

Chapter 1: Introduction

After completing this chapter, you will understand what this class is, what topics are covered, the overall class objectives, and you will understand what is contained in the ModusToolbox™ any cloud solution. In addition, you will understand the Java Script Object Notation (JSON) file format (which is widely used on the Internet).

Table of contents

1.1	What is this class?	2
1.2	Prerequisites	2
1.3	Required software	2
1.4	Development kits	3
1.4.1	PSoC™ 6 kit	3
1.4.2	Arduino compatible shield	4
1.5	Introduction to ModusToolbox™ any cloud	5
1.6	ModusToolbox™ any cloud library descriptions	6
1.6.1	Wi-Fi Middleware Core	6
1.6.2	Wi-Fi Connection Manager (WCM)	7
1.6.3	HTTP	7
1.6.4	MQTT.....	7
1.6.5	Over-The-Air (OTA) Bootloading.....	7
1.6.6	Bluetooth®/Bluetooth® LE Hosted Stack (BTSTACK and FreeRTOS Wrapper)	7
1.6.7	Low Power Assistant (LPA).....	8
1.7	Documentation	8
1.8	Code examples	9
1.9	JavaScript Object Notation (JSON)	11
1.10	Creating JSON	12
1.11	Reading JSON using a parser	12
1.11.1	cJSON library.....	12
1.11.2	JSON_parser library	13
1.12	Exercises	15
	Exercise 1: Exercise 1: Parse a JSON document using the library "cJSON"	15
	Exercise 2: Process a JSON document using "JSON_parser"	15

Document conventions

Convention	Usage	Example
Courier New	Displays code and text commands	CY_ISR_PROTO(MyISR) ; make build
<i>Italics</i>	Displays file names and paths	sourcefile.hex
[bracketed, bold]	Displays keyboard commands in procedures	[Enter] or [Ctrl] [C]
Menu > Selection	Represents menu paths	File > New Project > Clone
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the Debugger icon, and then click Next .

1.1 What is this class?

This is a class to teach how to use Wi-Fi in ModusToolbox™ applications. The descriptions and exercises use a PSoC™ 6 MCU as a host to a CYW43012 device.

After completing this class, you should be able to create and debug Wi-Fi applications using ModusToolbox™ tools. This class will introduce you to several concepts including: the network stack, sockets, TLS, HTTP, MQTT, and interacting with Amazon Web Services (AWS).

1.2 Prerequisites

This class assumes that you know the basics of using the ModusToolbox™ ecosystem, how to interact with PSoC™ 6 MCUs including using peripherals, and how to program a PSoC™ 6 device. If you are unfamiliar with these topics, the following classes should be reviewed first:

- ModusToolbox™ Software Training Level 1 – Getting Started
- ModusToolbox™ Software Training Level 2 – PSoC™ MCUs

1.3 Required software

- ModusToolbox™ tools

You should already have ModusToolbox™ tools installed on your system. If not, refer to the ModusToolbox™ Software Training Level 1 Getting Started class or visit <https://www.cypress.com/products/modustoolbox-software-environment> for instructions.

