Chapter 5b: AnyCloud Solution

After completing this chapter, you will understand various AnyCloud concepts and how to use it to connect to the Cloud.

5b.1 Introduction

In the previous chapter, we covered all the basics of using PSoC 6 in ModusToolbox: how to create and configure an application, how to write/program/debug those applications using various IDEs, how to use and manage libraries, how to use board support packages, etc.

But PSoC 6 is only half the story. The development kit you are using also has a 43xxx connectivity device (either a CYW4343W or a CYW43012) 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 cover the Wi-Fi part of the picture, and in the next one we will cover Bluetooth.

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 independently. You will see an example using Coex to use Bluetooth LE for onboarding a Wi-Fi device in the PSoC 6 Low Power chapter.

We provide 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 the data is 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. Both are covered in later chapters. Note that if you are using Amazon's cloud services but not their IoT Core device management, AnyCloud is a great solution for you, as you will see in some of the exercises.

AnyCloud 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).

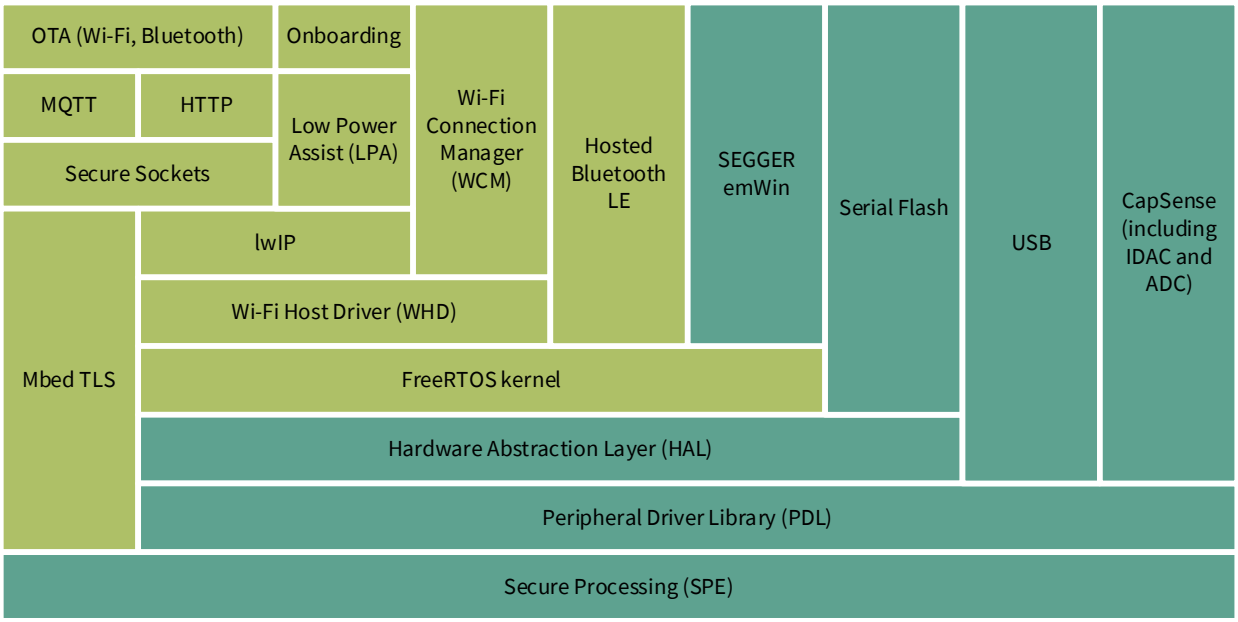
AnyCloud supports MQTT and HTTP application layer cloud protocols in addition to TCP and UDP.

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.

5b.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 us, while others use industry standard open source libraries. As you will see later, these can be pulled into a ModusToolbox application 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. When you include a library in your ModusToolbox application, the repository is downloaded into the application directory under the *libs* subdirectory.

You might notice that there is no Bluetooth LE library for the BLESS hardware on PSoC 63 devices. This is because AnyCloud is designed to support PSoC 6 MCU with a combo device. It is theoretically possible to use a PSoC 63 device, with on-chip Bluetooth LE, in conjunction with 43xxx Wi-Fi 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.

5b.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 necessary to integrate the WHD with lwIP, MbedTLS, and FreeRTOS. By adding this 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 our 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.
- Connectivity Utilities - A library of general-purpose utilities such as a JSON parser, link list library, string conversion utilities, etc.

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.

5b.2.2 Wi-Fi Connection Manager (WCM)

The WCM makes Wi-Fi connections easier and more reliable. Firstly, it implements 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*.

