Chapter 4: Mbed OS SDK & Reference Flow

After completing this chapter, you will understand a top-level view of the Mbed OS environment and how to use it with ModusToolbox.

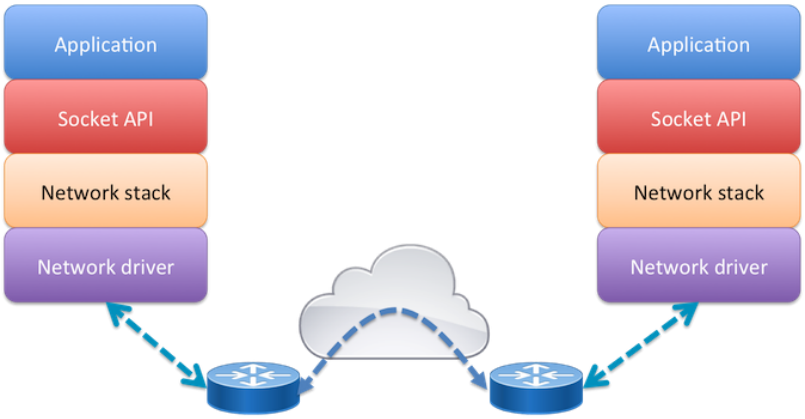
4.1 Tour

4.1.1 Connectivity Overview

Mbed OS offers a strong, integrated stack of standards-based networking technologies:

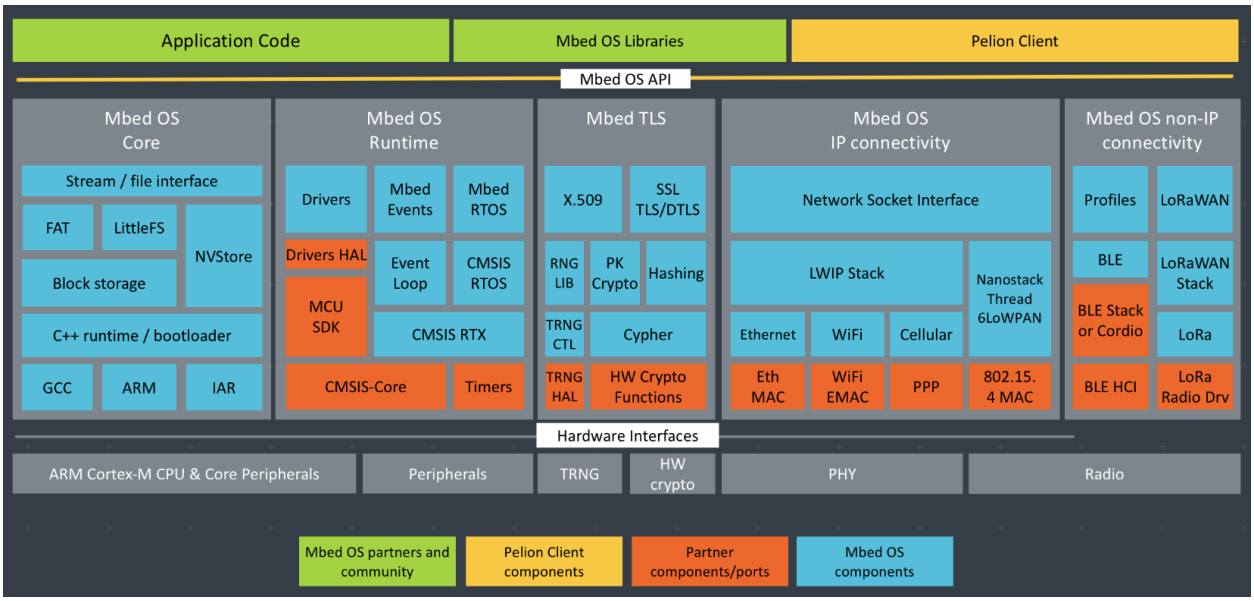
- For IP devices:
  - Cellular
  - NB-IoT
  - Thread
  - Wi-SUN
  - 6LoWPAN-ND
  - Bluetooth Low Energy (BLE)
- For Non-IP devices requiring a gateway:
  - LoRaWAN
  - Cellular

The Socket API standardizes all the connectivity options. It supports both IPv4 and IPv6. Applications are portable regardless of the final connectivity option. Mbed OS provides network drivers, such as Ethernet, Wi-Fi and Cellular.



4.1.2 Firmware Overview

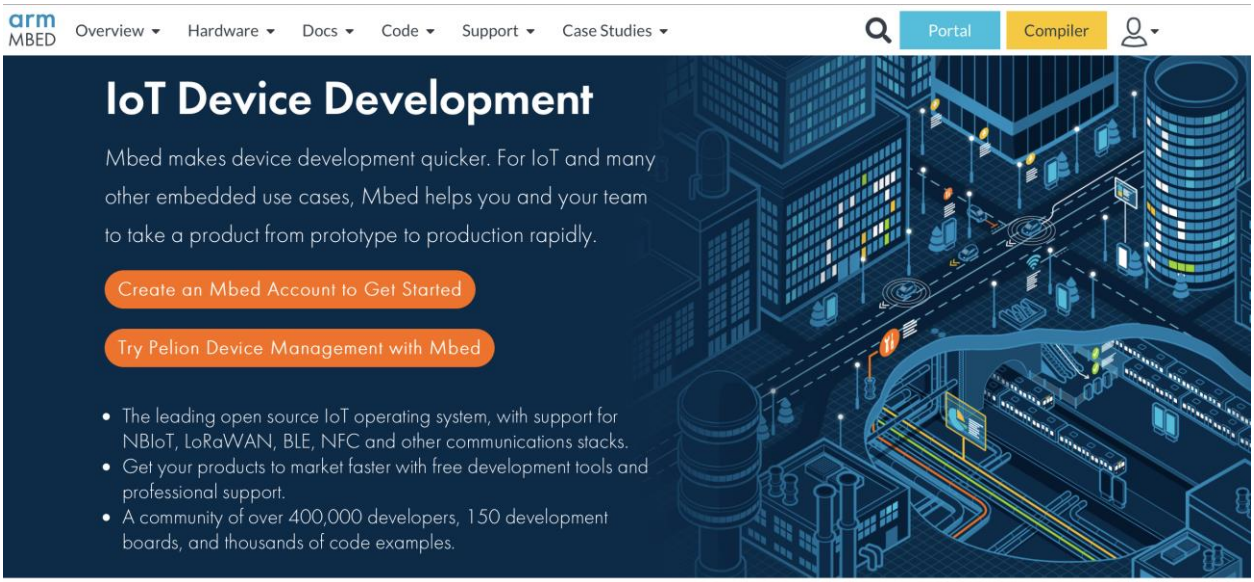
Mbed OS consists of a real-time operating system (RTOS) built on top of CMSIS RTX and a connectivity stack built on top of the lwIP TCP/IP stack, along with a suite of other functionality including BLE.



4.1.3 Mbed Website

From Arm's website www.mbed.com:

> "Arm Mbed OS is a free, open-source embedded operating system designed specifically for the "things" in the Internet of Things.
>
> It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS, and drivers for sensors and I/O devices."

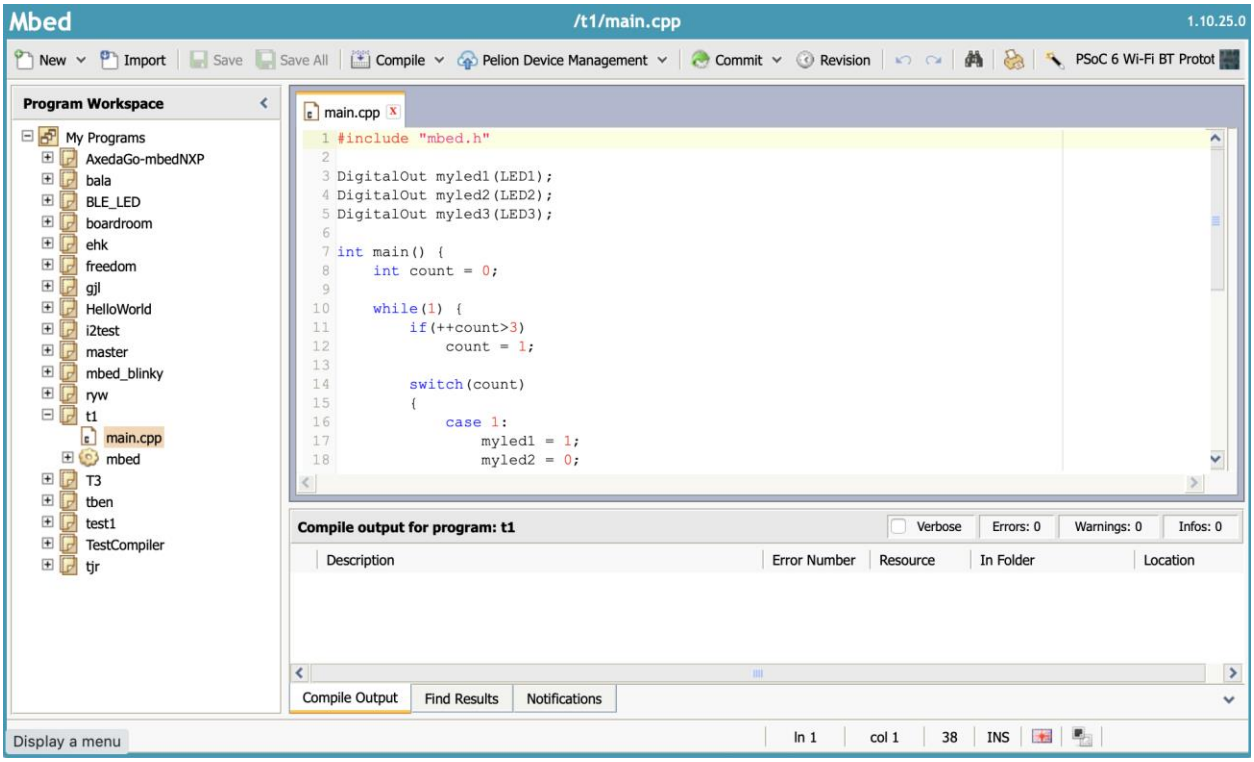


4.1.4 Development Environments

Online Compiler

The original design of the Mbed OS ecosystem was entirely based upon the online compiler environment. Without the ability to use "normal" development, code management, and debugging flows this had the result of essentially killing it for everything except for hobbyist uses. However, with the advent of Mbed OS 5, the addition of RTX plus an offline command line interface, this IoT platform became viable for professional developers.

https://ide.mbed.com/compiler/



Mbed Command Line Interface (CLI)

The Mbed CLI is a Python program that understands how to create projects, compile projects, manage libraries and program/debug chips. The entire interface is built around the command `mbed`, which has many subcommands to perform the different tasks required. A summary of the commands includes the following:

