Chapter 4: AnyCloud

Time 1 Hour

At the end of this chapter you should understand what is contained in the AnyCloud solution. In addition, you will understand the Java Script Object Notation (JSON) file format (which is widely used on the Internet).

4.1 Introduction

In addition to the PSoC 6, the development kit you are using has a CYW43012 connectivity device that supports both Wi-Fi and Bluetooth. Let's face it, most electronic devices these days have some sort of wireless connectivity. In this chapter we will discuss the Wi-Fi side of the AnyCloud Solution.

The connection between the PSoC 6 and the connectivity device is done using SDIO for Wi-Fi (or SPI on some devices) and an HCI UART interface for Bluetooth. Furthermore, wireless co-existence between Wi-Fi and Bluetooth is supported so both functions can operate simultaneously.

Infineon has several cloud solutions depending on how you decide to manage your connected devices. The AnyCloud solution was built for customers with their own cloud device management back end, whether hosted on AWS, Google, Azure, AliCloud, or any other cloud infrastructure.

If you are using Amazon Web Services IoT Core for your device management, then you may prefer to use our more customized AWS IoT Core solution. Likewise, if you are using Arm Pelion for your device management, you may prefer our customized Arm Pelion solution. Note that if you are using Amazon's cloud services for storage but not their IoT Core device management, AnyCloud is a great solution for you, as you will see in some of the upcoming exercises.

AnyCloud provides features such as the Wi-Fi Connection Manager, a Secure Socket layer, support for application layer cloud protocols, Bluetooth Low Energy (BLE) functionality, and Low Power Assist (LPA).

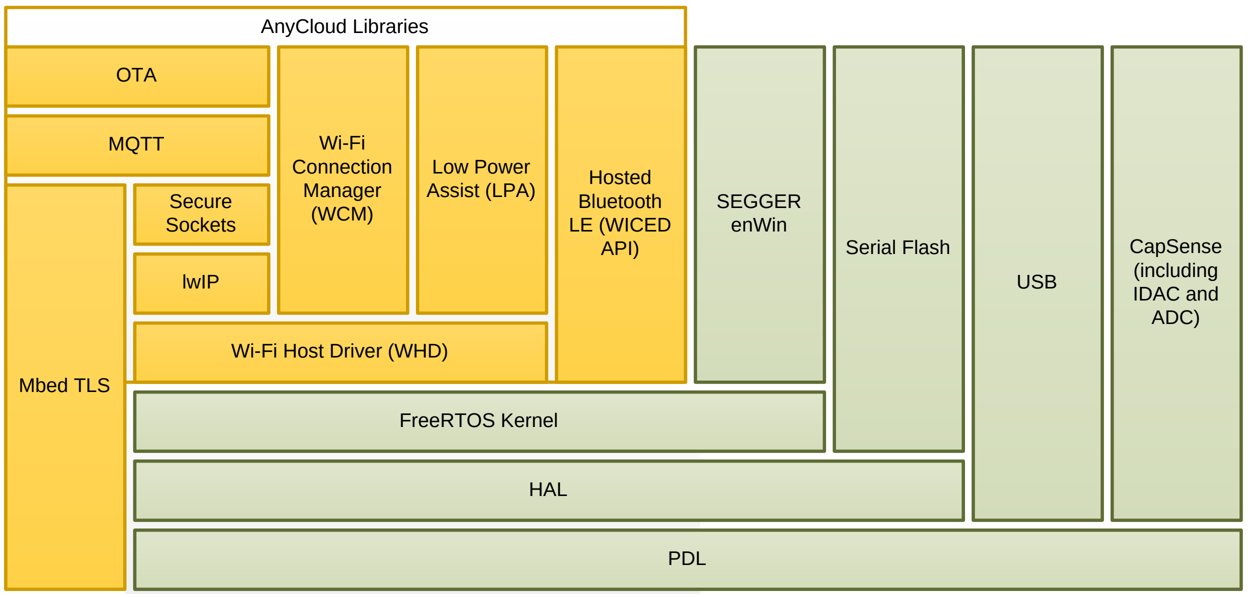
Currently, AnyCloud supports the MQTT application layer cloud protocol, but upcoming releases will add support for HTTP and AMQP.

While the AnyCloud solution provides core functionality, including connectivity, security, firmware upgrade support, and application layer protocols like MQTT, it is also flexible so you can modify or extend it to match your needs.

4.2 Library Descriptions

The AnyCloud solution is distributed as a collection of libraries that work together to help you easily get your IoT device up and running on the Cloud. Some of the libraries were written by Cypress, while others are industry standard open source libraries. As you have seen, these can be pulled into a ModusToolbox application easily by using the Library Manager.

The AnyCloud solution libraries fit together with the core PSoC 6 libraries like this:



Most libraries are available as GitHub repositories. These "repos" contain source files, readme files, and documentation such as an API reference.

