

AT15058: ATWINC1500 Weather Client Example Application

APPLICATION NOTE

Introduction

This application note describes the Weather Client demo application to retrieve weather information from a weather server (openweathermap.org) using Atmel® ATWINC1500 Wi-Fi® Network Controller.

Following topics are covered:

- Hardware setup
- Implementation of the Weather Client application
- Flow of the Weather Client application
- Openweathermap API features
- Extending the demo application

Features

- Weather client example based on ATWINC1500 and ATSAMW25 devices
- TCP socket connection
- Weather data fetched based on City name, ID, Zip code, and coordinates
- JSON, XML, and HTML formats
- Multilingual support
- Supports openweathermap APIs

Table of Contents

Int	roduc	etion	1
Fe	ature	S	1
1.	Ove	rview	3
2.	Hard	dware Setup	4
3.	Wea	ther Client Implementation	
	3.1.	Wi-Fi Host Driver Initialization	5
	3.2.	Socket Layer Callback Registration	5
	3.3.	HTTP Provisioning Mode Initialization	
	3.4.	Wi-Fi Host Driver Event Handling	
	3.5.	Wi-Fi Status Callback Handling	
	3.6.	Resolving Server IP from Domain Name	
	3.7.	Connecting to a TCP Server	
	3.8.	Socket Callback Handling	
	3.9.	Closing TCP Client Socket	12
4.	Pars	sing Response from Weather Server	13
5.	Gett	ing Started with ASF Weather Client Application	15
6.	Extending Weather Client Demo Application		
	6.1.	Methods to Fetch Current Weather for a Location	19
	6.2.	Methods to Change Unit of Temperature	20
	6.3.	Method to Change Language of Weather Data	21
	6.4.	Method to Change Weather Data Format	21
7.	Revi	ision History	23



1. Overview

The Weather Client demo application demonstrates retrieving of weather information from the Weather Server (openweathermap.org) by using the Atmel ATWINC1500 Wi-Fi Network Controller. In this application the HTTP provisioning method is used to configure the credentials of ATWINC1500. Provision mode is a facility to configure the SSID and Passphrase of the Access Point (AP) to the Wi-Fi client remotely.

The Weather Client demo has the following capabilities:

- ATWINC1500 starts in Access Point mode and any android mobile or laptop with Wi-Fi capability can be connected
- After successful connection, the HTTP provisioning page (atmelconfig.com) can be loaded to configure the ATWINC1500 credentials
- ATWINC1500 will disable the Access Point mode and initialize in the Station mode
- Based on the configured credentials, the ATWINC1500 searches and joins the desired Access Point
- After successful connection with the Access Point, the ATWINC1500 obtains the IP address from the DHCP server
- The DNS client in ATWINC1500 will resolve the Weather Server address (api.openweathermap.org)
- Creates TCP socket to send HTTP client requests
- Receives data message from the Weather HTTP Server
- Based on the HTTP response message format the weather information is parsed



2. Hardware Setup

Weather Client demo application uses one of the following hardware:

Option 1:

Atmel SAMD21 Xplained Pro + ATWINC1500 XPRO on EXT1 header

Option 2:

- Atmel SAM W25 Xplained Pro board
- A wireless access point with internet connectivity
- A mobile device for provisioning

Figure 2-1. Demo Setup





3. Weather Client Implementation

This section explains the implementation of the weather client example application.

3.1. Wi-Fi Host Driver Initialization

 System initialization will initialize the MCU's clock, hardware, events, and external hardware interfaces, if available.

```
/* Initialize the board. */
    system_init();
```

• The console UART interface should be configured for debug print output. The debug log level value can be set by using the M2M LOG LEVEL macro in the nm debug.h file.

```
/* Initialize the UART console. */
  configure_console();
  printf(STRING_HEADER);
```

In the BSP driver initialization the ATWINC1500 bring-up sequence will be followed. The sequence
of chip enable and reset pin of ATWINC1500 is to be followed. Refer the nm_bsp_init API
definition.

```
/* Initialize the BSP. */
   nm_bsp_init();
```