5b.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 MQTT library includes the Wi-Fi Middleware Core library as a dependency.

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

5b.2.4 HTTP

AnyCloud contains both server (http_server) and client (http_client) libraries. These libraries support non-secure (HTTP) and secure (HTTPS) access using HTTP version 1.1. The libraries support RESTful methods such as GET, PUT, POST, and HEAD.

5b.2.5 Over-The-Air (OTA) Bootloading

The OTA toolkit 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 OTA bootloading library includes the MCUBoot, WiFi-Connection-Manager, MQTT and Retarget-IO libraries as dependencies as well as BSPs for kits that are supported out of the box.

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

5b.2.6 WICED Bluetooth/Bluetooth LE Hosted Stack and BTSTACK

In addition to the great Wi-Fi support in AnyCloud, 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 AnyCloud examples will show you that exact thing.

The Library Manager name for the WICED Bluetooth/Bluetooth LE Hosted Stack library is *bluetooth-freertos*. The WICED Bluetooth/Bluetooth LE Hosted Stack library includes the BTSTACK library automatically.

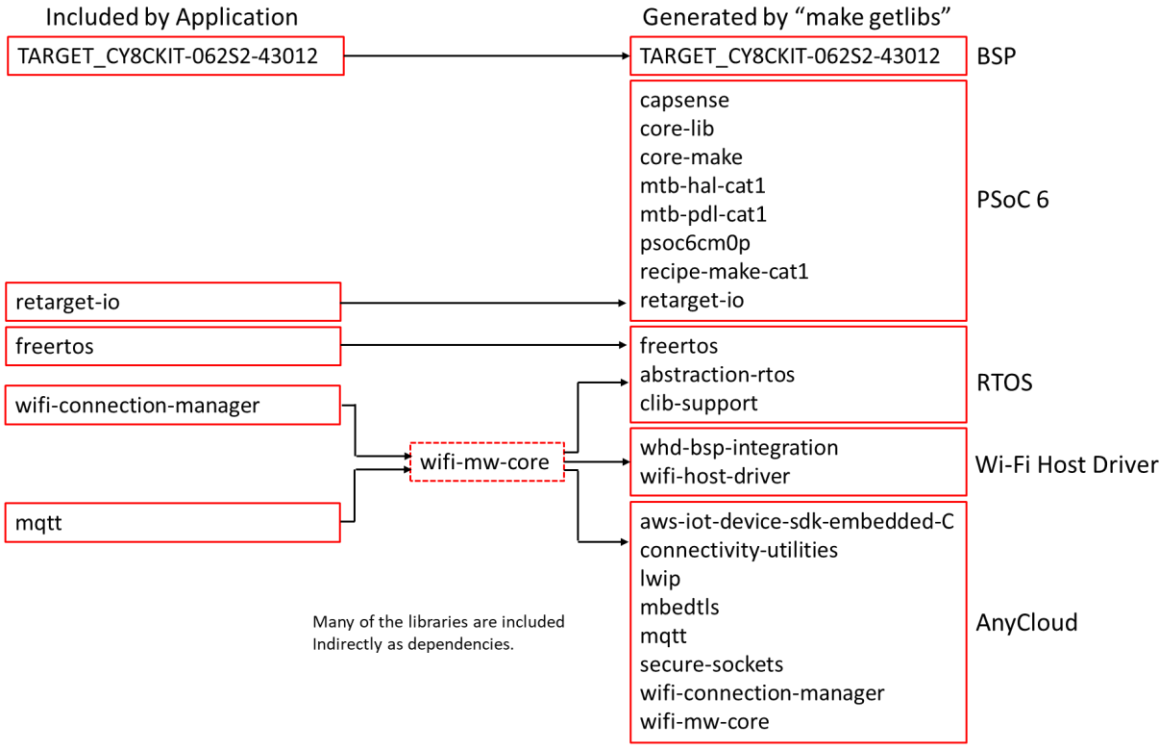
5b.2.7 Low Power Assistant (LPA)

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*.

