AN5305

ATWINC15x0/ATWINC3400 Wi-Fi® Network Controller Software Programming Guide

Introduction

This software programming guide describes the use of the ATWINC15x0/ATWINC3400 Wi-Fi Network Controller to build state-of-the-art Internet of Things (IoT) applications.

The following topics will be covered:

- How examples are organized
- Target board information
- Instruction for each example

Note: All the listed examples and folder structure/files are similar for both WINC15x0 and ATWINC3400. The diagrams and console logs in this document refer to ATWINC15x0.

Prerequisites

- Hardware Prerequisites:
  - SAM D21 Xplained Pro Evaluation Kit
  - ATWINC15x0 extension (For more details, refer to ATWINC15x0 user guide)
  - ATWINC3400 extension (For more details, refer to ATWINC3400 user guide)
  - IO1 extension
  - Micro-USB Cable (Micro-A/Micro-B)
- Software Prerequisites:
  - Atmel Studio 7.0 (For more details, refer to user guide)
  - Wi-Fi IoT Examples (for more details, refer to application notes)

Figure 1. SAM D21 XSTK Board Demo Setup

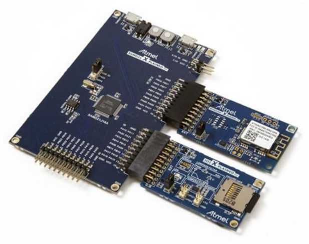


Table of Contents

1. How the Examples are Organized

This example package consists of several projects containing code examples. These examples are organized in a way that provides a wide coverage of the ATWINC15x0/ATWINC3400 API usage – from basic Wi-Fi operation to advanced topics. These examples are categorized as:

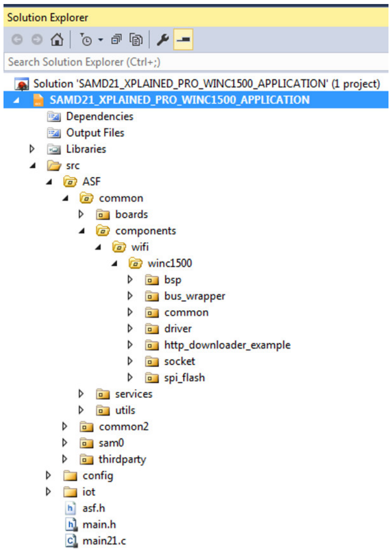
- Basic
- Protocol
- Advanced

2. Source Organization

There are some folders automatically allocated according to user configurations. The example source consists of `main.c` and `main.h`. The application source files are structured as follows:

- ./src/ASF
  - All ASF modules' source files are located in this folder. You can select various modules with the ASF wizard and it will configure the content in this folder.
- ./src/config
  - This folder consists of configuration header files for the SAM D21 and extension boards.
- ./src/iot
  - Some protocol/advanced examples include this folder. It contains source code to make use of certain protocols (for example, HTTP, MQTT, and so on), typically needed in IoT applications.
- ./src/ASF/common/components/wifi/winc1500
  - This is the folder which contains the driver source for the ATWINC15x0 Wi-Fi module.
- ./src/ASF/common/components/wifi/winc3400
  - This is the folder which contains the driver source for the ATWINC3400 Wi-Fi module.

Figure 2-1. ATWINC1500 Application Solution Explorer



```
Solution 'SAMD21_XPLAINED_PRO_WINC1500_APPLICATION' (1 project)
◢ SAMD21_XPLAINED_PRO_WINC1500_APPLICATION
      Dependencies
      Output Files
    ▷ Libraries
    ◢ src
        ◢ ASF
            ◢ common
                ▷ boards
                ◢ components
                    ◢ wifi
                        ◢ winc1500
                            ▷ bsp
                            ▷ bus_wrapper
                            ▷ common
                            ▷ driver
                            ▷ http_downloader_example
                            ▷ socket
                            ▷ spi_flash
                ▷ services
                ▷ utils
            ▷ common2
            ▷ sam0
            ▷ thirdparty
        ▷ config
        ▷ iot
          asf.h
          main.h
          main21.c
```

Note: Some examples may have additional source files, but the structure is similar across these examples.

3. Basic Operation Code

This section explains the basic code for using the SAM D21 and ATWINC15x0/ATWINC3400. The use of the ATWINC API may differ according to the purpose of your actual application.

3.1 Basic Examples

These examples describe basic Wi-Fi operation in a 'how-to' manner:

- How to read the Chip ID (to identify WINC15x0/ATWINC3400 H/W revision differences)
- How to set/get the MAC address of the Wi-Fi module
- How to start Wi-Fi in a specific operation mode, such as:
  - STA Mode (Station mode, known as a Wi-Fi client)
  - AP mode (Access Point mode)
- How to switch mode between the STA and AP modes at runtime
- How to scan for nearby APs
- How to set Deep Sleep mode
- How to connect to a secure Wi-Fi network using WEP/WPA/WPA2 security
- How to connect to an enterprise security network
- How to connect via Wi-Fi Protected Setup (WPS)
- How to get an RF signal status by reading the RSSI value
- How to set AP provision using Android App
- How to set HTTP provision

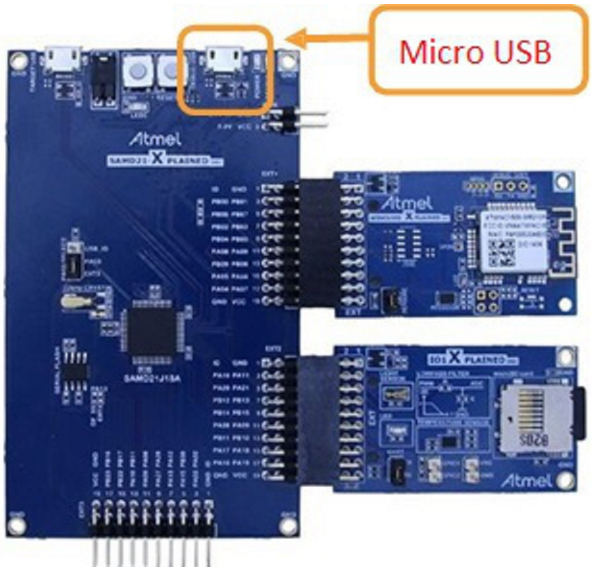
3.1.1 How to Read the Chip ID

This example demonstrates how to retrieve the chip information of the ATWINC15x0/ATWINC3400 using the SAM D21 Xplained Pro board. This is a basic operation to identify which HW version is used.

It is based on the following hardware:

- The SAM D21 Xplained Pro board
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 3-1. Get Chip ID Demo Setup



3.1.1.1 Execution

`main.c` - Initialize the ATWINC15x0/ATWINC3400 and retrieve information.

1. Code summary:
   – `nmi_get_chipid()` function returns the chip ID of the ATWINC15x0/ATWINC3400.
   – `nmi_get_rfrevid()` function returns the RF revision ID.

```
/* Display WINC1500 chip information. */
printf("Chip ID : \r\t\t\t%x\r\n", (unsigned int)nmi_get_chipid());
printf("RF Revision ID : \r\t\t\t%x\r\n", (unsigned int)nmi_get_rfrevid());
```

2. Build the program and download it into the board.
3. Start the application.

3.1.1.2 Get Chip ID Demo Console Log

Note: When the application executes successfully it provides the version information in the console.

```
-- WINC1500 chip information example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Chip ID : 1503a0
RF Revision ID : 1
Done.
```

Note: ATWINC15x0/ATWINC3400 behavior and corresponding log messages can be different depending upon the revision.

The following are the details of the ATWINC3400 chip ID for reference.

- Chip ID – 3400d1
- RF Revision ID – 6

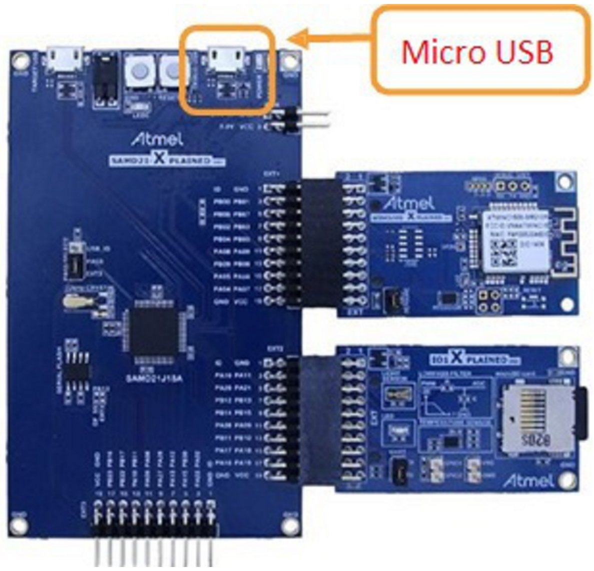
3.1.2 How to Set/Get the MAC Address

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to retrieve and set the MAC address of the Wi-Fi module.

The example is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 3-2. Demo Setup



3.1.2.1 Execution

Initialize the ATWINC15x0/ATWINC3400 and retrieve information.

1. Code summary:
   – The MAC address is typically stored in the OTP-ROM. You can get it via the `m2m_wifi_get_otp_mac_address()` function.

```
/* Get MAC Address from OTP. */
    m2m_wifi_get_otp_mac_address(mac_addr, &u8IsMacAddrValid);
```

   – To set the user defined MAC address in the program memory, the `m2m_wifi_set_mac_address()` function is used. This API needs to be called whenever the system resets. This API overwrites the program memory MAC address, which is loaded from the OTP memory during the initialization process.

```
/** User defined MAC Address. */
        const char main_user_defined_mac_address[] = {0xf8, 0xf0, 0x05, 0x20, 0x0b,
0x09};
       /* Cannot find MAC Address from OTP. Set user defined MAC address. */
    m2m_wifi_set_mac_address((uint8_t *)main_user_defined_mac_address);
```

   – The API `m2m_wifi_get_mac_address()` is used to read the program memory MAC address. It is currently used by the WLAN device.

```
/* Get MAC Address. */
    m2m_wifi_get_mac_address(mac_addr);
    printf("%02X:%02X:%02X:%02X:%02X:%02X\r\n",
            mac_addr[0], mac_addr[1], mac_addr[2],
            mac_addr[3], mac_addr[4], mac_addr[5]);
```

2. Build the program and download it into the board.
3. Start the application.

3.1.2.2 Get MAC Address Demo Console Log

The application should now be programmed and running. The following information will be displayed on the terminal window.

```
-- WINC1500 MAC Address example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
OTP MAC Address : F8:F0:05:F2:5F:62
```

Notes:

- The default MAC address is stored in One-Time-Programmable ROM (OTP ROM).
- The output of this example should display the OTP MAC address or USER MAC address.
- User defined MAC address: If you want to use a custom MAC address, there is an API available which allows for setting a user defined MAC address.

3.1.3 How to Get the Wi-Fi Network Signal Strength

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to check signal strength, such as RSSI.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 3-3. Get the Signal Status Demo Setup

3.1.3.1 Execution

`main.c` - Initialize the ATWINC15x0/ATWINC3400 and connect to the AP as a station.

1. Code summary:
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID        "DEMO_AP" /* < Destination SSID */
#define MAIN_WLAN_AUTH        M2M_WIFI_SEC_WPA_PSK /* < Security type */
#define MAIN_WLAN_PSK         "12345678" /* < Password for Destination SSID */
```

   – Connect the ATWINC15x0/ATWINC3400 to the AP via the `m2m_wifi_connect()` function.

```
/* Connect to defined AP. */
    m2m_wifi_connect((char *)MAIN_WLAN_SSID, sizeof(MAIN_WLAN_SSID),
                MAIN_WLAN_AUTH, (void *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);
```

   – Call `m2m_wifi_req_curr_rssi()` to request the current RSSI.

```

static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_REQ_DHCP_CONF:
    {
    /* Request RSSI for the connected AP. */
    m2m_wifi_req_curr_rssi();
```

   – The application will get the RSSI value when the `wifi_cb()` function is called with a `M2M_WIFI_RESP_CURRENT_RSSI` message.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_RESP_CURRENT_RSSI:
    {
    /* This message type is triggered by "m2m_wifi_req_curr_rssi()" function. */
    int8_t *rssi = (int8_t *)pvMsg;
    printf("RSSI for the current connected AP (%d)\r\n", (int8_t)(*rssi));
```

2. Build the program and download it into the board.
3. Start the application.

3.1.3.2 Signal Status Demo Console Log

The application should now be programmed and running. The following information will be displayed on the terminal window.

```
-- WINC1500 signal statistics example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Wi-Fi connected
Wi-Fi IP is 192.168.1.46
RSSI for the current connected AP (-37)
```

3.1.4 How to Run STA Mode

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to behave as a station.

The example is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 3-4. Demo Setup

3.1.4.1 Execution

`main.c` – Initialize the ATWINC15x0/ATWINC3400 and get the RSSI for the connected AP.

1. Code summary:
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID        "DEMO_AP" /* < Destination SSID */
#define MAIN_WLAN_AUTH        M2M_WIFI_SEC_WPA_PSK /* < Security manner */
#define MAIN_WLAN_PSK         "12345678" /* < Password for Destination SSID */
```

   – Connect to the AP with the given information.

```
/* Connect to defined AP. */
    m2m_wifi_connect((char *)MAIN_WLAN_SSID, sizeof(MAIN_WLAN_SSID),
                MAIN_WLAN_AUTH, (void *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);
```

   Note: The WINC1500 stores the last successful connection in its memory by default, so if the user disconnects from the AP, the user can call `m2m_wifi_default_connect();` instead of calling `m2m_wifi_connect()` with the same credentials again.

   – The `wifi_cb()` function is called with the `M2M_WIFI_RESP_CON_STATE_CHANGED` message, where an appropriate action can be executed as a response. The new Wi-Fi state is passed onto the callback as part of the message.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    switch (u8MsgType) {
    case M2M_WIFI_RESP_CON_STATE_CHANGED:
    {
       ...
    tstrM2mWifiStateChanged *pstrWifiState = (tstrM2mWifiStateChanged *)pvMsg;
    if (pstrWifiState->u8CurrState == M2M_WIFI_CONNECTED){
       /* Take action on connection successful */
       ...
```

   – The `wifi_cb()` function is called with the `M2M_WIFI_REQ_DHCP_CONF` message. If successful, the DHCP assigned IP address is passed to the application.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_REQ_DHCP_CONF:
    {
       ...
    uint8_t *pu8IPAddress = (uint8_t *)pvMsg;
    printf("Wi-Fi connected\r\n");
    printf("Wi-Fi IP is %u.%u.%u.%u\r\n",
            pu8IPAddress[0], pu8IPAddress[1], pu8IPAddress[2], pu8IPAddress[3]);
       ...
```

   After the IP is assigned, the next time the DHCP client running on WINC wants the IP, it can start the process with the DHCP request packet straight instead of the DHCP Discovery packet. It has been observed that, some DHCP servers do not respond to the DHCP REQ packet as the first packet but expect the DHCP Discovery to come in from the DHCP client. Ensure that WINC sends a DHCP Discovery packet every time an IP needs to be configured. It is recommended that the user call `m2m_wifi_disconnect()` just before `m2m_wifi_connect()`, which confirms that the DHCP exchange starts from the Discovery packet.
2. Build the program and download it into the board.
3. Start the application.

3.1.4.2 Station Mode Demo Console Log

Note: The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 station mode example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
```

3.1.5 How to Run AP Mode

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to behave as an AP.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 3-5. Demo Setup

3.1.5.1 Execution

`main.c` - Initialize the ATWINC15x0/ATWINC3400 and act as an AP.

1. Code summary:
   – Configure the network parameters in `main.h`.

```
/** Security mode Supported */
#define USE_WPAPSK        1 /* WPA/WPA2 PSK Security Mode*/
// #define USE_WEP            2 /* WEP Security Mode*/
// #define USE_OPEN        3 /* No Security or OPEN Authentication Mode*/

/** AP mode Settings */
#define MAIN_WLAN_SSID           "WINC1500_AP" /* < SSID */
#if (defined USE_WPAPSK)
#define MAIN_WLAN_AUTH           M2M_WIFI_SEC_WPA_PSK /* < Security type */
#define MAIN_WLAN_WPA_PSK        "1234567890" /* < Security Key in WPA PSK Mode */
#elif (defined USE_WEP)
#define MAIN_WLAN_AUTH           M2M_WIFI_SEC_WEP /* < Security type */
#define MAIN_WLAN_WEP_KEY        "1234567890" /* < Security Key in WEP Mode */
#define MAIN_WLAN_WEP_KEY_INDEX  (0)
#elif (defined USE_OPEN)
#define MAIN_WLAN_AUTH           M2M_WIFI_SEC_OPEN /* < Security type */
#endif
#define MAIN_WLAN_CHANNEL        (M2M_WIFI_CH_6) /* < Channel number */
```

   – In the `main()` function, initializes the AP mode configuration structure (`strM2MAPConfig`) , see the following example. User can enable the AP mode via `m2m_wifi_enable_ap()` function. Users may choose any of the available security methods according to the requirements.

```
/* Initialize AP mode parameters structure with SSID, channel and OPEN security
type. */
    memset(&strM2MAPConfig, 0x00, sizeof(tstrM2MAPConfig));
    strcpy((char *)&strM2MAPConfig.au8SSID, MAIN_WLAN_SSID);
    strM2MAPConfig.u8ListenChannel = MAIN_WLAN_CHANNEL;
    strM2MAPConfig.u8SecType = MAIN_WLAN_AUTH;

    strM2MAPConfig.au8DHCPServerIP[0] = 192;
    strM2MAPConfig.au8DHCPServerIP[1] = 168;
    strM2MAPConfig.au8DHCPServerIP[2] = 1;
    strM2MAPConfig.au8DHCPServerIP[3] = 1;
#if USE_WEP
    strcpy((char *)&strM2MAPConfig.au8WepKey, MAIN_WLAN_WEP_KEY);
    strM2MAPConfig.u8KeySz = strlen(MAIN_WLAN_WEP_KEY);
    strM2MAPConfig.u8KeyIndx = MAIN_WLAN_WEP_KEY_INDEX;
#endif
#if USE_WPAPSK
    strcpy((char *)&strM2MAPConfig.au8Key, MAIN_WLAN_WPA_PSK);
    strM2MAPConfig.u8KeySz = strlen(MAIN_WLAN_WPA_PSK);
#endif
    /* Bring up AP mode with parameters structure. */
    ret = m2m_wifi_enable_ap(&strM2MAPConfig);
    printf("AP mode started. You can connect to %s.\r\n", (char *)MAIN_WLAN_SSID);
```

2. Build the program and download it into the board.
3. Start the application.

3.1.5.2 AP Mode Demo Console Log

Note: The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 AP mode example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
AP mode started. You can connect to WINC1500_AP.
Station connected
Station IP is xxx.xxx.xxx.xxx
```

Note: The ATWINC15x0/ATWINC3400 supports AP mode with the following limitations:

1. Only one associated station is supported. After a connection is established with a station, further connections are rejected.
2. The device cannot work as a station in this mode (STA/AP concurrency is not supported).

3.1.6 How to Change Modes

This example demonstrates how to use the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board as a station, and change it to AP.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header

Figure 3-6. Demo Setup

3.1.6.1 Execution

`main.c` - Initialize the ATWINC15x0. By default, the device starts in station mode, then switches to AP mode. For details on each mode, refer to the MODE_STA, MODE_AP example.

1. Code summary:

- Configure the network parameters in `main.h`.