You might notice that there is no BLE library for the BLESS hardware on PSoC 63 devices. This is because AnyCloud is designed to support PSoC 6 MCUs with a combo device. It is theoretically possible to use a PSoC 63 device with on-chip BLE in conjunction with Wi-Fi from a 43xxx device, but the board would require two separate antennae and the software would not support value-add features like LPA or Co-Ex.

The following subsections describe the AnyCloud libraries.

4.2.1 Wi-Fi Middleware Core

This library is a collection of the core libraries that any Wi-Fi application will need plus a little bit of glue logic. By adding the Wi-Fi Middleware Core library to an application, you get:

- Wi-Fi Host Driver (WHD) - Embedded Wi-Fi Host Driver that provides a set of APIs to interact with Cypress WLAN chips.
- FreeRTOS - FreeRTOS kernel, distributed as standard C source files with configuration header file, for use with the PSoC6 MCU.
- CLib Support - This is a support library that provides the necessary hooks to make C library functions such as malloc and free thread safe. This implementation is specific to FreeRTOS.
- lwIP - A Lightweight open-source TCP/IP stack.
- MbedTLS - An open source, portable, easy to use, readable and flexible SSL library, that has cryptographic capabilities.
- RTOS Abstraction Layer - Minimalistic RTOS-agnostic kernel interface allowing middleware to be used in multiple ecosystems, such as Arm's Mbed OS and Amazon's FreeRTOS. It is not

recommended for use by the end application - the user should write code for their RTOS of choice such as FreeRTOS.

- Secure Sockets - Network abstraction APIs for underlying lwIP network stack and MbedTLS security library. The secure sockets library eases application development by exposing a socket like interface for both secure and non-secure socket communication.
- Predefined configuration files for FreeRTOS, lwIP and MbedTLS for typical embedded IoT use-cases.
- Associated glue layer between lwIP and WHD.

The Library Manager name for the Wi-Fi Middleware Core library is *wifi-mw-core*. It includes the other libraries listed above automatically.

4.2.2 Wi-Fi Connection Manager (WCM)

The WCM makes Wi-Fi connections easier and more reliable. Firstly, it implements Wi-Fi Protected Setup (WPS) to simplify the secure connection of a device to a Wi-Fi access point (AP). This enables applications to store the credentials in non-volatile memory so that future connections are just automatic whenever the AP is available. Secondly, it provides a monitoring service to detect problems and keep connections alive, improving reliability.

The Wi-Fi Connection Manager library includes the Wi-Fi Middleware Core library as a dependency.

The Library Manager name for this library is *wifi-connection-manager*.

4.2.3 MQTT

This library includes the open source AWS IoT device SDK embedded C library plus some glue to get it to work seamlessly in AnyCloud. It is based on MQTT client v3.1.1 and supports QoS levels 0 and 1. Both secure and non-secure TCP connections can be used.

The Library Manager name for this library is *MQTT*.

4.2.4 Over-The-Air (OTA) Bootloading

The OTA toolkit library is an extensible solution based on MCUBoot that can be modified to work with any third-party or custom IoT device management software. With it you can rapidly create efficient and reliable OTA schemes. It currently supports OTA over MQTT. Support for other protocols such as HTTP will be added in the future.

The Library Manager name for this library is *OTA*.

4.2.5 BT/BLE Hosted Stack (BTSTACK and FreeRTOS Wrapper)

In addition to the great Wi-Fi support in AnyCloud, you can use the Bluetooth LE functionality in the 43xxx combo device to enable BLE for your device. For example, it can easily be used to enable Wi-Fi onboarding so that you can safely and quickly connect your device to a Wi-Fi network using BLE to select the network and enter the password. One of the AnyCloud examples will show you that exact thing.

In the Library Manager, you can include the *bluetooth-freertos* library which will include the BTSTACK library automatically.