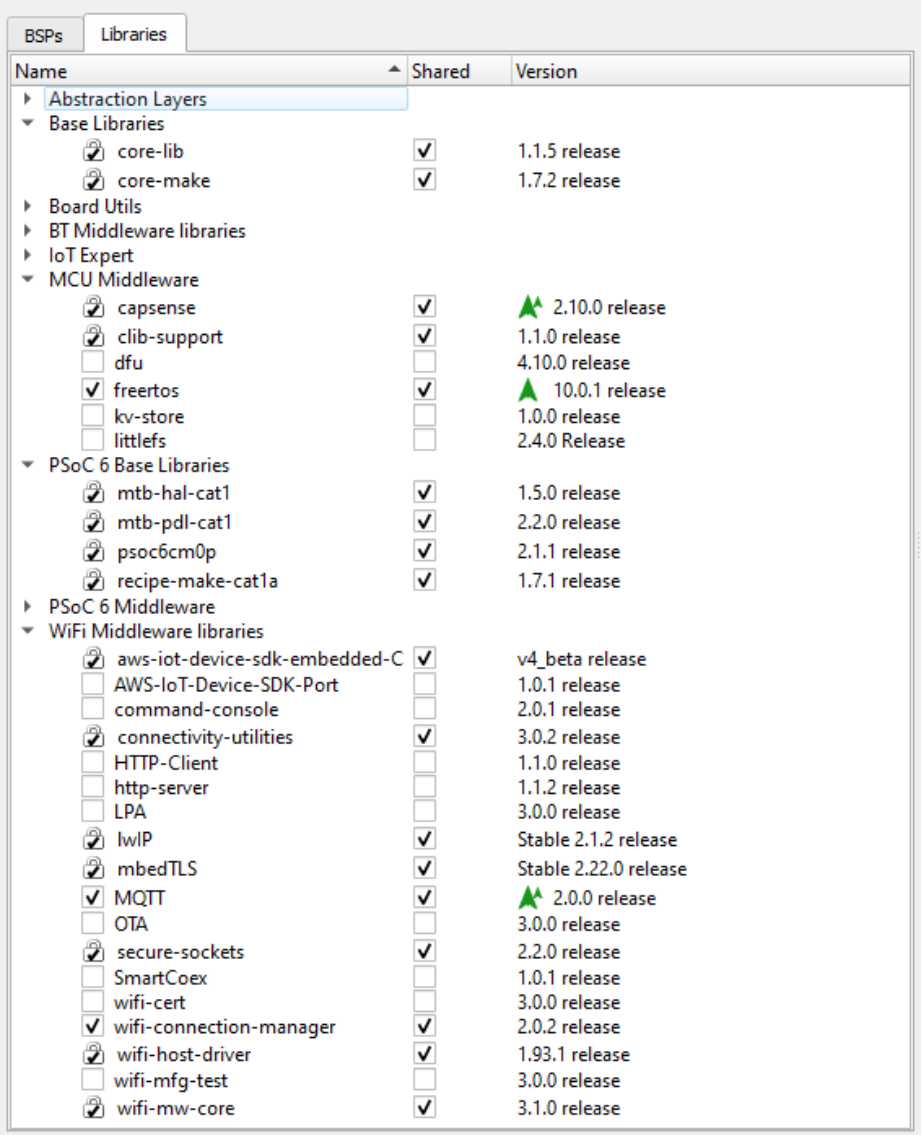
5b.3 Library Inclusion in an Application

As you have seen previously, libraries can include other dependent libraries, and AnyCloud libraries are no exception. As an example, the libraries for the AnyCloud MQTT Client code example and where they come from are shown here. The user application only includes 5 *.mtb* files but once the application has been created, it will contain 22 different libraries. Note that some code examples may still use the LIB flow instead of the MTB flow, but the concepts are the same.



5b.4 Library Manager

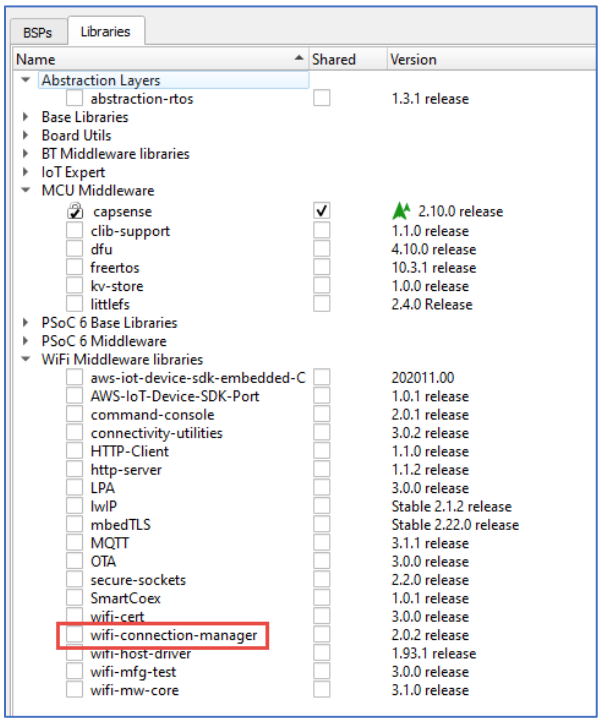
The following image shows the ModusToolbox Library Manager tool with the same AnyCloud MQTT Client code example and Wi-Fi libraries.