Command	Function
`mbed help`	Print the help menu
`mbed new`	Creates a new Mbed OS project
`mbed detect`	Prints a list of every development kit that is attached to the computer
`mbed config target CY8CKIT_062_WIFI_BT`	Configures the target kit to be the CY…
`mbed config toolchain GCC_ARM`	Configures the toolchain to be GCC_ARM
`mbed target CY8CKIT_062_WIFI_BT`	Same as “mbed config target”
`mbed toolchain GCC_ARM`	Same as “mbed config toolchain”
`mbed compile`	Compiles the project with the configured toolchain. `-f` means flash the target `-t` specify the toolchain (e.g. GCC_ARM) `-m` specify the target (e.g. CY8CPROTO_062_4343W)
`mbed import <URL>`	Creates a new Mbed OS project by reading the URL
`mbed add <URL>`	Add a library to your project
`mbed deploy`	Finds and adds missing libraries from the .lib files and brings them back into the project

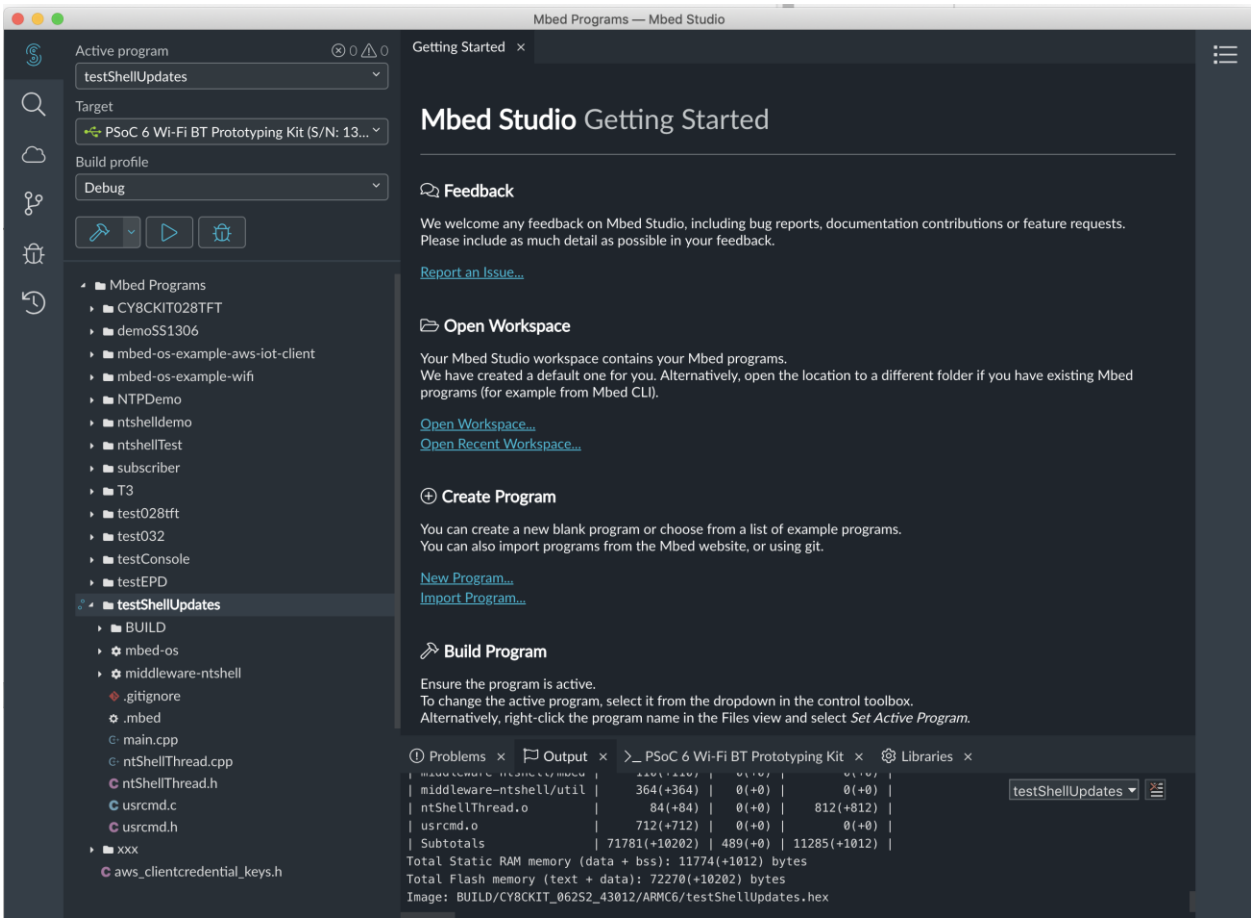
`mbed update <version>`	Update library to a “version”
	e.g.
	`cd mbed-os`
	`mbed update mbed-os-5.14.1`

`mbed export`	Generate project files for a specified IDE.

Note: During installation, we had you install Mbed CLI in a virtual Python environment. That is the safest thing to do so that Mbed CLI doesn't conflict with other applications on your system. It is possible to install Mbed CLI on the native OS, but you run the risk of different versions of tools conflicting.

Mbed Studio

Without the online compiler, there was not a good alternative for an Integrated Development Environment. As a result, Arm decided to invest in building their own tool – called Mbed Studio. This tool includes everything required to create, build, manage, program and debug Mbed OS projects.



Eclipse

Another option to develop Mbed OS projects is to "export" the project to an Eclipse project. To do this you run the `mbed export` command. For example:

```
mbed export -i gnuarmeclipse -m CY8CPROTO_062_4343W
```

This will create a .project file and a .cproject file, which you can then use to import the entire project into Eclipse (or Eclipse IDE). For step-by-step instructions, refer to the Eclipse IDE for ModusToolbox User Guide.