- The Wi-Fi host driver initialization starts with API m2m_wifi_init() and structure tstrWifiInitParam filled with appropriate information. The Wi-Fi Initialization sequence will configure the SPI communication interface and external interrupt with respect to the host MCU and ATWINC1500 pin connections. Apart from this Wi-Fi host application layer callback function, wifi cb() will also be registered with the initialization sequence.
- After successful initialization of communication interface between the ATWINC1500 and host MCU, ATWINC1500 is ready to accept the Wi-Fi initialization and configuration commands of the WLAN module. In this process, the ATWINC1500 WLAN module will be reset by the host MCU and wait for the confirmation from the module firmware.
- The Wi-Fi application callback function indicates the events such as connect and disconnect status.
 The function operates by obtaining the IP address from the DHCP server with respect to the DHCP request from the ATWINC1500 station. When ATWINC1500 act as softAP, providing the IP address to the WLAN client device will also be notified.

```
/* Initialize Wi-Fi parameters structure. */
   memset((uint8_t *)&param, 0, sizeof(tstrWifiInitParam));

/* Initialize Wi-Fi driver with data and status callbacks. */
   param.pfAppWifiCb = wifi_cb;
   ret = m2m wifi_init(&param);
   if (M2M_SUCCESS != ret) {
        printf("main: m2m_wifi_init call error!(%d)\r\n", ret);
        while (1) {
        }
   }
}
```

3.2. Socket Layer Callback Registration

During socket layer initialization, a general IP layer event callback and specific socket layer event handling callback function are registered. A general IP layer callback function is triggered whenever the host application receives an event from the TCP/IP stack of the ATWINC1500 WLAN module. Corresponding event specific data is read from the WLAN module.



After reading the socket event data from the ATWINC1500, the socket layer callback function is called. While initializing the socket, the <code>sockaddr_in</code> structure is filled with address family corresponding to IPv4 or IPv6, port number, and IP address.

Code Snippet:

```
/* 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);
```

3.3. HTTP Provisioning Mode Initialization

The ATWINC1500 HTTP provisioning mode is started by calling the asynchronous Wi-Fi provisioning function $m2m_wifi_start_provision_mode()$. This function triggers the ATWINC1500 to activate the Wi-Fi AP (hotspot) mode with the passed configuration parameters and then starts the HTTP provision web server. The provisioning status is returned through the $M2M_WIFI_RESP_PROVISION$ INFO event.

Table 3-1. m2m_wifi_start_provision_mode parameters

Direction	Parameter	Description
In	pstrAPConfig	AP configuration parameters as defined in tstrM2MAPConfig configuration structure. A NULL value passed in, will result in a negative error M2M_ERR_FAIL.
In	pcHttpServerDoma inName	Domain name of the HTTP Provision server which others will use to load the provisioning Home page. For example "atmelconfig.com".
in	bEnableHttpRedir ect	A flag to enable/disable the HTTP redirect feature. Possible values are: 0:Disable, 1:Enable

This function returns M2M_SUCCESS for successful operations and a negative value otherwise. After starting the HTTP provisioning mode, the provisioning server domain name and the access point's SSID are printed to the serial output terminal.

Code Snippet:

```
/* Start web provisioning mode. */
m2m_wifi_start_provision_mode((tstrM2MAPConfig *)&gstrM2MAPConfig, (char
*)gacHttpProvDomainName, 1);
```

3.4. Wi-Fi Host Driver Event Handling

The Host Interface Layer (HIF) interrupts the host MCU when one or more events are pending in ATWINC1500 firmware. In order to receive event callbacks, the host MCU application needs to call the $m2m_wifi_handle_events$ () to let the host driver retrieve and process the pending events from the ATWINC1500 firmware. It is recommended to call this function either:

- In the main loop of the application or in a dedicated task in the host MCU or
- At least once when host MCU receives an interrupt from WINC firmware



Code Snippet:

m2m wifi handle events(NULL);

3.5. Wi-Fi Status Callback Handling

M2M_WIFI_RESP_PROVISION_INFO

When the mobile device transmits the configuration information, wifi_cb() function will be called with M2M_WIFI_RESP_PROVISION_INFO message and you can connect to the AP with the given information. The access point's SSID, the security type, and the authentication information parameters are retrieved from provisioning mode callback. Connection will be established with the specific AP by calling asynchronous m2m wifi connect Wi-Fi connection function.

