before the listen operation.

**See also**
bind accept recv recvfrom send sendto

**Returns**
The function returns ZERO for successful operations and a negative value otherwise. The possible error values are:
- `SOCK_ERR_NO_ERROR` Indicating that the operation was successful.
- `SOCK_ERR_INVALID_ARG` Indicating passing invalid arguments such as negative socket ID.
- `SOCK_ERR_INVALID` Indicate socket listen failure.
Example

This example demonstrates the call of the listen socket operation after a socket operation.

```c
static void TCP_Socketcallback(SOCKET sock, uint8 u8Msg, *pvMsg)
{
    int ret = -1;

    switch(u8Msg)
    {
    case SOCKET_MSG_BIND:
    {
        tstrSocketBindMsg *pstrBind = (tstrSocketBindMsg*)pvMsg;
        if(pstrBind != NULL)
        {
            if(pstrBind->status == 0)
                {
                ret = listen(sock, 0);
            
            if(ret <0)
                printf("Listen failure! Error %d\n",ret);
            }
        }
    }
    else
    {
        M2M_ERR("bind Failure!\n");
    close(sock);
        }
    }
    break;

    case SOCKET_MSG_LISTEN:
```
{  
tstrSocketListenMsg *pstrListen =  
    (tstrSocketListenMsg*)pvMsg;  
if(pstrListen != NULL)  
    {  
      if(pstrListen->status == 0)  
      {  
        ret = accept(sock,NULL,0);  
      }  
      else  
      {  
        M2M_ERR("listen Failure!\n");  
        close(sock);  
      }  
    }  
  break;  
}
case SOCKET_MSG_ACCEPT:  
  {  
    tstrSocketAcceptMsg *pstrAccept =  
        (tstrSocketAcceptMsg*)pvMsg;  
    if(pstrAccept->sock >= 0)  
      {  
        TcpNotificationSocket = pstrAccept->sock;  
        recv(pstrAccept->sock,gau8RxBuffer,sizeof(gau8RxBuffer),TEST_RECV_TIMEOUT);  
      }  
    else  
      {  
        M2M_ERR("accept failure\n");  
      }  
    break;  
}
default:
    break;

<table>
<thead>
<tr>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket » Function</td>
</tr>
</tbody>
</table>
Functions

| NMI_API sint8 accept (SOCKET sock, struct sockaddr *addr, uint8 *addrlen) |  |
The function has no current implementation. An empty deceleration is used to prevent errors when legacy application code is used. For recent application use, the accept function can be safer as it has no effect and could be safely removed from any application using it.
Function Documentation
◆ accept()

```c
NMI_API sint8 accept ( SOCKET sock,
            struct sockaddr * addr,
            uint8 * addrlen
)
```

**Parameters**

- **[in]** `sock`  
  Socket ID, must hold a non negative value. A negative value will return a socket error `SOCK_ERR_INVALID_ARG`. Indicating that an invalid argument is passed in.

- **[in]** `addr`  
  Not used in the current implementation.

- **[in]** `addrlen`  
  Not used in the current implementation.

**Returns**

The function returns ZERO for successful operations and a negative value otherwise. The possible error values are:

- **SOCK_ERR_NO_ERROR**  
  Indicating that the operation was successful.

- **SOCK_ERR_INVALID_ARG**  
  Indicating passing invalid arguments such as negative socket ID.

---

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by [doxygen](http://www.stackoverflow.com) 1.8.13
connect

Socket » Function
<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NMI_API</strong> <strong>sint8</strong> <code>connect (SOCKET sock, struct sockaddr *pstrAddr, uint8 u8AddrLen)</code></td>
</tr>
</tbody>
</table>
Detailed Description

Establishes a TCP connection with a remote server. The asynchronous connect function must be called after receiving a valid socket ID from the socket function. The application socket callback function is notified of a successful new socket connection through the event SOCKET_MSG_CONNECT. A successful connect means the TCP session is active. The application is then required to make a call to the recv to receive any packets transmitted by the remote server, unless the application is interrupted by a notification of socket connection termination.
Function Documentation
connect() ▶

```c
NMI_API sint8 connect ( SOCKET sock,
                     struct sockaddr * pstrAddr,
                     uint8 u8AddrLen
                )
```

**Parameters**

- **[in]** `sock` Socket ID, must hold a non negative value. A negative value will return a socket error `SOCK_ERR_INVALID_ARG`. Indicating that an invalid argument is passed in.
- **[in]** `pstrAddr` Address of the remote server.
- **[in]** `pstrAddr` Pointer to socket address structure "sockaddr_in" `sockaddr_in`
- **[in]** `u8AddrLen` Size of the given socket address structure in bytes. Not currently used, implemented for BSD compatibility only.

**Precondition**

The socket function must be called to allocate a TCP socket before passing the socket ID to the bind function. If the socket is not bound, you do NOT have to call bind before the "connect" function.

**See also**

socket recv send close

**Returns**

The function returns ZERO for successful operations and a negative value otherwise. The possible error values are:

- **SOCK_ERR_NO_ERROR** Indicating that the operation was successful.
- **SOCK_ERR_INVALID_ARG** Indicating passing invalid arguments such as negative socket ID or NULL socket address structure.
- **SOCK_ERR_INVALID** Indicate socket connect failure.
Example

The example demonstrates a TCP application, showing how the asynchronous call to the connect function is made through the main function and how the callback function handles the SOCKET_MSG_CONNECT event.
UDP example

```c
struct sockaddr_in Serv_Addr;
SOCKET TcpClientSocket = -1;
int ret = -1

TcpClientSocket = socket(AF_INET, SOCK_STREAM, 0);
Serv_Addr.sin_family = AF_INET;
Serv_Addr.sin_port = htons(1234);
Serv_Addr.sin_addr.s_addr = inet_addr(SERVER);
printf("Connected to server via socket %u\n", TcpClientSocket);

do {
    ret = connect(TcpClientSocket,
                   (sockaddr_in*)&Serv_Addr, sizeof(Serv_Addr));
    if(ret != 0) {
        printf("Connection Error\n");
    } else {
        printf("Connection successful.\n");
        break;
    }
} while(1)
```
TCP example

```c
if(u8Msg == SOCKET_MSG_CONNECT)
{
    tstrSocketConnectMsg *pstrConnect =
        (tstrSocketConnectMsg*)pvMsg;
    if(pstrConnect->s8Error == 0)
    {
        uint8 acBuffer[GROWL_MSG_SIZE];
        uint16 u16MsgSize;

        printf("Connect success!\n");

        u16MsgSize = FormatMsg(u8ClientID, acBuffer);
        send(sock, acBuffer, u16MsgSize, 0);
        recv(pstrNotification->Socket,
             (void*)au8Msg,GROWLDESCRIPTION_MAX_LENGTH,
             GROWL_RX_TIMEOUT);
        u8Retry = GROWL.CONNECT_RETRY;
    }
    else
    {
        M2M_DBG("Connection Failed, Error:
               %d\n",pstrConnect->s8Error);
        close(pstrNotification->Socket);
    }
}
```
recv
Socket » Function
Functions

<table>
<thead>
<tr>
<th>NMI_API sint16 recv (SOCKET sock, void *pvRecvBuf, uint16 u16BufLen, uint32 u32Timeoutmsec)</th>
<th></th>
</tr>
</thead>
</table>
An asynchronous receive function, used to retrieve data from a TCP stream. Before calling the recv function, a successful socket connection status must have been received through any of the two socket events [SOCKET_MSG_CONNECT] or [SOCKET_MSG_ACCEPT], from the socket callback. Hence, indicating that the socket is already connected to a remote host. The application receives the required data in response to this asynchronous call through the reception of the event SOCKET_MSG_RECV in the socket callback.