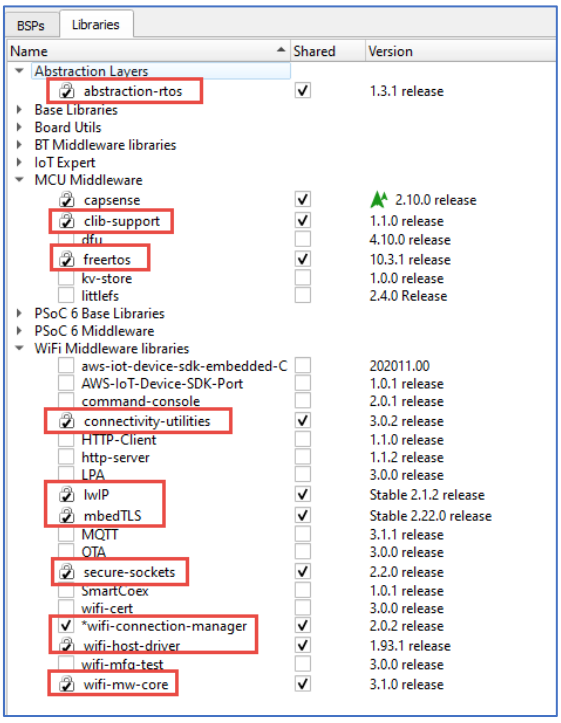


5b.4.1 Adding Libraries

As noted earlier, when the ModusToolbox build system encounters a *.mtb* file, it adds that library of code to the application. That library may specify additional dependent libraries in the manifest file. The ModusToolbox build system reads the manifest files so that all dependent libraries are added automatically.

The following images show the libraries included as part of an Empty PSoC 6 application before and after adding the WCM library.





When adding only the WCM library to the project, several other libraries are automatically included as well. The WCM library has dependencies specified in the manifest file for the libraries it depends upon. The same paradigm applies to various AnyCloud libraries that have dependencies. If you add a library that requires other libraries, they will be added automatically.

Dependencies are not necessarily bi-directional. The LPA, WCM, and mbedTLS libraries provide good examples to illustrate this concept. The WCM does not depend upon the LPA. The LPA is optional. So adding WCM to a project does not add the LPA library. However, the LPA requires the WCM. So when you add the LPA to a project, the WCM (and all its dependencies) are added automatically. Similarly, the WCM depends upon the MbedTLS library. However, the MbedTLS library does not depend upon the WCM. It has no dependencies of its own.

5b.5 Library Configuration Files

Some libraries provide configuration header file templates. You saw this previously with the *FreeRTOSConfig.h* file. When adding a library that has configuration templates to an application, the configuration files should be copied to the top-level application directory so that they can be customized. When there are multiple files with the same name in an application, the build system will automatically pick the one at the top-level.

If you want to put the application specific configuration files in a different location, you must specify the path to that directory (relative to the project's root directory) in the INCLUDES variable in the application's Makefile. Files in that path are guaranteed to be included before the build system searches through the application's hierarchy.

Some examples of configuration files that you may need:

Wi-Fi Middleware Core

The template files are in the *libs/wifi-mw-core/configs* subdirectory. See also the Quick Start in the library's README.md file (https://github.com/cypresssemiconductorco/wifi-mw-core).

- *FreeRTOSConfig.h*: Settings for FreeRTOS.

  Use this file as a template for FreeRTOS configuration instead of the one from the FreeRTOS library. It has some modifications specific to AnyCloud.

- *mbedtls_user_config.h*: Settings for Mbed TLS.
- *lwipopts.h*: Settings for lwIP

  Applications may choose to modify this file to optimize memory consumption based on the Wi-Fi characteristics of the application.

MQTT

The template file is in the *mqtt/cyport/include* subdirectory. See also the Quick Start in the library's README.md file (https://github.com/cypresssemiconductorco/mqtt).

- *iot_config.h*: Settings for MQTT.

OTA

The template file is in the *anycloud-ota/configs* subdirectory. See also the library's README.md file (https://github.com/cypresssemiconductorco/anycloud-ota).

- *cy_ota_config.h*: Settings for OTA.