Table 3-2. m2m_wifi_connect parameters

Direction	Parameter	Description	
In	pcSsid	A buffer holding the SSID corresponding to the requested AP.	
In	u8SsidLen	Length of the given SSID (not including the NULL termination). A length less than ZERO or greater than the maximum defined SSID M2M_MAX_SSID_LEN will result in a negative error M2M_ERR_FAIL.	
In	u8SecType	Wi-Fi security type.	
		The parameter can be belong to one of the following types: -M2M_WIFI_SEC_OPEN -M2M_WIFI_SEC_WEP -M2M_WIFI_SEC_WPA_PSK -M2M_WIFI_SEC_802_1X A value outside these possible values will result in a negative return error M2M_ERR_FAIL.	
In	pvAuthInfo	Authentication parameters required for completing the connection. Its type is based on the Security type. If the authentication parameters are NULL or are greater than the maximum length of the authentication parameters length as defined by M2M_MAX_PSK_LEN, a negative error will return M2M_ERR_FAIL(-12) indicating connection failure.	
In	u16Ch	Wi-Fi channel number as defined in tenuM2mScanCh enumeration. Channel number greater than M2M_WIFI_CH_14 returns a negative error M2M_ERR_FAIL(-12), except if the value is M2M_WIFI_CH_ALL(255), since this indicates that the firmware should scan all channels to find the SSID requested to connect to. Failure to find the connection match will return a negative error M2M_DEFAULT_CONN_SCAN_MISMATCH.	



Code Snippet:

M2M_WIFI_RESP_CON_STATE_CHANGED

The connection status is known when a response of M2M_WIFI_RESP_CON_STATE_CHANGED is received based on the states defined in tenuM2mConnState. Successful connection is defined by M2M_WIFI_CONNECTED. DHCP Client configuration is requested automatically after a successful Wi-Fi connection is established.

Note: The m2m_wifi_request_dhcp_client function call is legacy implementation. Will be removed in the future.

Code Snippet:

M2M WIFI REQ DHCP CONF

When the DHCP client in the ATWINC1500 firmware obtains an IP address from the local Access Point (AP), the wifi_cb() callback is invoked with the M2M_WIFI_REQ_DHCP_CONF event type. The obtained IP address is printed in the serial output terminal.

The weather server domain name is resolved by calling the <code>gethostbyname()</code> asynchronous Domain Name Server (DNS) resolving function.

Table 3-3. gethostbyname Parameter

Direction	Parameter	Description
In	pcHostName	NULL terminated string containing the domain name for the remote host. Its size must not exceed HOSTNAME_MAX_SIZE.

Code Snippet:



3.6. Resolving Server IP from Domain Name

The <code>gethostbyname()</code> function use the DNS client to resolve a domain name into the corresponding IP address. The DNS response will be delivered to the DNS callback function <code>resolve_cb()</code> which was registered by using the <code>registerSocketCallback()</code> function.

In the callback function the Weather Server Host name and the corresponding Host IP information are printed to serial output terminal.

Code Snippet:

```
static void resolve_cb(uint8_t *hostName, uint32_t hostIp)
{
    printf("resolve_cb: %s IP address is %d.%d.%d.%d\r\n\r\n", hostName,
    (int)IPV4_BYTE(hostIp, 0), (int)IPV4_BYTE(hostIp, 1),
     (int)IPV4_BYTE(hostIp, 2), (int)IPV4_BYTE(hostIp, 3));
}
```

3.7. Connecting to a TCP Server

A TCP client socket is the logical end-point to establish a connection with the TCP server. A socket can be created by using the <code>socket()</code> API.

```
/** Socket for client */
static SOCKET tcp_client_socket = -1;

/* 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;
}</pre>
```

- The first argument, AF_INET indicates that only IPv4 transport addresses are supported by the ATWINC1500 firmware
- The second argument, SOCK_STREAM specifies that this is a TCP socket in contrast to the SOCK_DGRAM for UDP sockets
- The third argument specifies the flag value. It shall be left to 0 for normal TCP sockets and if a secure socket is to be created, the argument shall be populated with value SOCKET_FLAGS_SSL.

After the TCP socket is created, the socket ID (in this case, tcp_client_socket) is used by most other socket functions and is also passed as an argument to the socket event callback function to identify which socket generated the event.

Connection to the TCP server is initiated through the connect () API.

```
/* Connect server */
ret = connect(tcp_client_socket, (struct sockaddr *)&addr, sizeof(struct sockaddr_in));
```

The connect () API uses the socket ID (tcp_client_socket) and the address of the TCP server to connect with the remote TCP server.