4.2.6 Low Power Assistant (LPA)

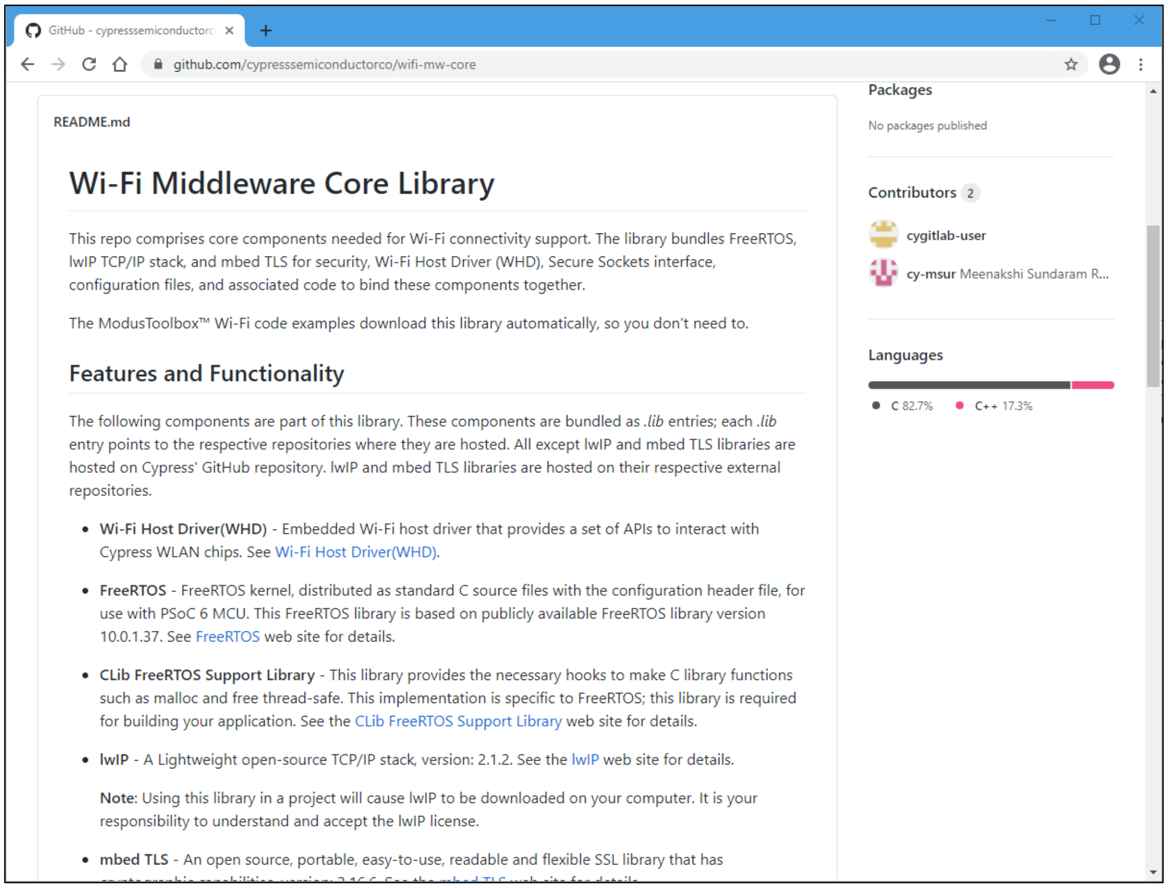
The LPA is a library and associated settings in the Device Configurator that allow you to configure a PSoC 6 Host and WLAN (Wi-Fi / BT Radio) device for optimized low-power operation. With LPA you can achieve the most aggressive power budgets by placing the host device into sleep or deep sleep modes while networks are quiet or when there is traffic that can be handled by the connectivity device.

The Library Manager name for this library is *LPA*.

4.3 Documentation

The best place to find documentation for AnyCloud is inside the individual libraries themselves. You can look on GitHub directly, you can open the documentation from a library that you have downloaded, or you can open the documentation for a downloaded library from inside the Eclipse IDE for ModusToolbox.

For example, here is the *README.md* for the Wi-Fi Middleware Core as seen on GitHub:


GitHub - cypresssemiconductorc
github.com/cypresssemiconductorco/wifi-mw-core
README.md
Packages
No packages published
Contributors 2
cygitlab-user
cy-msur Meenakshi Sundaram R...
Languages
● C 82.7%
● C++ 17.3%

Wi-Fi Middleware Core Library

This repo comprises core components needed for Wi-Fi connectivity support. The library bundles FreeRTOS, lwIP TCP/IP stack, and mbed TLS for security, Wi-Fi Host Driver (WHD), Secure Sockets interface, configuration files, and associated code to bind these components together.

The ModusToolbox™ Wi-Fi code examples download this library automatically, so you don't need to.