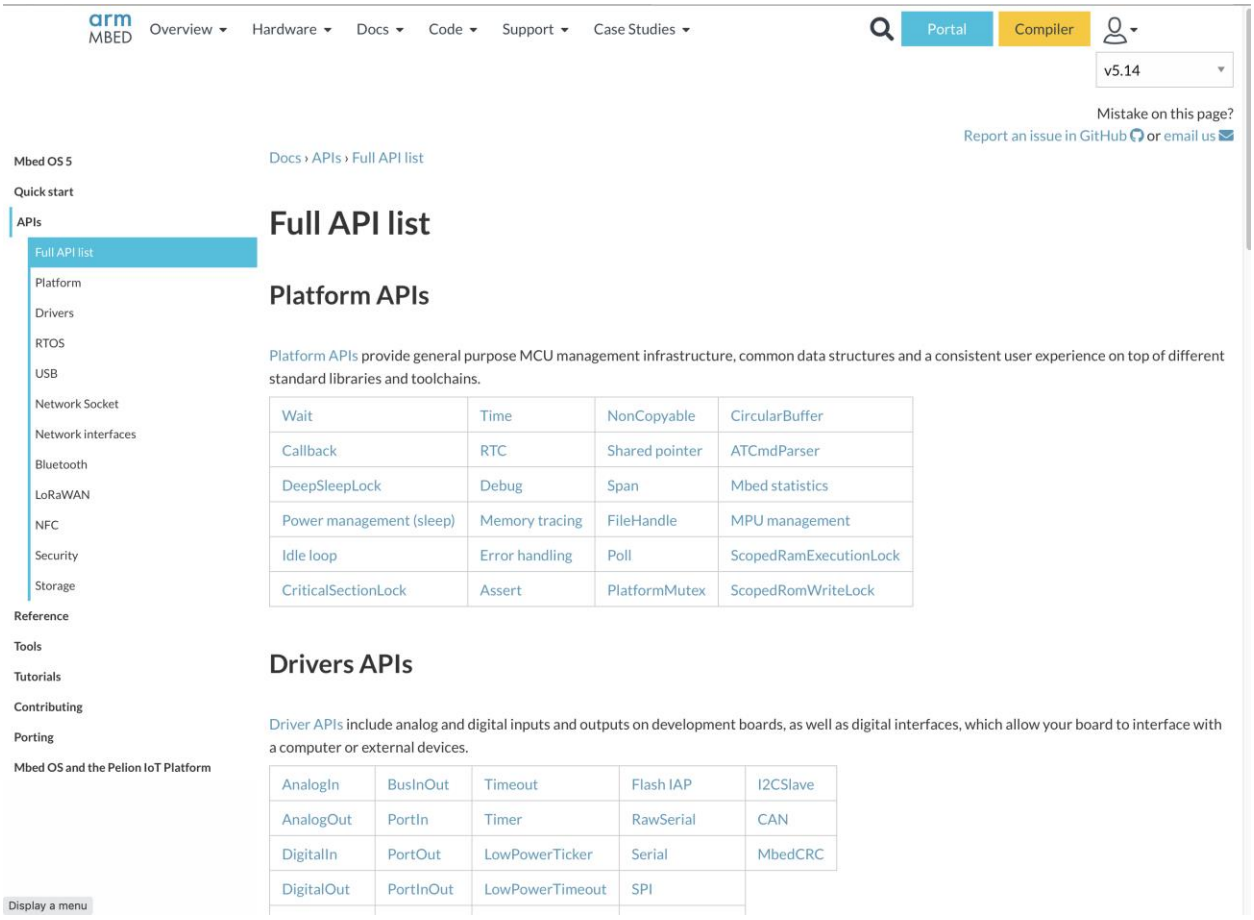
Visual Studio Code

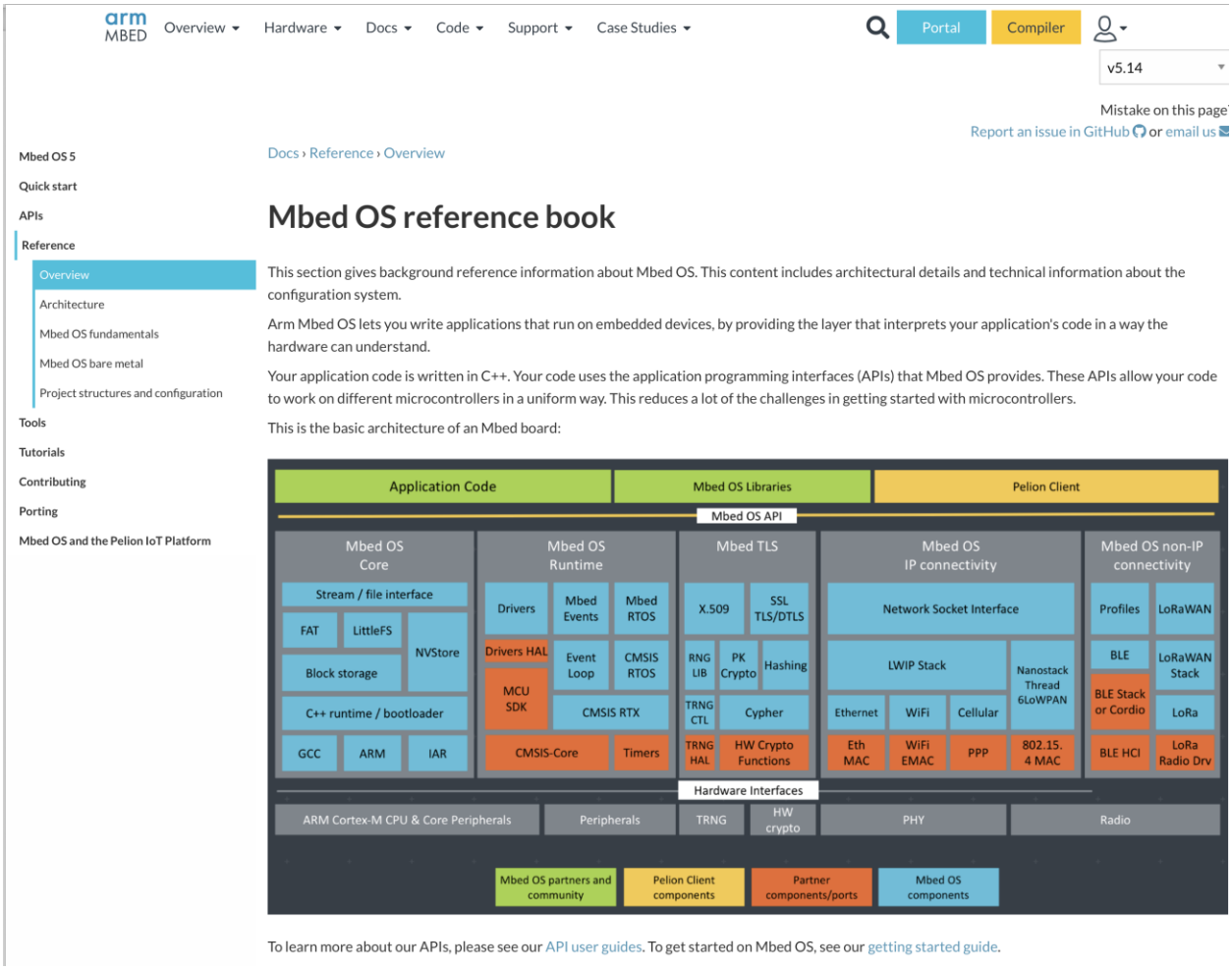
You can export your Mbed OS project to Visual Studio Code by using the `mbed export` command with the argument `vscode_gcc_arm`. For example:

```
mbed export -i vscode_gcc_arm -m CY8CPROTO_062_4343W
```

4.1.5 Documentation

There is extensive documentation available on the Arm Mbed OS website:





Issues/enhancements related to Mbed itself (not Cypress specific issues) can be filed on the Mbed GitHub site at https://github.com/ARMmbed/mbed-os/issues/new.

4.1.6 Mbed OS RTOS

The Mbed OS Real Time Operating System is based on a very mature operating system called RTX, which was developed by Keil and purchased by Arm. Mbed OS has implemented an RTOS HAL that abstracts the RTX API to act like Mbed OS standard. The RTOS has all the normal functions including tasks, semaphores, mutexes, queue, flags, etc.

4.1.7 Network Proxy Settings

Depending on the location of this class you may need to set up your network proxy settings for Mbed. Proxy settings will vary from site to site. When creating applications from inside a Cypress internal network, some proxy settings are necessary due to our security settings. This behavior will be improved in the future. The steps (for now) are:

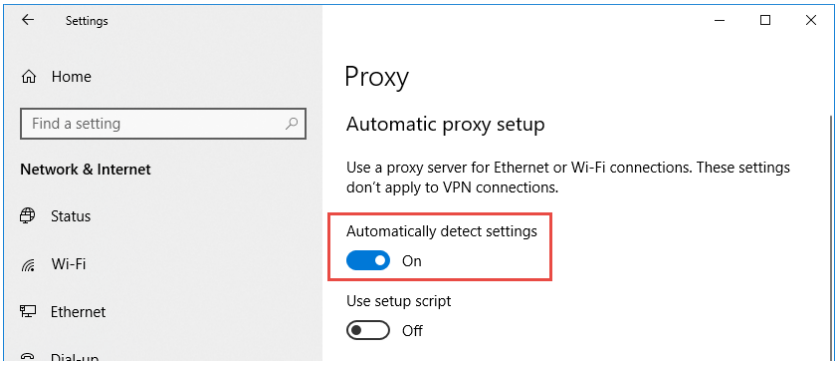
Proxies by Site

Find the proxy host name and port number for your site. A list of Cypress proxies can be found at: https://itsm.cypress.com/solutions/726105.portal.

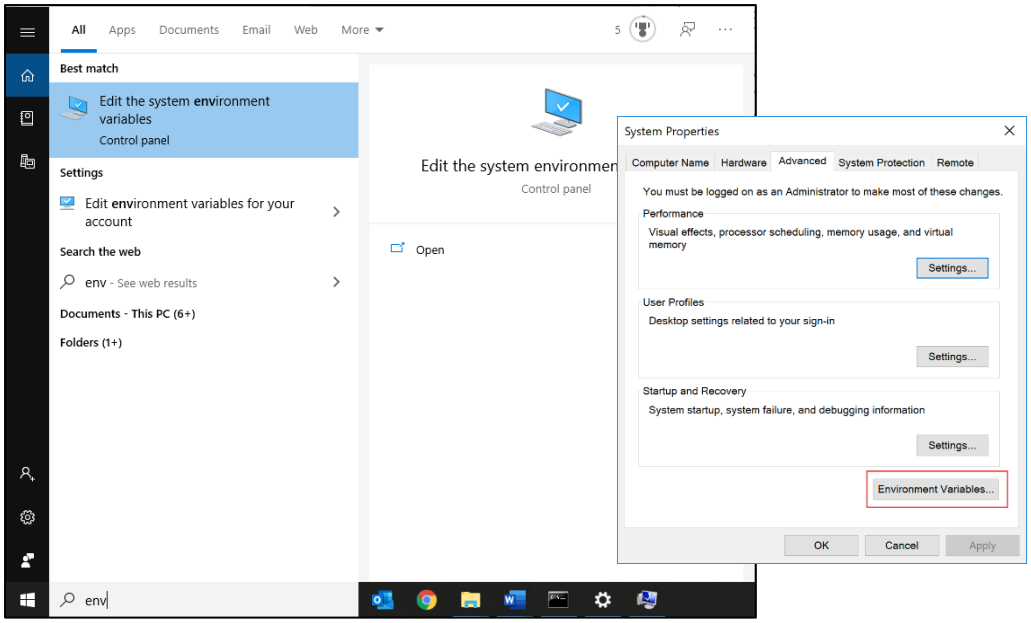
Create OS specific proxy variables:

Windows:

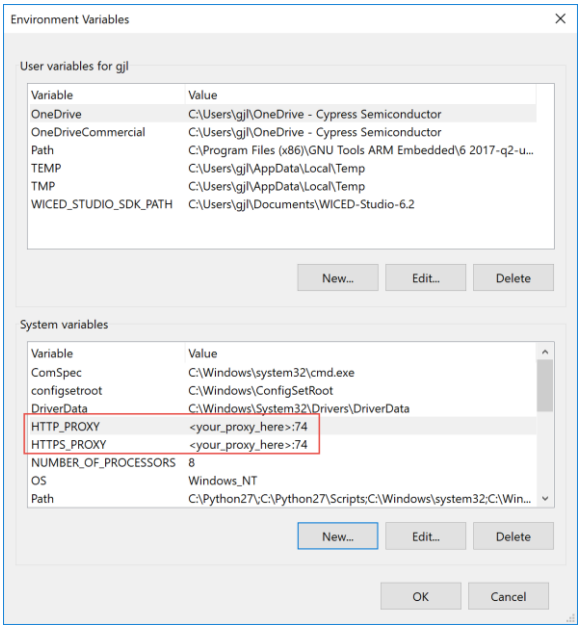
1. Set the Windows proxy settings (**Settings > Network & Internet > Proxy**) to "Automatically Detect Settings."



2. Open the Environment Variables dialog.



3. Create two Windows system variables (HTTP_PROXY and HTTPS_PROXY) using the appropriate proxy name and port number (see previous table):



macOS:

1. Set **System Preferences > Network > Advanced > Proxies > Auto Proxy Discovery**.

2. Open a terminal and check if the variables (http_proxy, https_proxy) are set in the terminal. For example:

```
$ export | grep proxy
declare -x http_proxy="bigbro.ore.cypress.com:74/"
declare -x https_proxy="bigbro.ore.cypress.com:74/"
```

Linux:

1. Set the "Network Proxy" to "Manual".
2. Open a terminal and check if the variables are set in the terminal. For example:

```
$ export | grep proxy
declare -x http_proxy="bigbro.ore.cypress.com:74/"
declare -x https_proxy="bigbro.ore.cypress.com:74/"
```

3. Setup Cypress CA Certificate.

This is Cypress-specific; see: https://confluence.cypress.com/pages/viewpage.action?spaceKey=MIDDLEWARE&title=Development+environment#Developmentenvironment-SetupCypressCACertificates.

4.2 Demo walkthrough

4.2.1 Demo Mbed Online Compiler mbed-os-example-blinky

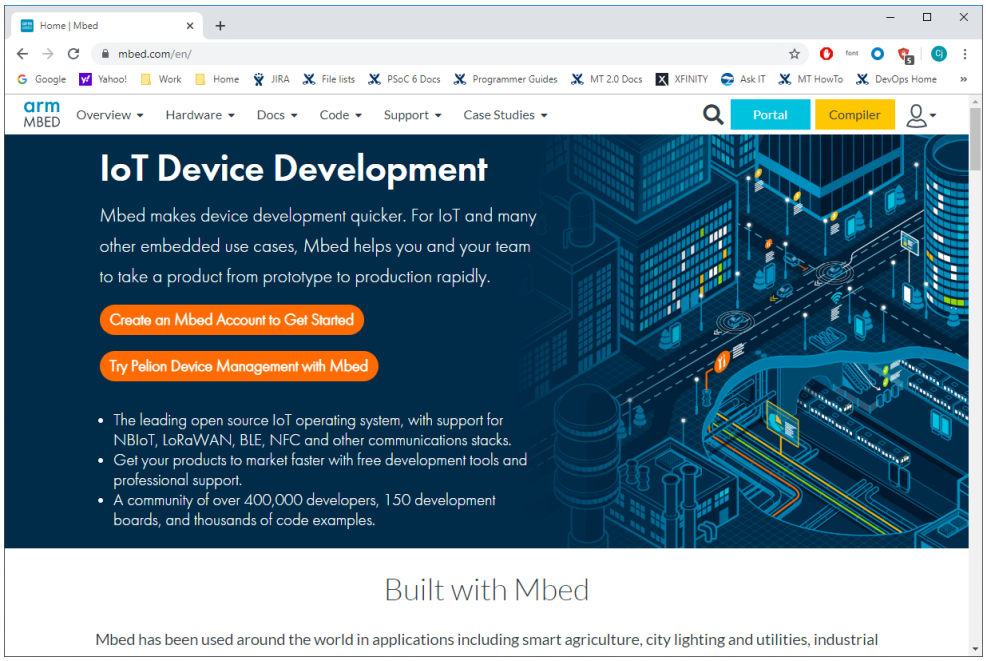
4.2.2 Demo Command Line mbed-os-example-blinky