```
/** Security mode Supported */
#define USE_WPAPSK        1 /* WPA/WPA2 PSK Security Mode*/
// #define USE_WEP            2 /* WEP Security Mode*/
// #define USE_OPEN        3 /* No Security or OPEN Authentication Mode*/

/** AP mode Settings */
#define MAIN_WLAN_SSID           "WINC1500_AP" /* < SSID */
#if (defined USE_WPAPSK)
#define MAIN_WLAN_AUTH           M2M_WIFI_SEC_WPA_PSK /* < Security type */
#define MAIN_WLAN_WPA_PSK        "1234567890" /* < Security Key in WPA PSK Mode */
#elif (defined USE_WEP)
#define MAIN_WLAN_AUTH           M2M_WIFI_SEC_WEP /* < Security type */
#define MAIN_WLAN_WEP_KEY        "1234567890" /* < Security Key in WEP Mode */
#define MAIN_WLAN_WEP_KEY_INDEX  (0)
#elif (defined USE_OPEN)
#define MAIN_WLAN_AUTH           M2M_WIFI_SEC_OPEN /* < Security type */
#endif
#define MAIN_WLAN_CHANNEL        (M2M_WIFI_CH_6) /* < Channel number */
```

- AP mode gets enabled in the `main()` function. For more details, refer to the “How to Run AP mode” example.

```
enable_disable_ap_mode();
nm_bsp_sleep(DELAY_FOR_MODE_CHANGE);
```

2. Build the program and download it into the board.
3. Start the application.

3.1.6.2 Mode Change Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 mode change example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
AP mode, start
AP mode, end
```

3.1.7 How to Scan for APs

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to explain how to scan for an AP when running as a station.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on EXT1 header

Figure 3-7. Scan APs Demo Setup

3.1.7.1 Execution

`main.c` - Initialize the ATWINC15x0/ATWINC3400 and scan for APs until the defined AP is found.

1. Code summary:
   - Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID        "DEMO_AP" /* < Destination SSID */
#define MAIN_WLAN_AUTH        M2M_WIFI_SEC_WPA_PSK /* < Security type */
#define MAIN_WLAN_PSK         "12345678" /* < Password for Destination SSID */
```

- Request to scan in all channels.

```
/* Request scan. */
    m2m_wifi_request_scan(M2M_WIFI_CH_ALL);
```

- The `wifi_cb()` function is called with the `M2M_WIFI_RESP_SCAN_DONE` message when scanning is done. At this point, a specific scan result can be requested by calling the `m2m_wifi_req_scan_result()` API with a specific index.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_RESP_SCAN_DONE:
    {
    tstrM2mScanDone *pstrInfo = (tstrM2mScanDone *)pvMsg;
    scan_request_index = 0;
    if (pstrInfo->u8NumofCh >= 1) {
        m2m_wifi_req_scan_result(scan_request_index);
        scan_request_index++;
    }
```

- The `wifi_cb()` function will be called again with the `M2M_WIFI_RESP_SCAN_RESULT` message. The application will get the AP information for the specific result index requested. If the scan result is the same as the AP information in `main.h`, then the device will connect to the AP.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_RESP_SCAN_RESULT:
    {
    tstrM2mWifiscanResult *pstrScanResult = (tstrM2mWifiscanResult *)pvMsg;
    uint16_t demo_ssid_len;
    uint16_t scan_ssid_len = strlen((const char *)pstrScanResult->au8SSID);
    /* display AP found. */
    printf("[%d] SSID:%s\r\n", scan_request_index, pstrScanResult->au8SSID);
    num_found_ap = m2m_wifi_get_num_found_ap();
    if (scan_ssid_len) {
    /* check same SSID. */
        demo_ssid_len = strlen((const char *)MAIN_WLAN_SSID);
        if((demo_ssid_len == scan_ssid_len) &&
        (!memcmp(pstrScanResult->au8SSID, (uint8_t *)MAIN_WLAN_SSID,
demo_ssid_len))) {
            printf("Found %s \r\n", MAIN_WLAN_SSID);
            m2m_wifi_connect((char *)MAIN_WLAN_SSID,
                        sizeof(MAIN_WLAN_SSID),
                        MAIN_WLAN_AUTH,
                        (void *)MAIN_WLAN_PSK,
                        M2M_WIFI_CH_ALL);
```

2. Build the program and download it into the board.
3. Start the application.

3.1.7.2 Scan APs Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 AP scan example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
[1] SSID:DEMO_AP1
[2] SSID:DEMO_AP2
[3] SSID:DEMO_AP
Found DEMO_AP
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
```

3.1.8 How to Set Power Save Mode

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to check the PS (Power Save) mode.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1

Figure 3-8. Power Save Mode Demo Setup

3.1.8.1 Execution

`main.c` - Initialize the ATWINC15x0, set PS Mode and get the RSSI for the connected AP.

1. Code summary:
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID         "DEMO_AP" /* < Destination SSID */
#define MAIN_WLAN_AUTH         M2M_WIFI_SEC_WPA_PSK /* < Security manner */
#define MAIN_WLAN_PSK          "12345678" /* < Password for Destination SSID */
```

   – Configure the Power Save parameters in `main.h`.

```
/** PowerSave mode Settings */
#define MAIN_PS_SLEEP_MODE          M2M_PS_MANUAL /* M2M_NO_PS /
M2M_PS_DEEP_AUTOMATIC / M2M_PS_MANUAL */
```

   – In the `main()` function, set the Power Save mode, based on the define above.

```
/* Set defined sleep mode */
    if (MAIN_PS_SLEEP_MODE == M2M_PS_MANUAL) {
        printf("M2M_PS_MANUAL\r\n");
        m2m_wifi_set_sleep_mode(MAIN_PS_SLEEP_MODE, 1);
    } else if (MAIN_PS_SLEEP_MODE == M2M_PS_DEEP_AUTOMATIC) {
        printf("M2M_PS_DEEP_AUTOMATIC\r\n");
        tstrM2mLsnInt strM2mLsnInt;
        m2m_wifi_set_sleep_mode(M2M_PS_DEEP_AUTOMATIC, 1);
        strM2mLsnInt.u16LsnInt = M2M_LISTEN_INTERVAL;
        m2m_wifi_set_lsn_int(&strM2mLsnInt);
    }
```

   – Connect to the AP.

```
/* Connect to defined AP. */
    m2m_wifi_connect((char *)MAIN_WLAN_SSID, sizeof(MAIN_WLAN_SSID),
                MAIN_WLAN_AUTH, (void *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);
```

   – The ATWINC15x0 goes to Sleep automatically if Power Save is configured for the `M2M_PS_DEEP_AUTOMATIC` mode. The ATWINC15x0 will wake up upon any request/callback (Wi-Fi/SOCKET) and the host driver will allow the SoC to sleep again after handling the request. However, if Power Save is configured for the `M2M_PS_MANUAL` mode, then the application is responsible for explicitly requesting the ATWINC15x0 to enter sleep while specifying the expected sleep time.

```
/* Request sleep mode */
if (gu8SleepStatus == MAIN_PS_REQ_SLEEP) {
    if (MAIN_PS_SLEEP_MODE == M2M_PS_MANUAL) {
        m2m_wifi_request_sleep(MAIN_REQUEST_SLEEP_TIME);
        gu8SleepStatus = MAIN_PS_SLEEP;
    }
}
```

2. Build the program and download it into the board.
3. Start the application.

3.1.8.2 Power Save Mode Demo Console Log

The application must be programmed and running. The following information will be displayed on the terminal window.

```
-- WINC1500 Power Save Mode example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
RSSI for the current connected AP (-xx)
```

3.1.9 HTTP Provision Mode

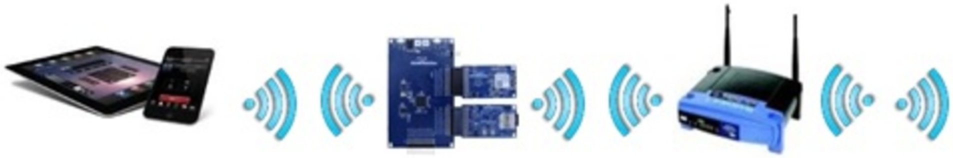
This example demonstrates the use of the ATWINC15x0 with the SAM D21 Xplained Pro board to start Provision Mode.

Note: This example application is not available in ASF for ATWINC3400

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header

Figure 3-9. HTTP Provision Mode Demo Setup



3.1.9.1 Execution

`main.c` - Initialize the ATWINC15x0 and start Provision Mode until one of the various APs is selected.

1. Code summary:
   – Configure the network parameters in `main.h`.

```
#define MAIN_HTTP_PROV_SERVER_DOMAIN_NAME    "atmelconfig.com"
    #define MAIN_M2M_DEVICE_NAME                 "WINC1500_00:00"
```

   – Start provision mode before entering the main loop.

```
m2m_wifi_start_provision_mode((tstrM2MAPConfig *)&gstrM2MAPConfig, (char
*)gacHttpProvDomainName, 1);
```

   – When your mobile device sends configuration information, the `wifi_cb()` function will be called with the `M2M_WIFI_RESP_PROVISION_INFO` message and you can connect to the AP with the given information.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_RESP_PROVISION_INFO:
    {
    tstrM2MProvisionInfo *pstrProvInfo = (tstrM2MProvisionInfo *)pvMsg;
    printf("wifi_cb: M2M_WIFI_RESP_PROVISION_INFO.\r\n");
    if (pstrProvInfo->u8Status == M2M_SUCCESS) {
        m2m_wifi_connect((char *)pstrProvInfo->au8SSID, strlen((char *)pstrProvInfo-
>au8SSID), pstrProvInfo->u8SecType,
        pstrProvInfo->au8Password, M2M_WIFI_CH_ALL);
```

2. Build the program and download it into the board.

3. Start the application.

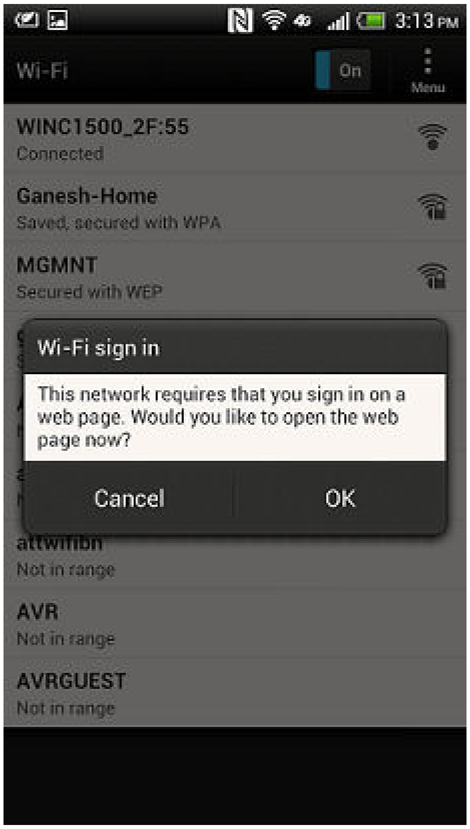
3.1.9.2 HTTP Provision Mode Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 HTTP provision example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Provision Mode started.
Connect to [atmelconfig.com] via AP[WINC1500_2F:55] and fill up the page.
```

1. Connect your mobile device to ATWINC15x0 (ATWINC1500_xx:xx).

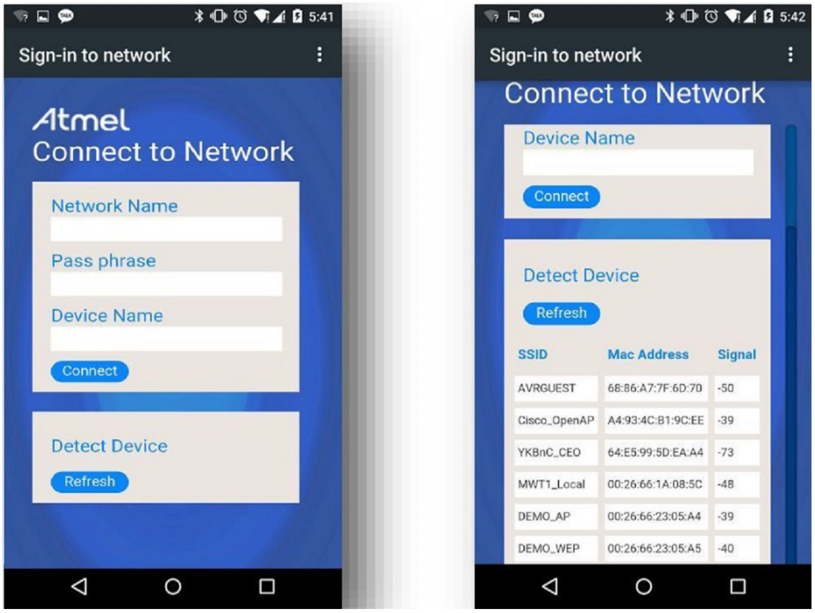
Figure 3-10. ATWINC1500 Provision AP



This network requires that you sign in on a web page. Would you like to open the web page now?

2. Browse to (www.microchip.com) to setup the AP, populate the page, and then press Connect.

Figure 3-11. HTTP Provisioning Web Page



SSID	Mac Address	Signal
AVRGUEST	68:86:A7:7F:6D:70	-50
Cisco_OpenAP	A4:93:4C:B1:9C:EE	-39
YKBnC_CEO	64:E5:99:5D:EA:A4	-73
MWT1_Local	00:26:66:1A:08:5C	-48
DEMO_AP	00:26:66:23:05:A4	-39
DEMO_WEP	00:26:66:23:05:A5	-40

3. If configured correctly, the ATWINC15x0 will connect to the specified AP.

```
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED: CONNECTED.
wifi_cb: M2M_WIFI_REQ_DHCP_CONF: IP is xxx.xxx.xxx.xxx
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED: DISCONNECTED.
wifi_cb: M2M_WIFI_RESP_PROVISION_INFO.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED: CONNECTED.
wifi_cb: M2M_WIFI_REQ_DHCP_CONF: IP is xxx.xxx.xxx.xxx
```

Note: Refer to the HTTP Provision Mode application note for more details.

3.1.10 AP Provision Mode

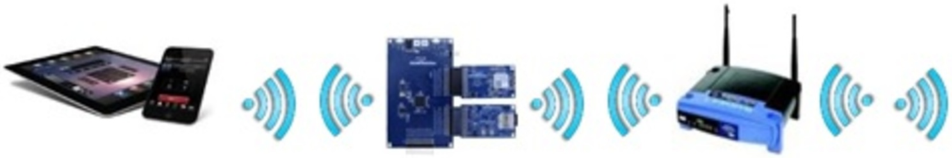
This example demonstrates the use of the ATWINC15x0 with the SAM D21 Xplained Pro board to start Provision mode.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header

Figure 3-12. AP Provision Demo Setup



3.1.10.1 Execution

`main.c` - Initialize the ATWINC15x0 and start Provision mode until one of various APs is selected.

1. Code summary:
   – Initialize the socket module and create the TCP server socket.

```
/* Initialize socket address structure. */
addr.sin_family = AF_INET;
addr.sin_port = _htons((MAIN_WIFI_M2M_SERVER_PORT));
addr.sin_addr.s_addr = 0;
/* Initialize Socket module */
socketInit();
```

```
registerSocketCallback(socket_cb, NULL);
while (1) {
    m2m_wifi_handle_events(NULL);
    if (tcp_server_socket < 0) {
    /* Open TCP server socket */
    if ((tcp_server_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    /* Bind service*/
    bind(tcp_server_socket, (struct sockaddr *)&addr, sizeof(struct sockaddr_in));
```

– Enable AP mode before the main loop. (Refer to the “How to Run AP Mode” example.)

```
/* Initialize AP mode parameters structure with SSID, channel and OPEN security
type. */
    memset(&strM2MAPConfig, 0x00, sizeof(tstrM2MAPConfig));
    strcpy((char *)&strM2MAPConfig.au8SSID, MAIN_WLAN_SSID);
    strM2MAPConfig.u8ListenChannel = MAIN_WLAN_CHANNEL;
    strM2MAPConfig.u8SecType = MAIN_WLAN_AUTH;
    strM2MAPConfig.au8DHCPServerIP[0] = 0xC0; /* 192 */
    strM2MAPConfig.au8DHCPServerIP[1] = 0xA8; /* 168 */
    strM2MAPConfig.au8DHCPServerIP[2] = 0x01; /* 1 */
    strM2MAPConfig.au8DHCPServerIP[3] = 0x01; /* 1 */

    /* Bring up AP mode with parameters structure. */
    ret = m2m_wifi_enable_ap(&strM2MAPConfig);
```

– After your Android device is connected to the ATWINC15x0 and sends the AP configuration, disable AP mode and connect to the AP with the given information.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    case SOCKET_MSG_RECV:
    {
        printf("Disable to AP.\r\n");
     m2m_wifi_disable_ap();
     nm_bsp_sleep(500);
     printf("Connecting to %s.\r\n", (char *)str_ssid);
     m2m_wifi_connect((char *)str_ssid, strlen((char *)str_ssid), sec_type, str_pw,
M2M_WIFI_CH_ALL);
```

– The `wifi_cb()` function is called with the `M2M_WIFI_REQ_DHCP_CONF` message and then receives an IP address.