5b.6 Makefile Changes

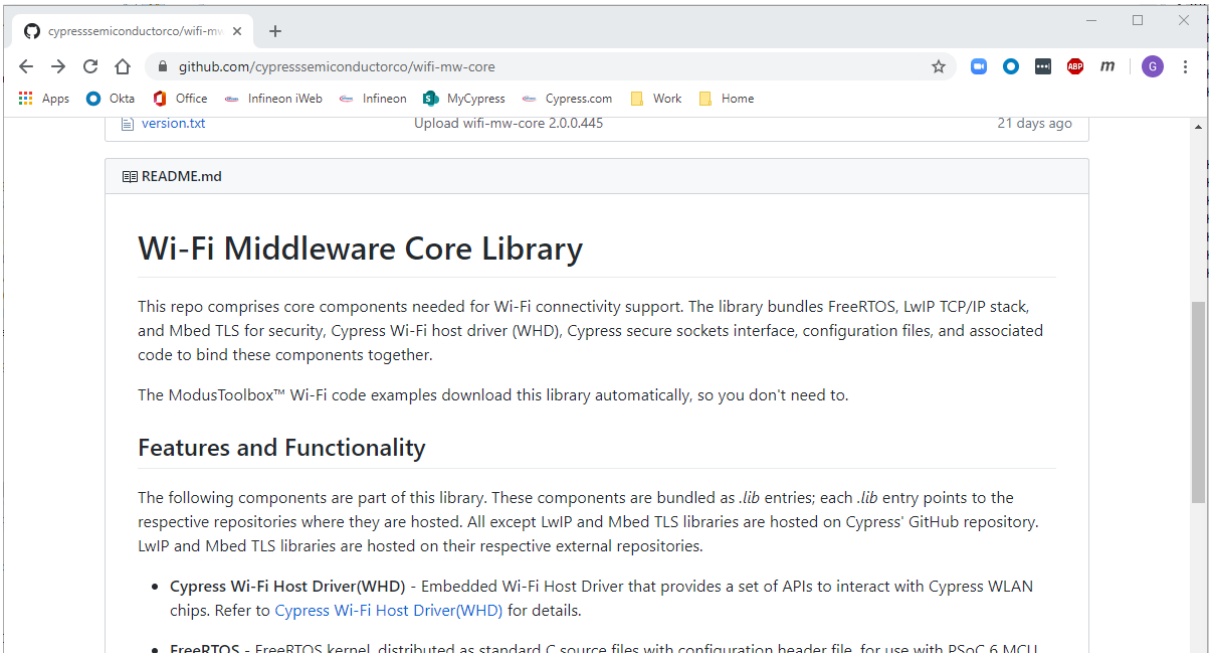
There are a few changes required to the Makefile to get the AnyCloud libraries included properly. If you use one of the AnyCloud code examples, all of this is already done for you. These changes are also shown in the Wi-Fi Middleware Core library documentation. The required changes are:

```
COMPONENTS+= FREERTOS PSOC6HAL LWIP MBEDTLS
DEFINES+= MBEDTLS_USER_CONFIG_FILE='"mbedtls_user_config.h"'
DEFINES+= CYBSP_WIFI_CAPABLE CY_RTOS_AWARE
```