4.2.3 Demo Mbed Studio mbed-os-example-blinky

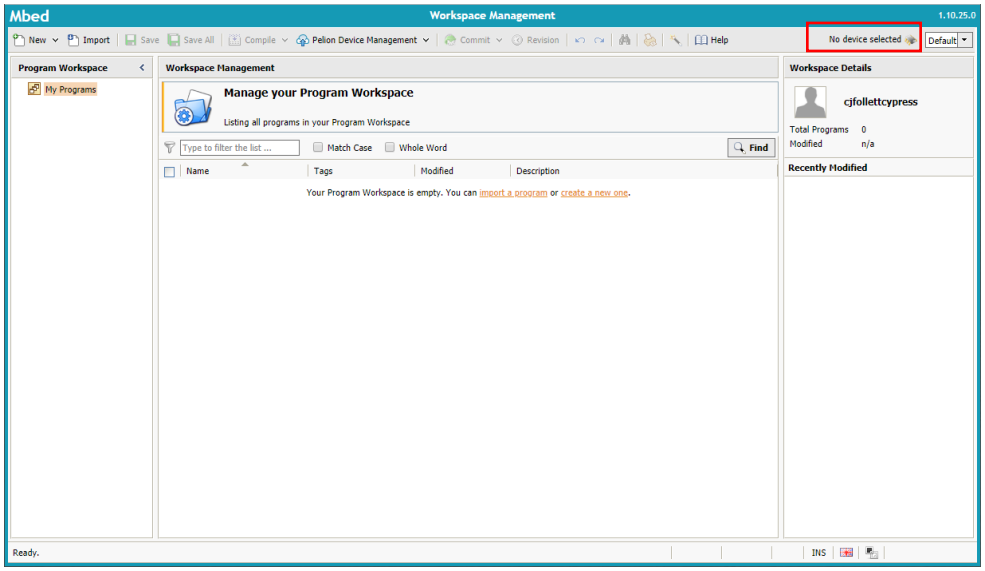
4.3 Exercises (Part 1)

4.3.1 Make a blinky project using the online compiler

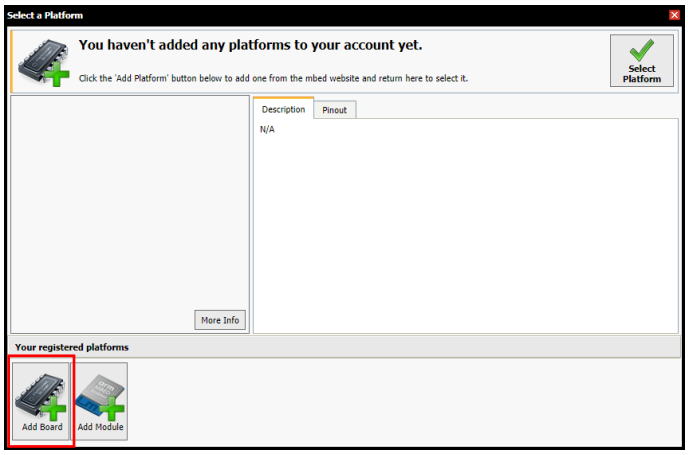
1. If you don’t have an mbed.com account, create it now at mbed.com.
2. Connect the CY8CPROTO_062_4343W kit to your computer and make sure it is in DAPLink mode (fast 2 Hz ramping LED).
3. Next, click the **Compiler** button.



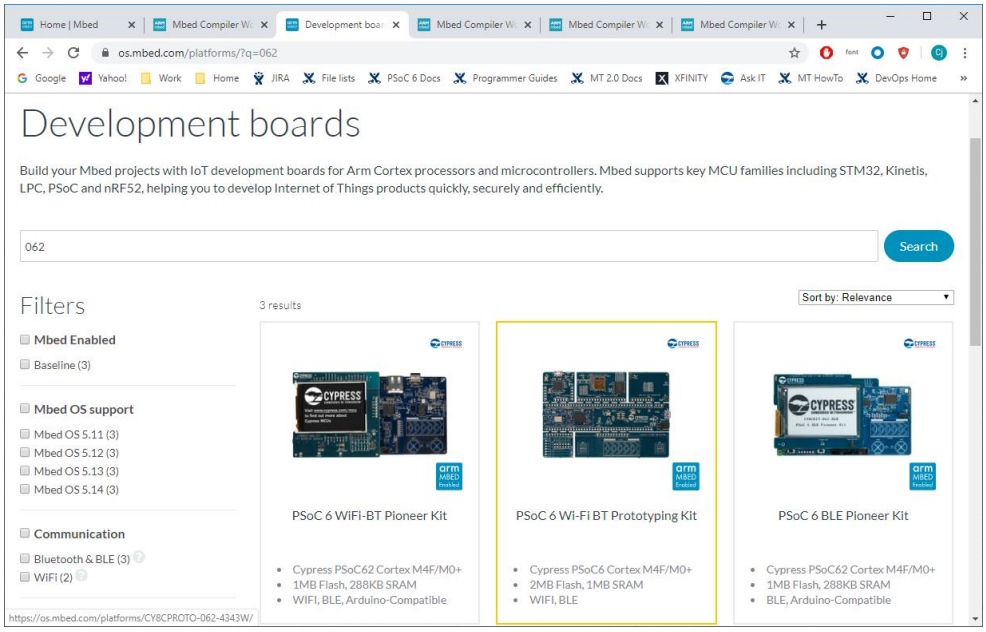
4. If you already have the CY8CPROTO-062-4343W target added to the compiler, skip ahead to step 9.
5. If you don't have the CY8CPROTO-062-4343W targets, add them by clicking **No device selected**.



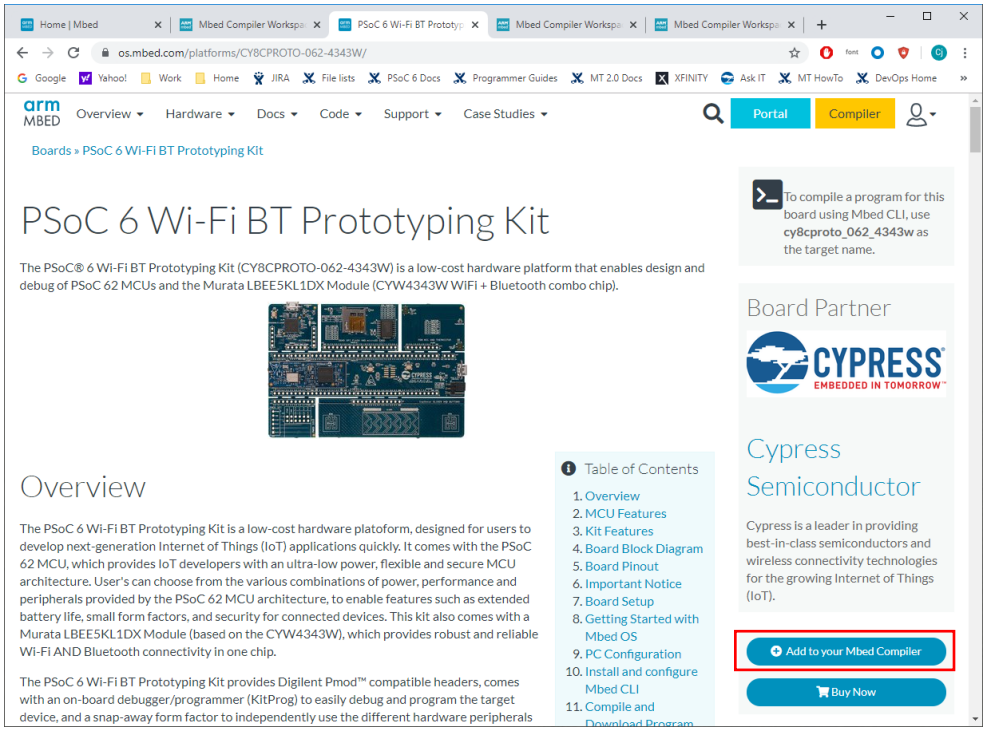
6. Then, click **Add Board**.



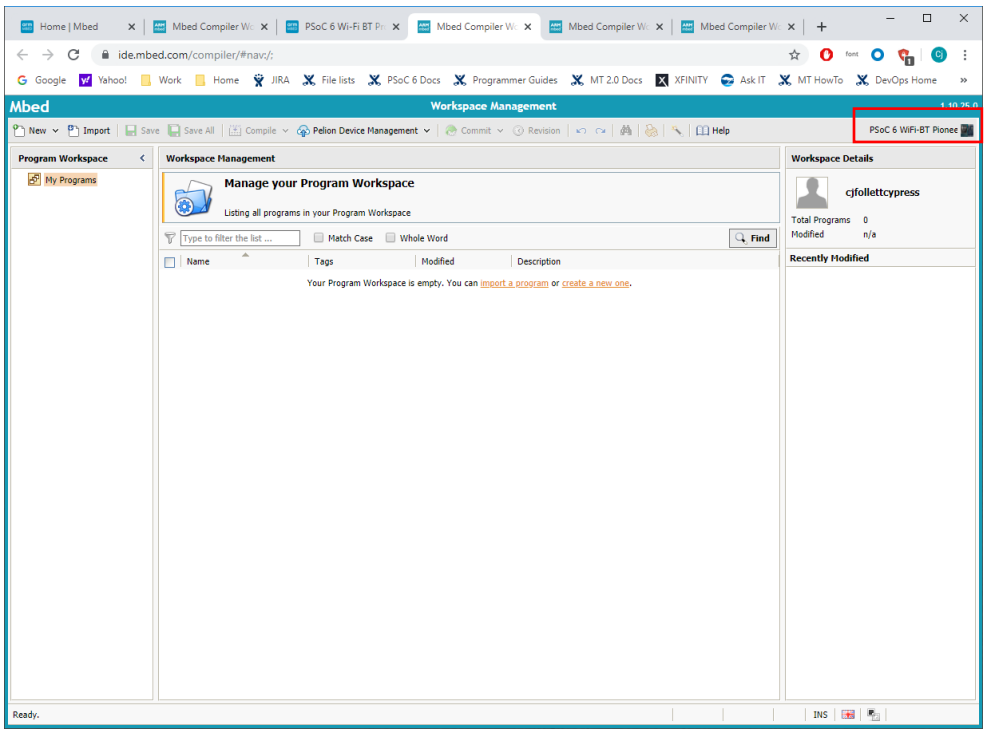
7. Find your board and click on it. You can filter the list to show only Cypress Semiconductor boards.



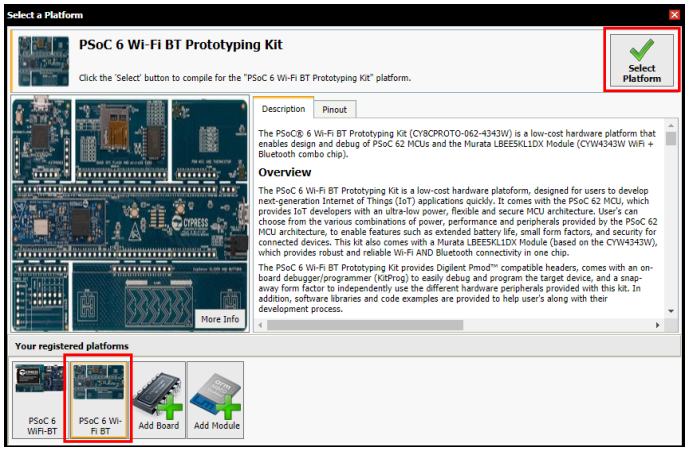
8. Then click **Add to your Mbed Compiler**.