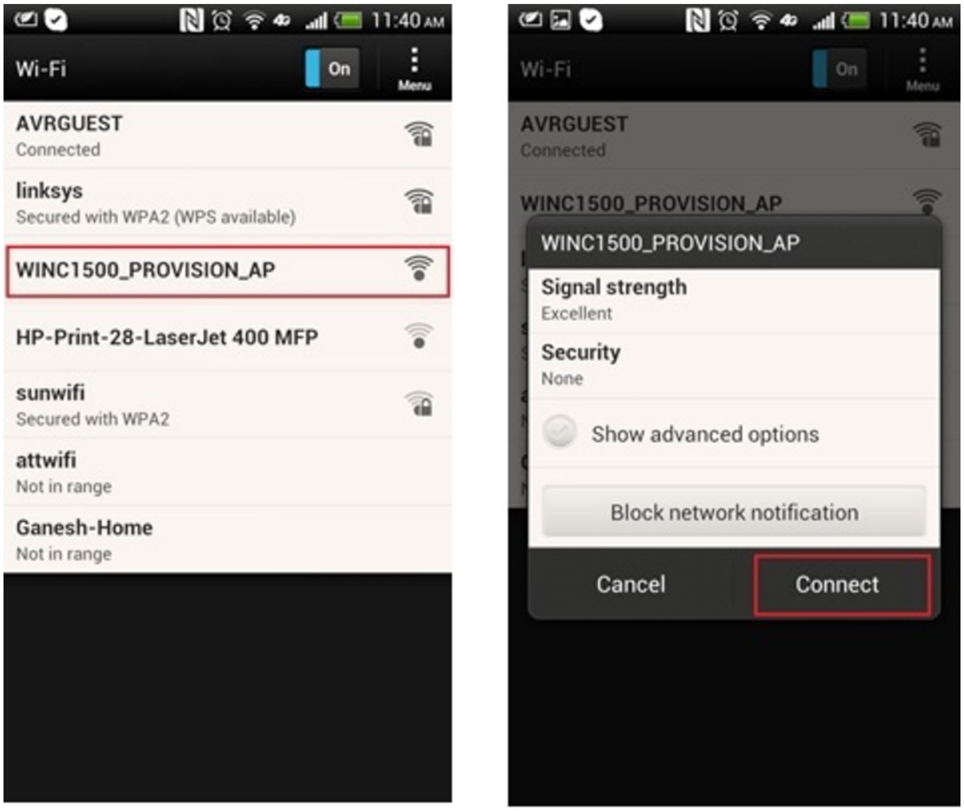
```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_REQ_DHCP_CONF:
    {
    uint8_t *pu8IPAddress = (uint8_t *)pvMsg;
    printf("Wi-Fi connected\r\n");
    printf("Wi-Fi IP is %u.%u.%u.%u\r\n",
            pu8IPAddress[0], pu8IPAddress[1], pu8IPAddress[2], pu8IPAddress[3]);
```

2. Build the program and download it into the board.
3. Start the application.

3.1.10.2 Android App Connection Process

1. Install *provision_ap.apk* in your Android device. You can also build the Android application source and install it.
2. Connect your Android device to the ATWINC15x0.

Figure 3-13. ATWINC1500 AP Connection



AVRGUEST
Connected

linksys
Secured with WPA2 (WPS available)

WINC1500_PROVISION_AP

HP-Print-28-LaserJet 400 MFP

sunwifi
Secured with WPA2

attwifi
Not in range

Ganesh-Home
Not in range

WINC1500_PROVISION_AP

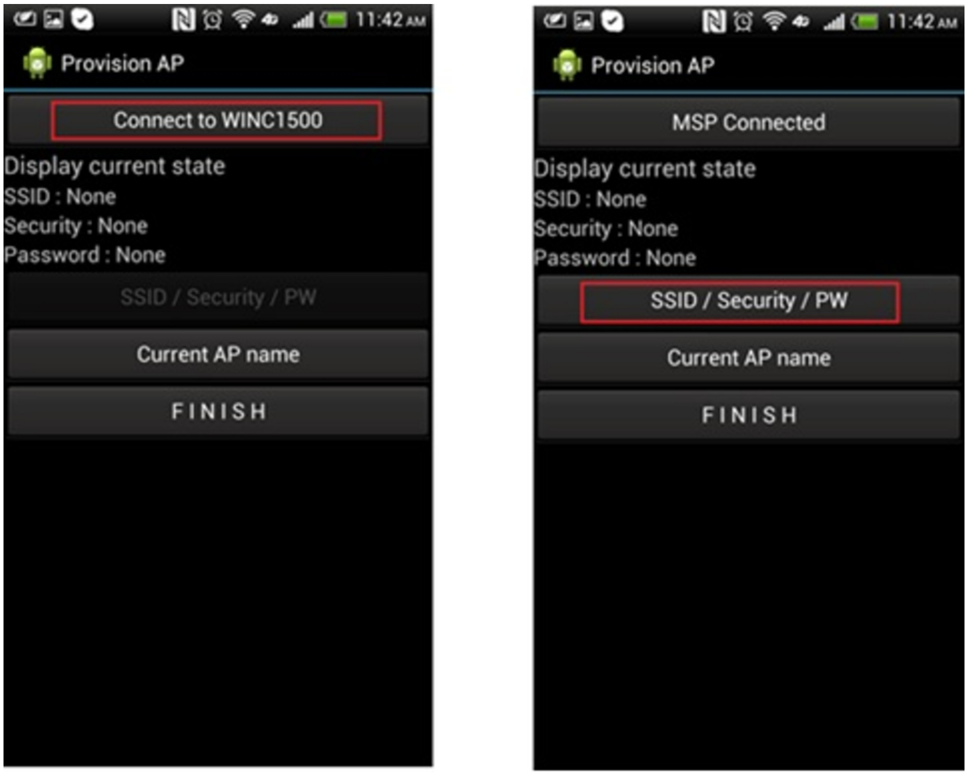
Signal strength
Excellent

Security
None

○ Show advanced options

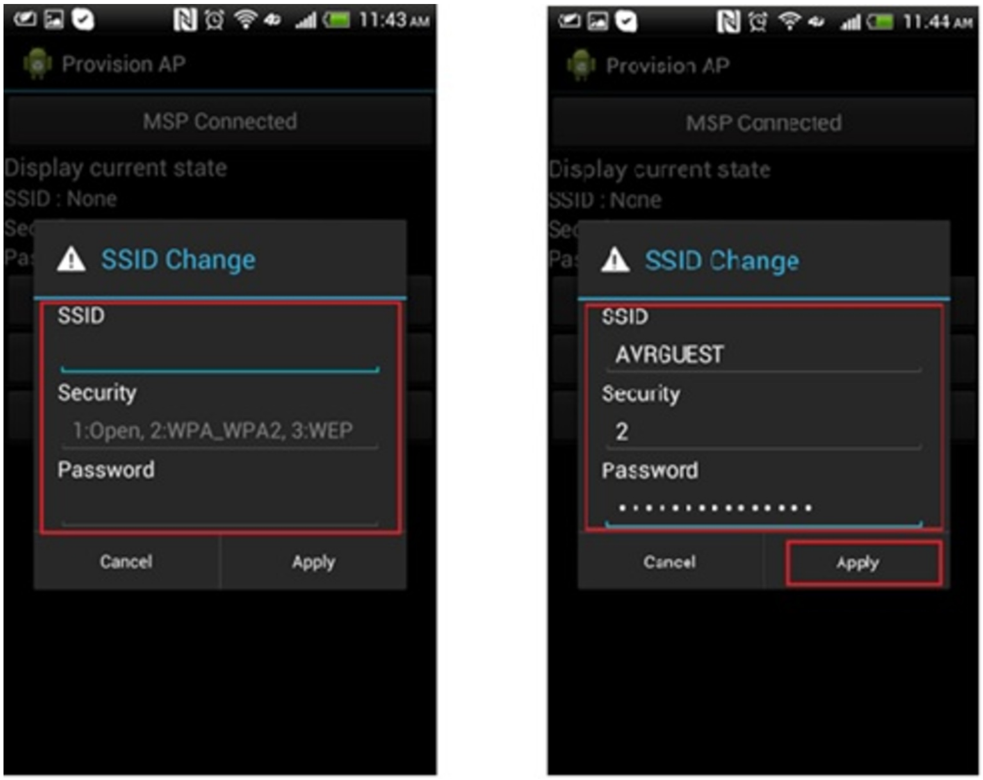
3. Launch the Android application to configure AP, press the Connect button, and the SSID button will then be available.

Figure 3-14. Connect to TCP Server and Enter Credentials



4. Input the connection information, and then press the Apply button.

Figure 3-15. Entering AP Credentials



SSID

Security

1:Open, 2:WPA_WPA2, 3:WEP

Password

SSID

AVRGUEST

Security

2

Password

•••••••••••••••

5. If configured correctly, the ATWINC15x0 will connect to the specified AP.

3.1.10.3 AP Provision Mode Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 AP provision example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
AP Provision mode started.
On the android device, connect to WINC1500_PROVISION_AP then run setting app.
(APP)(INFO)Socket 0 session ID = 1
socket_cb: Ready to listen.
Wi-Fi connected. IP is xxx.xxx.xxx.xxx
socket_cb: Client socket is created.
Disable to AP
Connecting to XXXXXX.
wifi_cb: DISCONNECTED
wifi_cb: CONNECTED
Wi-Fi connected. IP is xxx.xxx.xxx.xxx
```

Note: Refer to the AP Provision Mode application note for more details.

3.1.11 Connection to WPS Security

This example demonstrates how to connect the ATWINC15x0 Wi-Fi device to an AP with WPS Security with the SAM D21 Xplained Pro board as the host MCU.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1

Figure 3-16. Demo Setup

3.1.11.1 Execution

`main.c` - Initialize the ATWINC15x0 and connect to the AP using WPS.

1. Code summary:
   Case 1: Push Button Method
   - To test the WPS button method, configure the WPS push button feature in `main.h` as below and use case 1 in the `main()` function.

```
/** WPS Push Button Feature */
#define MAIN_WPS_PUSH_BUTTON_FEATURE     true
```

```
/* Device name must be set before enabling WPS mode. */
m2m_wifi_set_device_name((uint8 *)devName, strlen(devName));
if (MAIN_WPS_PUSH_BUTTON_FEATURE) {
/* case 1 WPS Push Button method. */
    if (!gbPressButton){
        btn_init();
    }
}
```

- When pressing the SW0 button on the SAMD21, it will trigger WPS in the `btn_press()` function.

```
/* Connect to defined AP. */
m2m_wifi_connect((char *)MAIN_WLAN_SSID, sizeof(MAIN_WLAN_SSID),
                MAIN_WLAN_AUTH, (void *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);
```

- The `wifi_cb()` will receive the `M2M_WIFI_REQ_WPS` message and it can connect to the AP with the given information.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_REQ_WPS:
    {
    tstrM2MWPSInfo *pstrWPS = (tstrM2MWPSInfo *)pvMsg;
    printf("Wi-Fi request WPS\r\n");
    printf("SSID : %s, authtyp : %d pw : %s\n", pstrWPS->au8SSID, pstrWPS-
>u8AuthType, pstrWPS->au8PSK);
    if (pstrWPS->u8AuthType == 0) {
        printf("WPS is not enabled OR Timedout\r\n");
        m2m_wifi_request_scan(M2M_WIFI_CH_ALL);
        /* WPS is not enabled by firmware OR WPS monitor timeout.*/
    } else{
        printf("Request Wi-Fi connect\r\n");
        m2m_wifi_connect((char *)pstrWPS->au8SSID, (uint8)m2m_strlen(pstrWPS-
>au8SSID),
                    pstrWPS->u8AuthType, pstrWPS->au8PSK, pstrWPS->u8Ch);
    }
```

Case 2: PIN Method

- To test the WPS PIN method, configure the WPS PIN number and WPS push button feature in `main.h` as below and use case 2 in the `main()` function.

```
/** WPS PIN number */
#define MAIN_WPS_PIN_NUMBER              "12345670"
/** WPS Push Button Feature */
#define MAIN_WPS_PUSH_BUTTON_FEATURE     false
```

```
/* Device name must be set before enabling WPS mode. */
m2m_wifi_set_device_name((uint8 *)devName, strlen(devName));
if (!MAIN_WPS_PUSH_BUTTON_FEATURE) {
    /* case 2 WPS PIN method */
    m2m_wifi_wps(WPS_PIN_TRIGGER, (const char *)MAIN_WPS_PIN_NUMBER);
}
```

- When pressing the SW0 button on the SAMD21, it will trigger WPS in the `btn_press()` function.

```
/* Connect to defined AP. */
m2m_wifi_connect((char *)MAIN_WLAN_SSID, sizeof(MAIN_WLAN_SSID),
                MAIN_WLAN_AUTH, (void *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);
```

- The `wifi_cb()` will receive the `M2M_WIFI_REQ_WPS` message and it can connect to the AP with the given information.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_REQ_WPS:
    {
    tstrM2MWPSInfo *pstrWPS = (tstrM2MWPSInfo *)pvMsg;
    printf("Wi-Fi request WPS\r\n");
    printf("SSID : %s, authtyp : %d pw : %s\n", pstrWPS->au8SSID, pstrWPS-
>u8AuthType, pstrWPS->au8PSK);
    if (pstrWPS->u8AuthType == 0) {
        printf("WPS is not enabled OR Timedout\r\n");
        m2m_wifi_request_scan(M2M_WIFI_CH_ALL);
        /* WPS is not enabled by firmware OR WPS monitor timeout.*/
    } else{
        printf("Request Wi-Fi connect\r\n");
        m2m_wifi_connect((char *)pstrWPS->au8SSID, (uint8)m2m_strlen(pstrWPS-
>au8SSID),
                    pstrWPS->u8AuthType, pstrWPS->au8PSK, pstrWPS->u8Ch);
    }
```

2. Prepare an AP that supports Wi-Fi Protected Setup (WPS).

3. Press the WPS button on the AP when using the WPS button method or enter the WPS PIN number in the AP setup menu and start the AP. (For more information, refer to the AP product documentation.)
4. Run the application. Press the SW0 button on the SAM D21 when using the WPS button method. The ATWINC15x0 will be connected to the AP automatically without security information.
5. Build the program and download it into the board.
6. Start the application.

3.1.11.2 WPS Mode Demo Console Log

Note: In the WPS button method, the following information will be displayed on the terminal window.

```
-- WINC1500 security connection with Wi-Fi Protected Setup(WPS) example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
SW0 button pressed
Device is connecting using WPS Push Button option
Wi-Fi request WPS
SSID : xxxxxx, AuthType : x, PW : xxxxxxxx
Request Wi-Fi connect
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
```

Note: In the WPS PIN method, the following information will be displayed on the terminal window.

```
-- WINC1500 security connection with Wi-Fi Protected Setup(WPS) example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Wi-Fi request WPS
SSID : xxxxxx, AuthType : x, PW : xxxxxxxx
Request Wi-Fi connect
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
```

3.1.12 Security with WEP/WPA

This example demonstrates how to connect the ATWINC15x0 Wi-Fi device to an AP with WEP or WPA security using the SAM D21 Xplained Pro board as the host MCU.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on EXT1 header

Figure 3-17. Demo Setup

3.1.12.1 Execution

`main.c` - Initialize the ATWINC15x0 and connect to the AP using WPS.

1. Code summary:
   Case 1: WEP Security Method

- To test WEP security, modify `MAIN_WLAN_DEVICE_NAME`, `MAIN_WLAN_WEP_KEY_INDEX` and `MAIN_WLAN_WEP_KEY_40` or `MAIN_WLAN_WEP_KEY_104` in `main.h`.

```
/** security information for Wi-Fi connection */
#define MAIN_WLAN_DEVICE_NAME           "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_WEP_KEY_INDEX         1 /**< WEP key index */
/**< 64 bit WEP key. In case of WEP64, 10 hexadecimal (base 16) characters (0-9 and
A-F) ) */
#define MAIN_WLAN_WEP_KEY_40            "1234567890"
/**< 128 bit WEP key. In case of WEP128, 26 hexadecimal (base 16) characters (0-9
and A-F) ) */
#define MAIN_WLAN_WEP_KEY_104           "1234567890abcdef1234567890"
```

- When pressing the SW0 button on the SAM D21, it will trigger WPS in the `btn_press()` function.

```
/** Security parameters for 64 bit WEP Encryption @ref m2m_wifi_connect */
tstrM2mWifiWepParams wep64_parameters = { MAIN_WLAN_WEP_KEY_INDEX,
                                         sizeof(MAIN_WLAN_WEP_KEY_40),
                                        MAIN_WLAN_WEP_KEY_40};
/** Security parameters for 128 bit WEP Encryption @ref m2m_wifi_connect */
tstrM2mWifiWepParams wep128_parameters = { MAIN_WLAN_WEP_KEY_INDEX,
                                          sizeof(MAIN_WLAN_WEP_KEY_104),
                                          MAIN_WLAN_WEP_KEY_104};
```

```
/* Case 1. Connect to AP with security type WEP. */
m2m_wifi_connect((char *)MAIN_WLAN_DEVICE_NAME, strlen((char
*)MAIN_WLAN_DEVICE_NAME),
     M2M_WIFI_SEC_WEP, &wep64_parameters, M2M_WIFI_CH_ALL);
```

Case 2: WPA-PSK Security Method

- To test WPA security, use case 2 in the `main()` function and modify `MAIN_WLAN_PSK` in `main.h`.

```
#define MAIN_WLAN_PSK     "12345678" /**< Password for Destination SSID */
```

- Connect to the AP with the given information.