Receiving the SOCKET_MSG_RECV message in the callback with zero or negative buffer length indicates the following:

- **SOCK_ERR_NO_ERROR** : Socket connection closed
- **SOCK_ERR_CONN_ABORTED** : Socket connection aborted
- **SOCK_ERR_TIMEOUT** : Socket receive timed out

The application code is expected to close the socket through the call to the close function upon the appearance of the above mentioned errors.
recv()

NMI_API sint16 recv ( SOCKET sock,
               void * pvRecvBuf,
               uint16 u16BufLen,
               uint32 u32Timeoutmsec
)

Parameters

[in] **sock**
Socket ID, must hold a non negative value. A negative value will return a socket error

**SOCK_ERR_INVALID_ARG**. Indicating that an invalid argument is passed in.

[in] **pvRecvBuf**
Pointer to a buffer that will hold the received data. The buffer is used in the recv callback to deliver the received data to the caller. The buffer must be resident in memory (heap or global buffer).

[in] **u16BufLen**
The buffer size in bytes.

[in] **u32Timeoutmsec**
Timeout for the recv function in milliseconds. If the value is set to ZERO, the timeout will be set to infinite (the recv function waits forever). If the timeout period is elapsed with no data received, the socket will get a timeout error.

Precondition

- The socket function must be called to allocate a TCP socket before passing the socket ID to the recv function.
- The socket in a connected state is expected to receive data through the socket interface.

See also

socket connect bind listen recvfrom close
Returns

The function returns ZERO for successful operations and a negative value otherwise. The possible error values are:

- **SOCK_ERR_NO_ERROR** Indicating that the operation was successful.
- **SOCK_ERR_INVALID_ARG** Indicating passing invalid arguments such as negative socket ID or NULL Receive buffer.
- **SOCK_ERR_BUFFER_FULL** Indicate socket receive failure.
Example

The example demonstrates a code snippet for the calling of the recv function in the socket callback upon notification of the accept or connect events, and the parsing of the received data when the SOCKET_MSG_RECV event is received.

```c
switch(u8Msg)
{
    case SOCKET_MSG_ACCEPT:
        {
            tstrSocketAcceptMsg *pstrAccept = (tstrSocketAcceptMsg*)pvMsg;

            if(pstrAccept->sock >= 0)
            {
                recv(pstrAccept->sock,gau8RxBuffer,sizeof(gau8RxBuffer),TEST_RECV_TIMEOUT);
            } else
            {
                M2M_ERR("accept\n");
            }
            break;

    case SOCKET_MSG_RECV:
        {
            tstrSocketRecvMsg *pstrRx = (tstrSocketRecvMsg*)pvMsg;

            if(pstrRx->s16BufferSize > 0)
            {
```
recv(sock, gau8RxBuffer, sizeof(gau8RxBuffer), TEST_RECV_TIMEOU
T);
    }
else
    {
        printf("Socet recv Error: %d\n", pstrR>
gs16BufferSize);
close(sock);
    }
break;
default:
    break;
}

recvfrom
Socket » Function
| NMI_API sint16 recvfrom (SOCKET sock, void *pvRecvBuf, uint16 u16BufLen, uint32 u32Timeoutmsec) |
Detailed Description

Receives data from a UDP Socket.

The asynchronous recvfrom function is used to retrieve data from a UDP socket. The socket must already be bound to a local port before a call to the recvfrom function is made (i.e. message SOCKET_MSG_BIND is received with successful status in the socket callback).

Upon calling the recvfrom function with a successful return code, the application is expected to receive a notification in the socket callback whenever a message is received through the SOCKET_MSG_RECVFROM event.

Receiving the SOCKET_MSG_RECVFROM message in the callback with zero, indicates that the socket is closed. Whereby a negative buffer length indicates one of the socket error codes such as socket timeout error:

The recvfrom callback can also be used to show the IP address of the remote host that sent the frame by using the "strRemoteAddr" element in the tstrSocketRecvMsg structure. (refer to the code example)
Function Documentation
recvfrom()

NMI_API sint16 recvfrom ( SOCKET sock,
       void * pvRecvBuf,
       uint16 u16BufLen,
       uint32 u32TimeoutSeconds )

Parameters

[in] sock
Socket ID, must hold a non negative value. A negative value will return a socket error
SOCK_ERR_INVALID_ARG.
Indicating that an invalid argument is passed in.

[in] pvRecvBuf
Pointer to a buffer that will hold the received data. The buffer shall be used in the recv callback to deliver the received data to the caller. The buffer must be resident in memory (heap or global buffer).

[in] u16BufLen
The buffer size in bytes.

[in] u32TimeoutSeconds
Timeout for the recv function in milliseconds. If the value is set to ZERO, the timeout will be set to infinite (the recv function waits forever).

Precondition

- The socket function must be called to allocate a UDP socket before passing the socket ID to the recvfrom function.
- The socket corresponding to the socket ID must be successfully bound to a local port through the call to a bind function.

See also

socket bind close
Returns

The function returns ZERO for successful operations and a negative value otherwise. The possible error values are:

- **SOCK_ERR_NO_ERROR** Indicating that the operation was successful.
- **SOCK_ERR_INVALID_ARG** Indicating passing invalid arguments such as negative socket ID or NULL Receive buffer.
- **SOCK_ERR_BUFFER_FULL** Indicate socket receive failure.
Example

The example demonstrates a code snippet for the calling of the recvfrom notification of a successful bind event, and the parsing of the received data when the SOCKET_MSG_RECVFROM event is received.

```
switch(u8Msg)
{
    case SOCKET_MSG_BIND:
        {
            tstrSocketBindMsg *pstrBind = (tstrSocketBindMsg*)pvMsg;

            if(pstrBind != NULL)
            {
                if(pstrBind->status == 0)
                {
                    recvfrom(sock, gau8SocketTestBuffer, TEST_BUFFER_SIZE, 0);
                }
                else
                {
                    M2M_ERR("bind\n");
                }
            }
            break;
        }

    case SOCKET_MSG_RECVFROM:
        {
            tstrSocketRecvMsg *pstrRx = (tstrSocketRecvMsg*)pvMsg;

            if(pstrRx->s16BufferSize > 0)
            {
                //get the remote host address and port number
```
uint16 u16port = pstrRx->strRemoteAddr.sin_port;
uint32 strRemoteHostAddr = pstrRx->strRemoteAddr.sin_addr;

printf("Received frame with size = %d
number = %d\n\n", pstrRx->s16BufferSize, strRemoteHostAddr, u16port);

ret = recvfrom(sock, gau8SocketTestBuffer, sizeof(gau8SocketTestBuffer), TEST_RECV_TIMEOUT);
else
{
    printf("Socket recv Error: %d\n", pstrRx->ret = close(sock);
}
break;
default:
break;
}
send
Socket » Function
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint16 send (SOCKET sock, void *pvSendBuffer, uint16 u16SendLength, uint16 u16Flags)</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Asynchronous sending function, used to send data on a TCP/UDP socket.

Called by the application code when there is outgoing data available required to be sent on a specific socket handler. The only difference between this function and the similar `sendto` function, is the type of socket the data is sent on and the parameters passed in. `send` function is most commonly called for sockets in a connected state. After the data is sent, the socket callback function registered using `registerSocketCallback()`, is expected to receive an event of type `SOCKET_MSG_SEND` holding information containing the number of data bytes sent.
Function Documentation
◆ send()

```c
NMI_API sint16 send ( SOCKET sock,
    void * pvSendBuffer,
    uint16 u16SendLength,
    uint16 u16Flags
 )
```

**Parameters**

- **sock** [in] Socket ID, must hold a non negative value. A negative value will return a socket error `SOCK_ERR_INVALID_ARG`. Indicating that an invalid argument is passed in.
- **pvSendBuffer** [in] Pointer to a buffer holding data to be transmitted.
- **u16SendLength** [in] The buffer size in bytes.
- **u16Flags** [in] Not used in the current implementation.

**Precondition**

Sockets must be initialized using socketInit.