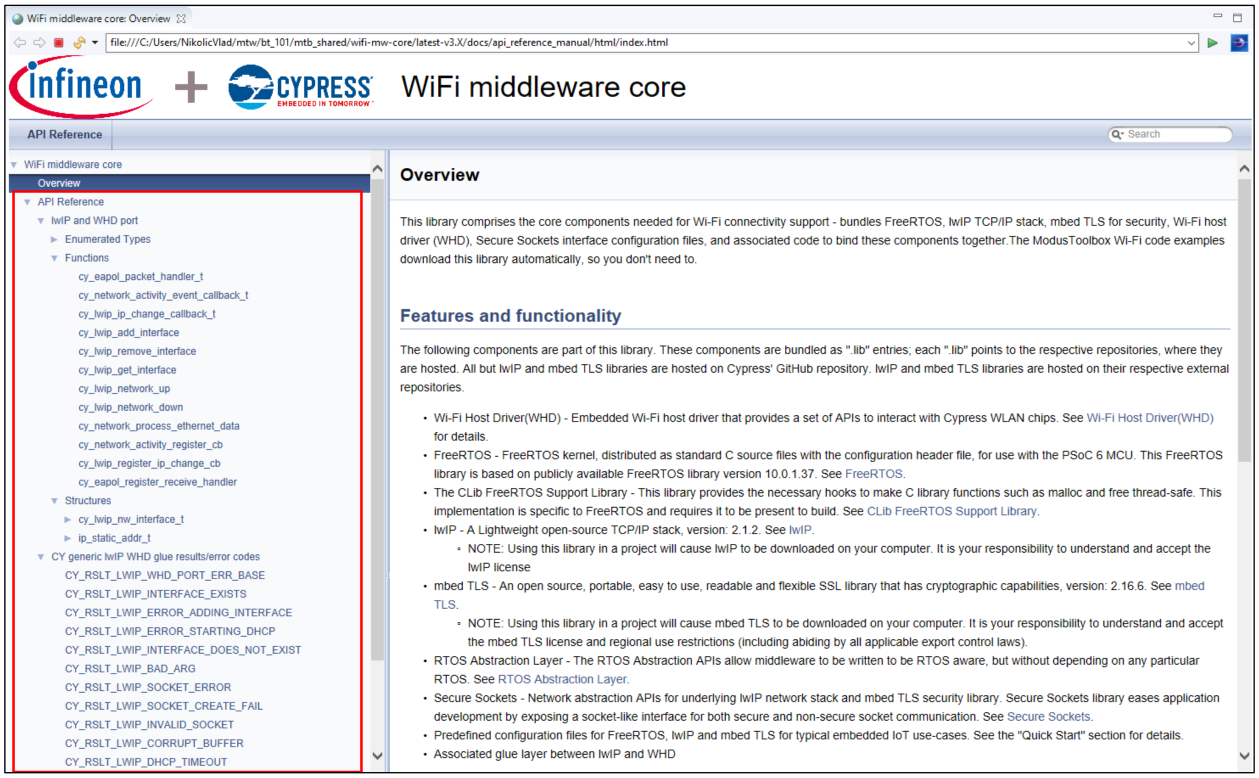
Features and Functionality

The following components are part of this library. These components are bundled as *.lib* entries; each *.lib* entry points to the respective repositories where they are hosted. All except lwIP and mbed TLS libraries are hosted on Cypress' GitHub repository. lwIP and mbed TLS libraries are hosted on their respective external repositories.

- **Wi-Fi Host Driver(WHD)** - Embedded Wi-Fi host driver that provides a set of APIs to interact with Cypress WLAN chips. See Wi-Fi Host Driver(WHD).
- **FreeRTOS** - FreeRTOS kernel, distributed as standard C source files with the configuration header file, for use with PSoC 6 MCU. This FreeRTOS library is based on publicly available FreeRTOS library version 10.0.1.37. See FreeRTOS web site for details.
- **CLib FreeRTOS Support Library** - This library provides the necessary hooks to make C library functions such as malloc and free thread-safe. This implementation is specific to FreeRTOS; this library is required for building your application. See the CLib FreeRTOS Support Library web site for details.
- **lwIP** - A Lightweight open-source TCP/IP stack, version: 2.1.2. See the lwIP web site for details.

  Note: Using this library in a project will cause lwIP to be downloaded on your computer. It is your responsibility to understand and accept the lwIP license.
- **mbed TLS** - An open source, portable, easy-to-use, readable and flexible SSL library that has

Once you have downloaded a library, you can go to the *docs* directory or you can open it from the Quick Panel inside Eclipse. In either case, you will typically find an *api_reference_manual.html* file with full descriptions of the API:


WiFi middleware core: Overview
file:///C:/Users/NikolicVlad/mtw/bt_101/mtb_shared/wifi-mw-core/latest-v3.X/docs/api_reference_manual/html/index.html
WiFi middleware core
API Reference
Search
WiFi middleware core
Overview
API Reference
lwIP and WHD port
Enumerated Types
Functions
cy_eapol_packet_handler_t
cy_network_activity_event_callback_t
cy_lwip_ip_change_callback_t
cy_lwip_add_interface
cy_lwip_remove_interface
cy_lwip_get_interface
cy_lwip_network_up
cy_lwip_network_down
cy_network_process_ethernet_data
cy_network_activity_register_cb
cy_lwip_register_ip_change_cb
cy_eapol_register_receive_handler
Structures
cy_lwip_nw_interface_t
ip_static_addr_t
CY generic lwIP WHD glue results/error codes
CY_RSLT_LWIP_WHD_PORT_ERR_BASE
CY_RSLT_LWIP_INTERFACE_EXISTS
CY_RSLT_LWIP_ERROR_ADDING_INTERFACE
CY_RSLT_LWIP_ERROR_STARTING_DHCP
CY_RSLT_LWIP_INTERFACE_DOES_NOT_EXIST
CY_RSLT_LWIP_BAD_ARG
CY_RSLT_LWIP_SOCKET_ERROR
CY_RSLT_LWIP_SOCKET_CREATE_FAIL
CY_RSLT_LWIP_INVALID_SOCKET
CY_RSLT_LWIP_CORRUPT_BUFFER
CY_RSLT_LWIP_DHCP_TIMEOUT