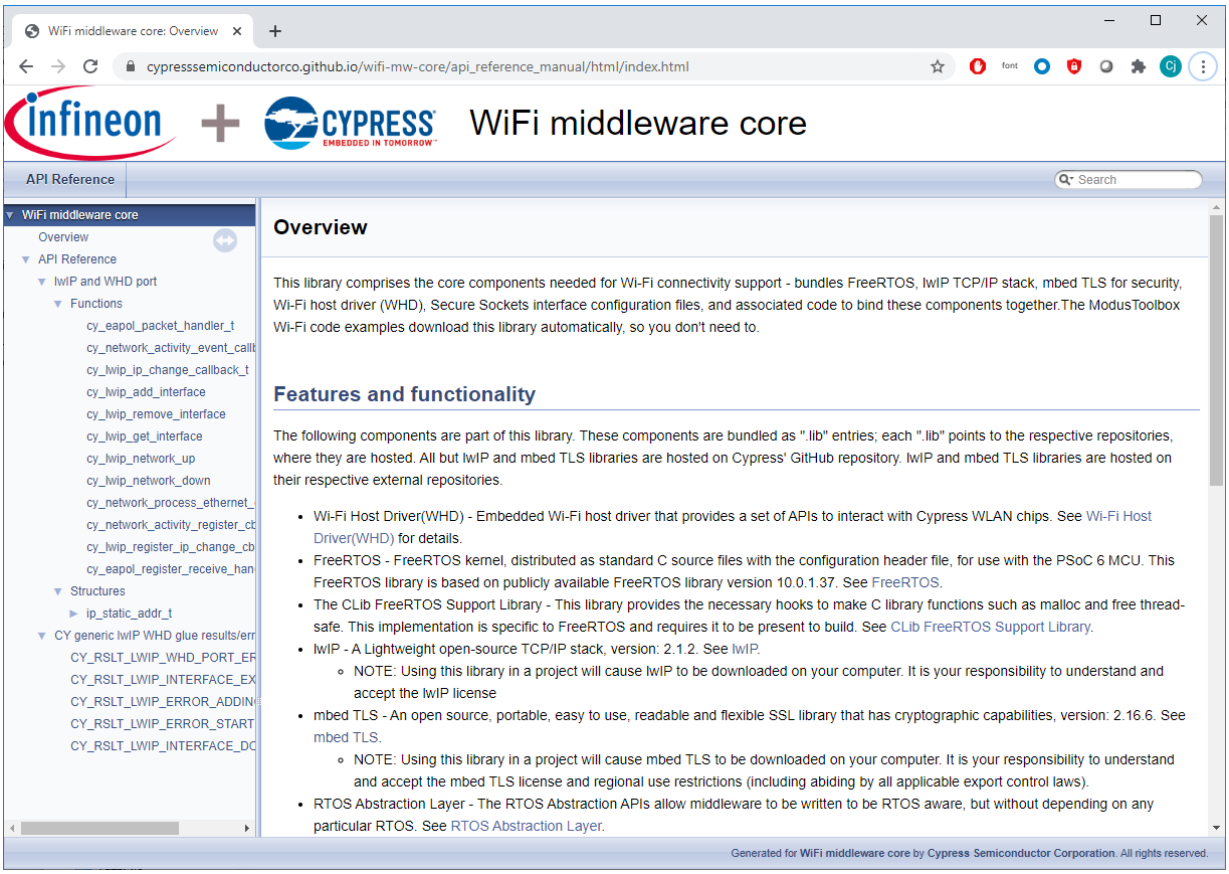
5b.7 Documentation

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 from inside the Eclipse IDE for ModusToolbox using links in the Quick Panel.

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



Once you have downloaded a library, you can go to the "docs" folder where you will typically find an *api_reference_manual.html* file with full descriptions of the API:



5b.8 Code Examples

5b.8.1 Finding Code Examples

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

All 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 following 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

5b.8.2 Code Example List

Here is a sample of what you will find. The complete list includes other applications and list is expanding all the time:

Wi-Fi Scan

This example demonstrates how to configure different scan filters provided in the Wi-Fi Connection Manager (WCM) middleware and scan for the available Wi-Fi.

WPS Enrollee

This example demonstrates how to use the connection management and WPS Enrollee features provided in the Wi-Fi Connection Manager (WCM) middleware to ease connection to a Wi-Fi access point that supports WPS (Wi-Fi Protected Setup).

Bluetooth LE Wi-Fi Onboarding

This example demonstrates how to use a Bluetooth LE client (e.g. a cellphone) to configure the device to connect to a Wi-Fi access point. The Wi-Fi SSID and password are stored in on-chip emulated EEPROM so that the device will automatically reconnect on subsequent power cycles.

We will talk in detail about using Bluetooth LE with AnyCloud in a later chapter.

TCP Keepalive Offload

This code example demonstrates the TCP Keepalive Offload functionality offered by our Wi-Fi devices when paired with a PSoC 6 MCU. It employs the Low Power Assistant (LPA) middleware library which helps in developing low power wireless applications.

We will talk in detail about the LPA in a later chapter.

TCP Server

This code example demonstrates a TCP server. Functionality can be demonstrated either using a provided Python script to act as a TCP client, or by programming a second kit with the TCP client example so that they two kits can communicate with each other. The server sends LED ON/OFF messages when a user button is pressed on the kit.

TCP Client

This code example demonstrates a TCP client. Functionality can be demonstrated either using a provided Python script to act as a TCP server, or by programming a second kit with the TCP server example so that they two kits can communicate with each other. The client receives LED ON/OFF messages from the server and controls its own LED based on the message received.

Secure TCP Server

This code example demonstrates a TCP server using SSL (Secure Sockets Layer) encryption. Functionality can be demonstrated using a provided Python script to act as a TCP client. Example certificates and keys are provided as part of the example for use with the provided Python client.

Secure TCP Client

This code example demonstrates a TCP client using SSL (Secure Sockets Layer) encryption. Functionality can be demonstrated using a provided Python script to act as a TCP server. Example certificates and keys are provided as part of the example for use with the provided Python server.