```
/* Case 2. Connect to AP with security type WPA. */
m2m_wifi_connect((char *)MAIN_WLAN_DEVICE_NAME, strlen((char
*)MAIN_WLAN_DEVICE_NAME),
     M2M_WIFI_SEC_WPA_PSK, (char *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);
```

2. Prepare an AP that supports WEP and WPA/WPA2 Security and configure its Wi-Fi Security. For more information, refer to the AP manufacturer's manual.
3. Run the application. If the device connected successfully, the IP address assigned by DHCP will be displayed on the terminal program.

3.1.12.2 WEP/WPA Security Mode Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 security connection with WEP,WPA security example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to XXXXXX.
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
```

3.1.13 Connection to an Enterprise Security Network

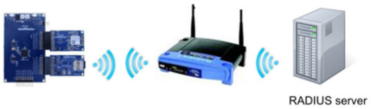
This example demonstrates how to connect the ATWINC15x0 Wi-Fi device to an AP with WPA/WPA2 enterprise security with the SAM D21 Xplained board as the host MCU.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header

Figure 3-18. Enterprise Mode Demo Setup



3.1.13.1 Execution

`main.c` - Initialize the ATWINC15x0 and connect to the AP with WPA/WPA2 enterprise security.

1. Code summary:
   – Modify `MAIN_WLAN_802_1X_USR_NAME`, `MAIN_WLAN_802_1X_PWD` to hold the appropriate username and password, respectively. Modify `MAIN_WLAN_DEVICE_NAME` to hold the desired wireless network name.

```
/** security information for Wi-Fi connection */
#define MAIN_WLAN_DEVICE_NAME           "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_802_1X_USR_NAME       "atmeluser" /**< RADIUS user account name */
#define MAIN_WLAN_802_1X_PWD            "12345678" /**< RADIUS user account password
*/
```

   – Connect to the AP with the given information.

```
/* Connect to the enterprise network. */
m2m_wifi_connect((char *)MAIN_WLAN_DEVICE_NAME, sizeof(MAIN_WLAN_DEVICE_NAME),
     M2M_WIFI_SEC_802_1X, (char *)&gstrCred1x, M2M_WIFI_CH_ALL);
```

   – The `wifi_cb()` function is called with the `M2M_WIFI_RESP_CON_STATE_CHANGED` message, where an appropriate action can be executed as a response. The new Wi-Fi state is passed onto the callback as part of the message.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    switch (u8MsgType) {
    case M2M_WIFI_RESP_CON_STATE_CHANGED:
    {
    tstrM2mWifiStateChanged *pstrWifiState = (tstrM2mWifiStateChanged *)pvMsg;
    if (pstrWifiState->u8CurrState == M2M_WIFI_CONNECTED) {
       /* Take action on connection successful */
```

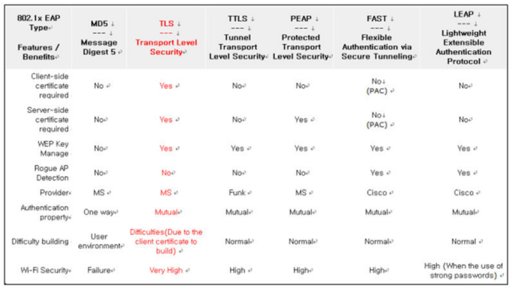
   – Once the DHCP negotiation completes, the `wifi_cb()` function is called with the `M2M_WIFI_REQ_DHCP_CONF` message.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_REQ_DHCP_CONF:
    {
    uint8_t *pu8IPAddress = (uint8_t *)pvMsg;
    printf("Wi-Fi connected\r\n");
    printf("Wi-Fi IP is %u.%u.%u.%u\r\n",
            pu8IPAddress[0], pu8IPAddress[1], pu8IPAddress[2], pu8IPAddress[3]);
```

2. Prepare an AP that supports WPA/WPA2 enterprise security. Before configuring the RADIUS server settings in the AP, you will need to know certain parameters. Ask your network administrator for this information and configure it in the AP.
   – Username
   – Password
   – Name of wireless network
   – Root certificate file
3. Download the root certificate generated from the previous step to the ATWINC15x0/ATWINC3400 using the *RootCertDownload.bat* file.
4. Build and run the application. If the device has connected successfully, the IP address assigned by DHCP will be displayed on the terminal program.

Note: To connect to the enterprise security network, the root certificate must be installed.

Figure 3-19. Supported 802.1x EAP (Extensible Authentication Protocol)



802.1x EAP Type Features / Benefits	MD5 --- Message Digest 5	TLS --- Transport Level Security	TTLS --- Tunnel Transport Level Security	PEAP --- Protected Transport Level Security	FAST --- Flexible Authentication via Secure Tunneling	LEAP --- Lightweight Extensible Authentication Protocol
Client-side certificate required	No	Yes	No	No	No (PAC)	No
Server-side certificate required	No	Yes	No	Yes	No (PAC)	No
WEP Key Manage	No	Yes	Yes	Yes	Yes	Yes
Rogue AP Detection	No	No	No	No	Yes	Yes
Provider	MS	MS	Funk	MS	Cisco	Cisco
Authentication property	One way	Mutual	Mutual	Mutual	Mutual	Mutual
Difficulty building	User environment	Difficulties(Due to the client certificate to build)	Normal	Normal	Normal	Normal
Wi-Fi Security	Failure	Very High	High	High	High	High (When the use of strong passwords)

3.1.13.2 Enterprise Security Network Mode Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 security connection with WPA/WPA2 enterprise example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
Connection successfully completed.
```

4. Protocol Examples

This chapter describes protocol examples in detail. These protocol examples can also be used for IoT applications.

- UDP protocol example:
  - Client
  - Server
- TCP protocol example:
  - Client
  - Server
- NTP time client – retrieves the network time for the IoT application
- Send email – send an email from an SMTP server

4.1 UDP Client

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to test the UDP socket.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 4-1. UDP Client Demo Setup

4.1.1 Execution

`main.c` - Initialize the Wi-Fi module and test with the UDP server.

1. Code summary.
   - Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                        "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                        M2M_WIFI_SEC_WPA_PSK /**< Security manner
*/
#define MAIN_WLAN_PSK                         "12345678" /**< Password for Destination
SSID */
#define MAIN_WIFI_M2M_PRODUCT_NAME            "NMCTemp"
#define MAIN_WIFI_M2M_SERVER_IP               0xFFFFFFFF /* 255.255.255.255 */
#define MAIN_WIFI_M2M_SERVER_PORT             (6666)
#define MAIN_WIFI_M2M_REPORT_INTERVAL         (1000)
```

   - Initialize the socket module.

```
/* 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 socket module */
socketInit();
```

   - Connect to the AP.

```
/* 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);
```

   - After the device is connected to the AP, create a TX socket in the main loop.

```
/* Create socket for Tx UDP */
if (tx_socket < 0) {
    if ((tx_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    printf("main : failed to create TX UDP client socket error!\r\n");
    continue;
    }
}
```

– Send data from the UDP client TX socket to the UDP server RX socket.

```
ret = sendto(tx_socket, &msg_wifi_product_main, sizeof(t_msg_wifi_product_main),
     0, (struct sockaddr *)&addr, sizeof(addr));
```

2. Build the program and download it into the board.
3. Start the application.

4.1.2 UDP Client Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 UDP Client example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxx
main: message sent
. . .
main: message sent
UDP client test Complete!
```

4.2 UDP Server

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to test the UDP socket.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 4-2. UDP Server Demo Setup

4.2.1 Execution

`main.c` - Initialize the Wi-Fi module and test with the UDP client.

1. Code summary.
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                      "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                      M2M_WIFI_SEC_WPA_PSK /**< Security manner
*/
#define MAIN_WLAN_PSK                       "12345678" /**< Password for Destination
SSID */
#define MAIN_WIFI_M2M_PRODUCT_NAME          "NMCTemp"
#define MAIN_WIFI_M2M_SERVER_IP             0xFFFFFFFF /* 255.255.255.255 */
#define MAIN_WIFI_M2M_SERVER_PORT           (6666)
#define MAIN_WIFI_M2M_REPORT_INTERVAL       (1000)
```

- Initialize the socket module and create the UDP server socket.

```
/* 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 socket module */
socketInit();
registerSocketCallback(socket_cb, NULL);
```

- Connect to the AP.

```
/* 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);
```

- After the device is connected to the AP, create an RX socket and bind it in the main loop.

```

if (rx_socket < 0) {
    if ((rx_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        printf("main : failed to create RX UDP Client socket error!\r\n");
        continue;
    }
    /* Socket bind */
    bind(rx_socket, (struct sockaddr *)&addr, sizeof(struct sockaddr_in));
}
```

- In the `socket_cb()` function, prepare a buffer to receive data.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    if (u8Msg == SOCKET_MSG_BIND) {
    recvfrom(sock, gau8SocketTestBuffer, MAIN_WIFI_M2M_BUFFER_SIZE, 0);
```

- You can receive data in the `socket_cb()` function with the `SOCKET_MSG_RECVFROM` message when a client device sends data. (Use “UDP Client” example.)

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    ...
    } else if (u8Msg == SOCKET_MSG_RECVFROM) {
    tstrSocketRecvMsg *pstrRx = (tstrSocketRecvMsg *)pvMsg;
    if (pstrRx->pu8Buffer && pstrRx->s16BufferSize) {
        printf("socket_cb: received app message.(%u)\r\n", packetCnt);
    /* Prepare next buffer reception. */
    recvfrom(sock, gau8SocketTestBuffer, MAIN_WIFI_M2M_BUFFER_SIZE, 0);
```

2. Build the program and download it into the board.
3. Start the application.

4.2.2 UDP Server Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 UDP Server example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxx
socket_cb: bind success!
socket_cb: received app message.(1)
. . .
socket_cb: received app message.(10)
UDP server test Complete!
```

4.3 TCP Client

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to test the TCP client.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 4-3. TCP Client Demo Setup

4.3.1 Execution

`main.c` – Initialize the Wi-Fi module and test the TCP client.

1. Code summary:
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                      "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                      M2M_WIFI_SEC_WPA_PSK /**< Security manner
*/
#define MAIN_WLAN_PSK                       "12345678" /**< Password for Destination
SSID */
#define MAIN_WIFI_M2M_PRODUCT_NAME          "NMCTemp"
#define MAIN_WIFI_M2M_SERVER_IP             0xc0a80164 //0xFFFFFFFF /* 255.255.255.255
*/
#define MAIN_WIFI_M2M_SERVER_PORT           (6666)
#define MAIN_WIFI_M2M_REPORT_INTERVAL       (1000)
```

   – Initialize the socket module and register the socket callback function.

```
/* 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 socket module */
socketInit();
registerSocketCallback(socket_cb, NULL);
```

   Note: The Server IP address in this example is 0xc0a80164, which translates to 192.168.1.100.

   – Connect to the AP.

```
/* 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);
```

   – After the device is connected to the AP, create a TCP client socket and connect to the server in the main loop.

```
/* Open client socket. */
if (tcp_client_socket < 0) {
    if ((tcp_client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("main: failed to create TCP client socket error!\r\n");
    continue;
}
/* Connect server */
ret = connect(tcp_client_socket, (struct sockaddr *)&addr, sizeof(struct
sockaddr_in));
```

– Requests to connect, send and recv can be executed sequentially in the `socket_cb()` function. Also, while using the send operation, the maximum data that can be transmitted using `socket API send()` is 1400 bytes.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
...
case SOCKET_MSG_CONNECT:
    {
        if (pstrConnect && pstrConnect->s8Error >= 0)
            send(tcp_client_socket, &msg_wifi_product, ...);
    }
    ...
    case SOCKET_MSG_SEND:
    {
        recv(tcp_client_socket, gau8SocketTestBuffer, ...);
    }
    ...
    case SOCKET_MSG_RECV:
    {
        tstrSocketRecvMsg *pstrRecv = (tstrSocketRecvMsg *)pvMsg;
        if (pstrRecv && pstrRecv->s16BufferSize > 0) {
            printf("socket_cb: recv success!\r\n");
            printf("TCP Client Test Complete!\r\n");
        }
    }
```

2. Build the program and download it into the board.
3. Start the application.

4.3.2 TCP Client Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 TCP client example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxx
socket_cb: connect success!
socket_cb: send success!
socket_cb: recv success!
TCP Client Test Complete!
```

4.4 TCP Server

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to test the TCP server.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 4-4. TCP Server Demo Setup

4.4.1 Execution

main.c – Initialize the Wi-Fi module and test the TCP server.

1. Code summary.
   – Configure the network parameters in main.h.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                      "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                      M2M_WIFI_SEC_WPA_PSK /**< Security manner
*/
#define MAIN_WLAN_PSK                       "12345678" /**< Password for Destination
SSID */
#define MAIN_WIFI_M2M_PRODUCT_NAME          "NMCTemp"
#define MAIN_WIFI_M2M_SERVER_IP             0xFFFFFFFF /* 255.255.255.255 */
#define MAIN_WIFI_M2M_SERVER_PORT           (6666)
#define MAIN_WIFI_M2M_REPORT_INTERVAL       (1000)
```

   – Initialize the socket module and register the socket callback function.

```
/* 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 socket module */
socketInit();
registerSocketCallback(socket_cb, NULL);
```

   – Connect to the AP.

```
/* 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);
```

   – After the device is connected to the AP, create a TCP server socket and bind it in the main loop.

```
if (tcp_server_socket < 0) {
/* Open TCP server socket */
    if ((tcp_server_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("main: failed to create TCP server socket error!\r\n");
        continue;
    }
/* Bind service*/
    bind(tcp_server_socket, (struct sockaddr *)&addr, sizeof(struct sockaddr_in));
}
```

   – Five operations (bind/listen/accept/recv/send) must be executed sequentially in the socket_cb() function.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    ...
    case SOCKET_MSG_BIND:
    {
        tstrSocketBindMsg *pstrBind = (tstrSocketBindMsg *)pvMsg;
        if (pstrBind && pstrBind->status == 0)
            listen(tcp_server_socket, 0);
    }
    case SOCKET_MSG_LISTEN:
    {
    tstrSocketListenMsg *pstrListen = (tstrSocketListenMsg *)pvMsg;
    if (pstrListen && pstrListen->status == 0)
        accept(tcp_server_socket, NULL, NULL);
    }
    case SOCKET_MSG_ACCEPT:
    {
        tstrSocketAcceptMsg *pstrAccept = (tstrSocketAcceptMsg *)pvMsg;
        if (pstrAccept) {
            tcp_client_socket = pstrAccept->sock;
            recv(tcp_client_socket, gau8SocketTestBuffer, ..., 0);
        }
    }
    case SOCKET_MSG_RECV:
    {
        tstrSocketRecvMsg *pstrRecv = (tstrSocketRecvMsg *)pvMsg;
        if (pstrRecv && pstrRecv->s16BufferSize > 0)
```

```
            send(tcp_client_socket, &msg_wifi_product, ..., 0);
    }
    case SOCKET_MSG_SEND:
    {
        printf("socket_cb: send success!\r\n");
        printf("TCP Server Test Complete!\r\n");
        printf("close socket\n");
    }
```

2. Build the program and download it into the board.
3. Start the application.
   Note: In case the Peer (TCP client) closes the socket, the only way for the application running on the MCU to find out about the closure of the socket is in the socket_cb() function in the recv operation, which returns an error -12 (SOCK_ERR_CONN_ABORTED) indicating the socket is closed by the peer.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
...
SOCKET_MSG_RECV:
{
tstrSocketRecvMsg *pstrRecv = (tstrSocketRecvMsg *)pvMsg;
if (pstrRecv && pstrRecv->s16BufferSize > 0) {

printf("r %s\r\n",gau8SocketTestBuffer);
recv(tcp_client_socket, gau8SocketTestBuffer, sizeof(gau8SocketTestBuffer), 0);
} else {

printf("socket_cb: recv error! %d\r\n",pstrRecv->s16BufferSize);
// Prints the error value in the receive callback.
close(tcp_server_socket);
tcp_server_socket = -1;
}
...
}
```

4.4.2 TCP Server Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 TCP server example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxxsocket_cb: bind success!
socket_cb: listen success!
socket_cb: accept success!
socket_cb: recv success!
socket_cb: send success!
TCP Server Test Complete!
close socket
```

4.5 NTP Time Client

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to retrieve time information from the time server.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 4-5. NTP Time Client Demo Setup

4.5.1 Execution

`main.c` - Initialize the socket and get the time from the NTP server.

1. Code summary.
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                                  "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                                  M2M_WIFI_SEC_WPA_PSK /**< Security
manner */
#define MAIN_WLAN_PSK                                   "12345678" /**< Password for
Destination SSID */
```

   – Initialize the socket module and register the socket callback function.

```
/* Initialize Socket module */
socketInit();
/* Register socket handler, resolve handler */
registerSocketCallback(socket_cb, resolve_cb);
```

   – Connect to the AP.

```
/* 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);
```

   – After the device is connected to the AP, create a UDP socket and bind it in the main loop.

```
if (udp_socket < 0) {
    udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (udp_socket < 0) {
    printf("main: UDP Client Socket Creation Failed.\r\n");
    continue;
    }
/* Initialize default socket address structure. */
addr_in.sin_family = AF_INET;
addr_in.sin_addr.s_addr = _htonl(MAIN_DEFAULT_ADDRESS);
addr_in.sin_port = _htons(MAIN_DEFAULT_PORT);
bind(udp_socket, (struct sockaddr *)&addr_in, sizeof(struct sockaddr_in));
```

   – Initialize the socket module and send an NTP time query to the NTP server in the `resolve_cb()` function

```
static void resolve_cb(uint8_t *pu8DomainName, uint32_t u32ServerIP)
{
    ...
if (udp_socket >= 0) {
        addr.sin_family = AF_INET;
        addr.sin_port = _htons(MAIN_SERVER_PORT_FOR_UDP);
        addr.sin_addr.s_addr = u32ServerIP;

        ret = sendto(udp_socket, (int8_t *)&cDataBuf, ...);
    }
```

   – Receive the NTP time from the server and convert it in the `socket_cb()` function.