Overview

This library comprises the core components needed for Wi-Fi connectivity support - bundles FreeRTOS, lwIP TCP/IP stack, mbed TLS for security, Wi-Fi host driver (WHD), Secure Sockets interface configuration files, and associated code to bind these components together.The ModusToolbox Wi-Fi code examples download this library automatically, so you don't need to.

Features and functionality

The following components are part of this library. These components are bundled as ".lib" entries; each ".lib" points to the respective repositories, where they are hosted. All but lwIP and mbed TLS libraries are hosted on Cypress' GitHub repository. lwIP and mbed TLS libraries are hosted on their respective external repositories.

- Wi-Fi Host Driver(WHD) - Embedded Wi-Fi host driver that provides a set of APIs to interact with Cypress WLAN chips. See Wi-Fi Host Driver(WHD) for details.
- FreeRTOS - FreeRTOS kernel, distributed as standard C source files with the configuration header file, for use with the PSoC 6 MCU. This FreeRTOS library is based on publicly available FreeRTOS library version 10.0.1.37. See FreeRTOS.
- The CLib FreeRTOS Support Library - This library provides the necessary hooks to make C library functions such as malloc and free thread-safe. This implementation is specific to FreeRTOS and requires it to be present to build. See CLib FreeRTOS Support Library.
- lwIP - A Lightweight open-source TCP/IP stack, version: 2.1.2. See lwIP.
  - NOTE: Using this library in a project will cause lwIP to be downloaded on your computer. It is your responsibility to understand and accept the lwIP license
- mbed TLS - An open source, portable, easy to use, readable and flexible SSL library that has cryptographic capabilities, version: 2.16.6. See mbed TLS.
  - NOTE: Using this library in a project will cause mbed TLS to be downloaded on your computer. It is your responsibility to understand and accept the mbed TLS license and regional use restrictions (including abiding by all applicable export control laws).
- RTOS Abstraction Layer - The RTOS Abstraction APIs allow middleware to be written to be RTOS aware, but without depending on any particular RTOS. See RTOS Abstraction Layer.
- Secure Sockets - Network abstraction APIs for underlying lwIP network stack and mbed TLS security library. Secure Sockets library eases application development by exposing a socket-like interface for both secure and non-secure socket communication. See Secure Sockets.
- Predefined configuration files for FreeRTOS, lwIP and mbed TLS for typical embedded IoT use-cases. See the "Quick Start" section for details.
- Associated glue layer between lwIP and WHD

4.4 Code Examples

It is always easier to start with an existing application rather than starting with a blank slate. Infineon provides a wealth of AnyCloud code examples for just that reason.

All Infineon code examples are hosted on GitHub so ultimately that's where you will find them, but there are a few ways to get there:

1. Inside the project creator tool:

   Look for starter applications whose names start with "AnyCloud"