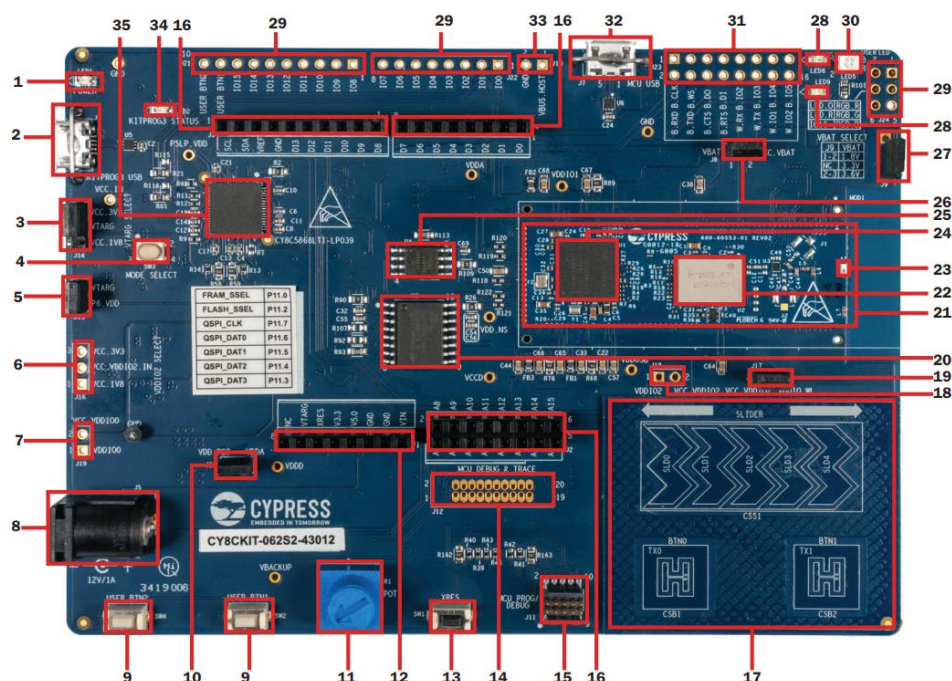
1.4 Development kits

For this class we will use the following development kits:

1.4.1 PSoC™ 6 kit

[CY8CKIT-062S2-43012](#) – A PSoC™ 6-2M MCU and a CYW43012 Wi-Fi + Bluetooth® Combo

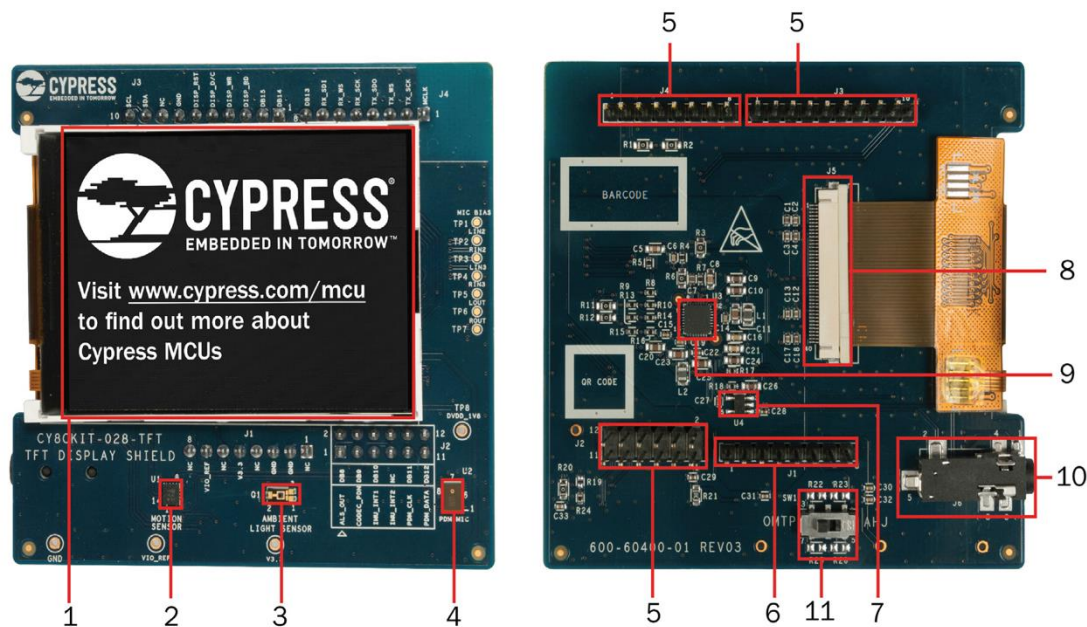
Note: If you are taking this course on your own, you will need two of these kits to complete some of the exercises in later chapters. If you are taking this course in a classroom setting, you will only need one of these kits.