For TCP Socket:
Must use a successfully connected Socket (so that the intended recipient address is known ahead of sending the data). Hence this function is expected to be called after a successful socket connect operation(in client case or accept in the the server case).

For UDP Socket:
UDP sockets most commonly use sendto function, where the destination address is defined. However, in-order to send outgoing data using the send function, at least one successful call must be made to the sendto function a priori the consecutive calls to the send function, to ensure that the destination address is saved in the firmware.

**See also**
socketInit recv sendto socket connect accept sendto

**Warning**

u16SendLength must not exceed **SOCKET_BUFFER_MAX_LENGTH**.
Use a valid socket identifier through the a prior call to the **socket** function. Must use a valid buffer pointer. Successful completion of a call to **send()** does not guarantee delivery of the message. A negative return value indicates only locally-detected errors.

**Returns**

The function shall return **SOCK_ERR_NO_ERROR** for successful operation and a negative value (indicating the error) otherwise.
sendto
Socket » Function
## Functions

| NMI_API | sint16 sendto (SOCKET sock, void *pvSendBuffer, uint16 u16SendLength, uint16 flags, struct sockaddr *pstrDestAddr, uint8 u8AddrLen) |
Detailed Description

Asynchronous sending function, used to send data on a UDP socket. Called by the application code when there is data required to be sent on a UDP socket handler. The application code is expected to receive data from a successful bounded socket node. The only difference between this function and the similar `send` function, is the type of socket the data is received on. This function works only with UDP sockets. After the data is sent, the socket callback function registered using `registerSocketCallback()`, is expected to receive an event of type `SOCKET_MSG_SENDTO`.
sendto()

NMI_API sint16 sendto ( SOCKET sock, void * pvSendBuffer, uint16 u16SendLength, uint16 flags, struct sockaddr * pstrDestAddr, uint8 u8AddrLen )

Parameters

[in] sock
Socket ID, must hold a non negative value. A negative value will return a socket error `SOCK_ERR_INVALID_ARG`. Indicating that an invalid argument is passed in.

[in] pvSendBuffer
Pointer to a buffer holding data to be transmitted. A NULL value will return a socket error `SOCK_ERR_INVALID_ARG`. Indicating that an invalid argument is passed in.

[in] u16SendLength
The buffer size in bytes. It must not exceed `SOCKET_BUFFER_MAX_LENGTH`.

[in] flags
Not used in the current implementation

[in] pstrDestAddr
The destination address.

[in] u8AddrLen
Destination address length in bytes. Not used in the current implementation, only included for BSD compatibility.

Precondition
Sockets must be initialized using socketInit.

See also
socketInit recvfrom sendto socket connect accept send
Warning

u16SendLength must not exceed
SOCKET_BUFFER_MAX_LENGTH.
Use a valid socket (returned from socket ). A valid buffer pointer
must be used (not NULL).
Successful completion of a call to sendto() does not guarantee
delivery of the message, A negative return value indicates only
locally-detected errors

Returns

The function returns SOCK_ERR_NO_ERROR for successful
operation and a negative value (indicating the error) otherwise.
close
Socket » Function
Functions

NMI_API sint8  close (SOCKET sock)
Detailed Description

Synchronous close function, releases all the socket assigned resources.
◆ close()

```c
NMI_API sint8 close(SOCKET sock)
```

**Parameters**

- `[in] sock` Socket ID, must hold a non negative value. A negative value will return a socket error `SOCK_ERR_INVALID_ARG`. Indicating that an invalid argument is passed in.

**Precondition**

Sockets must be initialized through the call of the `socketInit` function. `close` is called only for valid socket identifiers created through the `socket` function.

**Warning**

If `close` is called while there are still pending messages (sent or received) they will be discarded.

**See also**

- `socketInit`  
- `socket`

**Returns**

The function returned `SOCK_ERR_NO_ERROR` for successful operation and a negative value (indicating the error) otherwise.
nmi_inet_addr
Socket » Function
Functions

NMI_API uint32 nmi_inet_addr (char *pclpAddr)
Detailed Description

Synchronous function which returns a BSD socket compliant Internet Protocol (IPv4) socket address. This IPv4 address in the input string parameter could either be specified as a host name, or as a numeric string representation like n.n.n.n known as the IPv4 dotted-decimal format (i.e. "192.168.10.1"). This function is used whenever an ip address needs to be set in the proper format (i.e. for the tstrM2MIPConfig structure).
Function Documentation
nmi_inet_addr()

NMI_API uint32 nmi_inet_addr ( char * pClpAddr )

Parameters

[in] pClpAddr A null terminated string containing the IP address in IPv4 dotted-decimal address.

Returns

Unsigned 32-bit integer representing the IP address in Network byte order (eg. "192.168.10.1" will be expressed as 0x010AA8C0).
gethostbyname
Socket » Function
## Functions

| NMI_API sint8 gethostbyname (uint8 *pcHostName) |
Detailed Description

Asynchronous DNS resolving function. This function use DNS to resolve a domain name into the corresponding IP address. A call to this function will cause a DNS request to be sent and the response will be delivered to the DNS callback function registered using `registerSocketCallback()`
**gethostbyname()**

NMI_API sint8 gethostbyname ( uint8 * pcHostName )

Parameters

[in] **pcHostName** NULL terminated string containing the domain name for the remote host. Its size must not exceed HOSTNAME_MAX_SIZE.

See also

registerSocketCallback

Warning

Successful completion of a call to gethostbyname() does not guarantee success of the DNS request, a negative return value indicates only locally-detected errors.

Returns

- SOCK_ERR_NO_ERROR
- SOCK_ERR_INVALID_ARG

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
sslEnableCertExpirationCheck

Socket » Function
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI_API sint8 <code>sslEnableCertExpirationCheck</code></td>
<td>Enable SSL certificate expiration check</td>
<td><code>tenuSslCertExpSettings enuValidationSetting</code></td>
</tr>
</tbody>
</table>
Detailed Description

Configure the behavior of the SSL Library for Certificate Expiry Validation.
**sslEnableCertExpirationCheck()**

```plaintext
NMI_API sint8 sslEnableCertExpirationCheck (tenuSslCertExpSettings enuValidation)
```

**Parameters**

[in] `enuValidationSetting` See `tenuSslCertExpSettings` for details.

**Returns**

- `SOCK_ERR_NO_ERROR` for successful operation and negative code otherwise.

**See also**
- `tenuSslCertExpSettings`
setsockopt
Socket » Function
### Functions

| NMI_API | sint8 setsockopt (SOCKET socket, uint8 u8Level, uint8 option_name, const void *option_value, uint16 u16OptionLen) |
Detailed Description

The `setsockopt()` function shall set the option specified by the `option_name` argument, at the protocol level specified by the `level` argument, to the value pointed to by the `option_value` argument for the socket specified by the `socket` argument.

Possible protocol level values supported are `SOL_SOCKET` and `SOL_SSL_SOCKET`. Possible options when the protocol level is `SOL_SOCKET`:

- **SO_SET_UDP_SEND_CALLBACK**
  
  Enable/Disable callback messages for `sendto()`. Since UDP is unreliable by default the user maybe interested (or not) in receiving a message of `SOCKET_MSG_SENDTO` for each call of `sendto()`. Enabled if option value equals TRUE, disabled otherwise.

- **IP_ADD_MEMBERSHIP**
  
  Valid for UDP sockets. This option is used to receive frames sent to a multicast group. `option_value` shall be a pointer to Unsigned 32-bit integer containing the multicast IPv4 address.

- **IP_DROP_MEMBERSHIP**
  
  Valid for UDP sockets. This option is used to stop receiving frames sent to a multicast group. `option_value` shall be a pointer to Unsigned 32-bit integer containing the multicast IPv4 address.

Possible options when the protocol level is `SOL_SSL_SOCKET`:

- **Allow an opened SSL socket to bypass the X509 certificate**
**SO_SSL_BYPASS_X509_VERIF**

verification process. It is highly recommended **NOT** to use this socket option in production software applications. The option is supported for debugging and testing purposes. The option value should be casted to int type and it is handled as a boolean flag.