```
static void resolve_cb(uint8_t *pu8DomainName, uint32_t u32ServerIP)
{
    ...
if (udp_socket >= 0) {
        addr.sin_family = AF_INET;
        addr.sin_port = _htons(MAIN_SERVER_PORT_FOR_UDP);
        addr.sin_addr.s_addr = u32ServerIP;

        ret = sendto(udp_socket, (int8_t *)&cDataBuf, ...);
    }
```

– Parse the time from the received server response.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    ...
    case SOCKET_MSG_RECVFROM:
    {
        /* printf("socket_cb: socket_msg_recvfrom!\r\n"); */
        tstrSocketRecvMsg *pstrRx = (tstrSocketRecvMsg *)pvMsg;
        if (pstrRx->pu8Buffer && pstrRx->s16BufferSize) {
            uint8_t packetBuffer[48];
            memcpy(&packetBuffer, pstrRx->pu8Buffer, sizeof(packetBuffer));
            ...
            uint32_t secsSince1900 = packetBuffer[40] << 24 |
                                    packetBuffer[41] << 16 |
                                    packetBuffer[42] << 8 |
                                    packetBuffer[43];

            /* Now convert NTP time into everyday time.
             * Unix time starts on Jan 1 1970. In seconds, that's 2208988800.
            * Subtract seventy years. */
            const uint32_t seventyYears = 2208988800UL;
            uint32_t epoch = secsSince1900 - seventyYears;
        /* Print the hour, minute and second. GMT is the time at Greenwich Meridian.
*/
            printf("socket_cb: The GMT time is %lu:%02lu:%02lu\r\n",
                        (epoch  % 86400L) / 3600,           /* hour (86400 equals
secs per day) */
                        (epoch  % 3600) / 60,               /* minute (3600 equals
secs per minute) */
                        epoch % 60);                        /* second */
```

2. Build the program and download it into the board.
3. Start the application.

4.5.2 NTP Time Client Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 time client example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxx
m2m_ip_resolve_handler : DomainName pool.ntp.org
socket_cb: The GMT time is xx:xx:xx
```

Note: If the server connection is unstable, this example may not operate normally.

4.6 SMTP Send Email

This example demonstrates the use of the ATWINC15x0/ATWINC3400 with the SAM D21 Xplained Pro board to send email for an SMTP server.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 4-6. SMTP Send Email Demo Setup

4.6.1 Execution

`main.c` - Initialize the chip and send an email.

1. Code summary.
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                  "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                  M2M_WIFI_SEC_WPA_PSK /**< Security manner */
#define MAIN_WLAN_PSK                   "12345678" /**< Password for Destination
SSID */
#define MAIN_SENDER_RFC                 "<sender@gmail.com>" /* Set Sender Email
Address */
#define MAIN_RECIPIENT_RFC              "<recipient@gmail.com>"  /* Set Recipient
Email Address */
#define MAIN_EMAIL_SUBJECT              "Hello from WINC1500!"
#define MAIN_TO_ADDRESS                 "recipient@gmail.com" /* Set To Email
Address */
#define MAIN_FROM_ADDRESS               "sender@gmail.com" /* Set From Email Address
*/
```

   – Initialize the socket module and register the socket callback function.

```
/* Initialize socket module */
socketInit();
registerSocketCallback(socket_cb, resolve_cb);
```

   – Connect to the AP.

```
/* 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);
```

   – After the device is connected to the AP, try to connect to the SMTP server. Once connected, the smtpStatehandler will be executed sequentially until the socket status becomes SocketComplete.

```
if (gu8SocketStatus == SocketInit) {
if (tcp_client_socket < 0) {
    gu8SocketStatus = SocketWaiting;
    if (smtpConnect() != SOCK_ERR_NO_ERROR) {
            gu8SocketStatus = SocketInit;
    }
}
} else if (gu8SocketStatus == SocketConnect) {
    gu8SocketStatus = SocketWaiting;
    if (smtpStateHandler() != MAIN_EMAIL_ERROR_NONE) {
        ...
    }
} else if (gu8SocketStatus == SocketComplete) {
    printf("main: Email was successfully sent.\r\n");
    close_socket();
}
```

   – Connect to the socket and receive data following the SMTP status.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    ...
case SOCKET_MSG_CONNECT:
    {
        if (pstrConnect && pstrConnect->s8Error >= SOCK_ERR_NO_ERROR)
            recv(tcp_client_socket, gcHandlerBuffer, ..., 0);
    }
    ...
    case SOCKET_MSG_RECV:
    {
        switch (gu8SmtpStatus) {
            case SMTP_INIT:
                ...
            case SMTP_HELO:
                ...
            case SMTP_AUTH:
                ...
            case SMTP_AUTH_USERNAME:
                ...
case SMTP_AUTH_PASSWORD:
                ...
            case SMTP_FROM:
                ...
            case SMTP_RCPT:
                ...
            case SMTP_DATA:
                ...
            case SMTP_MESSAGE_DATAEND:
                ...
        }
    }
```

2. Build the program and download it into the board.
3. Start the application.

Notes:

- To use Gmail, the root certificate must be installed. For more details about downloading the root certificate, refer to the "Atmel-42417-SAMD21-ATWINC1500-Platform_Getting_Started_Guide" document.
- If the server connection is unstable, this example may not operate normally.
  Limitations:

  1. Email is sent to only one recipient.

  2. Only plain text email is supported.

4.6.2 SMTP Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 send email example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxx
Host IP is xxx.xxx.xxx.xxx
Host Name is smtp.gmail.com
Recipient email address is recipient@gmail.com
main: Email was successfully sent.
```

5. Advanced Examples

This chapter describes complex or advanced examples in detail. For a customer's IoT application, these examples are useful to understand how to use ATWINC15x0/ATWINC3400 APIs and implement a feature for an IoT application.

- Exosite Cloud Application
- Growl client – example using RESTful API over SSL (essential for IoT application)
- HTTP client file downloader application
- IoT temperature and Qtouch® sensor demo application using an Android app
- MQTT chat client – demonstrate how to send and receive IoT information using the MQTT protocol
- OTA firmware upgrade – ATWINC15x0/ATWINC3400 firmware upgrade via OTA server
- PubNub Cloud application
- SSL client connection – Set up an SSL client connection
- Weather client – get the current weather information from the network provider and utilize the IO1 sensor device
- Wi-Fi serial – useful for chatting or controlling a remote device
- Enterprise security – for details, refer to the ATWINC Enterprise Security Application Note
- Simple Roaming and ALPN – for details, refer to the ATWINC3400 Wi-Fi/BLE Network Controller - Software Design Guide
- ALPN connect example – demonstrates TLS ALPN extension to negotiate secure protocol (HTTP/2(preferred) or HTTP/1.1) between client and server

5.1 Exosite Cloud

This section demonstrates how to connect the SAM W25 Xplained Pro board to cloud back-end services such as Exosite.

Note: This example application is not available in ASF for ATWINC3400

The following topics will be covered:

- Hardware overview for both the SAM W25 Wi-Fi module and SAM W25 Xplained Pro evaluation board
- How to connect to Exosite Cloud
- How to create a dashboard to visualize the data
- How to build and execute the Exosite demo

It is based on the following hardware:

- The SAM W25 Xplained Pro
- The IO1 Xplained Pro extension on the EXT1

5.1.1 Execution

`main.c` – Initialize the board, connect to an Exosite Cloud and upload the temperature details to the cloud.

The SAM W25 module should now be in AP mode and be listed as a Wi-Fi network with the same name as shown in the image below (Atmel_SAMW25_XX:XX). Notice that the last two bytes of the MAC address are appended to the SSID. Simply connect your PC/smartphone to the module in AP mode.

```
#define MAIN_M2M_DEVICE_NAME    "Atmel_SAMW25_00:00"
```

Note: Once connected, open your favorite web browser at the following address http://atmelconfig.com and provide the required Network Name (SSID) and Passphrase (device name can be blank) fields of the Wi-Fi AP the SAM W25 is supposed to connect to.

Once the SAM W25 is connected to the AP with internet connectivity, it will immediately connect to the Exosite messaging service and will start sending temperature and light data.

For this application, you can start with the `main.c` file, which has all the code needed to perform the system initialization by configuring the necessary components such as UART, buttons, LEDs, BSP, temperature sensor, Wi-Fi driver and socket.

The WiFi initialization and any other configurations required should be done before the super loop function `for(;;system_sleep())`:

```
/* Initialize WINC1500 Wi-Fi driver with data and status callbacks. */
    param.pfAppWifiCb = wifi_cb;
    ret = m2m_wifi_init(&param);
    if (M2M_SUCCESS != ret) {
        DEBUG(DEBUG_CONF_WIFI "m2m_wifi_init call error!(%d)" DEBUG_EOL, ret);
        while (1) {
    }
    }
    m2m_wifi_set_sleep_mode(M2M_PS_AUTOMATIC, 1);

    /* Initialize socket. */
    socketInit();
    registerSocketCallback(http_client_socket_event_handler,
http_client_socket_resolve_handler);
    /* Connect using stored SSID and Password. */
    m2m_wifi_default_connect();
     ap_exosite_connection_state = MAIN_CHECKING_AP_INFORMATION;
    for(;;system_sleep())
    {
        /* Handle pending events from network controller. */
        ret = m2m_wifi_handle_events(NULL);
```

The first step is to initialize the Exosite client module by calling function `Exosite_example_init()`. This function receives one parameter, which is a function pointer to the callback of the http module. The role of this function is to initialize the http_module and also initialize a timer_module for it. Add the following code just before the `for(;;system_sleep())` inside the `main()` function.

```
/* Initialize Exosite. */
exosite_example_init(main_http_client_callback);
```

`main_http_client_callback()` will be registered by `Exosite_example_init()` and will receive and process all socket events. This function can be located in the `main.c` file or in a separate file as shown below:

```
/** brief Callback to get the Data from socket.*/
    static void main_http_client_callback(struct http_client_module *module_list, int type,
                                                union http_client_data *data)
    {
        Switch(type)
        {
            case HTTP_CLIENT_CALLBACK_SOCK_CONNECTED: break;
            case HTTP_CLIENT_CALLBACK_REQUESTED: break;
            case HTTP_CLIENT_CALLBACK_RECV_RESPONSE: break;
            case HTTP_CLIENT_CALLBACK_DISCONNECTED: break;
        }
    }
```

Register the callback function for the sockets by calling the following function in `main()`:

```
/* Initialize socket. */
socketInit();
registerSocketCallback(http_client_socket_event_handler, http_client_socket_resolve_handler);
```

The above function receives two parameters. The first one is a function pointer to deliver socket messages from the socket. The other one is a function pointer that is used for resolving DNS. The two parameters are located in `http_client.h`.

```
/**
 * \brief Event handler of socket event.
 *
 * \param[in]  sock            Socket descriptor.
 * \param[in]  msg_type        Event type.
 * \param[in]  msg_data        Structure of socket event.
 */
void http_client_socket_event_handler(SOCKET sock, uint8_t msg_type, void *msg_data);

/**
 * \brief Event handler of gethostbyname.
 *
 * \param[in]  domain_name     Domain name.
```

```
  * \param[in]  server_ip       Server IP.
  */
void http_client_socket_resolve_handler(uint8_t *domain_name, uint32_t server_ip);
```

In order to send the data from the temperature and light sensors on the IO1 to the Exosite cloud, the function `exosite_example_read_and_write()` must be used.

```
/* publish the temp measurements every interval */
if (tick_counter_check_timer())
{
    Char send_buf[100]; int dTemp = 0;
    int dLight = 0;

    /* prepare to sensor data in the I/o1 Board */ io1_board_prepare_to_get_info();
    dTemp = io1_board_get_temperature(); dLight = io1_board_get_lightvalue();
    sprintfsend_buf,”degree=%d&voltage=%d”, fintƀdTemp, fintƀdLightƀ;
    if( exosite_example_read_and_write(send_buf, (char*)p_board_info->cik))
    ...
}
```

The function sends the data to the cloud continuously at regular intervals.

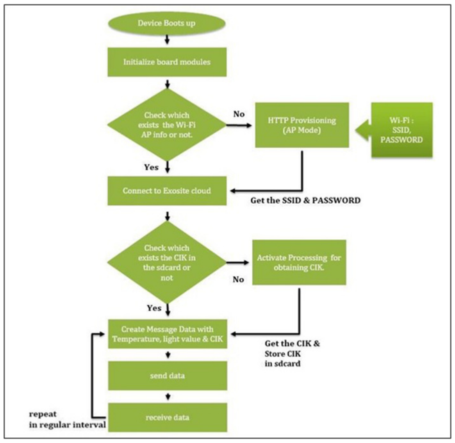
In order to receive the response messages from Exosite, the `main_http_client_callback()` function must be used.

```
/** brief Callback to get the Data from socket. */
static void main_http_client_callback(struct http_client_module *module_list,
                                        int type, union http_client_data *data)
{
    case HTTP_CLIENT_CALLBACK_RECV_RESPONSE:
    parsing_http_response_data( data->recv_response.response_code,
    data->recv_response.content,
    data->recv_response.content_length);
    break;
}
```

Then you can add the relevant code to process the received messages without the `parsing_http_response_data()` function.

Note: This function is required to receive the CIK from Exosite.

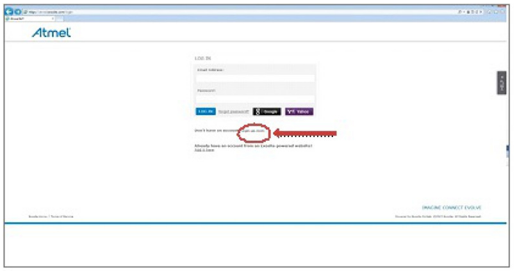
Figure 5-1. Exosite Application Flow Chart


Device Boots up
Initialize board modules
Check which exists the Wi-Fi AP info or not.
No
HTTP Provisioning (AP Mode)
Wi-Fi : SSID, PASSWORD
Yes
Connect to Exosite cloud
Get the SSID & PASSWORD
Check which exists the CIK in the sdcard or not
Activate Processing for obtaining CIK.
No
Yes
Create Message Data with Temperature, light value & CIK
Get the CIK & Store CIK in sdcard
send data
receive data
repeat in regular interval

5.1.2 How to Set the Exosite Cloud to View the Published Data

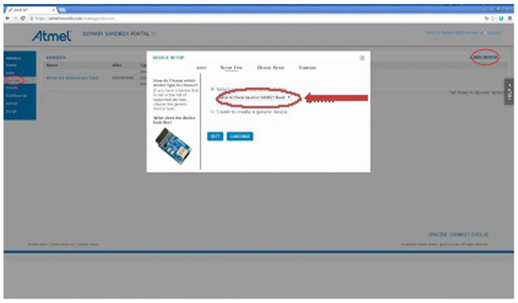
1. Open a free account on the Exosite Portal.
   – Go to the Exosite Portal web site http://atmel.exosite.com
   – Click on “Create an Account” and then log in

Figure 5-2. Atmel Exosite Login Page



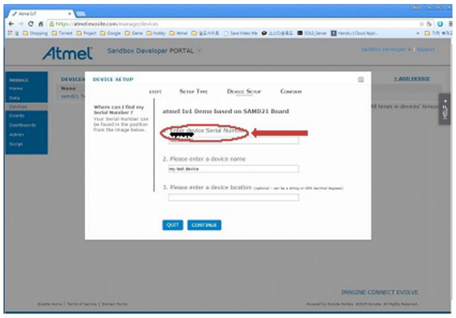
2. Register your device and input your device serial number on the I/O1 extension.
   - You can see the device setup window (see below).
   - Select Device -> ADD DEVICE -> Atmel I/O1 Demo based on the SAM W25 Board.

   Figure 5-3. Atmel Exosite Add Device



   - Populate the S/N written on the back of I/O1 Xplained Pro.

   Figure 5-4. Atmel Exosite Add Device Details



DEVICE SETUP

Where can I find my Serial Number ?
Your Serial number can be found in the position from the image below.

atmel Io1 Demo based on SAMD21 Board

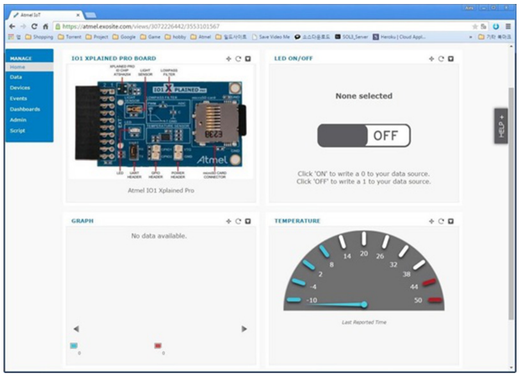
Enter device Serial Number [redacted]

2. Please enter a device name: my test device

3. Please enter a device location (optional - can be a string or GPS decimal degrees)

3. The DashBoard of my device will be shown.

   Figure 5-5. Atmel Exosite Device Update



Click 'ON' to write a 0 to your data source.
Click 'OFF' to write a 1 to your data source.

5.2 Growl Notification

This example demonstrates the use of the ATWINC15x0 with the SAM D21 Xplained Pro.

This example transmits a notification from the ATWINC15x0 device (based on a certain trigger) to a public remote server which in turn sends it to a phone application.

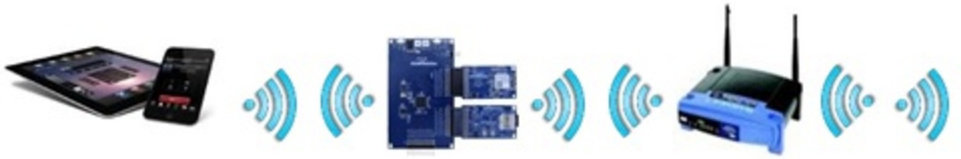
The initiated notification from the ATWINC15x0 device is directed to a certain subscriber on the server. The supported applications are PROWL (for iPhone notifications) and NMA (for ANDROID notifications).

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header

Figure 5-6. Demo Setup



5.2.1 Execution

`main.c` - Initialize Growl and send a notification message.