MQTT Client

This code example demonstrates connecting to an AWS IoT using MQTT. Example certificates and keys are provided as part of the example.

MQTT OTA Firmware Update

This code example demonstrates Over-the-Air (OTA) firmware update MQTT. The device establishes a connection with a designated MQTT broker and then periodically checks to see if a firmware update is available. When an update is available, it is downloaded and installed on the device.

HTTPS Server

This code example demonstrates a secure HTTP server. It allows a secure connection from an HTTPS client though SSL handshake. Once the handshake completes, the client can make GET, POST and PUT requests with the server.

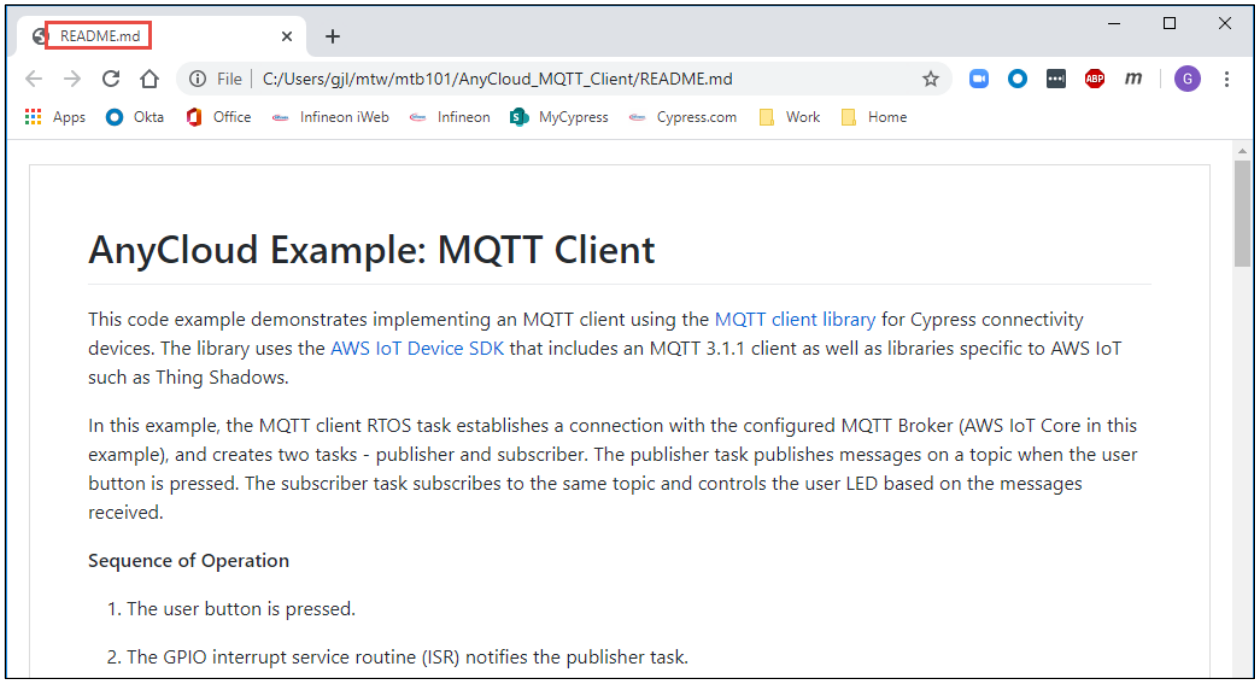
HTTPS OTA Firmware Update

This code example demonstrates Over-the-Air (OTA) firmware update using HTTPS. The device establishes a connection with a designated HTTPS server and then periodically checks to see if a firmware update is available. When an update is available, it is downloaded and installed on the device.

5b.8.3 Code Example Documentation

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 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 will be formatted strangely since Eclipse doesn't support everything that Markdown can do.



5b.9 Walk-Through

The Cypress GitHub site contains many AnyCloud example applications. As a starting point, we will create and run the TCP Client example to get an understanding of how a WiFi connection is established and how data is sent using TCP.

The TCP Client and Server applications each come with a Python program that can be used to act as the other side of the TCP connection (i.e. a Python TCP Server to interact with the TCP Client example, and vice versa). It is also possible to program one kit as a server and a second kit as a client to get them to talk to each other.