Set the Server Name Indicator (SNI) for an SSL socket. The SNI is a null terminated string containing the server name associated with the connection. It must not exceed the size of `HOSTNAME_MAX_SIZE`.

This option allow the TLS to cache the session information for fast TLS session establishment in future connections using the TLS Protocol session resume features.

**SO_SSL_SNI**

**SO_SSL_ENABLE_SESSION_CACHING**
setsockopt()

NMI_API sint8 setsockopt ( SOCKET socket, uint8 u8Level, uint8 option_name, const void * option_value, uint16 u16OptionLen )

Parameters

[in] sock Socket handler.
[in] level protocol level. See description above.
[in] option_name option to be set. See description above.
[in] option_value pointer to user provided value.
[in] option_len length of the option value in bytes.

Returns

The function shall return SOCK_ERR_NO_ERROR for successful operation and a negative value (indicating the error) otherwise.

See also

SOL_SOCKET, SOL_SSL_SOCKET, IP_ADD_MEMBERSHIP, IP_DROP_MEMBERSHIP
getsockopt
Socket » Function
### Functions

| NMI_API sint8 | getsockopt (SOCKET sock, uint8 u8Level, uint8 u8OptName, const void *pvOptValue, uint8 *pu8OptLen) |
Detailed Description

Get socket options retrieves This Function isn't implemented yet but this is the form that will be released later.
◆ getsockopt()

```c
sint8 getsockopt ( SOCKET sock,
                 uint8 u8Level,
                 uint8 u8OptName,
                 const void * pvOptValue,
                 uint8 * pu8OptLen )
```

**Parameters**

- **[in]** `sock` Socket Identifier.
- **[in]** `u8Level` The protocol level of the option.
- **[in]** `u8OptName` The `u8OptName` argument specifies a single option to get.
- **[out]** `pvOptValue` The `pvOptValue` argument contains pointer to a buffer containing the option value.
- **[out]** `pu8OptLen` Option value buffer length.

**Returns**

The function shall return ZERO for successful operation and a negative value otherwise.
m2m_ping_req

Socket » Function
### Functions

| NMI_API | sint8 | m2m_ping_req (uint32 u32DstIP, uint8 u8TTL, tpfPingCb fpPingCb) |
Detailed Description

The function sends ping request to the given IP Address.
Function Documentation
m2m_ping_req()

```c
NMI_API sint8 m2m_ping_req ( uint32 u32DstIP,
                              uint8  u8TTL,
                              tpfPingCb fpPingCb
)
```

### Parameters

- **[in]** `u32DstIP` Target Destination IP Address for the ping request. It must be represented in Network byte order. The function `nmi_inet_addr` could be used to translate the dotted decimal notation IP to its Network bytes order integer representative.

- **[in]** `u8TTL` IP TTL value for the ping request. If set to ZERO, the default value SHALL be used.

- **[in]** `fpPingCb` Callback will be called to deliver the ping statistics.

### See also

- `nmi_inet_addr`

### Returns

The function returns **M2M_SUCCESS** for successful operations and a negative value otherwise.
## Data Structures

Here are the data structures with brief descriptions:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_addr</td>
<td>IPv4 address representation</td>
</tr>
<tr>
<td>sockaddr</td>
<td>Generic socket address structure</td>
</tr>
<tr>
<td>sockaddr_in</td>
<td>Socket address structure for IPV4 addresses. Used to specify socket address information on which to connect to. Can be cast to other structures.</td>
</tr>
<tr>
<td>tstr1xAuthCredentials</td>
<td>Credentials for the user to authenticate with the AAA server (WPA-Enterprise Mode</td>
</tr>
<tr>
<td>tstrCryptoResp</td>
<td>Crypto response</td>
</tr>
<tr>
<td>tstrEthInitParam</td>
<td>Structure to hold Ethernet interface parameters. Structure is to be defined according to the application's functionality before a call is made to initialize the Wi-Fi operations by calling the <code>m2m_wifi_init</code> function. This structure is part of the Wi-Fi configuration structure and is defined in the <code>tstrWifiInitParam</code>. Applications should only define this structure if the bypass mode is not defined.</td>
</tr>
<tr>
<td>tstrM2MAPConfig</td>
<td>AP Configuration</td>
</tr>
<tr>
<td>tstrM2mBatteryVoltage</td>
<td>Battery Voltage</td>
</tr>
<tr>
<td>Scotte ID</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tstrM2mClientState</td>
<td>PS Client State</td>
</tr>
<tr>
<td>tstrM2MConnInfo</td>
<td>M2M Provisioning Information obtained from the HTTP Provisioning server</td>
</tr>
<tr>
<td>tstrM2MDefaultConnResp</td>
<td>Response error of the m2m_default_connect</td>
</tr>
<tr>
<td>tstrM2MDeviceNameConfig</td>
<td>Device name</td>
</tr>
<tr>
<td>tstrM2mEnableLogs</td>
<td>Enable Firmware logs</td>
</tr>
<tr>
<td>tstrM2MIPConfig</td>
<td>Static IP configuration</td>
</tr>
<tr>
<td>tstrM2mIpCtrlBuf</td>
<td>Structure holding the incoming buffer size information, indicating the data size information, indicating the data size of the incoming buffer and the remaining buffer's data size. The data of the buffer which holds the packet sent to the host when in the bypass mode, is placed in the tstrEthInitParam structure's au8ethRcvBuf attribute. This following information is retrieved in the host when an event M2M_WIFI_RESP_ETHERNET_RX_PACKET is received in the Wi-Fi callback function tpfAppWifiCb.</td>
</tr>
<tr>
<td>tstrM2mIpRsvdPkt</td>
<td>Received Packet Size and Data Offset</td>
</tr>
<tr>
<td>tstrM2mLsnInt</td>
<td>Listen interval</td>
</tr>
<tr>
<td>tstrM2MMulticastMac</td>
<td>M2M add/remove multi-cast mac address</td>
</tr>
<tr>
<td>tstrM2MP2PConnect</td>
<td>Set the device to operate in the Wi-Fi Direct (P2P) mode</td>
</tr>
<tr>
<td>tstrM2MProvisionInfo</td>
<td>M2M Provisioning Information obtained from the HTTP Provisioning server</td>
</tr>
<tr>
<td>tstrM2MProvisionModeConfig</td>
<td>M2M Provisioning Mode Configuration</td>
</tr>
<tr>
<td>tstrM2mPsType</td>
<td>Power Save Configuration</td>
</tr>
<tr>
<td>tstrM2mPwrMode</td>
<td>Power Mode</td>
</tr>
<tr>
<td>tstrM2mPwrState</td>
<td>Power Mode</td>
</tr>
<tr>
<td>tstrM2mReqScanResult</td>
<td>Scan Result Request</td>
</tr>
<tr>
<td>tstrM2MScan</td>
<td>Wi-Fi Scan Request</td>
</tr>
<tr>
<td>tstrM2mScanDone</td>
<td>Wi-Fi Scan Result</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tstrM2MScanOption</td>
<td>Scan options and configurations</td>
</tr>
<tr>
<td>tstrM2MScanRegion</td>
<td>Wi-Fi channel regulation region info</td>
</tr>
<tr>
<td>tstrM2Mservercmd</td>
<td>PS Server CMD</td>
</tr>
<tr>
<td>tstrM2mServerInit</td>
<td>PS Server initialization</td>
</tr>
<tr>
<td>tstrM2mSetMacAddress</td>
<td>Sets the MAC address from application. WINC load the mac address from the</td>
</tr>
<tr>
<td></td>
<td>by default to the WINC configuration but that function is used to let the</td>
</tr>
<tr>
<td></td>
<td>app overwrite the configuration memory with the mac address from the host</td>
</tr>
<tr>
<td>tstrM2mSlpReqTime</td>
<td>Manual power save request sleep time</td>
</tr>
<tr>
<td>tstrM2mTxPwrLevel</td>
<td>Tx power level</td>
</tr>
<tr>
<td>tstrM2mWifiConnect</td>
<td>Wi-Fi Connect Request</td>
</tr>
<tr>
<td>tstrM2mWifiGainsParams</td>
<td>Gain Values</td>
</tr>
<tr>
<td>tstrM2MWifiMonitorModeCtrl</td>
<td>Wi-Fi Monitor Mode Filter</td>
</tr>
<tr>
<td>tstrM2MWifiRxPacketInfo</td>
<td>Wi-Fi RX Frame Header</td>
</tr>
<tr>
<td>tstrM2mWifiscanResult</td>
<td>Wi-Fi Scan Result</td>
</tr>
<tr>
<td>tstrM2MWifiSecInfo</td>
<td>Authentication credentials to connect to a Wi-Fi network</td>
</tr>
<tr>
<td>tstrM2mWifiStateChanged</td>
<td>Wi-Fi Connection State</td>
</tr>
<tr>
<td>tstrM2MWifiTxPacketInfo</td>
<td>Wi-Fi TX Packet Info</td>
</tr>
<tr>
<td>tstrM2mWifiWepParams</td>
<td>WEP security key parameters</td>
</tr>
<tr>
<td>tstrM2MWPSConnect</td>
<td>WPS Configuration parameters</td>
</tr>
<tr>
<td>tstrM2MWPSInfo</td>
<td>WPS Result</td>
</tr>
<tr>
<td>tstrOtaControlSec</td>
<td>Control section structure is used to define the working image and the</td>
</tr>
<tr>
<td></td>
<td>validity of the image and its offset, also both firmware versions is kept in</td>
</tr>
<tr>
<td></td>
<td>that structure</td>
</tr>
<tr>
<td>tstrOtalInitHdr</td>
<td>OTA Image Header</td>
</tr>
<tr>
<td>tstrOtaUpdateInfo</td>
<td>OTA Update Information</td>
</tr>
<tr>
<td>tstrOtaUpdateStatusResp</td>
<td>OTA Update Information</td>
</tr>
<tr>
<td>tstrPrng</td>
<td>M2M Request PRNG</td>
</tr>
<tr>
<td><strong><code>tstrSocketAcceptMsg</code></strong></td>
<td>Socket accept status</td>
</tr>
</tbody>
</table>
![]()  
<p>| <strong><code>tstrSocketBindMsg</code></strong> | Socket bind status |
| <strong><code>tstrSocketConnectMsg</code></strong> | Socket connect status |
| <strong><code>tstrSocketListenMsg</code></strong> | Socket listen status |
| <strong><code>tstrSocketRecvMsg</code></strong> | Socket recv status |
| <strong><code>tstrSslSetActiveCsList</code></strong> | |
| <strong><code>tstrSystemTime</code></strong> | Used for time storage |
| <strong><code>tstrTlsCrlEntry</code></strong> | Certificate data for inclusion in a revocation list (CRL) |
| <strong><code>tstrTlsCrlInfo</code></strong> | Certificate revocation list details |
| <strong><code>tstrTlsSrvSecFileEntry</code></strong> | This struct contains a TLS certificate file entry |
| <strong><code>tstrTlsSrvSecHdr</code></strong> | This struct contains a set of TLS certificates |
| <strong><code>tstrWifiInitParam</code></strong> | Structure, holding the Wi-fi configuration attributes such as the wi-fi callback, monitoring mode callback and Ethernet parameter initialization structure. Such configuration parameters are required to be set before calling the wi-fi initialization function <code>m2m_wifi_init</code>. <code>pfAppWifiCb</code> attribute must be set to handle the wi-fi callback operations. <code>pfAppMonCb</code> attribute, is optional based on whether the application requires the monitoring mode configuration, and cannot be set before the initialization. |
| <strong><code>tuniM2MWifiAuth</code></strong> | Wi-Fi Security Parameters for all supported security modes |</p>
<table>
<thead>
<tr>
<th>Data Fields</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tstrEthInitParam Struct</td>
<td>Reference</td>
</tr>
</tbody>
</table>
### Data Fields