1. Code summary:
   – Configure the Growl parameters for your account in `main.h`.

```
/** Growl Options */
#define PROWL_API_KEY              "6ce3b9ff6c29e5c5b8960b28d9e987aec5ed603a"
#define NMA_API_KEY                "91787604ed50a6cfc2d3f83d1ee196cbc30a3cb08a7e69a0"
```

   – Get the MAC address and set the device name with the MAC address.

```
m2m_wifi_get_mac_address(gau8MacAddr);

set_dev_name_to_mac((uint8_t *)gacDeviceName, gau8MacAddr);
set_dev_name_to_mac((uint8_t *)gstrM2MAPConfig.au8SSID, gau8MacAddr);
m2m_wifi_set_device_name((uint8_t *)gacDeviceName, ...);
```

   – Start Provision mode.

```
m2m_wifi_start_provision_mode((tstrM2MAPConfig *)&gstrM2MAPConfig,
            (char *)gacHttpProvDomainName, 1);
```

   – When your mobile device sends the configuration information, the `wifi_cb()` function will be called with the `M2M_WIFI_RESP_PROVISION_INFO` message and you can connect to the AP with the given information.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_RESP_PROVISION_INFO:
    tstrM2MProvisionInfo *pstrProvInfo = (tstrM2MProvisionInfo *)pvMsg;
    if (pstrProvInfo->u8Status == M2M_SUCCESS) {
    m2m_wifi_connect((char *)pstrProvInfo->au8SSID,
        strlen((char *)pstrProvInfo->au8SSID),
        pstrProvInfo->u8SecType,
        pstrProvInfo->au8Password, M2M_WIFI_CH_ALL);
    }
```

   – After the device is connected to the AP, initialize the Growl key and execute the message handler.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    ...
    case M2M_WIFI_REQ_DHCP_CONF:
    {
    ...
    NMI_GrowlInit((uint8_t *)PROWL_API_KEY, (uint8_t *)NMA_API_KEY);
    growl_send_message_handler();
    }
    ...
}
```

– The notification message will be sent through the function shown below.

```
static int growl_send_message_handler(void)
{
    ...
    NMI_GrowlSendNotification(NMA_CLIENT, (uint8_t *)"Growl_Sample",
        (uint8_t *)"Growl_Event", (uint8_t *)"growl_test", NMA_CONNECTION_TYPE);
    return 0;
}
```

2. Build the program and download it into the board.
3. Start the application.
4. Connect your mobile device to ATWINC15x0/ATWINC3400 AP [WINC1500_08:CA].
5. Browse to the webpage (www.microchip.com) to setup AP, populate the page, then press Connect.
6. The ATWINC15x0/ATWINC3400 will be connected to the AP that you entered.
7. The Growl message will be sent.

This example supports sending growl notifications to the following servers:

- Prowl for iOS push notifications (https://www.prowlapp.com/)
- NMD for Android push notifications (www.notifymydevice.com/)

In order to enable the Growl application (for sending notifications), you need to set your own API key to represent your account. Create your own by:

Create an NMD account at www.notifymydevice.com/ and create an API key. Copy the obtained key string to the NMA_API_KEY macro in the `main.h` as follows:

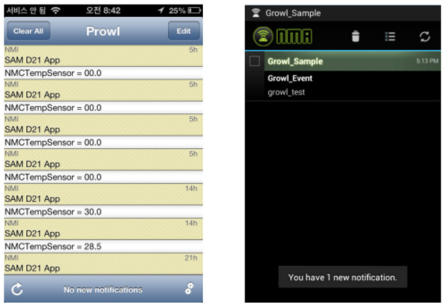
- #define NMA_API_KEY "f8bd3e7c9c5c10183751ab010e57d8f73494b32da73292f6"

Create a Prowl account at https://www.prowlapp.com/ and create an API key. Copy the obtained API key string to the PROWL_API_KEY macro in the file `main.h` as follows:

- #define PROWL_API_KEY "117911f8a4f2935b2d84abc934be9ff77d883678"

Note: For using Growl, the root certificate must be installed.

Figure 5-7. Launch the Growl or NMA application to receive notification



NMI 5h
SAM D21 App
NMCTempSensor = 00.0
NMI 5h
SAM D21 App
NMCTempSensor = 00.0
NMI 5h
SAM D21 App
NMCTempSensor = 00.0
NMI 5h
SAM D21 App
NMCTempSensor = 00.0
NMI 14h
SAM D21 App
NMCTempSensor = 30.0
NMI 14h
SAM D21 App
NMCTempSensor = 28.5
NMI 21h
SAM D21 App

Growl_Sample 5:13 PM
Growl_Event
growl_test

You have 1 new notification.

5.2.2 Growl Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 simple growl example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
```

```
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Provision Mode started.
Connect to [atmelconfig.com] via AP[WINC1500_XX:XX] and fill up the page.
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
wifi_cb: M2M_WIFI_RESP_PROVISION_INFO.
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
send Growl message
Growl CB : 20
```

5.3 HTTP File Downloader

This example demonstrates the use of the ATWINC15x0 with the SAM D21 Xplained Pro board to download the file in the SD card connected to the IO1 Xplained Pro board.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header
- The IO1 XPRO board on the EXT2 header

Figure 5-8. HTTP File Downloader Demo Setup

5.3.1 Execution

`main.c` – Initialize the ATWINC15x0 and download the file from the URL.

1. Code summary:
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                      "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                      M2M_WIFI_SEC_WPA_PSK /**< Security manner
*/
#define MAIN_WLAN_PSK                       "12345678" /**< Password for Destination
SSID */
```

   – Configure the HTTP URL file to be downloaded into the SD card.

```
/** Content URI for download. */
#define MAIN_HTTP_FILE_URL        "http://www.atmel.com/Images/Atmel-42502-
SmartConnect-WINC1500-MR210PB_Datasheet.pdf"
```

   – Configure the HTTP client.

```
configure_http_client();
```

   – Get the default config data and specify the user configuration. The `http_client_init()` and `http_client_register_callback()` functions execute sequentially.

```
static void configure_http_client(void)
{
    ...
    http_client_get_config_defaults(&httpc_conf);

    httpc_conf.recv_buffer_size = MAIN_BUFFER_MAX_SIZE;
    httpc_conf.timer_inst = &swt_module_inst;

    ret = http_client_init(&http_client_module_inst, &httpc_conf);
    if (ret < 0) {
    while (1) {
    } /* Loop forever. */
    }
    http_client_register_callback(&http_client_module_inst, http_client_callback);
}
```

- Initialize the socket module and register the socket callback functions.

```
socketInit();
registerSocketCallback(socket_cb, resolve_cb);
```

- Connect to the AP.

```
/* 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);
```

- After the device is connected to the AP, an HTTP request will be sent.

```
static void wifi_callback(uint8_t msg_type, void *msg_data)
{
    ...
    case M2M_WIFI_REQ_DHCP_CONF:
    {
        start_download();
    }
```

```
static void start_download(void)
{
    ...
    /* Send the HTTP request. */
    printf("start_download: sending HTTP request...\r\n");
    http_client_send_request(&http_client_module_inst, MAIN_HTTP_FILE_URL,
HTTP_METHOD_GET, NULL, NULL);
}
```

- Four operations will execute sequentially. The files are stored in the SD card when they are received by the HTTP request and the callback.

```
static void http_client_callback(...)
{
    switch (type) {
    case HTTP_CLIENT_CALLBACK_SOCK_CONNECTED:
        printf("Connected\r\n");
        break;
    case HTTP_CLIENT_CALLBACK_REQUESTED:
    printf("Request complete\r\n");
    break;
    case HTTP_CLIENT_CALLBACK_RECV_RESPONSE:
        printf("http_client_callback: received response %u data size %u\r\n",
                (unsigned int)data->recv_response.response_code,
                (unsigned int)data->recv_response.content_length);
        if ((unsigned int)data->recv_response.response_code == 200) {
            http_file_size = data->recv_response.content_length;
            received_file_size = 0;
        }
        if (data->recv_response.content_length <= MAIN_BUFFER_MAX_SIZE) {
            store_file_packet(data->recv_response.content, data-
>recv_response.content_length);
        }
        break;
    case HTTP_CLIENT_CALLBACK_RECV_CHUNKED_DATA:
        store_file_packet(data->recv_chunked_data.data, data-
>recv_chunked_data.length);
    case HTTP_CLIENT_CALLBACK_DISCONNECTED:
        printf("Disconnected reason:%d\r\n", data->disconnected.reason);
        ...
    }
}
```

- The first sequence begins with the socket connected. After the request completes, the third sequence is executed and the file header is followed by the file data.

2. Build the program and download it into the board.
3. Start the application.

Notes:

- If the disconnect reason is equal to -ECONNRESET(-104), it means the server disconnected your connection due to the keep alive timeout. This operation is normal.
- If the server connection is unstable, this example may not operate normally.

5.3.2 HTTP File Downloader Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 HTTP file downloader example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
This example requires the AP to have internet access.

init_storage: please plug an SD/MMC card in slot...
init_storage: mounting SD card...
init_storage: SD card mount OK.
main: connecting to WiFi AP DEMO_AP...
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: IP address is xxx.xxx.xxx.xxx
start_download: sending HTTP request...
resolve_cb: www.microchip.com IP address is xxx.xxx.xxx.xxx

http_client_callback: HTTP client socket connected.
http_client_callback: request completed.
http_client_callback: received response 200 data size 1147097
store_file_packet: creating file [0:WINC1500-MR210PB_Datasheet.pdf]
store_file_packet: received[xxx], file size[1147097]
...
store_file_packet: received[1147097], file size[1147097]
store_file_packet: file downloaded successfully.
main: please unplug the SD/MMC card.
main: done.
```

5.4 IoT Temperature and Qtouch Sensor Demo

The purpose of this demo is to connect various kinds of sensors to your home network using a Wi-Fi access point and remotely access the sensors' information via an Android device.

A sensor typically implements a basic discovery system where UDP broadcast frames are sent each second to advertise the sensor presence on the network. The Android app can then detect such packets and communicate with the sensor to receive the sensor data stream.

These are implemented in a similar way and use a similar communication protocol with the Android application. Both can work at the same time and report sensor information to the same “Atmel IoT Sensor” Android application.

Note: This example application is not available in ASF for ATWINC3400.

The Temperature Sensor Demo is based on the following hardware:

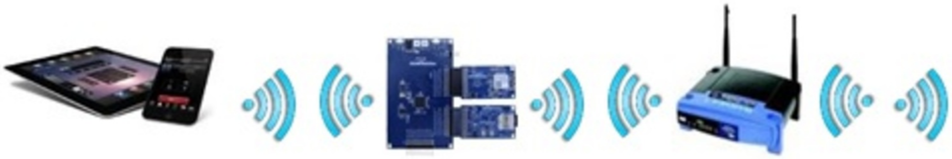
- The SAM D21 Xplained Pro
- The ATWINC15x0 on EXT1 header
- The IO1 XPRO board on the EXT2
- The QT1 XPRO board on the EXT1 and EXT2

The Qtouch Sensor Demo is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT3 header
- The QT1 XPRO board on the EXT1 and EXT2

Note: The Qtouch mutual capacitance QT1 XPRO board needs to be connected to EXT1 and EXT2 header together.

Figure 5-9. IoT Demo Setup



5.4.1 Execution

`main.c` - Initialize the board, connect to an AP and communicate using the Android App.

- Configure AP SSID in `demo.h` for the *Temperature Sensor Demo*.

```
#define DEMO_WLAN_AP_NAME    "WINC1500_MyAP"// Access Point Name.
```

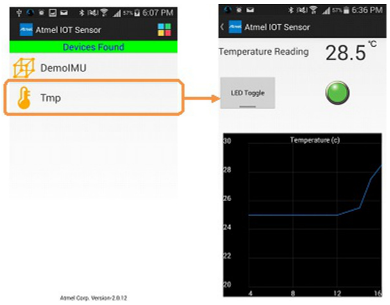
- Configure AP SSID in `demo.h` for the *Qtouch Sensor Demo*.

```
#define DEMO_WLAN_AP_NAME    "WINC1500_TOUCH_MyAP"// Access Point Name.
```

- **Note:** Refer to the HTTP provision mode application note for more details.

Once your sensor Wi-Fi module is provisioned, connect your Android device to the same Access Point and open the Atmel IoT Sensor Application. Your sensor will display on the Android application's main screen as shown below:

Figure 5-10. Demo Android App



By tapping the sensor's name, you will be able to access the real time data flow. In the case of a temperature sensor, you can see the real time temperature data in the graph. The "LED Toggle" button toggles the LED status. A green or red light next to the "LED Toggle" button gives the current status of the LED.

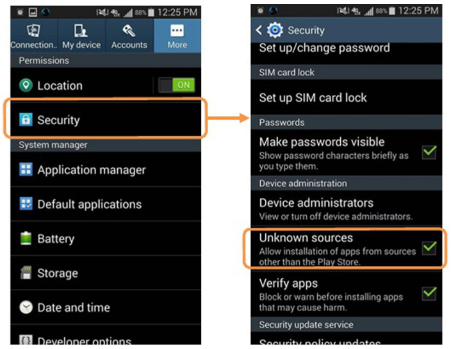
Note: If your Android device is not connected to the same Access Point as the sensor, you will not be able see any device.

5.4.2 Install the IoT Sensor Android Application

To Install the IoT Sensor APK located in the android_app folder of your ASF project, do the following:

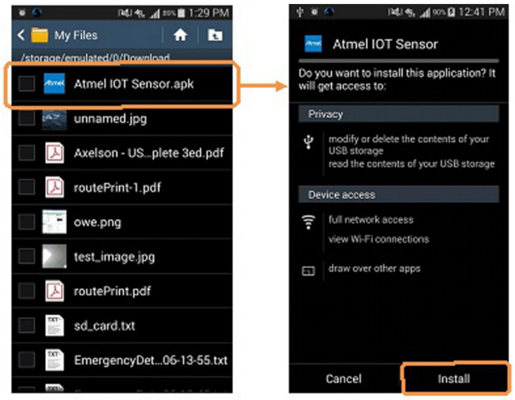
- Connect your Android device to your Windows laptop
- Open the start menu and double click on "Computer"
- Your phone will be listed under the "Portable Devices" list
- Open and browse the existing folders
- Locate the "Download" folder, then simply drag and drop the provided "Atmel_IOT_Sensor_v2.10.18.apk" application
- Go to your Android phone settings and allow unknown application sources

Figure 5-11. Android App Install Settings



- Open your favorite file browser like “MyFiles” (available on Google market for free)
- Go to your “Downloads” folder
- Click on the Atmel IoT Sensor Application file (APK)
- Press the “Install” button

Figure 5-12. Install Android App


1:29 PM
My Files
/storage/emulated/0/Download
Atmel IOT Sensor.apk
unnamed.jpg
Axelson - US...plete 3ed.pdf
routePrint-1.pdf
owe.png
test_image.jpg
routePrint.pdf
sd_card.txt
EmergencyDet...06-13-55.txt
12:41 PM
Atmel IOT Sensor
Do you want to install this application? It will get access to:
Privacy
modify or delete the contents of your USB storage
read the contents of your USB storage
Device access
full network access
view Wi-Fi connections
draw over other apps
Cancel
Install

5.5 MQTT Chat

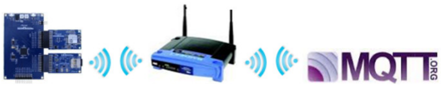
This example demonstrates the use of the ATWINC15x0 with the SAM D21 Xplained Pro board to chat using the MQTT protocol.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header

Figure 5-13. MQTT Chat Demo Setup



5.5.1 Execution

This example uses Paho MQTT library for the MQTT protocol implementation.

`main.c` - Initialize the board, connect to an MQTT broker and chat with other devices.

1. Code summary:
   – Configure the network parameters and the MQTT broker in `main.h`.

```
/* Chat MQTT topic. This is only demo User can create their own topic*/
#define MAIN_CHAT_TOPIC "atmel/sample/chat_demo/"
```

```
/** A MQTT broker server which was connected.
 * test.mosquitto.org is public MQTT broker.*/
static const char main_mqtt_broker[] = "test.mosquitto.org";

/** Wi-Fi Settings */
#define MAIN_WLAN_SSID        "DEMO_AP" /* < Destination SSID */
#define MAIN_WLAN_AUTH        M2M_WIFI_SEC_WPA_PSK /* < Security manner */
#define MAIN_WLAN_PSK         "12345678" /* < Password for Destination SSID */
```

- Configure the MQTT module. When calling the `configure_mqtt()` function, several parameters like read/write buffers and buffer size are configured along with the callback function.

```
/* Initialize the MQTT service. */
configure_mqtt();
```

```
static void configure_mqtt(void)
{
    ...
    mqtt_get_config_defaults(&mqtt_conf);
    /* To use the MQTT service, it is necessary to always set the buffer and the 
timer. */
       mqtt_conf.read_buffer = mqtt_read_buffer;
    mqtt_conf.read_buffer_size = MAIN_MQTT_BUFFER_SIZE;
    mqtt_conf.send_buffer = mqtt_send_buffer;
    mqtt_conf.send_buffer_size = MAIN_MQTT_BUFFER_SIZE;

    result = mqtt_init(&mqtt_inst, &mqtt_conf);
    result = mqtt_register_callback(&mqtt_inst, mqtt_callback);
```

- The Paho MQTT platform implementation uses SysTick for timing. Therefore, systick must be configured before using any part of the library.

```
if (SysTick_Config(system_cpu_clock_get_hz() / 1000))
    {
        puts("ERR>> Systick configuration error\r\n");
        while (1);
    }
```

- Setup the username first and then the topic value will be set with MAIN_CHAT_TOPIC and username.

```
    /* Setup username first */
    printf("Enter the username (Max %d characters)\r\n", MAIN_CHAT_USER_NAME_SIZE);
    scanf("%64s", mqtt_user);
    printf("User : %s\r\n", mqtt_user);
    sprintf(topic, "%s%s", MAIN_CHAT_TOPIC, mqtt_user);
```

- Initialize the socket module and register the socket callback function.

```

socketInit();
registerSocketCallback(socket_cb, NULL);
```

- Connect to the AP.

```
/* 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);
```

- After the device is connected to the AP, call the `mqtt_connect()` function to connect the socket.

```
static void wifi_callback(uint8 msg_type, void *msg_data)
{
    ...
    case M2M_WIFI_REQ_DHCP_CONF:
    ...
    /* Try to connect to MQTT broker when Wi-Fi was connected. */
    mqtt_connect(&mqtt_inst, main_mqtt_broker);
}
```

- The MQTT callback will receive the `MQTT_CALLBACK_SOCK_CONNECTED` message and then start sending a CONNECT message to the MQTT broker.