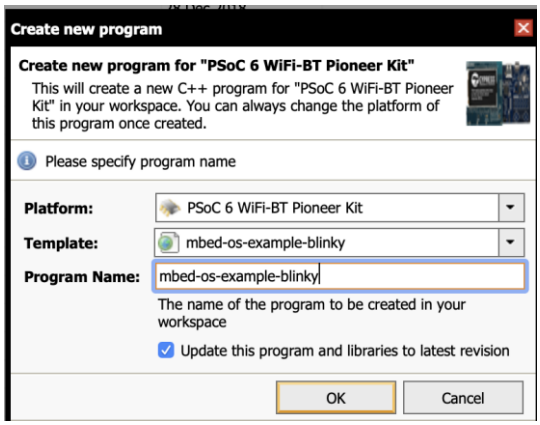
9. Go back to the compiler. Make sure you have the correct target selected. If not, click on the selected target



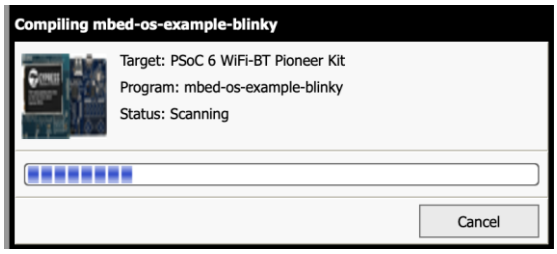
10. Then, select the correct target and click **Select Platform**.



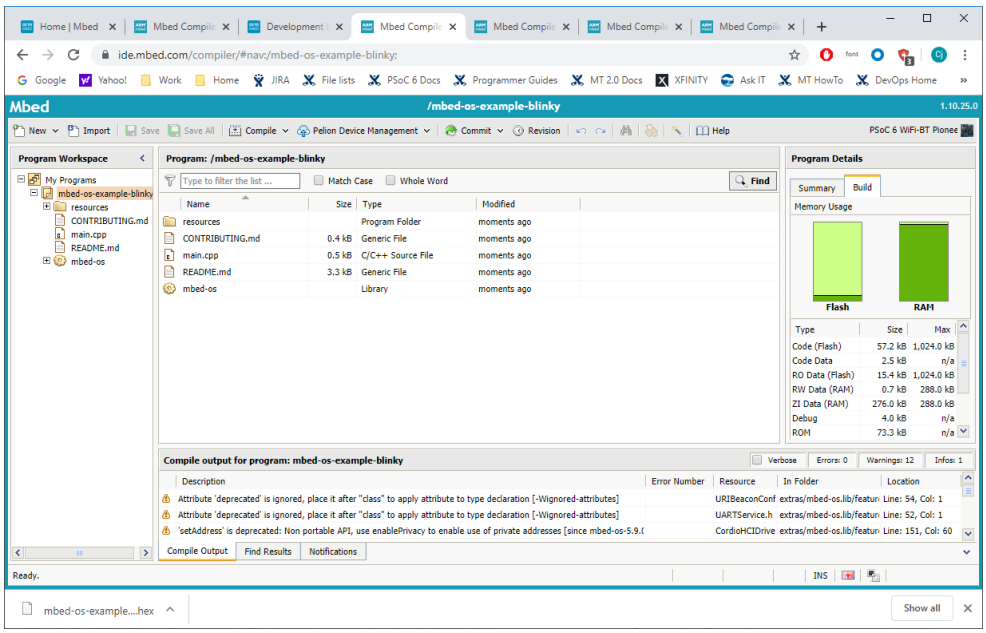
11. On the Workspace Manage page, click the **New** button. Then on the dialog, select the mbed-os-example-blinky template.



12. On the Workspace Manage page, click the **Compile** button and the program begins compiling.

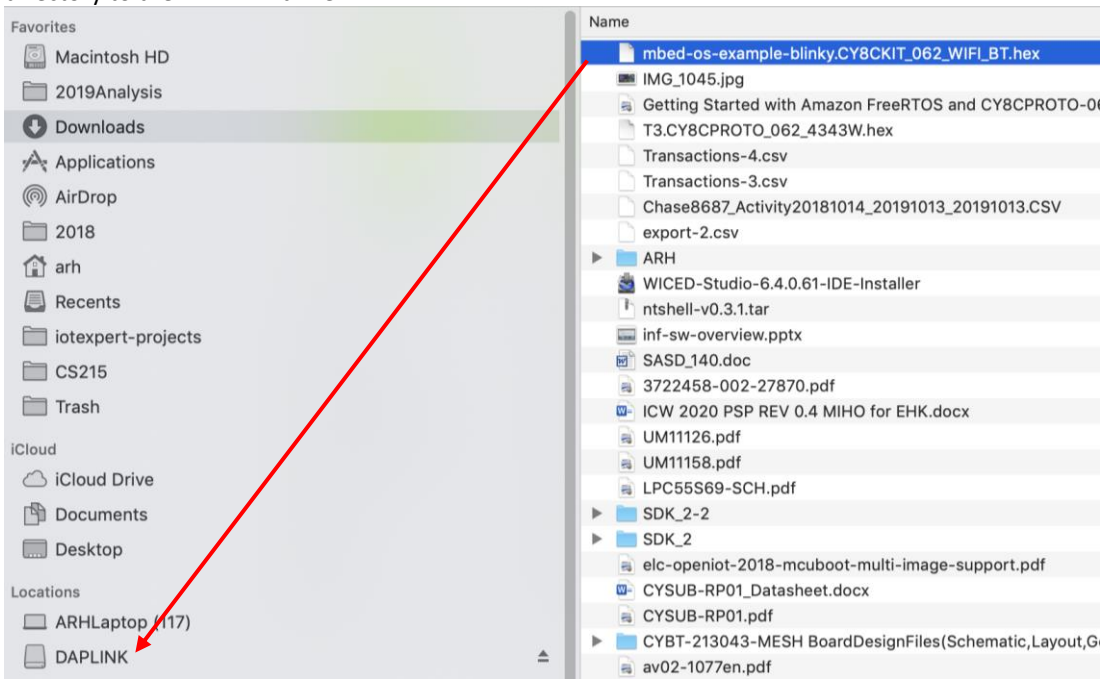


13. When it is done, it will show the compiler results and download a hex file to your computer.



14. To program your board, copy the hex file into the USB DAPLink device using the File Explorer.

Note that the KitProg3 must be in DAPLink/Mass Storage mode. When DAPLink mode is selected, the "DAPLINK" directory appears as a flash drive. Just drag and drop the hex file from the Downloads directory to the DAPLINK drive.



The LED should start blinking on the board once copying is complete.

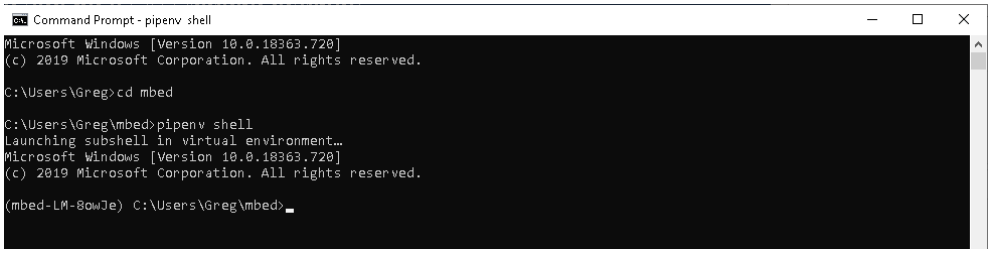
4.3.2 Make a blinky project using the Mbed CLI

This is the simple blinking LED project that we did in the previous example. Except this time, you will do it on your computer with the command line interface. Refer to the Mbed documentation for appropriate steps to set up Mbed CLI.

1. Startup a command line using either `Command Prompt` or `PowerShell` for Windows, or the CLI for Mac.
2. Go to the directory where you created the virtual environment such as C:\Users\<username>\mbed.

   The directory will have a file called "Pipfile". If you run from a directory that doesn't contain a "Pipfile" or whose parents don't contain a "Pipfile", it will create a new environment instead of opening an existing environment. In that case, you probably will want to exit the new environment and remove it. See the SW installation instructions for info on removing a virtual environment.

3. Run `pipenv shell` to start a shell in the virtual environment.



4. Import the project with the `mbed import` command: (It knows to bring it from the Arm GitHub repository.)

   ```
   mbed import mbed-os-example-blinky
   cd mbed-os-example-blinky
   ```

   If you get an error, see Mbed Troubleshooting.

5. Type `mbed detect` to ensure that the Mbed CLI detects the device.
6. Set the target:

   ```
   mbed config target CY8CKIT_062_WIFI_BT
   ```

7. Set the toolchain:

   ```
   mbed config toolchain GCC_ARM
   ```

8. Using a code editor, open the main.cpp file and change `BLINKING_RATE_MS` from 500 to 1000. This will allow you to see a change in behavior when you re-program the kit.
9. Build and program the kit (`-f` means to program the device flash memory when compilation finishes):

```
mbed compile -f
```

10. Once you are done, you can either leave the window open or you can type `exit` to leave the virtual environment and return to the normal terminal. To restart, just run `pipenv shell` again from the proper director.

Mbed Troubleshooting

Command Not Found

Running `mbed --version` returns error:

```
'mbed' is not recognized as an internal or external command, operable program or
batch file.
```

Debugging:

Use `echo %PATH%` to see the current path.

Use `where mbed` to see if the Windows command tool can find the mbed executable.

Solution:

If the PATH environment variable is incorrect, fix it, close all command windows, and try again.

For example, if you find mbed.exe at *C:\Python37\Scripts*, make sure both *C:\Python37* and C*:\Python37\Scripts* are set in the PATH environment variable.

Mbed Import Failed

Running `mbed import <project>` fails during clone.

Debugging:

1. Check you can go to https://github.com in your web browser.
2. Run `curl http://github.com` to check you can pull data from GitHub.
3. Run `git clone <project uri>` to see if you can clone outside of Mbed.
4. Run `mbed import -vv <project>` to get more details about the cloning process.

Solution 1:

GitHub is not accessible. Wait for GitHub to come back up or use a different internet connection.

Solution 2:

Proxy server is set incorrectly. If you're able to access GitHub via a web browser but not via `curl` this is likely the problem.