3.8. Socket Callback Handling

In weather client socket callback function, we check for different socket events such as SOCKET_MSG_CONNECT and SOCKET_MSG_RECV.



SOCKET MSG CONNECT

The SOCKET_MSG_CONNECT event callback provides a tstrSocketConnectMsg structure containing an error code.

- If the error code is 0 then the connection to the remote host is successful. Now the application can send and receive data through the socket.
- If the error code is a negative value, it indicates an error due to a timeout condition. The socket is closed after printing the error message.

SEND function:

An asynchronous function, send (SOCKET sock, void *pvSendBuffer, uint16 u16SendLength, uint16 u16Flags) is called by the application to send data containing the HTTP GET request.

Table 3-4, send Parameters

Direction	Parameter	Description
In	sock	Socket ID, must hold a non-negative value. A negative value will return a socket error SOCK_ERR_INVALID_ARG. Indicating that an invalid argument is passed in.
In	pvSendBuffer	Pointer to a buffer holding data to be transmitted.
In	u16SendLength	The buffer size in bytes.
In	u16Flags	Not used in the current implementation.

RECEIVE function:

An asynchronous receive function, recv (SOCKET sock, void *pvRecvBuf, uint16 u16BufLen, uint32 u32Timeoutmsec) is used to retrieve data from a TCP stream. The application receives the required data in response to this asynchronous call through the reception of the event SOCKET MSG RECV in the socket callback.

Table 3-5. recv Parameters

Direction	Parameter	Description
In	sock	Socket ID, must hold a non-negative value. A negative value will return a socket error SOCK_ERR_INVALID_ARG, indicating that an invalid argument is passed in.
In	pvRecvBuf	Pointer to a buffer that will hold the received data. The buffer is used in the recv callback to deliver the received data to the caller. The buffer must be resident in memory (heap or global buffer).
In	u16BufLen	The buffer size in bytes.
In	u32Timeoutmse	Timeout for the recv function in milliseconds. If the value is set to ZERO, the timeout will be set to infinite (the recv function waits forever). If the timeout period is elapsed with no data received, the socket will get a timeout error.



Code Snippet:

SOCKET_MSG_RECV

In SOCKET_MSG_RECV event, the tstrSocketRecvMsg structure is parsed to receive information from the weather server.

If the received data from the weather server is larger than the user buffer size defined during the asynchronous call to the recv function, then the data will be delivered to the user in a number of consecutive chunks according to the user buffer size. If the user buffer size contains a negative or zero value then it indicates an error.

The received message from the weather server is then parsed to get the weather information.

Table 3-6. tstrSocketRecvMsg Structure

Data field	Description	
uint8* pu8Buffer	Pointer to the user buffer (passed to recv function) containing the received data chunk.	
sint16 s16BufferSize	The received data chunk size. Holds a negative value if there is a receive error or ZERO on success upon reception of close socket message.	
uint16 u16RemainingSize	The number of bytes remaining in the current recv operation.	
struct sockaddr_in strRemoteAddr	Socket address structure for the remote peer. It is valid for SOCKET_MSG_RECVFROM event.	

Code Snippet:



3.9. Closing TCP Client Socket

Asynchronous close function will release all the resources assigned to the socket.

Code Snippet:

```
close(tcp_client_socket);
```

Table 3-7. close parameter

Direction	Parameter	Description
In	sock	Socket ID, must hold a non-negative value. A negative value will return a socket error SOCK_ERR_INVALID_ARG, indicating that an invalid argument is passed in.



4. Parsing Response from Weather Server

HTTP Request to Server:

Buffer gau8ReceivedBuffer used in function send() contains below HTTP GET Request which will be sent to weather server.

```
"GET /data/2.5/weather?q=Paris&appid=APIKEY&mode=xml&units=metric HTTP/1.1\r\nHost: api.openweathermap.org\r\nAccept: */*\r\n\r\n"
```