- | | |
|---|--|
| 1. Power LED (LED1) | 19. CYW43012 VDDIO current measurement jumper (J17) |
| 2. KitProg3 USB connector (J6) | 20. Infineon serial NOR flash memory (S25FL512S, U3) |
| 3. PSoC™ 6 MCU VDD power selection jumper (J14) | 21. Infineon PSoC™ 6 (2M) MCU with CYW43012 Carrier Module (CY8CMOD-062S2-43012, MOD1) |
| 4. KitProg3 programming mode selection button (SW3) | 22. CYW43012 based Murata Type 1LV module |
| 5. PSoC™ 6 MCU VDD current measurement jumper (J15) | 23. Wi-Fi/Bluetooth® antenna |
| 6. PSoC™ 6 MCU VDDIO2 and CYW43012 VDDIO power selection jumper (J16) | 24. PSoC™ 6 MCU |
| 7. PSoC™ 6 MCU VDDIO0 current measurement jumper (J19) | 25. Infineon serial Ferroelectric RAM (CY15B104QSN, U4) |
| 8. External power supply VIN connector (J5) | 26. CYW43012 VBAT current measurement jumper (J8) |
| 9. PSoC™ 6 MCU user buttons (SW2 and SW4) | 27. CYW43012 VBAT power selection jumper (J9): |
| 10. Potentiometer connection jumper (J25) | 28. PSoC™ 6 MCU user LEDs (LED8 and LED9) |
| 11. Potentiometer (R1) | 29. PSoC™ 6 I/O header (J21, J22, J24) |
| 12. Arduino-compatible power header (J1) | 30. RGB LED (LED5) |
| 13. PSoC™ 6 MCU reset button (SW1) | 31. Wi-Fi/Bluetooth® GPIO header (J23) |
| 14. PSoC™ 6 MCU debug and trace header (J12) | 32. PSoC™ 6 USB device connector (J7) |
| 15. PSoC™ 6 MCU program and debug header (J11) | 33. Optional USB Host power supply header (J10) |
| 16. Arduino Uno R3-compatible I/O headers (J2, J3, and J4) | 34. KitProg3 status LED (LED2) |
| 17. CAPSENSE™ slider (SLIDER) and buttons (N0 and BTN1) | 35. KitProg3 (PSoC™ 5LP MCU) programmer and debugger (CY8C5868LTI-LP039, U2) |
| 18. PSoC™ 6 MCU VDDIO2 current measurement jumper (J18) | 36. MicroSD Card holder (J20) (on back of board) |

1.4.2 Arduino compatible shield

[CY8CKIT-028-TFT](#)



- | | |
|--|--|
| 1. 2.4-inch TFT display | 7. TFT display power control load switch (U4) |
| 2. Motion Sensor (U1) | 8. TFT display connector (J5) |
| 3. Ambient Light Sensor (Q1) | 9. Audio CODEC (U3) |
| 4. PDM microphone (U2) | 10. Audio Jack (J6) |
| 5. Arduino compatible I/O headers (J2, J3, J4) | 11. Audio Jack Selection (OMTP/AHJ) Switch (SW1) |
| 6. Arduino compatible power header (J1) | |

1.5 Introduction to ModusToolbox™ any cloud

In addition to the PSoC™ 6, the development kit used in this course 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 focus on the Wi-Fi piece of the puzzle. 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 ModusToolbox™ any cloud 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, ModusToolbox™ any cloud is a great solution for you, as you will see in some of the upcoming exercises.