Set the proxy environment variables as described in the "Network Proxy" section earlier in this chapter. If you want to set the variables for a single terminal window instead of setting environment varialbes, you can use:

Windows Command Prompt:

```
set http_proxy=<proxy server>:<port>
set https_proxy=<proxy server>:<port>
```

Windows PowerShell:

```
#env:http_proxy="<proxy server>:<port>"
#env:https_proxy="<proxy server>:<port>"
```

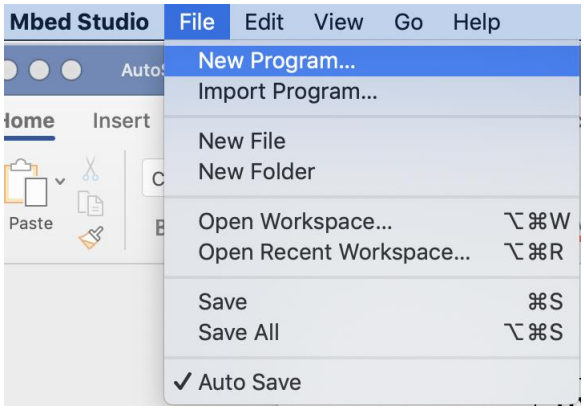
Solution 3:

Run `cmd` as administrator. Use this if `git clone <project url>` works, but `mbed import -vv <project>` shows a permission error.

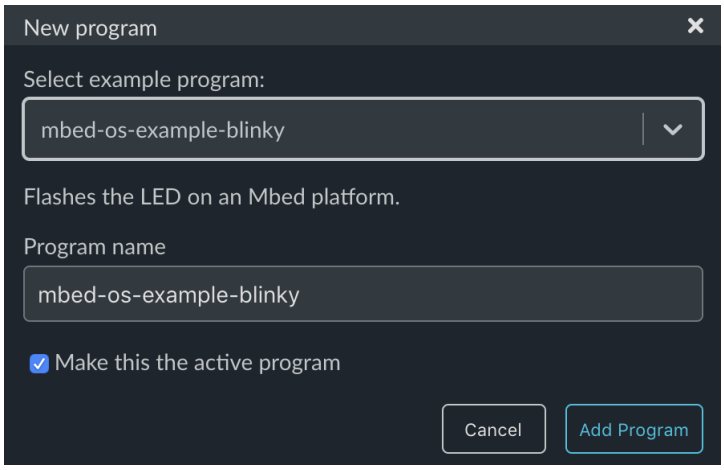
4.3.3 Make a blinky project using Mbed Studio

In this example project, we will use Mbed Studio to create, build and program. To do this run Mbed Studio. Then:

1. Use the **File > New Program** menu.

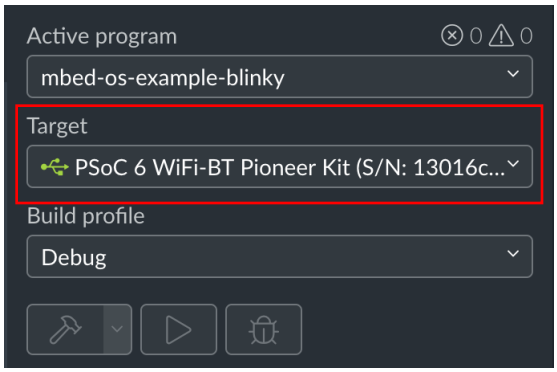


2. Select the mbed-os-example-blinky example program.

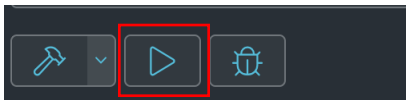


3. Select the correct target.

   Notice the green symbol means that the target was automatically detected by Mbed Studio if the board is connected to your computer and is in DAPLink mode.



4. Click Program (looks like a play) button (which will build and then program).



5. Once programming completes, the LED on the kit should be blinking at the faster rate again.

4.4 Mbed OS Libraries

Mbed OS programs can use a rich source of community, Arm, and Cypress libraries. These libraries reside on the internet in Git Repositories. To add them to your project you call the function `mbed add <URL>`, which will create a file called libraryname.lib. This file will have the URL for the library and a Git Tag. For instance, if you type:

```
mbed add http://github.com/cypresssemiconductorco/emwin.git
```

You will end up with a file in your project called emwin.lib. This will have the following content:

https://github.com/cypresssemiconductorco/emwin/#dd87764b00041bc3d901381b992f23ba18f4f1b2

The long string of numbers is the git commit hash (meaning which version of the library). You can also specify a specific tag by appending #tag. For example, "#mbed-os-5.15.0".

In addition to the .lib file, you will have a directory called "emwin". This will have all the code for the library. All this code will be compiled and linked into your project by the `mbed compile` command

Interestingly, mbed-os is just a library. So, you will have the file "mbed-os.lib" in your project with a link to the version of mbed-os that you are using. If you are missing any of the actual libraries, you can

always use the `mbed deploy` command, which will bring the libraries back into your project for each ".lib" file.

At any point, you can upgrade a library to a newer version by running the `mbed update` command. For instance, to move your mbed-os library to 5.15.0, from the mbed-os directory you would run:

```
mbed update mbed-os-5.15.0
```

The argument to the `mbed update` command is the commit hash or tag that you want to update to. You can see a full list of Mbed OS release and your currently selected release using the command `mbed releases`.

4.5 Mbed CLI Configuration System

When you compile your program, one of the first things that happens is that Mbed CLI will take the file "mbed_app.json" (which resides in your project directory) as well as all of the files "mbed_lib.json" (which reside in the libraries) and convert them into a C-Header file called "mbed_config.h".

The Mbed CLI Configuration System conversion system is well documented in the online docs here.

Basically, the system knows your target (because you set it in the config or on the command line), it then matches your target against a regular expression in the json files. If it matches, then it creates #defines in the mbed_config.h.

For example, this mbed_app.json:

```
{
    "target_overrides": {
        "*": {
            "platform.stdio-convert-newlines": true
        }
    }
}
```

Will result in the mbed_config.h having this:

```
#define MBED_CONF_PLATFORM_STDIO_CONVERT_NEWLINES       1
```

Another example from the emWIN-SSD1306 library:

```
{
    "name" : "SSD1306_OLED",
    "config": {
        "SDA":"P6_1",
        "SCL":"P6_0",
        "I2CADDRESS":"0x78",
        "I2CFREQ":"400000"
    }
}
```

Will result in:

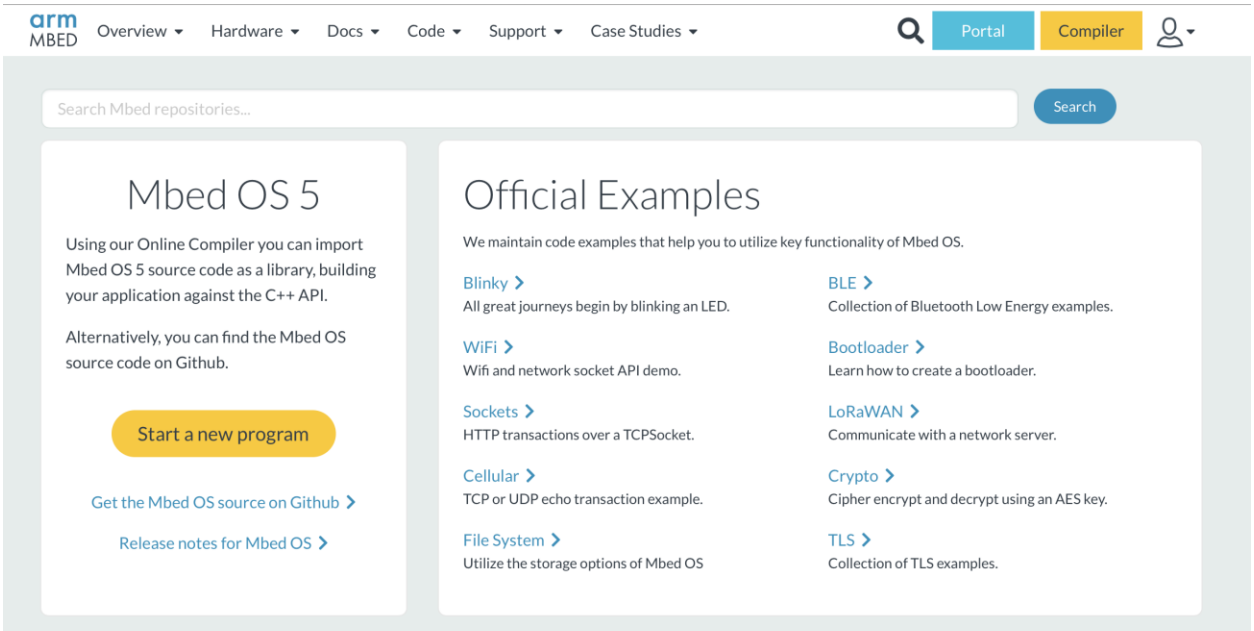
```
#define MBED_CONF_SSD1306_OLED_I2CADDRESS   0x78    // set by library:SSD1306_OLED
#define MBED_CONF_SSD1306_OLED_I2CFREQ      400000  // set by library:SSD1306_OLED
#define MBED_CONF_SSD1306_OLED_SCL          P6_0    // set by library:SSD1306_OLED
#define MBED_CONF_SSD1306_OLED_SDA          P6_1
```

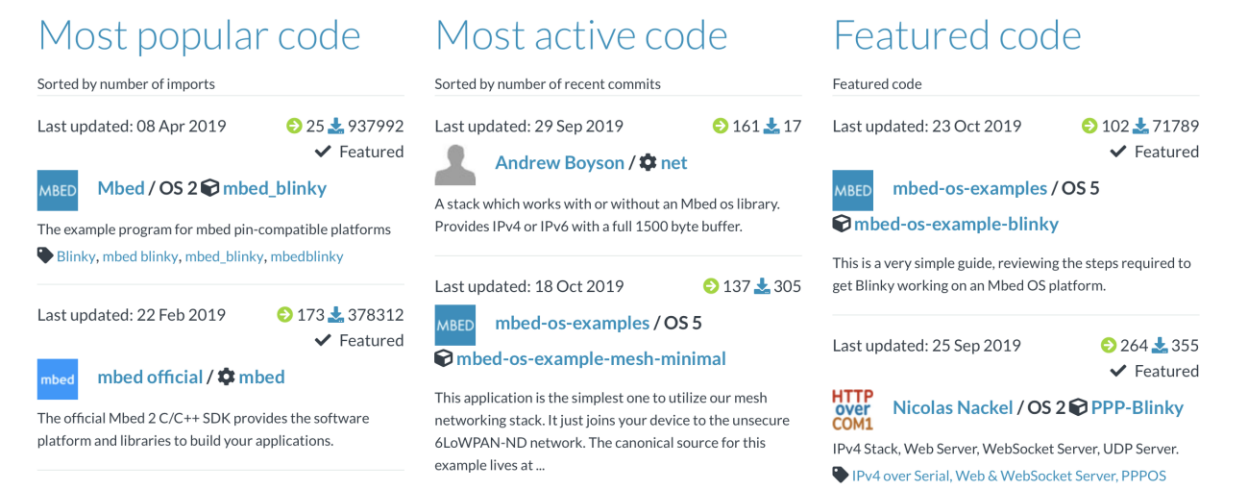
You can override these definitions in your project for specific targets. For example you could override the I2C frequency for the TFT display on the CY89CKIT_062S2_43012 kit like this in your mbed_app.json :

```
{
    "target_overrides": {
        "*": {
               "target.components_add": ["EMWIN_OSNTS"]
        },
        "CY8CKIT_062S2_43012" : {
               "SSD1306_OLED.I2CFREQ": "1000000"
        }
   }
}
```

4.6 Library & Example Code

There are many example projects by Arm, Cypress and even Alan Hawse available on the web. On the Mbed website you can search their libraries and example code here.