HTTP Response from Server:

```
<current>
   <city id="2988507" name="Paris">
        <coord lon="2.35" lat="48.85"/>
       <country>FR</country>
       <sun rise="2015-12-22T07:41:47" set="2015-12-22T15:56:42"/>
   </city>
   <temperature value="11.4" min="10" max="12.22" unit="metric"/>
   <humidity value="76" unit="%"/>
   sure value="1027" unit="hPa"/>
   <wind>
       <speed value="5.1" name="Gentle Breeze"/>
       <direction value="200" code="SSW" name="South-southwest"/>
   </wind>
   <clouds value="75" name="broken clouds"/>
   <visibility/>
   cipitation mode="no"/>
   <weather number="803" value="broken clouds" icon="04d"/>
   <lastupdate value="2015-12-22T10:04:46"/>
</current>
```

Based on this server response, in the embedded code we search for keywords such as "name=" for moving below pointers,

- The pcIndxPtr pointer is moved to the start of a city name for example, address of P in Paris
- The pcEndPtr pointer is moved to the address next to the end of a city name for example, address
 of S+1 in Paris and the value is written as 0

Printing string value from address pcIndxPtr displays the corresponding city name. Similarly, temperature value and weather conditions are parsed and printed on the serial output terminal.

```
/* Get city name. */
pcIndxPtr = strstr((char *)pstrRecv->pu8Buffer, "name=");
printf("City: ");
if (NULL != pcIndxPtr) {
    pcIndxPtr = pcIndxPtr + strlen("name=") + 1;
    pcEndPtr = strstr(pcIndxPtr, "\">");
if (NULL != pcEndPtr) {
         *pcEndPtr = 0;
    printf("%s\r\n", pcIndxPtr);
     } else {
         printf("N/A\r\n");
         break;
    /* Get temperature. */
    pcIndxPtr = strstr(pcEndPtr + 1, "temperature value");
    printf("Temperature: ");
    if (NULL != pcIndxPtr) {
    pcIndxPtr = pcIndxPtr + strlen("temperature value") + 2;
pcEndPtr = strstr(pcIndxPtr, "\" ");
    if (NULL != pcEndPtr) {
         *pcEndPtr = 0;
```



```
printf("%s\r\n", pcIndxPtr);
} else {
   printf("N/A\r\n");
   break;
}
```



5. Getting Started with ASF Weather Client Application

This topic explains the steps for demonstrating ATWINC1500 projects using Atmel Studio ASF example applications.

- Open Atmel Studio 7 File > New > Example Projects.
- 2. In the **Search for Example Projects** type Weather client.
- Select and Open WINC1500 Weather Client Example SAM D21 Xplained Pro. Figure 5-1. Atmel Studio ATWINC1500 Project Creation

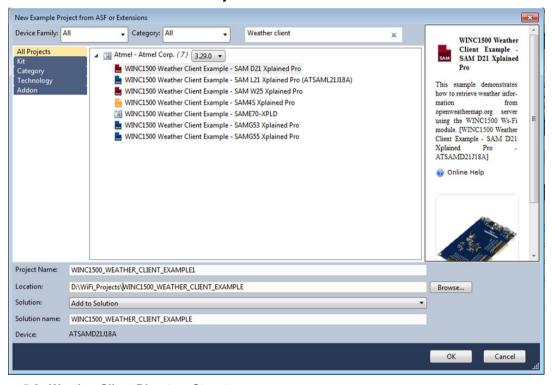
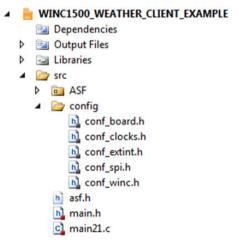


Figure 5-2. Weather Client Directory Structure





4. The ATWINC1500 will start as a softAP using open security mode (no security method) and broadcasts the beacon frames with SSID WINC1500_00:00. The following macros are defined in the main.h file.

- Register to Openweathermap to get the API key.
- 6. Enter the API key by replacing APIKEY in the MAIN POST BUFFER definition.

```
#define MAIN_POST_BUFFER "&appid= APIKEY
```

- 7. Open the serial port terminal application by using the COM port configuration 115200,8,none, 1,none.
- 8. Compile and download the image into the SAM D21 XPRO board.
- 9. Run the application. Success or error messages will appear in the serial port terminal.
- 10. The ATWINC1500 softAP is now available and can be connected to a mobile device or any WLAN client supported devices.
- 11. Connect a mobile to the ATWINC1500 softAP and after a successful connection, the HTTP configuration web page will be triggered automatically.

Figure 5-3. HTTP Web Triggering after AP Connection



12. Refresh to scan all the available APs or routers. This scan results will be provided through the ATWINC1500 HTTP server to the provisioning page.