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

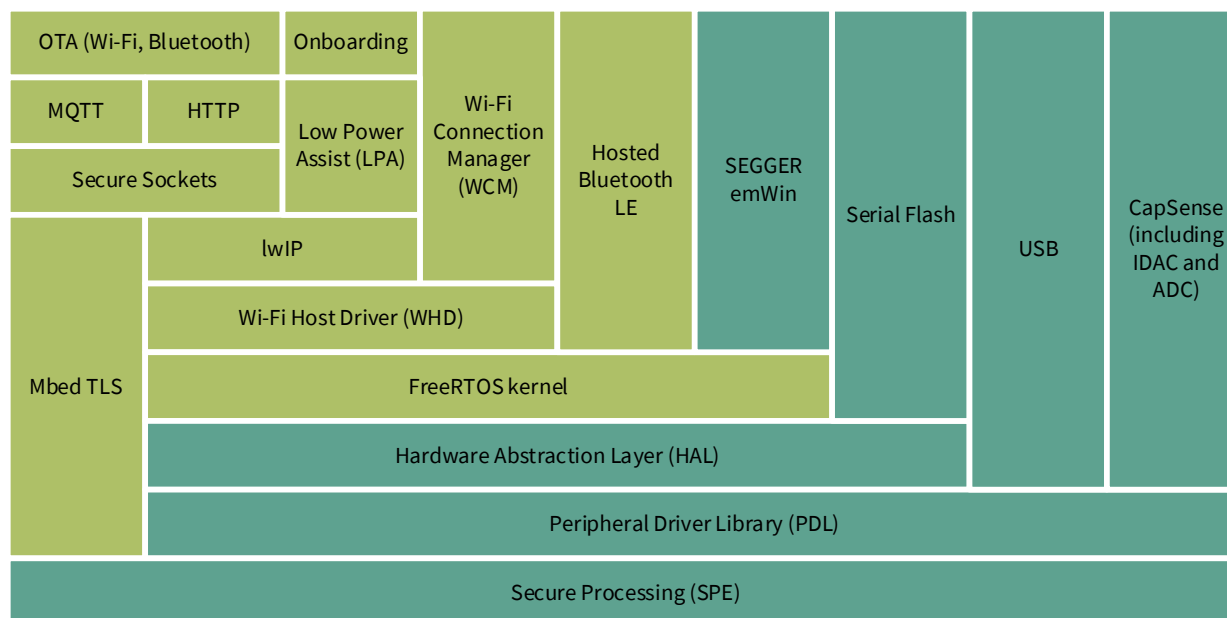
Currently, ModusToolbox™ any cloud supports the MQTT and HTTP application layer cloud protocols.

While the ModusToolbox™ any cloud 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.

1.6 ModusToolbox™ any cloud library descriptions

The ModusToolbox™ any cloud 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 Infineon, 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 ModusToolbox™ any cloud 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.

The following subsections describe the ModusToolbox™ any cloud libraries.

1.6.1 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 PSoC™ 6 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. lwIP stands for "lightweight IP".
- 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.

1.6.2 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*.

1.6.3 HTTP

This is actually two separate libraries – one for HTTP servers and one for HTTP clients. The libraries enable both secure (https) and non-secure (http) modes of connection. They support RESTful HTTP methods: HEAD, GET, PUT, and POST.

The Library Manager names for these libraries are *http-client* and *http-server*.

1.6.4 MQTT

This library includes the open source AWS IoT device SDK embedded C library plus some glue to get it to work seamlessly in ModusToolbox™ any cloud. 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*.

1.6.5 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 and HTTP.

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

1.6.6 Bluetooth®/Bluetooth® LE Hosted Stack (BTSTACK and FreeRTOS Wrapper)

In addition to the great Wi-Fi support in ModusToolbox™ any cloud, you can use the Bluetooth® LE functionality in the 43xxx combo device to enable Bluetooth® LE 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 Bluetooth® LE to select the network and enter the password. One of the ModusToolbox™ any cloud 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.