You can also search the GitHub repositories with "mbed-os" in the name. Generally, these projects are built on the Arm HAL and will run on the Cypress development kits.

- GitHub ARM - https://github.com/armmbed/
- GitHub Cypress – https://github.com/cypresssemiconductorco/
- GitHub IoT Expert – https://github.com/iotexpert/

Projects can be copied onto your computer with the CLI command `mbed import` and libraries can be added to your project with `mbed add`. For example, import projects with:

```
mbed import mbed-os-example-blinky
mbed import https://github.com/iotexpert/mbed-os-example-ntp
```

Or, for libraries you can, for example, add the emWin middleware by using:

```
mbed add https://github.com/cypresssemiconductorco/emwin.git
```

Note that for libraries on the Mbed GitHub site, the full URL is not required, but it is for projects or libraries from any other location.

4.7 Project Directory Organization

Inside of your project there are several interesting files and directories.

4.7.1 The ".lib" files

As discussed in Mbed OS Libraries, each of the lib files will be here.

4.7.2 mbed-os

This directory contains a complete copy of all the Mbed OS firmware, as well as all the board support packages (Targets).

4.7.3 mbed-os/targets

This directory contains all the Mbed OS board support packages for all vendors.

4.7.4 mbed-os/targets/targets.json

This file defines all the legal targets in Mbed OS and their characteristics. For example, the CY8CKIT_062_WiFi_BT kit is defined in this file like this:

```
"CY8CKIT_062_WIFI_BT": {
    "inherits": ["MCU_PSOC6_M4"],
    "features": ["BLE"],
    "supported_form_factors": ["ARDUINO"],
    "extra_labels_add": [
        "PSOC6_01",
        "CM0P_SLEEP",
        "WHD",
        "4343W",
        "CYW43XXX",
        "CORDIO"
    ],
    "macros_add": ["CY8C6247BZI_D54", "CYHAL_UDB_SDIO", "CYBSP_WIFI_CAPABLE"],
    "detect_code": ["1900"],
    "post_binary_hook": {
        "function": "PSOC6Code.complete"
    },
    "bootloader_supported": true,
    "sectors": [[268435456, 512]],
    "overrides": {
        "network-default-interface-type": "WIFI"
    },
    "program_cycle_s": 10
},
```

4.7.5 mbed-os/targets/TARGET_Cypress/TARGET_PSOC6

This directory contains all the board support packages for PSoC 6. The files in this directory contain the Mbed OS C++ HAL functions.

Inside of this directory there is one directory per Cypress target that contains the specific BSP information for each development kit.

4.7.6 mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/ TARGET_CY8CKIT_062_WIFI_BT/COMPONENT_BSP_DESIGN_MODUS

This directory contains the files used to run various configurators (e.g. design.modus for the device configurator, design.cycapsense for the CapSense configurator, design.cyqspi for the Quad SPI configurator, etc.), as well as the generated source. Take care when modifying these files so as not to collide with other things in the BSP. In particular there are resources in the device configuration such as clocks that are required for Mbed to work properly.

4.7.7 mbed_app.json

This file is used by the configuration system to create #defines to configure your project.

4.7.8 .mbedignore

Files with the name ".mbedignore" contain a list of files (or directories) which you do NOT want to include in your build. For example, if you have a library with a "sample" directory you could ignore all the code in that directory by:

Creating .mbedignore in **that** directory with a wildcard "*", or

Creating .mbedignore in **the directory above** with the text "sample".

4.7.9 BUILD

The directory "BUILD" contains all the compilation artifacts including the .o, .a, .hex, .elf, .map etc. Hierarchy in the build subdirectory is BUILD/target/toolchain/ so you can build for multiple targets and/or toolchains without over-writing other build results. You can effectively do a clean by running:

```
rm -rf BUILD
```

4.8 Mbed OS Targets

The target directories (e.g. TARGET_CY8CKIT_062_WIFI_BT) are used for selecting sources specific to an MCU or MCU family. The set of directories included in a given build is all the names of the targets in the inheritance hierarchy described by the targets.json file.

There are a large number of configuration options for Mbed OS targets. In addition to the vendor defined targets, such as the CY8CKIT-062-WIFI-BT provided by Cypress, users can create their own custom targets. The creation of custom targets is a critical step when moving from evaluation of Mbed OS using a kit into creating a production product.

User documentation on creating custom targets is available at https://os.mbed.com/docs/mbed-os/latest/reference/adding-and-configuring-targets.html.

4.9 Cypress Configurators

You can modify the Board Support Package by using the Cypress device configurator. To do this run the configurator from:

- macOS: /Application/ModusToolbox/tools_2.1/device-configurator
- Linux: ~/ModusToolbox/tools_2.1/device-configurator
- PC: C:\Users\<your user>\ModusToolbox\tools_2.1\device-configurator

Then use **File > Open** to open the design.modus for your board (from above). Take care when modifying these files. If you do change the design.modus or generate new code from it, you will have modified your mbed-os library. This means that the `mbed update` and associated commands will **no longer work**.

If you must make changes to the BSP, make your own target. An exercise later in this training will show you how to do this.

When opening the file, you may be asked to provide the path to the device support library. If so, that file is in the project at: mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/psoc6pdl/devicesupport.xml.

Other configurators may also be used depending on the capabilities of the board. For example:

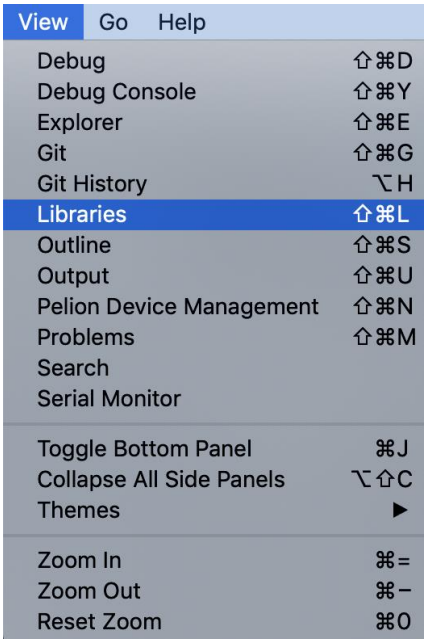
- CapSense: …/ModusToolbox/tools_2.1/capsense-configurator
- Bluetooth: …/ModusToobox/tools_2.1/bt-configurator

4.10 Low Power

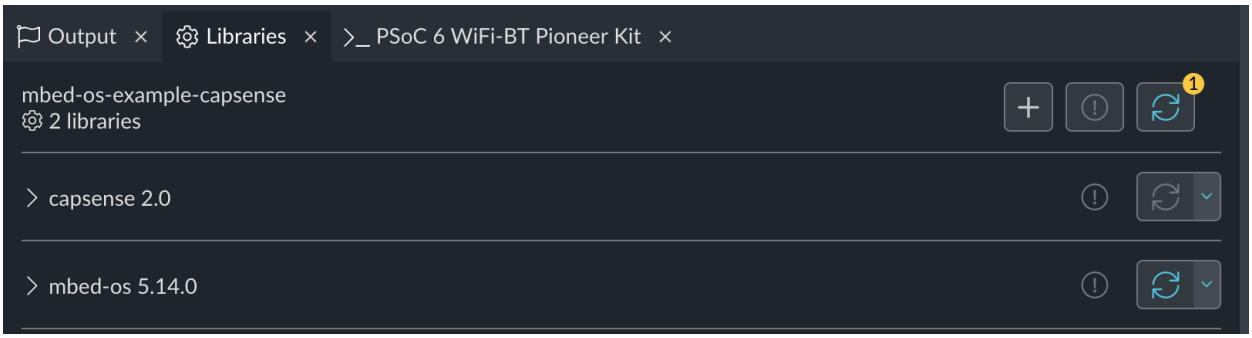
See Chapter 7.

4.11 Mbed Studio Library Manager

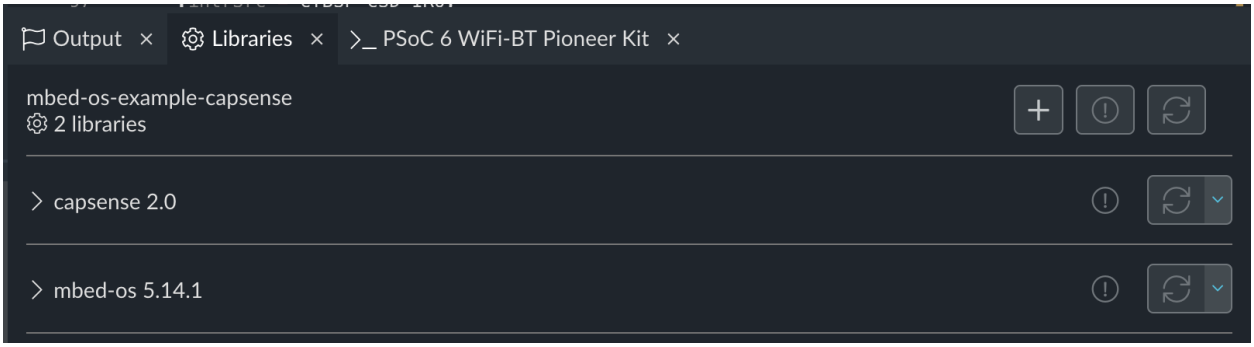
Mbed Studio has a built-in tool to let you manage libraries including adding, updating and removing them from your project. To run the tool, choose **View > Libraries**.



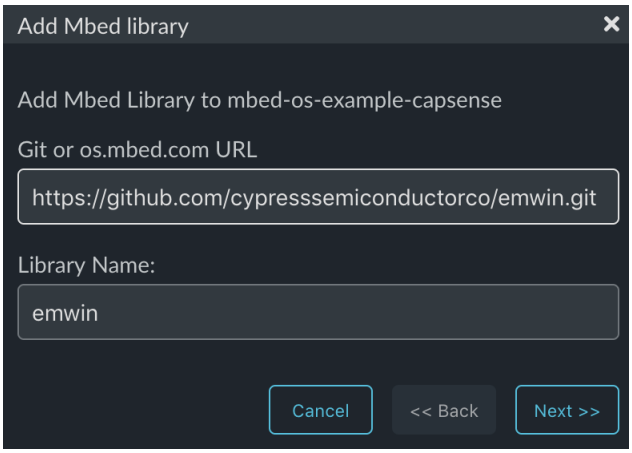
The first thing that you might want to do on this screen is update the mbed-os library to the current version. Notice that there is a "1" which indicates that there is one library which is not the latest. It shows that library by having the active circle arrow icon. If you press that icon it will update the library to the latest version.