Let's start with the AnyCloud TCP Client project:

1. Create the application (call it **ch05b_ex01_tcp_client**)
2. Review README.md
3. Update the values for WIFI_SSID, WIFI_PASSWORD, and WIFI_SECURITY_TYPE in *source/tcp_client.c*

4. Program the Application
5. Run the Python TCP Server Program (tcp_server.py) from the example project directory to test the Client
    a. Note: You need to make sure your computer's firewall is not setup to block traffic on port 50007 since that is the port used by this example.
6. Review the code:
    a. Initialize the BSP, the Retarget-IO library, and the LED pin
    b. Start a FreeRTOS task that will:
        i. Initialize the Wi-Fi connection manager
        ii. Connect to Wi-Fi
        iii. Initialize the Secure Socket library
        iv. Connect to the TCP Server
        v. Receive messages to control the LED and send a response back to the Server

5b.10 Exercises

5b.10.1 Exercise 1: TCP Server and Client

In this exercise, you will create the TCP Server and Client applications and test them using the provided Python scripts. You will then team up with another student to get two kits communicating with one another.

Note For the exercises to work properly using the Python script with your computer as the client or server, you will need to allow access to port 50007 through the firewall.

1. Repeat the TCP Client exercise from the walk-through to create and run **ch05b_ex01_tcp_client**.
2. Create the TCP Server application - call it **ch05b_ex01_tcp_server**.
3. Follow the instructions in the README.md file to configure, program, and test the Server with the Python program.
4. Team up with another student to create a Client and Server that talk to each-other.
5. Review the code and answer the questions below.

Questions

1. What FreeRTOS task is created to handle the TCP Server connection? What does it do?

2. What function is called when the button is pressed in the Server application? What does it do?

3. What callback events are registered for the TCP socket in the Server application?

4. What happens when the Server receives a message from the Client?

5b.10.2 Exercise 2: Secure TCP Server and Client

In this exercise, you will create the Secure TLS versions of the TCP Server and Client applications and test them using the provided Python scripts. You will then team up with another student to get two kits communicating with one another.

Note For the exercises to work properly using the Python script on your computer as the client or server, you will need to allow access to port 50007 through the firewall.

1. Create the Secure TCP Client application - call it **ch05b_ex02_secure_tcp_client**.
2. Follow the instructions in the README.md file to configure, program, and test the Client with the Python program.
3. Create the Secure TCP Server application - call it **ch05b_ex02_secure_tcp_server**.
4. Follow the instructions in the README.md file to configure, program, and test the Server with the Python program
5. Team up with another student to create a Client and Server that talk to each-other.
6. Review the code and answer the questions below.

Questions

1. What functions are added to enable TLS?

2. What is changed in the socket creation function calls to enable TLS?

5b.10.3 Exercise 3: MQTT Client

In this exercise, you will connect an MQTT Client to the Amazon Web Services Cloud to send and receive data from an MQTT Broker. The sequence of operations is:

1. The application connects to Wi-Fi and establishes an MQTT connection with an AWS broker.
2. The application subscribes to LED control messages from the broker.
3. When the user presses the button on the kit, the client publishes an LED control message to the broker.
4. The broker receives the message and sends the message on to any subscribers.
5. The kit receives the message from the broker via its subscription and turns the LED ON/OFF based on the incoming message.

On a given kit, there are three ways to use MQTT to turn the LED ON/OFF:

1. Press the button to send messages to the broker which are then forwarded back to the kit from the broker due to the kit's subscription.
2. Use the test client on the AWS console to send messages to control the LED.
3. Press the button on a second kit to send the message while the first kit receives the message via its subscription (or vice versa).

Steps

1. Create the AnyCloud MQTT Client application - call it **ch05b_ex03_mqtt**.
2. Follow the instructions in the README.md file to configure, program and test the MQTT client.

   When you setup the AWS Cloud side of things (Thing, Certificate, Policy), either follow the Getting Started with AWS tutorial referenced from the README.md file or use the AWS IoT Core chapter that is provided as part of the class material.

   If you use the class AWS account, you MUST change the `MQTT_TOPIC` to include your initials (e.g. <inits>_ledstatus). If not, anyone publishing to the same broker will be able to control your kit.
3. Send messages both by pressing the button on the kit and by using the test client on the AWS console.

Questions

1. How many tasks does `main` create? What are they and what is their purpose?

2. How many tasks does `mqtt_client_task` create? What are they and what is their purpose?

3. What happens in the infinite while loop in `mqtt_client_task`, `publisher_task`, and `subscriber_task`?

4. What is the sequence of function calls when the user presses the button?

5b.10.4 Exercise 4: HTTPS Server

In this exercise, you will create the HTTPS Server application and connect to it using Curl or a web browser.

Note For the exercise to work properly on your computer, you will need to allow access to port 50007 through the firewall.

1. Create the HTTPS Server application - call it **ch05b_ex04_https_server**.
2. Follow the instructions in the README.md file to configure, program, and test the Server.