1.6.7 Low Power Assistant (LPA)

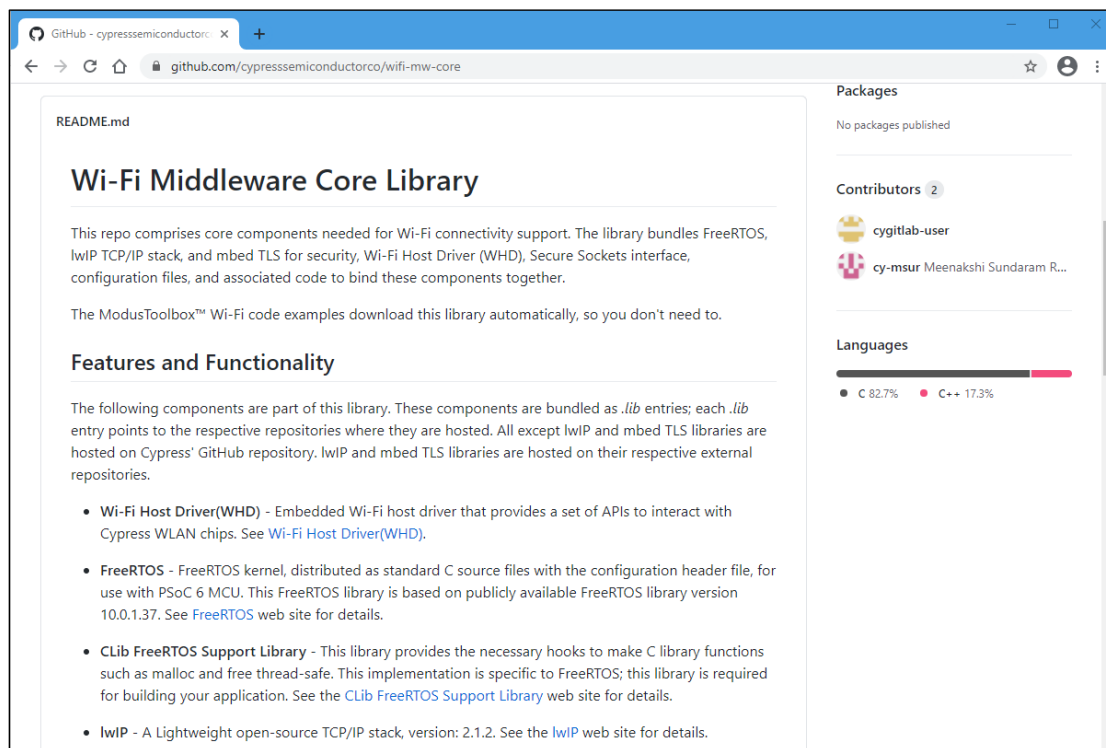
The LPA is a library and associated settings in the Device Configurator that allow you to configure a PSoC™ 6 Host and WLAN (Wi-Fi / Bluetooth® Radio) device for optimized low-power operation. With LPA you can achieve the most aggressive power budgets by placing the host device into sleep or deep sleep 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*.

1.7 Documentation

The best place to find documentation for ModusToolbox™ any cloud 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:



The screenshot shows the GitHub repository page for the 'Wi-Fi Middleware Core Library' by Cypress Semiconductor. The page includes a README section with the following content:

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

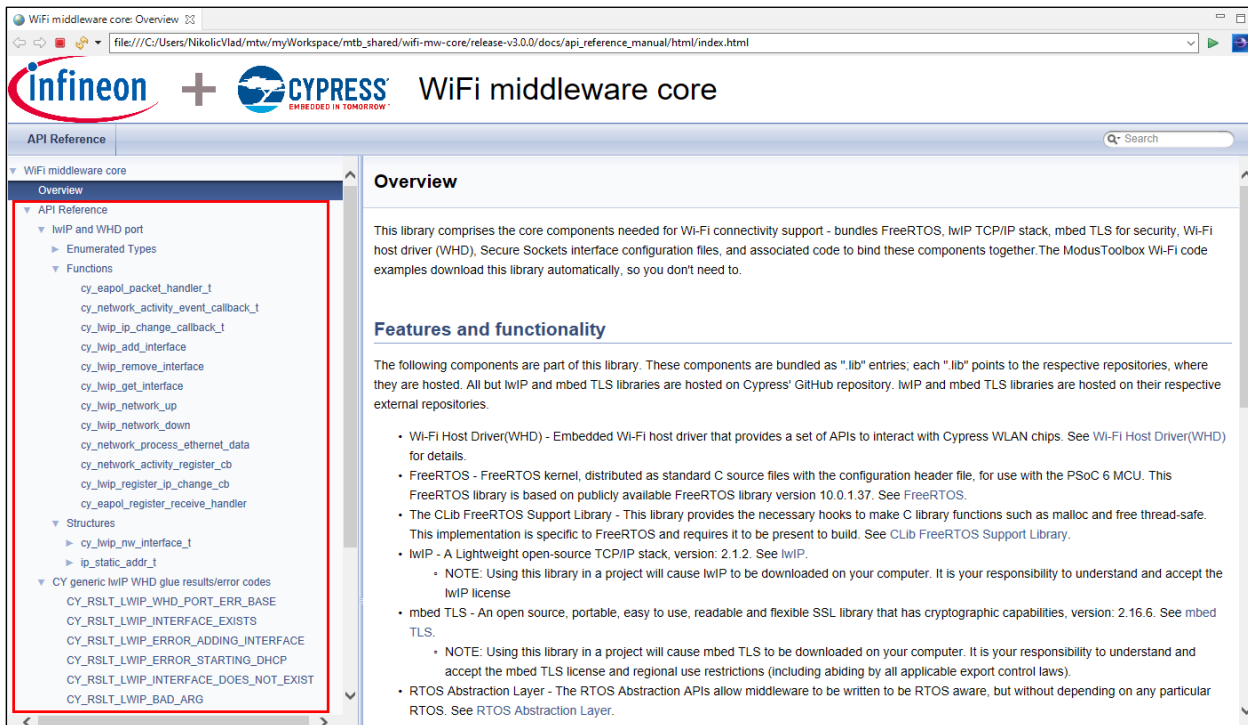
The following components are part of this library. These components are bundled as *.lib* entries; each *.lib* entry points to the respective repositories where they are hosted. All except lwIP and mbed TLS libraries are hosted on Cypress' GitHub repository. lwIP and mbed TLS libraries are hosted on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