2. Directly from the Cypress GitHub web site:

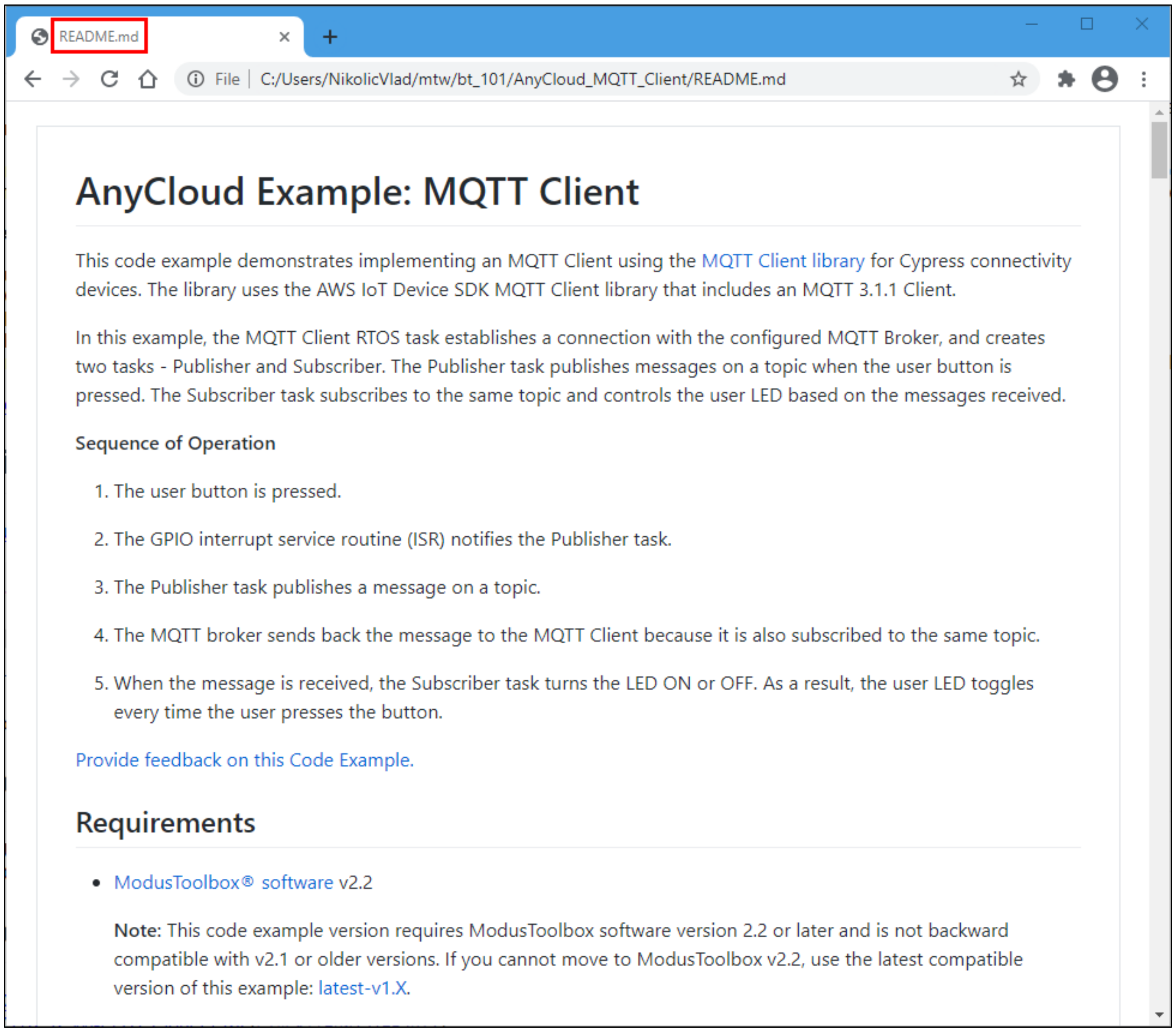
   https://github.com/cypresssemiconductorco?q=mtb-example-anycloud%20NOT%20Deprecated

3. From the Cypress website:

   https://www.cypress.com/documentation/code-examples/anycloud-sdk-examples-modustoolbox-software

Of course, you can learn a ton by looking at the source code and reading the comments from the examples. On top of that, every Infineon code example has a README.md file that explains the example, tells you which kits it will run on, shows how to set it up, explains how to use it, and explains the design. The README.md is always a good starting point.

You will need a markdown reader to get the most out of the README.md documents. There are plugins for most web browsers as well as an extension for VS Code. Markdown will also show inside the Eclipse IDE, but some things may be formatted strangely since Eclipse doesn't support everything that Markdown can do.


README.md
File
C:/Users/NikolicVlad/mtw/bt_101/AnyCloud_MQTT_Client/README.md

AnyCloud Example: MQTT Client

This code example demonstrates implementing an MQTT Client using the MQTT Client library for Cypress connectivity devices. The library uses the AWS IoT Device SDK MQTT Client library that includes an MQTT 3.1.1 Client.

In this example, the MQTT Client RTOS task establishes a connection with the configured MQTT Broker, and creates two tasks - Publisher and Subscriber. The Publisher task publishes messages on a topic when the user button is pressed. The Subscriber task subscribes to the same topic and controls the user LED based on the messages received.

Sequence of Operation

1. The user button is pressed.
2. The GPIO interrupt service routine (ISR) notifies the Publisher task.
3. The Publisher task publishes a message on a topic.
4. The MQTT broker sends back the message to the MQTT Client because it is also subscribed to the same topic.
5. When the message is received, the Subscriber task turns the LED ON or OFF. As a result, the user LED toggles every time the user presses the button.

Provide feedback on this Code Example.

Requirements

- ModusToolbox® software v2.2

  Note: This code example version requires ModusToolbox software version 2.2 or later and is not backward compatible with v2.1 or older versions. If you cannot move to ModusToolbox v2.2, use the latest compatible version of this example: latest-v1.X.