<table>
<thead>
<tr>
<th>tpfAppWifiCb</th>
<th>pfAppWifiCb</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpfAppEthCb</td>
<td>pfAppEthCb</td>
</tr>
<tr>
<td>uint8 * au8ethRcvBuf</td>
<td></td>
</tr>
<tr>
<td>uint16 u16ethRcvBufSize</td>
<td></td>
</tr>
<tr>
<td>uint8 u8EthernetEnable</td>
<td></td>
</tr>
<tr>
<td>uint8 <strong>PAD8</strong></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Structure to hold Ethernet interface parameters. Structure is to be defined and have its attributes set, based on the application's functionality before a call is made to initialize the Wi-Fi operations by calling the `m2m_wifi_init` function. This structure is part of the Wi-Fi configuration structure `tstrWifiInitParam`. Applications shouldn't need to define this structure, if the bypass mode is not defined.

See also
- tpfAppEthCb tpfAppWifiCb `m2m_wifi_init`

Warning
- Make sure that application defines ETH_MODE before using `tstrEthInitParam`. 
Field Documentation
 pfAppWifiCb

Callback for wifi notifications.
◆ pfAppEthCb

Callback for Ethernet interface.
◆ au8ethRcvBuf

uint8* au8ethRcvBuf

Pointer to Receive Buffer of Ethernet Packet
◆ u16ethRcvBufSize

uint16 u16ethRcvBufSize

Size of Receive Buffer for Ethernet Packet
<table>
<thead>
<tr>
<th><strong>u8EthernetEnable</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint8 u8EthernetEnable</strong></td>
</tr>
</tbody>
</table>

Enable Ethernet mode flag
### Padding

```
uint8 __PAD8__
```
<p>| tstrM2mlpCtrlBuf Struct Reference | Data Fields |</p>
<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16</td>
<td>u16DataSize</td>
</tr>
<tr>
<td>uint16</td>
<td>u16RemainingDataSize</td>
</tr>
</tbody>
</table>
Detailed Description

Structure holding the incoming buffer's data size information, indicating the data size of the buffer and the remaining buffer's data size. The data of the buffer which holds the packet sent to the host when in the bypass mode, is placed in the `tstrEthInitParam` structure in the au8ethRcvBuf attribute. This following information is retrieved in the host when an event `M2M_WIFI_RESP_ETHERNET_RX_PACKET` is received in the Wi-Fi callback function `tpfAppWifiCb`.

The application is expected to use this structure's information to determine if there is still incoming data to be received from the firmware.

See also
- `tpfAppEthCb` `tstrEthInitParam`

Warning
- Make sure that ETHERNET/bypass mode is defined before using `tstrM2mIpCtrlBuf`
Field Documentation
◆ u16DataSize

`uint16 u16DataSize`

Size of the received data in bytes.
◆ u16RemainigDataSize

<table>
<thead>
<tr>
<th>uint16 u16RemainigDataSize</th>
</tr>
</thead>
</table>

Size of the remaining data bytes to be delivered to host.
<table>
<thead>
<tr>
<th>tstrWifiInitParam</th>
<th>Struct Reference</th>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Data Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpfAppWifiCb</td>
<td>pfAppWifiCb</td>
</tr>
<tr>
<td>tpfAppMonCb</td>
<td>pfAppMonCb</td>
</tr>
<tr>
<td>tstrEthInitParam</td>
<td>strEthInitParam</td>
</tr>
</tbody>
</table>
Detailed Description

Structure, holding the Wi-fi configuration attributes such as the wi-fi callback, monitoring mode callback and Ethernet parameter initialization structure. Such configuration parameters are required to be set before calling the wi-fi initialization function `m2m_wifi_init`. `pfAppWifiCb` attribute must be set to handle the wi-fi callback operations. `pfAppMonCb` attribute, is optional based on whether the application requires the monitoring mode configuration, and can there not be set before the initialization. `strEthInitParam` structure, is another optional configuration based on whether the bypass mode is set.