The right sidebar of the screenshot shows repository statistics:

- Packages:** No packages published
- Contributors:** 2
 - cygitlab-user
 - cy-msur Meenakshi Sundaram R...
- Languages:**
 - C: 82.7%
 - C++: 17.3%

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:



1.8 Code examples

It is always easier to start with an existing application rather than starting with a blank slate. Infineon provides a wealth of ModusToolbox™ any cloud 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 Infineon GitHub web site:

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

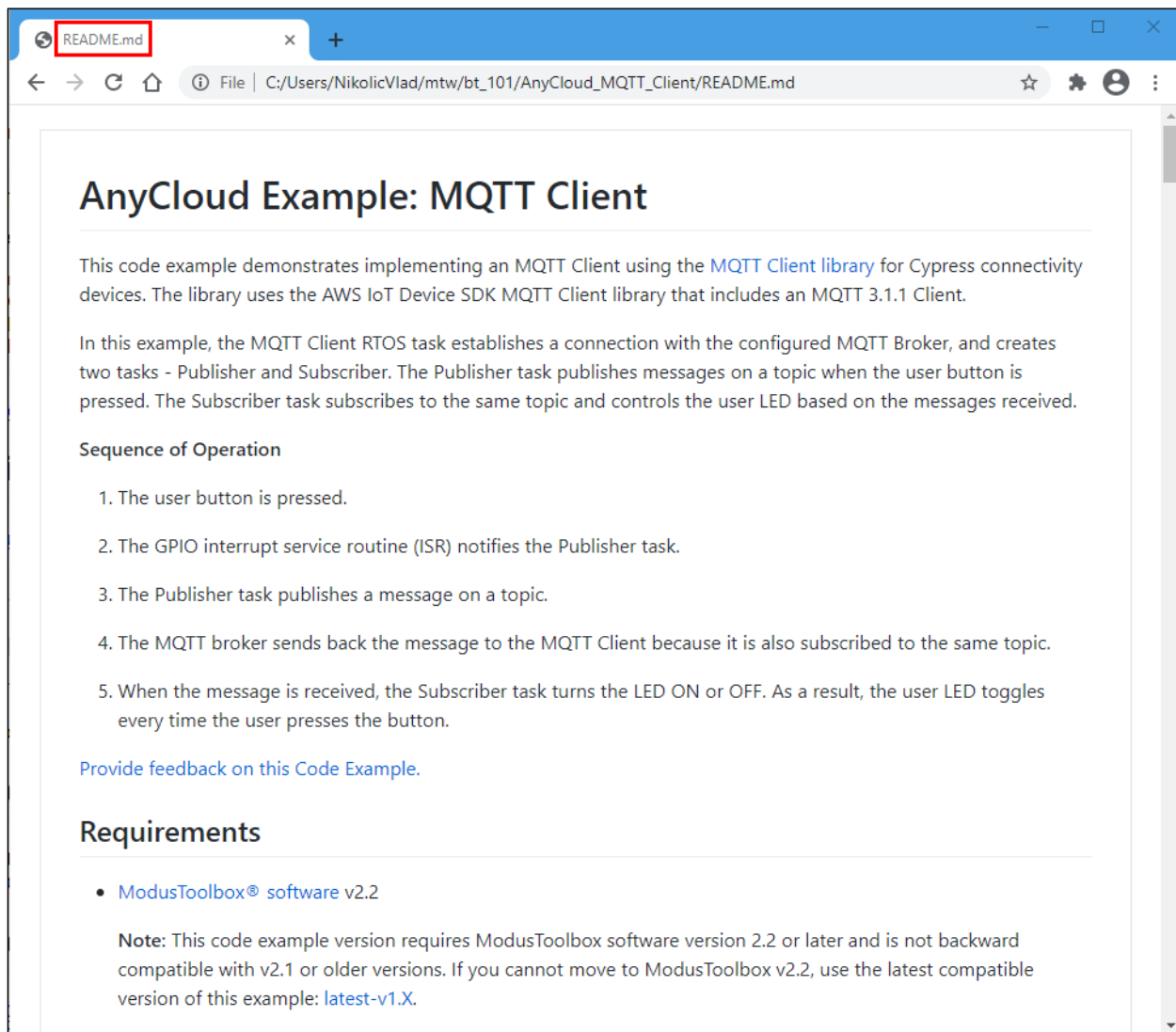
3. From the Cypress website:

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

The first method is recommended if you want to create an application from the code example since it handles all of the BSP and library downloads automatically.

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.



The screenshot shows a web browser window with a single tab titled 'README.md'. The address bar shows the file path: 'C:/Users/NikolicVlad/mtw/bt_101/AnyCloud_MQTT_Client/README.md'. The main content area displays the title 'AnyCloud Example: MQTT Client' in a large, bold font. Below the title, there is a paragraph explaining that the code example demonstrates implementing an MQTT Client using the MQTT Client library for Cypress connectivity devices. It mentions that the library uses the AWS IoT Device SDK MQTT Client library that includes an MQTT 3.1.1 Client. Another paragraph describes the example's functionality: the MQTT Client RTOS task establishes a connection with the configured MQTT Broker, creates two tasks (Publisher and Subscriber), and 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. A section titled 'Sequence of Operation' lists five steps: 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. Below this list is a link 'Provide feedback on this Code Example.' and a section titled 'Requirements' which lists 'ModusToolbox® software v2.2'. A note states that 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.

1.9 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": { "number": 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>

1.10 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}}}
```

1.11 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 was created in 2009 by Dave Gamble which you can find online. A copy of this library that has been verified to work with PSoC™ 6 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.

1.11.1 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. `valuelstring`, `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": { "number": 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"
```

1.11.2 JSON_parser library

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 complex 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 was 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": { "number": 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) &&
        (strcmp(obj_p->parent_object->object_string, "address", strlen("address")) == 0) &&
        (strcmp(obj_p->object_string, "zipcode", strlen("zipcode")) == 0) &&
        (obj_p->value_type == JSON_NUMBER_TYPE) ){
        /* Get zipcode value and convert to an integer */
        sprintf(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 *wifi-connection-manager* library in your project. That library includes the *freertos* library and the *wifi-mw-core* library. The *wifi-mw-core* library includes the *connectivity-utilities* library containing the JSON parser functions.
 - a. 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"
```


1.12 Exercises

Exercise 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 **ch01_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`.

Note: Refer to the section titled [cJSON library](#).

Note: Don't forget to add and initialize the *retarget-io* library.

Exercise 2: Process a JSON document using "JSON_parser"

Repeat the cJSON exercise using the *JSON_parser* library.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2021 Infineon Technologies AG.
All Rights Reserved.

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.