4.5 JavaScript Object Notation (JSON)

JSON (https://en.wikipedia.org/wiki/JSON) is an open-standard format that uses human-readable text to transmit data. It is the de facto standard for communicating data to/from the cloud. JSON supports the following data types:

- Integers
- Double precision floating point
- Strings
- Boolean (true or false)
- Arrays (use "[]" to specify the array with values separated by ",")
- Key/Value (keymap) pairs as "key":value (use "{}" to specify the keymap) with "," separating the pairs
    - Key/Value pair values can be arrays or can be other key/value pairs
    - Arrays can hold Key/Value pairs

For example, a legal JSON file looks like this:

```
{
    "name" : "alan",
    "age" : 49,
    "badass" : true,
    "children":  ["Anna","Nicholas"],
    "address" : {
            "number":201,
            "street": "East Main Street",
            "city": "Lexington",
            "state":"Kentucky",
            "zipcode":40507
    }
}
```

Note that carriage returns and spaces (except within the strings themselves) don't matter. For example, the above JSON code could be written as:

```
{"name":"alan","age":49,"badass":true,"children":["Anna","Nicholas"],"address":{"nu
mber":201,"street":"East Main
Street","city":"Lexington","state":"Kentucky","zipcode":40507}}
```

While this is more difficult for a person to read, it is easier to create such a string in the firmware when you need to send JSON documents.

Unfortunately, quotes mean something to the C compiler so if you are including a JSON string inside a C program you need to escape the quotes that are inside the JSON with a backslash (\). The above JSON would be represented like this inside a C program:

```
{\"name\":\"alan\",\"age\":49,\"badass\":true,\"children\":[\"Anna\",\"Nicholas\"],
\"address\":{\"number\":201,\"street\":\"East Main
Street\",\"city\":\"Lexington\",\"state\":\"Kentucky\",\"zipcode\":40507}}
```

There is a website available which can be used to do JSON error checking. It can be found at:

https://jsonformatter.curiousconcept.com

4.6 Creating JSON

If you need to create JSON to send out (e.g. to the cloud) you can create a string using `snprintf` With standard formatting codes to substitute in values for variables. For example, the following could be used to send the temperature as a floating-point value from an IoT device to a cloud service provider:

```
char json[100];
snprintf(json, sizeof(json), "{\"state\" : {\"reported\" :
{\"temperature\":%.1f} } }", psoc_data.temperature);
```

The %.1f is replaced in the string with `psoc_data.temperature` as a floating point value with one place after the decimal. If the actual temperature is 25.4, the resulting string created in the array *json* would be:

```
{"state" : {"reported" : {"temperature":25.4} } }
```

4.7 Reading JSON using a Parser

If you need to receive JSON (e.g. from the cloud) and then pull out a specific value, you can use a JSON parser. A parser will read the JSON and will then find and return values for keys that you specify.

The JSON parser in ModusToolbox is called *JSON_parser*, and is included as a part of the connectivity-utilities library. However, in this class we will also cover the JSON parser *cJSON*, which is available in this GitHub repository: https://github.com/markgsaunders/cJSON. cJSON is a Document Object Model Parser, meaning it reads the whole JSON in one gulp, whereas JSON_parser is iterative, and as such, enables you to parse larger files. cJSON is easier to use but it may be necessary to use JSON_parser for very large JSON files. For IoT devices, cJSON will almost always be sufficient.

4.7.1 cJSON Library

The *cJSON* library reads and processes the entire document at one time, then lets you access data in the document with an API to find elements. You will traverse the JSON hierarchy one level at a time until you reach the key:value pair that you are interested in. The functions generally return a pointer to a structure of type cJSON which has elements for each type of return data (i.e. valuestring, valueint, valuedouble, etc.)

For example, if you have a char array called ***data*** with the JSON related to Alan:

```
{"name":"alan","age":49,"badass":true,"children":["Anna","Nicholas"],"address":{"nu
mber":201,"street":"East Main
Street","city":"Lexington","state":"Kentucky","zipcode":40507}}
```

The code to get Alan's zip code would look like this:

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
#include "cJSON.h"

int main(void)
{
     int zipcodeValue;
     cJSON *root = cJSON_Parse(data); //Read the JSON
     cJSON *address = cJSON_GetObjectItem(root,"address"); // Search for the key "address"
     cJSON *zipcode = cJSON_GetObjectItem(address,"zipcode"); // Search for the key "zipcode"
under address
     zipcodeValue = zipcode->valueint; // Get the integer value associated with the key
zipcode
}
```

To include the cJSON library in your project:

1. Clone the cJSON repository from here: https://github.com/markgsaunders/cJSON into your application's root directory.
2. Include *cJSON.h* in the C source file:

```
#include "cJSON.h"
```

4.7.2 JSON_parser Library (Advanced)

The JSON_Parser library is an iterative parser, meaning that it reads one chunk at a time. This kind of parser is good for situations where you have very large structures where it is impractical to read the entire thing into memory at once but it is generally more difficult to use than the cJSON parser. You won't normally need it for IoT devices since they typically transmit data in small batches. To use it you:

1. Use `cy_JSON_parser_register_callback` to register a callback function that is executed whenever a JSON item is received.
2. Use `cy_JSON_parser` to pass in the JSON data itself.
3. Wait for the callback function to be called and process the data as necessary.

The callback function receives a structure of the type `cy_JSON_object_t` which is:

```
typedef struct cy_JSON_object {

    char*                   object_string;          /**< JSON object as a string */
    uint8_t                 object_string_length;   /**< Length of the JSON string */
    cy_JSON_type_t          value_type;             /**< JSON data type of value parsed */
    char*                   value;                  /**< JSON value parsed */
    uint16_t                value_length;           /**< JSON length of value parsed */
    struct cy_JSON_object*  parent_object;          /**< Pointer to parent JSON object */
} cy_JSON_object_t;
```

You can use conditional statements to check the name of the object that were received, check the type of value received, or even check values of parent objects.

The value types are:

```
typedef enum
{
    JSON_STRING_TYPE,
    JSON_NUMBER_TYPE,
    JSON_VALUE_TYPE,
    JSON_ARRAY_TYPE,
    JSON_OBJECT_TYPE,
    JSON_BOOLEAN_TYPE,
    JSON_NULL_TYPE,
    UNKNOWN_JSON_TYPE
} cy_JSON_types_t;
```

Note that the value itself is returned as a string (`char*`) no matter what so you will need to use `atof` to convert the string to a floating-point value or atoi to convert to an integer if that is what you need.

You must make sure a parent_object is not `NULL` before trying to access it or else your device will reboot.

Using the previous example, if you have a char array called ***data*** with the JSON related to Alan:

```
{"name":"alan","age":49,"badass":true,"children":["Anna","Nicholas"],"address":{"nu
mber":201,"street":"East Main
Street","city":"Lexington","state":"Kentucky","zipcode":40507}}
```

The code to get Alan's zip code would look like this:

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
#include "cy_retarget_io.h"
#include "cy_json_parser.h"

float zipcodeValue;
char zipcodeString[6];

cy_rslt_t jsonCallback(cy_JSON_object_t *obj_p){
     /* Verify that the JSON path is address: zipcode and that zipcode is a number */
     if( (obj_p->parent_object != NULL) &&
          (strncmp(obj_p->parent_object->object_string, "address", strlen("address")) == 0)
&&
          (strncmp(obj_p->object_string, "zipcode", strlen("zipcode")) == 0 ) &&
          (obj_p->value_type == JSON_NUMBER_TYPE) ){
            /'* Get zipcode value and convert to an integer */
            snprintf(zipcodeString, (obj_p->value_length)+1, "%s", obj_p->value);
            zipcodeValue = atoi(zipcodeString);
     }
     return CY_RSLT_SUCCESS ;
}

int main(void){
    cy_JSON_parser_register_callback(jsonCallback, NULL);
    cy_JSON_parser(data, strlen(data));
}
```

To use the JSON_parser library in your project:

1. Verify that you have the *connectivity-utilities* library in your project. It contains the JSON parser library.
    a. If you do not, you can add it by adding the *wifi-mw-core* library in the Library Manager.
    b. Don’t forget to copy the config files from *libs/wifi-mw-core/configs* to your root project directory and update the *Makefile* by adding:
    `COMPONENTS=FREERTOS LWIP MBEDTLS`
    `DEFINES+=MBEDTLS_USER_CONFIG_FILE='"mbedtls_user_config.h"'`
2. Include *cy_json_parser.h* in the C source file:
    `#include “cy_json_parser.h”`

4.8 Exercises

4.8.1 Exercise 1: Parse a JSON document using the library "cJSON"

Write a program that will read JSON from a hard-coded character array, parse out specific values, and print them to a UART terminal window.

☐ 1. Create a new project called **ch04_ex01_cJSON** based on the Empty PSoC 6 template.

☐ 2. Add the *cJSON* library to your application.

   a. From the command line in the application's root directory:
```
git clone https://github.com/markgsaunders/cJSON
```
   b. In *main.c*:
```
#include "cJSON.h"
```

☐ 3. Make a JSON string that contains the reported temperature. In a real IoT device, this would likely have been received from the cloud, but for now we will just hard-code it:

```
const char *jsonString = "{\"state\" : {\"reported\" : {\"temperature\":25.4} } }";
```

☐ 4. Use the cJSON parser to get the value of the temperature and print it to a terminal window using `printf`.

Hint Refer to the section titled *cJSON Library*.
Hint Don't forget to add and initialize the *retarget-io* library.

4.8.2 Exercise 2: (Advanced) Process a JSON document using "JSON_parser"

Repeat the cJSON exercise using the *JSON_parser* library.