See also
- `tpfAppEthCb`
- `tpfAppMonCb`
- `tstrEthInitParam`
Field Documentation
pfAppWifiCb

Callback for Wi-Fi notifications.
pfAppMonCb

Callback for monitoring interface.
◆ strEthInitParam

<table>
<thead>
<tr>
<th>tstrEthInitParam</th>
<th>strEthInitParam</th>
</tr>
</thead>
</table>

Structure to hold Ethernet interface parameters.

Generated on Thu Jan 26 2017 22:15:21 for WINC1500 IoT Software APIs by doxygen 1.8.13
Here is a list of all struct and union fields with links to the structures/unions they belong to:

- __PAD16__: tstrM2mBatteryVoltage, tstrM2mLsnInt, tstrM2mPsType, tstrM2mScanDone, tstrM2MScanRegion, tstrM2mSetMacAddress, tstrM2MWifiMonitorModeCtrl, tstrM2mWifiStateChanged, tstrPrng
- __PAD24__: tstrCryptoResp, tstrM2MAPConfig, tstrM2mClientState, tstrM2MConnInfo, tstrM2MDefaultConnResp, tstrM2mEnableLogs, tstrM2MP2PConnect, tstrM2MProvisionModeConfig, tstrM2mPwrMode, tstrM2mReqScanResult, tstrM2Mservercmd, tstrM2mServerInit, tstrM2mTxPwrLevel, tstrM2MWifiRxPacketInfo, tstrM2mWifiWepParams, tstrM2MWPSConnect, tstrTlsCrlEntry
- __PAD8__: tstrEthInitParam, tstrM2MMulticastMac, tstrOtaUpdateInfo, tstrSystemTime
- __PAD__: tstrM2mWifiConnect, tstrM2MWifiSecInfo
- __RSVD8__: tstrM2MScan
- __PAD16__: tstrOtaUpdateStatusResp
- __PAD8__: tstrM2mWifiscanResult
Here is a list of all struct and union fields with links to the structures/unions they belong to:

- a -

- acFileName : `tstrTlsSrvSecFileEntry`
- acHttpServerDomainName : `tstrM2MProvisionModeConfig`
- acPinNumber : `tstrM2MWPSConnect`
- acSSID : `tstrM2MConnInfo`
- astrEntries : `tstrTlsSrvSecHdr`
- astrTlsCrl : `tstrTlsCrlInfo`
- au8BSSID : `tstrM2MWifiMonitorModeCtrl`, `tstrM2MWifiRxPacketInfo`, `tstrM2mWifiscanResult`
- au8Data : `tstrTlsCrlEntry`
- au8DeviceName : `tstrM2MDeviceNameConfig`
- au8DHCPServerIP : `tstrM2MAPConfig`
- au8DstMacAddress : `tstrM2MWifiMonitorModeCtrl`, `tstrM2MWifiRxPacketInfo`
- au8ethRcvBuf : `tstrEthInitParam`
- au8IPAddr : `tstrM2MConnInfo`
- au8Key : `tstrM2MAPConfig`
- au8Mac : `tstrM2mSetMacAddress`
- au8MACAddress : `tstrM2MConnInfo`
- au8macaddress : `tstrM2MMulticastMac`
- au8Passwd : `tstr1xAuthCredentials`
- au8Password : `tstrM2MProvisionInfo`
- au8PSK : `tstrM2MWPSInfo`, `tuniM2MWifiAuth`
- au8SecStartPattern : `tstrTlsSrvSecHdr`
- au8SrcMacAddress : `tstrM2MWifiMonitorModeCtrl`, `tstrM2MWifiRxPacketInfo`
• au8SSID : tstrM2MAPConfig, tstrM2MProvisionInfo, tstrM2mWifiConnect, tstrM2mWifiscanResult, tstrM2MWPSInfo
• au8UserName : tstr1xAuthCredentials
• au8WepKey : tstrM2MAPConfig, tstrM2mWifiWepParams
Here is a list of all struct and union fields with links to the structures/unions they belong to:

- p -

- pfAppEthCb : tstrEthInitParam
- pfAppMonCb : tstrWifiInitParam
- pfAppWifiCb : tstrEthInitParam, tstrWifiInitParam
- pu8Buffer : tstrSocketRecvMsg
- pu8RngBuff : tstrPrng
Here is a list of all struct and union fields with links to the structures/unions they belong to:

- s -

- s16BufferSize : **tstrSocketRecvMsg**
- s8Error : **tstrSocketConnectMsg**
- s8ErrorCode : **tstrM2MDefaultConnResp**
- s8Resp : **tstrCryptoResp**
- s8RSSI : **tstrM2MConnInfo**, **tstrM2MWifiRxPacketInfo**
- s8rssi : **tstrM2mWifiscanResult**
- s8RssiThresh : **tstrM2MScanOption**
- s8ScanState : **tstrM2mScanDone**
- s_addr : **in_addr**
- sa_data : **sockaddr**
- sa_family : **sockaddr**
- sin_addr : **sockaddr_in**
- sin_family : **sockaddr_in**
- sin_port : **sockaddr_in**
- sin_zero : **sockaddr_in**
- sock : **tstrSocketAcceptMsg**, **tstrSocketConnectMsg**
- status : **tstrSocketBindMsg**, **tstrSocketListenMsg**
- strAddr : **tstrSocketAcceptMsg**
- strApConfig : **tstrM2MProvisionModeConfig**
- strCred1x : **tuniM2MWifiAuth**
- strEthInitParam : **tstrWifiInitParam**
- strRemoteAddr : **tstrSocketRecvMsg**
- strSec : **tstrM2mWifiConnect**
- strWepInfo : **tuniM2MWifiAuth**
Here is a list of all struct and union fields with links to the structures/unions they belong to:

- u -

- u16BattVolt : tstrM2mBatteryVoltage
- u16Ch : tstrM2mWifiConnect
- u16DataLength : tstrM2MWifiRxPacketInfo
- u16DataSize : tstrM2mIpCtrlBuf
- u16ethRcvBufSize : tstrEthInitParam
- u16FrameLength : tstrM2MWifiRxPacketInfo
- u16HeaderLength : tstrM2MWifiTxPacketInfo
- u16LsnInt : tstrM2mLsnInt
- u16PacketSize : tstrM2MWifiTxPacketInfo
- u16PassiveScanTime : tstrM2MScan
- u16PktOffset : tstrM2mIpRsvdPkt
- u16PktSz : tstrM2mIpRsvdPkt
- u16PrngSize : tstrPrng
- u16RemainingDataSize : tstrM2mIpCtrlBuf
- u16RemainingSize : tstrSocketRecvMsg
- u16ScanRegion : tstrM2MScanRegion
- u16Year : tstrSystemTime
- u32CsBMP : tstrSsslSetActiveCsList
- u32DataRateKbps : tstrM2MWifiRxPacketInfo
- u32DhcpLeaseTime : tstrM2MIPConfig
- u32DNS : tstrM2MIPConfig
- u32FileAddr : tstrTlsSrvSecFileEntry
- u32FileSize : tstrTlsSrvSecFileEntry
- u32Gateway : tstrM2MIPConfig
- u32nEntries : tstrTlsSrvSecHdr
• u32NextWriteAddr: tstrTlsSrvSecHdr
• u32OtaControlSecCrc: tstrOtaControlSec
• u32OtaCortusAppRollbackOffset: tstrOtaControlSec
• u32OtaCortusAppRollbackValidSts: tstrOtaControlSec
• u32OtaCortusAppRollbackVer: tstrOtaControlSec
• u32OtaCortusAppWorkingOffset: tstrOtaControlSec
• u32OtaCortusAppWorkingValidSts: tstrOtaControlSec
• u32OtaCortusAppWorkingVer: tstrOtaControlSec
• u32OtaCurrentworkingImagFirmwareVer: tstrOtaControlSec
• u32OtaCurrentworkingImagOffset: tstrOtaControlSec
• u32OtaFormatVersion: tstrOtaControlSec
• u32OtaLastCheckTime: tstrOtaControlSec
• u32OtaMagicValue: tstrOtaControlSec, tstrOtaInitHdr
• u32OtaPayloadSzie: tstrOtaInitHdr
• u32OtaRollbackImageOffset: tstrOtaControlSec
• u32OtaRollbackImageValidStatus: tstrOtaControlSec
• u32OtaRollbackImagFirmwareVer: tstrOtaControlSec
• u32OtaSequenceNumber: tstrOtaControlSec
• u32SleepTime: tstrM2mSlpReqTime
• u32StaticIP: tstrM2MIPConfig
• u32SubnetMask: tstrM2MIPConfig
• u8AddRemove: tstrM2MMulticastMac
• u8AuthType: tstrM2mWifiscanResult, tstrM2MWPSInfo
• u8BcastEn: tstrM2mPsType
• u8ch: tstrM2mWifiscanResult
• u8Ch: tstrM2MWPSInfo
• u8Channel: tstrM2mServerInit
• u8ChannelID: tstrM2MWifiMonitorModeCtrl
• u8ChNum: tstrM2MScan
• u8CipherType: tstrM2MWifiRxPacketInfo
• u8cmd: tstrM2Mservercmd
• u8CrlType: tstrTlsCrlInfo
• u8CurrState: tstrM2mWifiStateChanged
• u8DataLen: tstrTlsCrlEntry
• u8Day: tstrSystemTime
• u8DownloadUrlOffset: tstrOtaUpdateInfo
• u8DownloadUrlSize: tstrOtaUpdateInfo
• u8Enable: tstrM2mEnableLogs
• u8EnableRedirect: tstrM2MProvisionModeConfig
• u8EnRecvHdr: tstrM2MWifiMonitorModeCtrl
• u8ErrCode : tstrM2mWifiStateChanged
• u8EthernetEnable : tstrEthInitParam
• u8FrameSubtype : tstrM2MWifiMonitorModeCtrl, tstrM2MWifiRxPacketInfo
• u8FrameType : tstrM2MWifiMonitorModeCtrl, tstrM2MWifiRxPacketInfo
• u8HeaderLength : tstrM2MWifiRxPacketInfo
• u8Hour : tstrSystemTime
• u8Index : tstrM2mReqScanResult
• u8Index : tstrM2mWifiscanResult
• u8KeyIndx : tstrM2MAPConfig, tstrM2mWifiWepParams
• u8KeySz : tstrM2MAPConfig, tstrM2mWifiWepParams
• u8ListenChannel : tstrM2MAPConfig, tstrM2MP2PConnect
• u8Minute : tstrSystemTime
• u8Month : tstrSystemTime
• u8NcdRequiredUpgrade : tstrOtaUpdateInfo
• u8NcdUpgradeVersion : tstrOtaUpdateInfo
• u8NcfCurrentVersion : tstrOtaUpdateInfo
• u8NcfUpgradeVersion : tstrOtaUpdateInfo
• u8NoSaveCred : tstrM2mWifiConnect
• u8NumofCh : tstrM2mScanDone
• u8NumOfSlot : tstrM2MScanOption
• u8OtaUpdateStatus : tstrOtaUpdateStatusResp
• u8OtaUpdateStatusType : tstrOtaUpdateStatusResp
• u8PPAGFor11B : tstrM2mWifiGainsParams
• u8PPAGFor11GN : tstrM2mWifiGainsParams
• u8Priority : tstrM2MWifiRxPacketInfo
• u8ProbesPerSlot : tstrM2MScanOption
• u8PsType : tstrM2mPsType
• u8PwrMode : tstrM2mPwrMode
• u8Rsv1 : tstrTlsCrlInfo
• u8Rsv2 : tstrTlsCrlInfo
• u8Rsv3 : tstrTlsCrlInfo
• u8Second : tstrSystemTime
• u8SecType : tstrM2MAPConfig, tstrM2MConnInfo, tstrM2M ProvisionInfo, tstrM2MWifiSecInfo
• u8ServiceClass : tstrM2MWifiRxPacketInfo
• u8SlotTime : tstrM2MScanOption
• u8SsidHide : tstrM2MAPConfig
• u8State : tstrM2mClientState
- u8Status : tstrM2MProvisionInfo
- u8TriggerType : tstrM2MWPSConnect
- u8TxPwrLevel : tstrM2mTxPwrLevel
- uniAuth : tstrM2MWifiSecInfo
__PAD16__ : tstrM2mBatteryVoltage, tstrM2mLsnInt, tstrM2mPsType, tstrM2mScanDone, tstrM2MScanRegion, tstrM2mSetMacAddress, tstrM2MWifiMonitorModeCtrl, tstrM2mWifiStateChanged, tstrPrng

__PAD24__ : tstrCryptoResp, tstrM2MAPConfig, tstrM2mClientState, tstrM2MConnInfo, tstrM2MDefaultConnResp, tstrM2mEnableLogs, tstrM2MP2PConnect, tstrM2MProvisionModeConfig, tstrM2mPwrMode, tstrM2mReqScanResult, tstrM2MServercmd, tstrM2mServerInit, tstrM2mTxPwrLevel, tstrM2MWifiRxPacketInfo, tstrM2mWifiWepParams, tstrM2MWPSConnect, tstrTlsCrlEntry

__PAD8__ : tstrEthInitParam, tstrM2MMulticastMac, tstrOtaUpdateInfo, tstrSystemTime

__PAD__ : tstrM2mWifiConnect, tstrM2MWifiSecInfo

__RSVD8__ : tstrM2MScan

__PAD16__ : tstrOtaUpdateStatusResp

__PAD8__ : tstrM2mWifiscanResult
- a -

- acFileName : tstrTlsSrvSecFileEntry
- acHttpServerDomainName : tstrM2MProvisionModeConfig
- acPinNumber : tstrM2MWPSConnect
- acSSID : tstrM2MConnInfo
- astrEntries : tstrTlsSrvSecHdr
- astrTlsCrl : tstrTlsCrlInfo
- au8BSSID : tstrM2MWifiMonitorModeCtrl, tstrM2MWifiRxPacketInfo, tstrM2mWifiscanResult
- au8Data : tstrTlsCrlEntry
- au8DeviceName : tstrM2MDeviceNameConfig
- au8DHCPServerIP : tstrM2MAPConfig
- au8DstMacAddress : tstrM2MWifiMonitorModeCtrl, tstrM2MWifiRxPacketInfo
- au8ethRcvBuf : tstrEthInitParam
- au8IPAddr : tstrM2MConnInfo
- au8Key : tstrM2MAPConfig
- au8Mac : tstrM2mSetMacAddress
- au8MACAddress : tstrM2MConnInfo
- au8macaddress : tstrM2MMulticastMac
- au8Password : tstr1xAuthCredentials
- au8PSK : tstrM2MWPSInfo, tuniM2MWifiAuth
- au8SecStartPattern : tstrTlsSrvSecHdr
- au8SrcMacAddress : tstrM2MWifiMonitorModeCtrl, tstrM2MWifiRxPacketInfo
- au8SSID : tstrM2MAPConfig, tstrM2MProvisionInfo,
tstrM2mWifiConnect, tstrM2mWifiscanResult, tstrM2MWPSInfo

- au8UserName: tstr1xAuthCredentials
- au8WepKey: tstrM2MAPConfig, tstrM2mWifiWepParams
- p -

- pfAppEthCb : tstrEthInitParam
- pfAppMonCb : tstrWifiInitParam
- pfAppWifiCb : tstrEthInitParam, tstrWifiInitParam
- pu8Buffer : tstrSocketRecvMsg
- pu8RngBuff : tstrPrng
- s -