```
static void mqtt_callback(struct mqtt_module *module_inst, int type, union mqtt_data 
*data)
```

```
{
    ...
    case MQTT_CALLBACK_SOCK_CONNECTED:
    {
    mqtt_connect_broker(module_inst, 1, NULL, NULL, mqtt_user, NULL, NULL, 0, 0, 0);
```

– The MQTT callback will receive the `MQTT_CALLBACK_CONNECTED` message and then register a subscription for a specific topic. Each subscription has a separate callback function which is registered along with the subscribe call.

```
static void mqtt_callback(struct mqtt_module *module_inst, int type, union mqtt_data
*data)
{
    ...
    case MQTT_CALLBACK_CONNECTED:
    {
        mqtt_subscribe(module_inst, MAIN_CHAT_TOPIC "#", 0, SubscribeHandler)
```

– If another device sends a message with this topic, then the corresponding subscribe callback is called.

```
void SubscribeHandler(MessageData *msgData)
{
...
}
```

– If the user inputs a string via a terminal, the MQTT will publish the following message:

```
static void check_usart_buffer(char *topic)
{
    if (uart_buffer_written >= MAIN_CHAT_BUFFER_SIZE) {
        mqtt_publish(&mqtt_inst, topic, uart_buffer,
        MAIN_CHAT_BUFFER_SIZE, 0, 0);
        uart_buffer_written = 0;
    }
```

– To poll for a message in any of the subscribed topic, mqtt_yield must be called. This function must be called frequently.

```
while (1) {
        /* Handle pending events from network controller. */
        m2m_wifi_handle_events(NULL);
        ...
        if(mqtt_inst.isConnected)
            mqtt_yield(&mqtt_inst, 0);
    }
```

2. Build the program and download it into the board.
3. Start the application.
4. On the terminal window, enter the username through the terminal window.
5. After the initialization completes, you can chat.
   Note:
   - Initialization may take up to a few minutes depending on the network environment.
   - The application must be programmed and running. The following information will be displayed on the terminal window.

Note: The maximum message length should be shorter than 128 bytes. If the server connection is unstable, this example may not operate normally.

5.5.2 Demo Console Log

```
-- WINC1500 Wi-Fi MQTT chat example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Preparation of the chat has been completed.
```

```
Enter the user name (Max 64 characters)
User : demo_user
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
Preparation of the chat has been completed.
atmel/sample/chat_demo/demo_user >> hi!
atmel/sample/chat_demo/demo_user >> anybody there?
atmel/sample/chat_demo/demo_user >> I'm here
atmel/sample/chat_demo/demo_user >> hi
```

Note: The username should not contain any blank spaces.

Note: This demo is currently supported for WINC15x0 only.

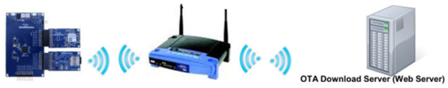
5.6 OTA Firmware Upgrade

This example demonstrates how to upgrade the ATWINC15x0/ATWINC3400 firmware via OTA. It downloads the ATWINC15x0/ATWINC3400 firmware from an OTA download server, which is a web server. You can upload a new firmware image and download it to your device.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 5-14. OTA Firmware Upgrade Demo Setup



5.6.1 Execution

`main.c` - Initialize the device and set the server information. It then connects to the OTA download server.

1. Set your OTA download server.
2. Upload the OTA firmware binary to the root folder in your server. (ATWINC15x0: e.g. `http://192.168.0.137/m2m_ota.bin`, ATWINC3400: e.g. `http://192.168.1.11/m2m_ota_3400.bin`).
3. Code summary:
   – Configure the network and OTA parameters in `main.h`. The default port number is 80. If the user application uses a different port number, update it in the URL. "`http://192.168.0.137/m2m_ota.bin`".

```
#define MAIN_WLAN_SSID        "DEMO_AP"
#define MAIN_WLAN_AUTH        M2M_WIFI_SEC_WPA_PSK
#define MAIN_WLAN_PSK         "12345678"
#define MAIN_OTA_URL          "http://192.168.0.137/m2m_ota.bin"
```

   – Connect to the AP.

```
/* 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);
```

   – Initialize the OTA function.

```
m2m_ota_init(OtaUpdateCb, OtaNotifCb);
```

   – After the device is connected to the AP, the `m2m_ota_start_update()` function will be executed in the `wifi_cb()` function.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    ...
    case M2M_WIFI_REQ_DHCP_CONF:
        m2m_ota_start_update((uint8_t *)MAIN_OTA_URL);
```

   – If successfully downloaded, the `m2m_ota_switch_firmware()` function will be called through `OtaUpdateCb()`.

```
static void OtaUpdateCb(uint8_t u8OtaUpdateStatusType, uint8_t u8OtaUpdateStatus)
{
    m2m_ota_switch_firmware();
}
```

   After that and if no errors occur, a message will display: “OTA success. Press reset your board.”

4. Build the program and download it into the board.
5. Start the application.
   When you receive the IP address, then the OTA update will start.
6. You can find the firmware version and the build time information.

```
(3)NMI M2M SW  VERSION xx.xx.xx
(3)NMI MIN DRV VERSION xx.xx.xx
(3)Built at xxx xx xxxx xx:xx:xx
```

7. Once connected to the OTA server, the following text will appear on the terminal window:

```
(770)(M2M)(OTA) Invalidate RB Image
Start Writing.. at 40000 Size 204380
(809)STATS    0   0   5  25   0, err: 16 - stuck: 0
(809)!@#$ Rate DN (48.0 Mbps) !@#$
(910)!@#$ Rate DN (MCS-4) !@#$
(1079)(M2M)Verification OK 204380
(1080)(M2M)(OTA) Update image done successfully
(1080)(M2M)Socket 6 Closed
(1083)(M2M)(OTA) Swap image done successfully
(1084)(M2M)(OTA) Switching firmware done.
```

8. After the successful OTA firmware upgrade, the following text will appear on the terminal window:

```
Wi-Fi IP is xxx.xxx.xxx.xxx
OtaUpdateCb 1 0
OtaUpdateCb m2m_ota_switch_firmware start.
OtaUpdateCb 2 0
OTA Success. Press reset your board.
```

9. The application reads data from the serial interface.

```
(1)Chip ID = 1502b1
(1)Flash ID = c21320c2, Size = 4 MBit
(1)Working Image offset = 0x3000 Rollback =  0x40000
(2)(M2M)(Efuse) successfully loaded from bank 1.
(2)EFUSE:MAC
(2)(M2M)MAC_ADDR  = xx:xx:xx:xx:xx:xx
(3)NMI M2M SW  VERSION xx.xx.xx
(3)NMI MIN DRV VERSION xx.xx.xx
(3)Built at xxx xx xxxx xx:xx:xx
(3)__ROM_FIRMWARE__
```

The OTA download procedure for WINC15x0 is:

1. Download the ATWINC15x0 driver version 19.4.4 OTA application from ASF.
2. Use any HTTP server or hfs.exe from http://www.rejetto.com/hfs/
3. Run the hfs.exe
4. Add the OTA firmware from the “\src\Tools\firmware\ota_firmware\m2m_ota_3a0.bin” 19.5.3 release package to the root folder in the hfs.exe tool.
5. Change the MAIN_OTA_URL "http://192.168.1.138/m2m_ota.bin" to the IP address in the hfs tool. The IP address is the HTTP server IP address and the user can use their own port number or port number 80.
6. Program the MCU with the OTA application available in the ASF (19.4.4) after compiling these changes.
7. Run the application and wait for it to complete.

The OTA download procedure for ATWINC3400 is:

1. Download the ATWINC3400 driver version 1.0.8 OTA application from ASF.
2. Use any HTTP server or hfs.exe from http://www.rejetto.com/hfs/
3. Run the hfs.exe

4. Add the OTA firmware from the “\src\firmware\m2m_ota_3400.bin” 1.2.2 release package to the root folder in the hfs.exe tool.
5. Change the MAIN_OTA_URL "http://192.168.1.11/m2m_ota_3400.bin" to the IP address in the hfs tool. The IP address is the HTTP server IP address and the user can use their own port number or port number 80.
6. Program the MCU with the OTA application available in the ASF (1.2.1) after compiling these changes.
7. Run the application and wait for it to complete.

5.6.2 OTA Firmware Upgrade Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 OTA firmware upgrade example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxx
```

The following text will appear on the terminal window of the ATWINC15x0/ATWINC3400 debug UART interface:

```
(1)Chip ID = 1502b1
(1)Flash ID = c21320c2, Size = 4 MBit
(1)Working Image offset = 0x3000 Rollback =  0x40000
(2)(M2M)(Efuse) successfully loaded from bank 1.
(2)EFUSE:MAC
(2)(M2M)MAC_ADDR  = xx:xx:xx:xx:xx:xx
(3)NMI M2M SW  VERSION xx.xx.xx
(3)NMI MIN DRV VERSION xx.xx.xx
(3)Built at xxx xx xxxx xx:xx:xx
(3)__ROM_FIRMWARE__
(4)(M2M)LOAD SEC
(6)(M2M)1000 400 2f000 2fc00 38000
(7)(M2M)Wifi Connect
(7)(M2M)SSID    : NW01
(7)(M2M)AUTH    : WPA-Personal
(7)(M2M)PSK     : nmisemi2
(8)(M2M)Channel : 256
(8)Reset MAC
(9)>> Sleep clk src <= Int. osc
(9)>> wakeup_delay = 1500 us
(9)>> pds = [652 652 6526 1957 3 ]
(9)---------------------------------------------
(489)MAC State <3>
(494)MAC State <4>
(494)Join on 11
(494)>> sta_wait_join 179
(494)MAC State <5>
(494)MAC State <6>
(495)Init Auth.
(495)MAC State <7>
(495)MAC State <9>
(495)MAC State <10>
(496)MAC State <1>
(496)!@#$ Rate DN (MCS-5) !@#$
(496)Assoc Success.
(498)(M2M)WIFI Connected
(499)(DHCP)<- DISCOVER
(500)Tsf join
(510)Tsf join Done
(532)(DHCP)-> OFFER
```

```
(563)(DHCP)-> ACK
(563)(DHCP)Self IP          : "xxx.xx.xxx.xxx"
```

5.7 PubNub Cloud

This example demonstrates how to use the SAM D21 Xplained Pro board with a PubNub Data Stream Network and how it can be used to publish and subscribe messages between Atmel SAM D21 and PubNub.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware and software:

- The SAM D21 Xplained Pro.
- The ATWINC15x0 on EXT1 header.
- The Atmel IO1 Xplained Pro extension on EXT2.

Figure 5-15. PubNub Cloud Demo Setup

5.7.1 Execution

`main.c` - Initialize the board, connect to a PubNub Data Stream Network and communicate with the cloud.

1. Code summary:
   - Modify the following 3 string values in `main.h` file. Those must be set in the PubNub console.

```
/** PubNub settings. */
#define MAIN_PUBNUB_PUBLISH_KEY                         "demo" // "pub-c-
e71e5bed-0fee-4263-a843-c9de85c8825e"
#define MAIN_PUBNUB_SUBSCRIBE_KEY                       "demo" // "sub-
c-746522be-7e35-11e5-98ab-0619f8985a4f"
#define MAIN_PUBNUB_CHANNEL                 "WINC1500_00:00" /**< Do not change -
last digits will be updated with MAC address. */
```

   The user can create their publish key and subscribe key in the PubNub web page.

   - Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID        "DEMO_AP" /* < Destination SSID */
#define MAIN_WLAN_AUTH        M2M_WIFI_SEC_WPA_PSK /* < Security manner */
#define MAIN_WLAN_PSK         "12345678" /* < Password for Destination SSID */
```

   - The `main.c` file has all the code needed to do the system initialization by configuring the necessary components such as UART, temperature sensor, Wi-Fi driver and socket.

```
/* Initialize Socket API. */
socketInit();
registerSocketCallback(m2m_tcp_socket_handler, socket_resolve_cb);
...
/* Connect to AP using Wi-Fi settings from main.h. */
printf("main: Wi-Fi connecting to AP using hardcoded credentials...\r\n");
m2m_wifi_connect((char *)MAIN_WLAN_SSID, sizeof(MAIN_WLAN_SSID),
            MAIN_WLAN_AUTH, (char *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);

while (1) {
    m2m_wifi_handle_events(NULL)

    /* Device is connected to AP. */
    ....
    printf("main: subscribe event, PNR_OK\r\n");
    pubnub_subscribe(pPubNubCfg, PubNubChannel);
    }

    /* Process any received messages from the channel we subscribed. */
    while (1) {
        char const *msg = pubnub_get(pPubNubCfg);
    ....
        /* Any other type of JSON message. */
        printf("main: received message: %s\r\n", msg);
        ...
        pubnub_subscribe(pPubNubCfg, PubNubChannel);
```

- The first step is to initialize the PubNub client module by calling the function `pubnub_init()`. This function receives three parameters which are `publishkey`, `subscribe key` and a pointer to the context retrieved from the `pubnub_get_ctx` function.

```
/* Initialize PubNub API. */
printf("main: PubNub configured with following settings:\r\n");
printf("main:  - Publish key: \"%s\", Subscribe key: \"%s\", Channel: \"%s\".\r\n\r
\n",
PubNubPublishKey, PubNubSubscribeKey, PubNubChannel);
pPubNubCfg = pubnub_get_ctx(0);
pubnub_init(pPubNubCfg, PubNubPublishKey, PubNubSubscribeKey);
```

- In order to send the data from the temperature sensor on the I/O1 to the PubNub server, the function `pubnub_publish()` should be used as shown below:

```
/* Publish the temperature measurements periodically. */
if (gu32MsTicks - gu32publishDelay > MAIN_PUBNUB_PUBLISH_INTERVAL) {
    gu32publishDelay = gu32MsTicks;
    adc_start_conversion(&adc_instance);
    temperature = at30tse_read_temperature();
    adc_read(&adc_instance, &light);
    ...
    pubnub_publish(pPubNubCfg, PubNubChannel, buf);
```

- In order to subscribe to a Pubnub channel, we need to use `pubnub_subscribe()` and `pubnub_get()` functions as shown below:

```
char const *msg = pubnub_get(pPubNubCfg);
...
pubnub_subscribe(pPubNubCfg, PubNubChannel);
```

2. Build the program and download it into the board.
3. Start the application.
4. Once the SAM D21 is connected to the AP with internet connectivity, it will immediately connect to the PubNub server and will start sending the temperature data at regular intervals.

5.7.2 Demo Console Log

On the terminal window, enter the username through the terminal window.

```
-- WINC1500 pubnub cloud example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : 19.5.2 Svnrev 13445
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : 19.3.0
(APP)(INFO)Driver ver: 19.5.2
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
main: PubNub configured with following settings:

main:  - Publish key: "demo", Subscribe key: "demo", Channel: "WINC1500_F8:C6".

main: Wi-Fi connecting to AP using hardcoded credentials...

m2m_wifi_state: M2M_WIFI_RESP_CON_STATE_CHANGED: CONNECTED

m2m_wifi_state: M2M_WIFI_REQ_DHCP_CONF: IP is 172.16.1.105

main: subscribe event, PNR_OK

main: subscribe event, interval.

socket_resolve_cb: pubsub.pubnub.com resolved with IP 54.249.82.178
```

```
main: subscribe event, interval.

main: publish event: {{"device":"WINC1500_F8:C6", "temperature":"25.50",
"light":"95", "led":"0"}}

main: subscribe event, PNR_OK

main: received message: {"device":"WINC1500_F8:C6", "temperature":"25.50",
"light":"95", "led":"0"}

main: subscribe event, interval.

main: publish event: {{"device":"WINC1500_F8:C6", "temperature":"25.50",
"light":"96", "led":"0"}}

main: subscribe event, PNR_OK

main: subscribe event, interval.

main: received message: {"device":"WINC1500_F8:C6", "temperature":"25.50",
"light":"96", "led":"0"}

main: subscribe event, interval.

main: publish event: {{"device":"WINC1500_F8:C6", "temperature":"25.50",
"light":"96", "led":"0"}}
```

When the PubNub cloud console is configured as mentioned, then SAM D21 will get a message as shown below.

```
main: received message: {"device":"WINC1500_F9:A7", "temperature":"26.0",
"light":"95", "led":"0"}

main: subscribe event, interval.

main: received LED control message: {"led":"on"}

main: subscribe event, interval.

main: publish event: {{"device":"WINC1500_F9:A7", "temperature":"26.0", "light":"96",
"led":"1"}}

main: subscribe event, interval.

main: received LED control message: {"led":"off"}
```

5.8 Serial Bridge

This application is used to download the firmware into the ATWINC15x0 using the host MCU. This application resides in the host MCU, and the *image_downloader.exe/winc_programmer_UART.exe* tool communicates with the ATWINC15x0/ATWINC3400 using a serial bridge application.

- The SAM D21 Xplained Pro
- The ATWINC15x0/ATWINC3400 on the EXT1 header

It is possible to use this application to port into a different Microchip MCU or another vendor MCU as well.
Note: For more details on using the serial bridge application, refer to Integrated Serial Flash Memory Download Procedure.

5.9 SSL Client Connection

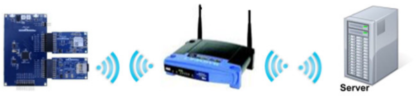
This example demonstrates how to set up an SSL connection.

It is based on the following hardware:

- The SAM D21 Xplained Pro

- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 5-16. SSL Client Demo Setup



5.9.1 Execution

`main.c` - Initialize the device and connect to a server using SSL.

1. Code summary.
   - Configure the network and server parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                          "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                          M2M_WIFI_SEC_WPA_PSK /**< Security type */
#define MAIN_WLAN_PSK                           "12345678" /**< Password for Destination
SSID */

/** All SSL defines */
#define MAIN_HOST_NAME                          "www.google.com"
#define MAIN_HOST_PORT                          443
```

   - Initialize the socket module and create the UDP server socket.

```
socketInit();
registerSocketCallback(socket_cb, resolve_cb);
```

   - Connect to the AP.