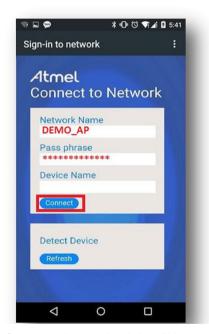
Figure 5-4. ATWINC1500 Provisioning Page





13. Enter the AP credentials to connect with the specific AP.

Figure 5-5. Entering AP Credentials





- 14. When pressing Connect, the credentials will be transferred to the ATWINC1500 HTTP server.
- 15. The ATWINC1500 HTTP server will trigger the provisioning response event with provisioning information to the host MCU application driver.
- 16. Host MCU will handle the provisioning response event in Wi-Fi callback function and parse the AP credentials such as SSID, Security type, passphrase.
- 17. With this AP's configuration, the ATWINC1500 application driver will connect with the desired AP and obtain the IP address by using the DHCP client request.
- 18. After the device is connected to the AP, the <code>gethostbyname()</code> function will be called to determine the IP address of the <code>api.openweathermap.org</code>.



- The TCP client socket is created and a connection is established to the weather server.
- 20. The socket callback function will receive the <code>SOCKET_MSG_CONNECT</code> event. Then it requests weather information to the server with a city name.
- 21. The socket callback function will receive the SOCKET MSG RECV event with weather information.
- 22. The Weather Client example terminal log appears as shown.

Figure 5-6. Weather Client Example Application Log

```
-- WINC1500 weather client example --
-- SAMW25_XPLAINED_PRO --
-- Compiled: Dec 24 2015 15:47:54 --
(APP>(INFO)Chip ID 1503a0
(APP>(INFO)Firmware ver : 19.4.4
(APP>(INFO)Min driver ver: 19.3.0
(APP>(INFO)Curr driver ver: 19.3.0

Provision Mode started.
Connect to [atmelconfig.com] via AP[WINC1500_31:CF] and fill up the page.
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: IP address is 192.168.1.103
wifi_cb: M2M_WIFI_DISCONNECTED
wifi_cb: M2M_WIFI_RESP_PROVISION_INFO
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: IP address is 192.168.4.27
resolve_cb: api.openweathermap.org IP address is 128.199.103.33

City: Paris
Temperature: 10.54
Weather Condition: Sky is Clear
```



6. Extending Weather Client Demo Application

In the weather client example application, the main.c and main.h was modified to configure different openweathermap API options. The modified files are available as attachment along with this application note.

In main.c file, the event $SOCKET_MSG_RECV$ from $socket_cb$ () function was modified to parse more weather information such as pressure, humidity, wind direction, and speed.

Note: The example implementation is applicable only for XML data format. Application users must write their own logic for other formats such as JSON and HTML.

Figure 6-1. Modified Weather Client Application Log

```
-- WINC1500 weather client example --
-- SAMW25_XPLAINED_PRO --
-- Compiled: Dec 24 2015 15:44:50 --
(APP>(INFO)Chip ID 1503a0
(APP>(INFO)Firmware ver : 19.4.4
(APP>(INFO)Firmware ver : 19.3.0
(APP>(INFO)Curr driver ver: 19.3.0
Provision Mode started.
Connect to [atmelconfig.com] via AP[WINC1500_31:CF] and fill up the page.

wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: IP address is 192.168.1.103
wifi_cb: M2M_WIFI_DISCONNECTED
wifi_cb: M2M_WIFI_RESP_PROWISION_INFO
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: M2M_WIFI_RESP_PROWISION_INFO
wifi_cb: M2M_WIFI_CONNECTED
wifi_cb: IP address is 192.168.4.27
resolve_cb: api.openweathermap.org IP address is 128.199.103.33

HTTP Response: 200 OK
Content-Type: text/xml

City: Chennai
Temperature: 31 Celsius
Humidity: 62 %
Pressure: 1012 hPa
Wind Speed: Light breeze
Wind Direction: NorthEast
Weather Condition: scattered clouds
```

Note: The links provided in the upcoming sections contains example web URLs. The variable "APIKEY" must be substituted with appropriate APIKEY value. To get the APIKEY, refer to http://openweathermap.org/appid.