- s16BufferSize : tstrSocketRecvMsg
- s8Error : tstrSocketConnectMsg
- s8ErrorCode : tstrM2MDefaultConnResp
- s8Resp : tstrCryptoResp
- s8RSSI : tstrM2MConnInfo, tstrM2MWifiRxPacketInfo
- s8ssi : tstrM2mWifiscanResult
- s8RssiThresh : tstrM2MScanOption
- s8ScanState : tstrM2mScanDone
- s_addr : in_addr
- sa_data : sockaddr
- sa_family : sockaddr
- sin_addr : sockaddr_in
- sin_family : sockaddr_in
- sin_port : sockaddr_in
- sin_zero : sockaddr_in
- sock : tstrSocketAcceptMsg, tstrSocketConnectMsg
- status : tstrSocketBindMsg, tstrSocketListenMsg
- strAddr : tstrSocketAcceptMsg
- strApConfig : tstrM2MProvisionModeConfig
- strCred1x : tuniM2MWifiAuth
- strEthInitParam : tstrWifiInitParam
- strRemoteAddr : tstrSocketRecvMsg
- strSec : tstrM2mWifiConnect
- strWepInfo : tuniM2MWifiAuth
- u -

- u16BattVolt : tstrM2mBatteryVoltage
- u16Ch : tstrM2mWifiConnect
- u16DataLength : tstrM2MWifiRxPacketInfo
- u16DataSize : tstrM2mlpCtrlBuf
- u16ethRcvBufSize : tstrEthInitParam
- u16FrameLength : tstrM2MWifiRxPacketInfo
- u16HeaderLength : tstrM2MWifiTxPacketInfo
- u16LsnInt : tstrM2mLsnInt
- u16PacketSize : tstrM2MWifiTxPacketInfo
- u16PassiveScanTime : tstrM2MScan
- u16PktOffset : tstrM2mlpRsvdPkt
- u16PktSz : tstrM2mlpRsvdPkt
- u16PrngSize : tstrPrng
- u16RemainingDataSize : tstrM2mlpCtrlBuf
- u16RemainingSize : tstrSocketRecvMsg
- u16ScanRegion : tstrM2MScanRegion
- u16Year : tstrSystemTime
- u32CsBMP : tstrSslSetActiveCsList
- u32DataRateKbps : tstrM2MWifiRxPacketInfo
- u32DhcpLeaseTime : tstrM2MIPConfig
- u32DNS : tstrM2MIPConfig
- u32FileAddr : tstrTlsSrvSecFileEntry
- u32FileSize : tstrTlsSrvSecFileEntry
- u32Gateway : tstrM2MIPConfig
- u32nEntries : tstrTlsSrvSecHdr
- u32NextWriteAddr : tstrTlsSrvSecHdr
- u32OtaControlSecCrc: tstrOtaControlSec
- u32OtaCortusAppRollbackOffset: tstrOtaControlSec
- u32OtaCortusAppRollbackValidSts: tstrOtaControlSec
- u32OtaCortusAppRollbackVer: tstrOtaControlSec
- u32OtaCortusAppWorkingOffset: tstrOtaControlSec
- u32OtaCortusAppWorkingValidSts: tstrOtaControlSec
- u32OtaCortusAppWorkingVer: tstrOtaControlSec
- u32OtaCurrentworkingImagFirmwareVer: tstrOtaControlSec
- u32OtaCurrentworkingImagOffset: tstrOtaControlSec
- u32OtaFormatVersion: tstrOtaControlSec
- u32OtaLastCheckTime: tstrOtaControlSec
- u32OtaMagicValue: tstrOtaControlSec, tstrOtalInitHdr
- u32OtaPayloadSzie: tstrOtalInitHdr
- u32OtaRollbackImageOffset: tstrOtaControlSec
- u32OtaRollbackImageValidStatus: tstrOtaControlSec
- u32OtaRollbackImagFirmwareVer: tstrOtaControlSec
- u32OtaSequenceNumber: tstrOtaControlSec
- u32SleepTime: tstrM2mSlpReqTime
- u32StaticIP: tstrM2MIPConfig
- u32SubnetMask: tstrM2MIPConfig
- u8AddRemove: tstrM2MMulticastMac
- u8AuthType: tstrM2mWifiscanResult, tstrM2MWPSInfo
- u8BcastEn: tstrM2mPsType
- u8Ch: tstrM2MWPSInfo
- u8Channel: tstrM2mServerInit
- u8ChannelID: tstrM2MWifiMonitorModeCtrl
- u8ChNum: tstrM2MScan
- u8CipherType: tstrM2MWifiRxPacketInfo
- u8cmd: tstrM2Mservcmd
- u8CrlType: tstrTlsCrlInfo
- u8CurrState: tstrM2mWifiStateChanged
- u8DataLen: tstrTlsCrlEntry
- u8Day: tstrSystemTime
- u8DownloadUrlOffset: tstrOtaUpdateInfo
- u8DownloadUrlSize: tstrOtaUpdateInfo
- u8Enable: tstrM2mEnableLogs
- u8EnableRedirect: tstrM2MProvisionModeConfig
- u8EnRecvHdr: tstrM2MWifiMonitorModeCtrl
- u8ErrCode: tstrM2mWifiStateChanged
- u8EthernetEnable: tstrEthInitParam
- u8FrameSubtype: tstrM2MWifiMonitorModeCtrl, tstrM2MWifiRxPacketInfo
- u8FrameType: tstrM2MWifiMonitorModeCtrl, tstrM2MWifiRxPacketInfo
- u8HeaderLength: tstrM2MWifiRxPacketInfo
- u8Hour: tstrSystemTime
- u8Index: tstrM2mReqScanResult
- u8index: tstrM2mWifiscanResult
- u8KeyIdx: tstrM2MAPConfig, tstrM2mWifiWepParams
- u8KeySz: tstrM2MAPConfig, tstrM2mWifiWepParams
- u8ListenChannel: tstrM2MAPConfig, tstrM2MP2PConnect
- u8Minute: tstrSystemTime
- u8Month: tstrSystemTime
- u8NcdRequiredUpgrade: tstrOtaUpdateInfo
- u8NcdUpgradeVersion: tstrOtaUpdateInfo
- u8NcfCurrentVersion: tstrOtaUpdateInfo
- u8NcfUpgradeVersion: tstrOtaUpdateInfo
- u8NoSaveCred: tstrM2mWifiConnect
- u8NumofCh: tstrM2mScanDone
- u8NumOfSlot: tstrM2MScanOption
- u8OtaUpdateStatus: tstrOtaUpdateStatusResp
- u8OtaUpdateStatusType: tstrOtaUpdateStatusResp
- u8PPAGFor11B: tstrM2mWifiGainsParams
- u8PPAGFor11GN: tstrM2mWifiGainsParams
- u8Priority: tstrM2MWifiRxPacketInfo
- u8ProbesPerSlot: tstrM2MScanOption
- u8PsType: tstrM2mPsType
- u8PwrMode: tstrM2mPwrMode
- u8Rsv1: tstrTlsCrlInfo
- u8Rsv2: tstrTlsCrlInfo
- u8Rsv3: tstrTlsCrlInfo
- u8Second: tstrSystemTime
- u8SecType: tstrM2MAPConfig, tstrM2MConnInfo, tstrM2MProvisionInfo, tstrM2MWifiSecInfo
- u8ServiceClass: tstrM2MWifiRxPacketInfo
- u8SlotTime: tstrM2MScanOption
- u8SsidHide: tstrM2MAPConfig
- u8State: tstrM2mClientState
- u8Status: tstrM2MProvisionInfo
- `u8TriggerType`: `tstrM2MWPSConnect`
- `u8TxPwrLevel`: `tstrM2mTxPwrLevel`
- `uniAuth`: `tstrM2MWifiSecInfo`