```
/* 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);
```

   - After the device is connected to the AP, the `gethostbyname()` function will be executed.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_REQ_DHCP_CONF:
    {
        /* Obtain the IP Address by network name */
        gethostbyname((uint8_t *)MAIN_HOST_NAME);
```

   - In the main loop, try to connect to the SSL server after resolving the domain host name.

```
if (sslConnect() != SOCK_ERR_NO_ERROR) {
    gu8SocketStatus = SocketInit;
}
```

   - If successfully connected, the `socket_cb()` function will be called with the SOCKET_MSG_CONNECT message.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    switch (u8Msg) {
    case SOCKET_MSG_CONNECT:
        if (pstrConnect && pstrConnect->s8Error >= SOCK_ERR_NO_ERROR)
            printf("Successfully connected.\r\n");
    break;
```

   - You can receive data in the `socket_cb()` function with the `SOCKET_MSG_RECVFROM` message when a client device sends data. (Use the “UDP Client” example.)

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    ...
    } else if (u8Msg == SOCKET_MSG_RECVFROM) {
    tstrSocketRecvMsg *pstrRx = (tstrSocketRecvMsg *)pvMsg;
    if (pstrRx->pu8Buffer && pstrRx->s16BufferSize) {
        printf("socket_cb: received app message.(%u)\r\n", packetCnt);
    /* Prepare next buffer reception. */
    recvfrom(sock, gau8SocketTestBuffer, MAIN_WIFI_M2M_BUFFER_SIZE, 0);
```

2. Build the program and download it into the board.
3. Start the application.

Note: To set up an SSL connection, a root certificate must be installed.

5.9.2 SSL Client Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window. The device is connected to a server using SSL.

```
-- WINC1500 SSL example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxx
socket_cb: bind success!
Host IP is 173.194.127.115
Host Name is www.google.com
Successfully connected
```

5.10 Weather Client

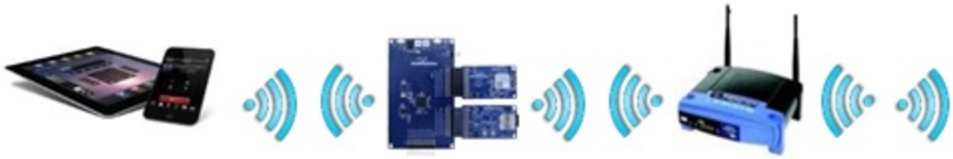
This example demonstrates the use of the ATWINC15x0 with the SAM D21 Xplained Pro board to retrieve weather information from a weather server (openweathermap.org).

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header

Figure 5-17. Weather Client Demo Setup



5.10.1 Execution

`main.c` - Initialize the chip and retrieve information.

1. Code summary:
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                    "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                    M2M_WIFI_SEC_WPA_PSK /**< Security manner
*/
#define MAIN_WLAN_PSK                     "12345678" /**< Password for Destination
SSID */
```

   – Initialize the socket module and register the socket callback function.

```
/* 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 socket module */
socketInit();
registerSocketCallback(socket_cb, NULL);
```

– Get the MAC address and set the device name with the MAC address.

```
m2m_wifi_get_mac_address(gau8MacAddr);

set_dev_name_to_mac((uint8_t *)gacDeviceName, gau8MacAddr);
set_dev_name_to_mac((uint8_t *)gstrM2MAPConfig.au8SSID, gau8MacAddr);
m2m_wifi_set_device_name((uint8_t *)gacDeviceName, ...);
```

– Start the Provision mode.

```
m2m_wifi_start_provision_mode((tstrM2MAPConfig *)&gstrM2MAPConfig,
        (char *)gacHttpProvDomainName, 1);
```

– When the mobile device sends configuration information, the `wifi_cb()` function is called with the `M2M_WIFI_RESP_PROVISION_INFO` message and the user can connect to the AP with the given information.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    case M2M_WIFI_RESP_PROVISION_INFO:
        tstrM2MProvisionInfo *pstrProvInfo = (tstrM2MProvisionInfo *)pvMsg;
        if (pstrProvInfo->u8Status == M2M_SUCCESS) {
        m2m_wifi_connect((char *)pstrProvInfo->au8SSID,
                strlen((char *)pstrProvInfo->au8SSID),
                pstrProvInfo->u8SecType,
                pstrProvInfo->au8Password, M2M_WIFI_CH_ALL);
    }
```

– After the device is connected to the AP, the `gethostbyname()` function is called.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
    ...
    case M2M_WIFI_REQ_DHCP_CONF:
    {
    ...
    gbConnectedWifi = true;
    gethostbyname((uint8_t *)MAIN_WEATHER_SERVER_NAME);
    }
    ...
}
```

– Create a TCP client socket and connect to the server in the main loop.

```
if (gbConnectedWifi && !gbTcpConnection) {
    if (gbHostIpByName) {
    if (tcp_client_socket < 0) {
        if ((tcp_client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        continue;
            }
        }
    if (connect(tcp_client_socket, ...) != SOCK_ERR_NO_ERROR) {
        continue;
    }
    gbTcpConnection = true;
    }
}
```

– The socket callback function receives the `SOCKET_MSG_CONNECT` message and then it requests weather information from the server with a city name.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    case SOCKET_MSG_CONNECT:
    sprintf(gau8ReceivedBuffer, ..., MAIN_CITY_NAME, ...);
    ...
    send(tcp_client_socket, gau8ReceivedBuffer, ...);
    recv(tcp_client_socket, (struct sockaddr *)&addr_in, sizeof(struct sockaddr_in));
    break;
}
```

– The socket callback function receives a `SOCKET_MSG_RECV` message with weather information. The user can also get the current temperature via an IO1 sensor board as shown in the following.

```
static void socket_cb(SOCKET sock, uint8_t u8Msg, void *pvMsg)
{
    case SOCKET_MSG_RECV:
    ...
    /** From the received Josan format buffer City Name, Temperature, Weather Number 
will be parsed */
    /* Get city name. */
    pcIndxPtr = strstr((char *)pstrRecv->pu8Buffer, "name=");
    printf("City: ");
    ...
    /* Get temperature. */
    pcIndxPtr = strstr(pcEndPtr + 1, "temperature value");
    printf("Temperature: ");
   ...
    /* Get weather condition. */
    pcIndxPtr = strstr(pcEndPtr + 1, "weather number");
    if (NULL != pcIndxPtr) {
        printf("Weather Condition: ");

}
```

2. Build the program and download it into the board.
3. Start the application.
4. Connect the mobile device to the ATWINC15x0/ATWINC3400 AP [WINC1500_XX:XX].
5. Browse the web page (www.microchip.com) to setup the AP, populate the page and press Connect.
6. The ATWINC15x0/ATWINC3400 will be connected to the selected AP.
7. The weather information will be printed.

Note: If the server connection is unstable, this example may not operate normally.

5.10.2 Weather Client Demo Console Log

The application must be programmed and running. The following information is displayed on the terminal window.

```
-- WINC1500 weather client example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Provision Mode started.
Connect to [atmelconfig.com] via AP[WINC1500_08:CA] and fill up the page.
Wi-Fi IP is xxx.xxx.xxx.xxx
wifi_cb: M2M_WIFI_RESP_PROVISION_INFO.
Wi-Fi connected
Wi-Fi IP is xxx.xxx.xxx.xxx
Host IP is 144.76.102.166
Host Name is openweathermap.org
City: Seoul
Weather Condition: sky is clear

Temperature from sensor : 27 degrees
Temperature from server : 3 degrees
Temperature difference : 24 degrees
```

5.11 Wi-Fi Serial

This example demonstrates how to emulate serial ports between two devices. It reads input data from the serial interface and sends it via a Wi-Fi connection and the terminal window will display the messages which you typed or received. It can be useful for chatting or controlling a remote device. It is based on the hardware listed below, and the user needs to prepare two pairs of SAM D21 and ATWINC15x0 boards.

Note: This example application is not available in ASF for ATWINC3400.

It is based on the following hardware:

- The SAM D21 Xplained Pro
- The ATWINC15x0 on the EXT1 header

Figure 5-18. Wi-Fi Serial Demo Setup

5.11.1 Execution

`main.c` - Initialize the device and the USART interface. Create the TCP sockets, send/receive messages and display them on the terminal window.

1. Code summary.
   – Configure the network parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                          "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                          M2M_WIFI_SEC_WPA_PSK /**< Security type */
#define MAIN_WLAN_PSK                           "12345678" /**< Password for Destination
SSID */
```

   – Configure the USART module to read user input data.

```
static void configure_console(void)
{
    ...
    stdio_serial_init(&cdc_uart_module, CONF_STDIO_USART_MODULE,&usart_conf);
    /* Register USART callback for receiving user input. */
    usart_register_callback(&cdc_uart_module, uart_callback,
SART_CALLBACK_BUFFER_RECEIVED);
    usart_enable_callback(&cdc_uart_module, USART_CALLBACK_BUFFER_RECEIVED);
    usart_enable(&cdc_uart_module);
}
```

   – Initialize the socket module and register the socket callback function.

```
/* Initialize socket module */
socketInit();
registerSocketCallback(socket_cb, resolve_cb);
```

   – Connect to the AP.

```
/* 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);
```

   – After the device is connected to the AP, create the TCP server socket and bind it in the main loop.

```
if ((tcp_server_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    continue;
}
...
bind(tcp_server_socket, (struct sockaddr *)&addr, sizeof(struct sockaddr_in));
```

   – If there is user input data, then the `handle_input_message()` function in the main loop calls the `parse_command()` function to parse the user data and execute the handler function according to the command.

```
uint8_t parse_command(char *buffer, uint8_t remote)
{
    for (i = i_st; i < i_ed; i++) {
        if (!strcmp(cmd_list[i].cmd_string, cmd_buf)) {
            cmd_list[i].cmd_handler(buffer + strlen(cmd_list[i].cmd_string) + 1);
            break;
            }
        }
```

   – The `handle_input_message()` function prints user data in the terminal window and sends it to the remote device.

```
void handle_input_message(void)
{
    ...
    if (tcp_connected == 1) {
        PRINT_LOCAL_MSG(uart_buffer);
        send(tcp_client_socket, uart_buffer, msg_len + 1, 0);
    }
```

   – When receiving data from the remote device, it handles the received message to display it or execute a command.
   – There are several commands for Wi-Fi serial functionality in this example.

2. Build the program and download it into the board.
3. Start the application.
4. Check the IP address of each board and execute the connection on one device by typing the `<<connect xxx:xxx:xxx:xxx` command on the terminal window with the other device's address. Use prefix "<<" to execute local commands.
5. When connected, the `socket_cb:connect success` message will appear:
6. Type messages on the terminal window and you will see the sent/received messages.
7. You can control the LED on the remote device by typing the `control ledon (Or) control ledoff` command. Use prefix ">>" to execute remote commands.
8. The application reads data from the serial interface. User commands can be modified to execute various actions.

5.11.2 Wi-Fi Serial Demo Console Log

The application must be programmed and running. The following information will be displayed on the terminal window.

```
-- WINC1500 Wi-Fi Serial example --
-- XXXXX_XPLAINED_PRO --
-- Compiled: xxx  x xxxx  xx:xx:xx --
(APP)(INFO)Chip ID 1503a0
(APP)(INFO)Firmware ver   : xx.x.x Svnrev xxxxx
(APP)(INFO)Firmware Build xxx  x xxxx  Time xx:xx:xx
(APP)(INFO)Firmware Min driver ver : xx.x.x
(APP)(INFO)Driver ver: xx.x.x
(APP)(INFO)Driver built at xxx  x xxxx  xx:xx:xx
Connecting to DEMO_AP.
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED : CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF : IP is xxx.xxx.xxx.xxxsocket_cb: bind success!
socket_cb: bind success.
socket_cb: listen success.
<<connect xxx.xxx.xxx.xxx
(Local device)
Connecting to [xxx.xxx.xxx.xxx] ...
socket_cb: connect success.
(Remote device)
socket_cb: accept success.
>>control ledon
(Or)
>>control ledoff
```

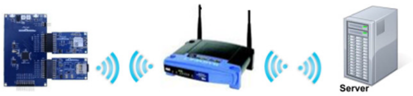
5.12 ALPN Client Connection

This example demonstrates how to negotiate securely to a protocol (HTTP/2(preferred) or HTTP/1.1) using TLS extension ALPN.

It is based on the following hardware:

- The SAM D21 Xplained Pro Board
- The ATWINC15x0/ATWINC3400 on the EXT1 header

Figure 5-19. ALPN Client Demo Setup



5.12.1 Execution

`main.c` - Initialize the device and connect to a server using SSL.

1. Code summary
   – Configure the network and server parameters in `main.h`.

```
/** Wi-Fi Settings */
#define MAIN_WLAN_SSID                  "DEMO_AP" /**< Destination SSID */
#define MAIN_WLAN_AUTH                  M2M_WIFI_SEC_WPA_PSK /**< Security type */
#define MAIN_WLAN_PSK                   "12345678" /**< Password for Destination
SSID */

/** All SSL defines */
#define MAIN_HOST_NAME                  "www.google.com"
#define MAIN_HOST_PORT                  443
```

   – Initialize the socket module and create the UDP server socket.

```
socketInit();
registerSocketCallback(socket_cb, resolve_cb);
```

   – Connect to the AP.

```
/* Connect to AP. */
m2m_wifi_connect((char *)MAIN_WLAN_SSID, sizeof(MAIN_WLAN_SSID),
        MAIN_WLAN_AUTH, (char *)MAIN_WLAN_PSK, M2M_WIFI_CH_ALL);
```

   – After the device is connected to the AP, the `gethostbyname()` function will be executed.

```
static void wifi_cb(uint8_t u8MsgType, void *pvMsg)
{
      case M2M_WIFI_REQ_DHCP_CONF:
      {
            /* Obtain the IP Address by network name */
            gethostbyname((uint8_t *)MAIN_HOST_NAME);
```

   – Ensure the following configuration in `sslConnect()`:
   Call `socket()` with the SOCKET_FLAGS_SSL bit set in u8Flags parameter.

```
/* Create secure socket */
    if (tcp_client_socket < 0) {
        tcp_client_socket = socket(AF_INET, SOCK_STREAM, SOCKET_FLAGS_SSL);
        }
```

   – Configure the SNI for the socket via `setsockopt()` and option SO_SSL_SNI.

```
/*Configure the SNI for the socket */
setsockopt(tcp_client_socket, SOL_SSL_SOCKET, SO_SSL_SNI, MAIN_HOST_NAME,
sizeof(MAIN_HOST_NAME));
```

   – Call `set_alpn_list()` with parameter pointing to "h2 http/1.1" (this means that HTTP/2 and HTTP/1.1 are both supported).

```
/* Sets the protocol list to be used for ALPN */
set_alpn_list(tcp_client_socket,"h2 http/1.1");
```

   – After the configuration is applied, try to connect to the SSL server by calling `sslConnect()`.

```
if (sslConnect() != SOCK_ERR_NO_ERROR) {
    gu8SocketStatus = SocketInit;
}
```

- If successfully connected, the `socket_cb()` function will be called with the SOCKET_MSG_CONNECT message. The user can check for the ALPN negotiation type in the `socket_cb()` function with the `get_alpn_index()` function.

```
/* Check for ALPN negotiation type. */
          switch (alpn_index)
          {
        case 1:
        printf("Negotiated HTTP/2.\r\n");
        break;
        case 2:
        printf("Negotiated HTTP/1.1.\r\n");
        break;
        case 0:
        printf("Protocol negotiation did not occur.\r\n");
        break;
          }
```

2. Build the program and download it into the board.
3. Start the application.
   Note: To set up an SSL connection, a root certificate must be installed.

5.12.2 ALPN Client Connection Console Log

The application must be programmed and running. The following information is displayed on the terminal window. The device is connected to a server using TLS ALPN extension.

```
-- ALPN example --
-- XXXXXX_XPLAINED_PRO --
-- Compiled: xxx xx xxxx xx:xx:xx --
(APP)(INFO)Chip ID 3400d2
(APP)(INFO)Curr driver ver: x.x.x
(APP)(INFO)Curr driver HIF Level: (2) x.x
(APP)(INFO)Fw HIF: xxxx
(APP)(INFO)Firmware HIF (2) : x.x
(APP)(INFO)Firmware ver   : x.x.x
(APP)(INFO)Firmware Build xxx xx xxxx Time xx:xx:xx
(APP)(INFO)Ota HIF: 0000
(APP)(INFO)No valid Ota image
wifi_cb: M2M_WIFI_RESP_CON_STATE_CHANGED: CONNECTED
wifi_cb: M2M_WIFI_REQ_DHCP_CONF: IP is xxx.xxx.xxx.xxx
Host IP is 172.217.163.68
Host Name is www.google.com
(APP)(INFO)Socket 0 session ID = 1
Successfully connected.
Negotiated HTTP/2.
```

6. Reference Documentation

The following reference documentation can be used to ease integration and device ramp. Refer to the Documentation Web Page for the latest documents.

- BLE Example Profiles Applications User Guide
- Certificates Update from Host via OTA(HTTPS) User Guide
- Iperf Application Note
- AWS IoT Demo with RSA Application Note
- Power Saving User Guide
- Knowledge Base (FAQs to debug/ understand the winc driver/ applications) at microchip.secure.force.com/CustomerCommunity/hottopics

7. Document Revision History

Revision	Date	Section	Description
D	09/2020	• 3.1.4.1 Execution • 4.3.1 Execution • 4.4.1 Execution • 6. Reference Documentation	• Added note for "Connect to the AP with the given information" and description for "Code summary". • Added more description for "Connect, send and recv operations will be executed sequentially in the socket_cb() function". • Added point 4 for "Execution" of TCP server. • Added Knowledge Base reference.
C	09/2019	Advanced Examples	Added reference for ALPN, Enterprise security and Simple roaming.
B	02/2018	• Document • MQTT Chat • Basic Operation code	• Initial release for ATWINC3400 • Revision of configuration codes • Revision of basic example codes of the demo console logs
A	04/2017	Document	• Updated from Atmel to Microchip template. • Assigned a new Microchip document number. Previous version is Atmel 42418 revision B. • ISBN number added.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-6661-1

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra’anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820