6.1. Methods to Fetch Current Weather for a Location

Current weather information can be fetched based on a city name, city ID, or geographic coordinates or ZIP code. Based on the user requirement, the corresponding define given below shall be used. For more information on city ID and ZIP codes, refer http://bulk.openweathermap.org/sample/.

```
/** Location configuration */
#define MAIN_CITY_NAME "q=paris" /*Fetch by city name*/
#define MAIN_CITY_NAME "id=1264527" /*Fetch by city ID*/
#define MAIN_CITY_NAME "lat=13.09&lon=80.28" /*Fetch by geographic coordinates*/
#define MAIN_CITY_NAME "zip=600001" /*Fetch by zip code*/
```



By City name

API Call:

api.openweathermap.org/data/2.5/weather?q={city name}

api.openweathermap.org/data/2.5/weather?q={city name},{country code}

Web URL:

http://api.openweathermap.org/data/2.5/weather?q=chennai&APPID=APIKEY&mode=xml&units=metric

By City ID

API Call:

api.openweathermap.org/data/2.5/weather?id={city id}

Web URL:

http://api.openweathermap.org/data/2.5/weather?id=1264527&APPID=APIKEY&mode=xml&units=metric

By Geographic Coordinates

API Call:

api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}

Web URL:

http://api.openweathermap.org/data/2.5/weather? lat=13.09&lon=80.28&APPID=APIKEY&mode=xml&units=metric

By Zip Code

API Call:

api.openweathermap.org/data/2.5/weather?zip={zip code},{country code}

Web URL:

http://api.openweathermap.org/data/2.5/weather? zip=600001.IN&APPID=APIKEY&mode=xml&units=metric

6.2. Methods to Change Unit of Temperature

The weather temperature unit can be configured for Celsius, Fahrenheit, or Kelvin. Based on the user requirement, appropriate define from the following options may be used.

Temperature in Kelvin

API Call:

api.openweathermap.org/data/2.5/weather?q={city name}

Web URL:

http://api.openweathermap.org/data/2.5/weather?q=chennai&APPID=APIKEY&mode=xml



Temperature in Fahrenheit

API Call:

api.openweathermap.org/data/2.5/weather?q={city name}&units=imperial

Web URL:

http://api.openweathermap.org/data/2.5/weather?q=chennai&APPID=APIKEY&mode=xml&units=imperial

Temperature in Celsius

API Call:

api.openweathermap.org/data/2.5/weather?q={city name}&units=metric

Web URL:

http://api.openweathermap.org/data/2.5/weather?q=chennai&APPID=APIKEY&mode=xml&units=metric

6.3. Method to Change Language of Weather Data

The weather condition information language can be configured for different languages. Based on the user requirement, one of the following defines can be used. For more openweathermap supported languages, refer http://openweathermap.org/current#multi.

API Call:

api.openweathermap.org/data/2.5/weather?q={city name}&lang={lang}

Web URL:

http://api.openweathermap.org/data/2.5/weather? q=chennai&APPID=APIKEY&mode=xml&units=metric&lang=en

6.4. Method to Change Weather Data Format

The weather server response format can be configured json, xml, or html. Based on the user requirement, one of the following defines can be used.

JSON

API Call:

api.openweathermap.org/data/2.5/weather?q={city name}

Web URL:

http://api.openweathermap.org/data/2.5/weather?g=chennai&APPID=APIKEY

XML

API Call:



api.openweathermap.org/data/2.5/weather?q={city name}&mode=xml

Web URL:

http://api.openweathermap.org/data/2.5/weather?q=chennai&APPID=APIKEY&mode=xml

HTML

API Call:

api.openweathermap.org/data/2.5/weather?q={city name}&mode=html

Web URL:

http://api.openweathermap.org/data/2.5/weather?q=chennai&APPID=APIKEY&mode=html



7. Revision History

Doc. Rev.	Date	Comments
42654A	03/2016	Initial document release.







Enabling Unlimited Possibilities®











Atmel Corporation

1600 Technology Drive, San Jose, CA 95110 USA

T: (+1)(408) 441.0311

F: (+1)(408) 436.4200

www.atmel.com

© 2016 Atmel Corporation. / Rev.: Atmel- 42654A-ATWINC1500-Weather-Client-Example-Application_AT15058_Application Note-03/2016

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. ARM®, ARM Connected® logo and others are the registered trademarks or trademarks of ARM Ltd Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death ("Safety-Critical Applications") without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.