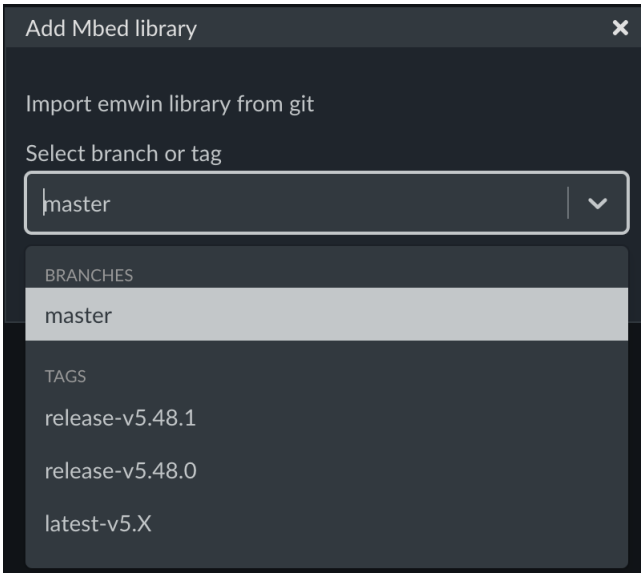
You will have a tab in your window that shows all the libraries that you are currently using. If you press the "+" at the top, you will be given screens to add libraries (notice I have updated to mbed-os-5.14.1).



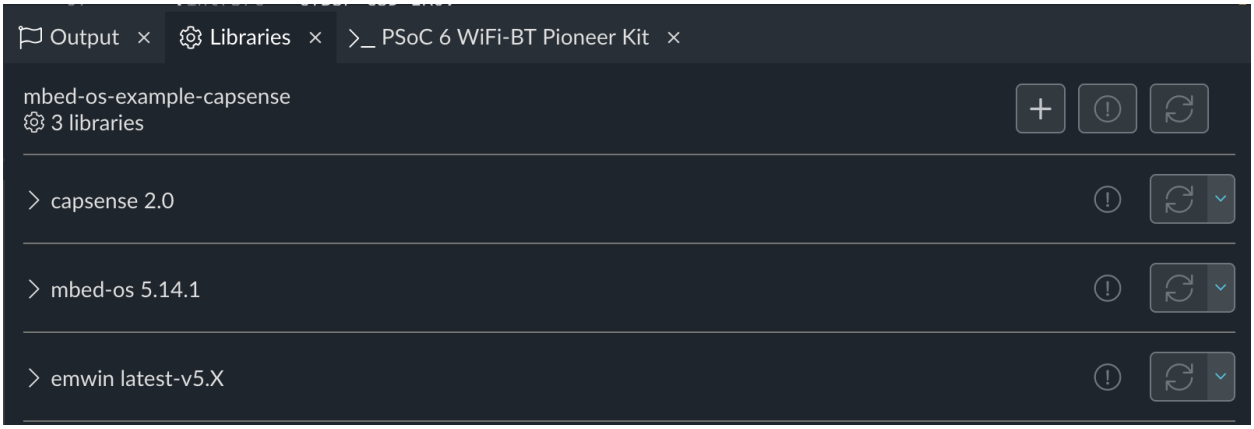
When you add the library, you need to give it the URL.



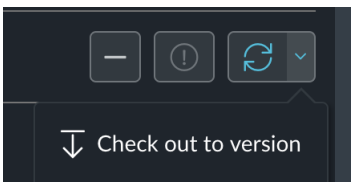
Then you need to pick out which version of the library you want.



Finally, you will see the libraries in your project:



If you click the little down arrow to the right of any of the libraries, you will be given the choice to move to a specific version of the library.



4.12 Exercises (Part 2)

Remember that the Mbed OS baud rate is 9600.

4.12.1 Go find answers to these questions

- [ ] 1. What are the supported options for development environments with Mbed OS?

- [ ] 2. What is the CLI command to import an Mbed OS example?

- [ ] 3. What is command to import a Cypress example?

- [ ] 4. What Cypress targets are supported in Mbed OS?

- [ ] 5. What are the dependencies to use the Cypress Enterprise Security?

- [ ] 6. What is the example that demonstrates how to connect a kit to Wi-Fi by provisioning via BLE with Cypress targets?

4.12.2 TFT

For this project, you will use the Cypress emWin middleware library and the library for the CY8CKIT-028-TFT using the Mbed CLI.

- [ ] 1. Create a new Mbed OS project and add the libraries:

```
mbed new tftproject
cd tftproject
mbed add https://github.com/cypresssemiconductorco/emwin.git
mbed add https://github.com/iotexpert/mbed-os-emwin-st7789v.git
```

2. Create a main.cpp file with this code (use notepad ++):

```
#include "mbed.h"
#include "GUI.h"
int main()
{
    GUI_Init();
    GUI_SetColor(GUI_WHITE);
    GUI_SetBkColor(GUI_BLACK);
    GUI_SetFont(GUI_FONT_32B_1);
    GUI_SetTextAlign(GUI_TA_CENTER);
    GUI_DispStringAt("Hello World", GUI_GetScreenSizeX()/2,
                     GUI_GetScreenSizeY()/2 - GUI_GetFontSizeY()/2);
}
```

3. Create an mbed_app.json:

```
{   "target_overrides": {
        "*": {
            "target.components_add": ["EMWIN_NOSNTS"]
        }
    }
}
```

4. Run the deploy command to make sure all your libraries are using the latest version:

```
mbed deploy
```

5. Set the target:

```
mbed config target CY8CKIT_062_WIFI_BT
```

6. Set the toolchain:

```
mbed config toolchain GCC_ARM
```

7. Compile and program:

```
mbed compile -f
```

When done, your project should do this:

8. Look at the mbed-os-emwin-st7789v.git GitHub repository and read the instructions in the README.md file.

4.12.3 mbed-os-example-wifi

For this project, you will use the mbed-os-example-wifi project.

1. The steps for this project are:

```
mbed import mbed-os-example-wifi
cd mbed-os-example-wifi
```

2. Edit the mbed_app.json file to set the correct Wi-Fi access point SSID and password.

```
"wifi-ssid": {
    "help": "WiFi SSID",
    "value": "\"Enter SSID Here\""
},
"wifi-password": {
    "help": "WiFi Password",
    "value": "\"Enter password here\""
```

3. Set the target:

```
mbed config target CY8CKIT_062_WIFI_BT
```

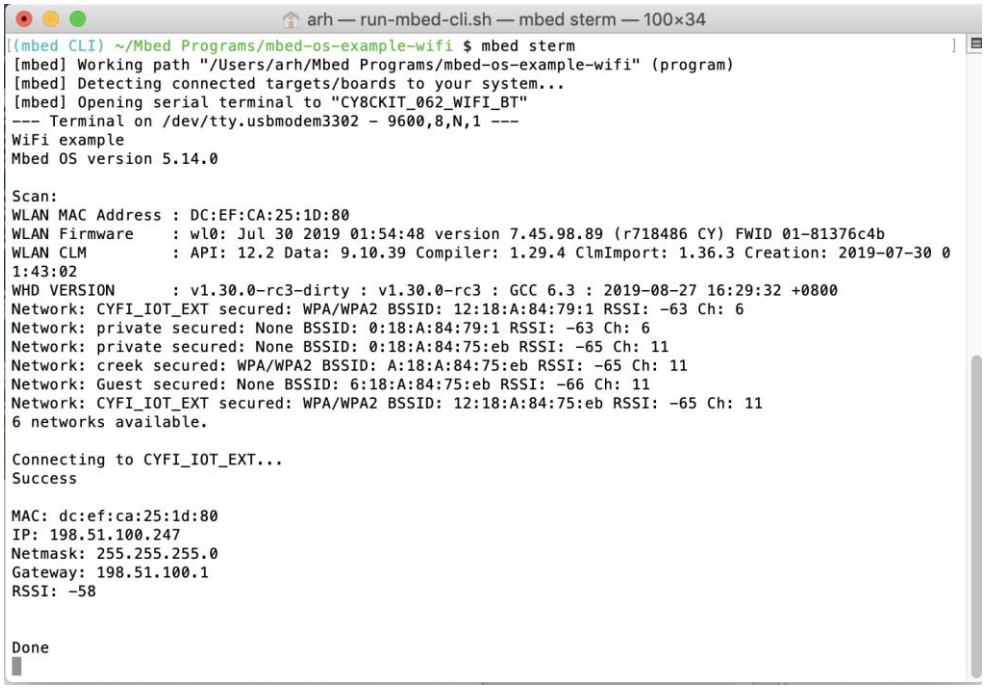
4. Set the toolchain:

```
mbed config toolchain GCC_ARM
```

5. Compile and program:

```
mbed compile -f
```

When the project finishes programming, your terminal using a baud rate of 9600 should look like this:



4.12.4 NTP Server

The Network Time Protocol is an old internet standard for distributing time. This program will go to the internet and get the time. Then it will save the time into the PSoC RTC. Finally, once per second, it will update the time on the screen using the emWin library and the IoT Expert TFT configuration files. The project will use one of the community libraries for getting the network time.

1. Start by importing the example code:

```
mbed import https://github.com/iotexpert/mbed-os-example-ntp
cd mbed-os-example-ntp
```

2. Modify the mbed_app.json to set the correct SSID and password.
3. Set the target:

```
mbed config target CY8CKIT_062_WIFI_BT
```

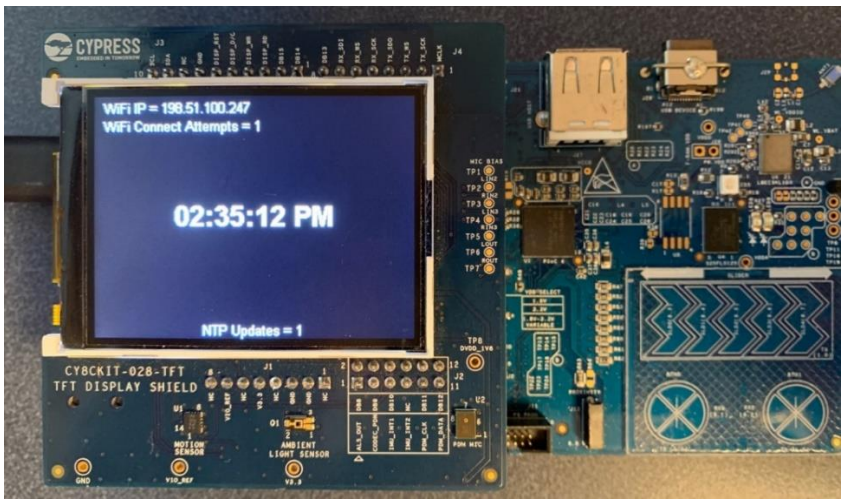
4. Set the toolchain:

```
mbed config toolchain GCC_ARM
```

5. Compile and program:

```
mbed compile -f
```

When programming completes, you should see something like this:

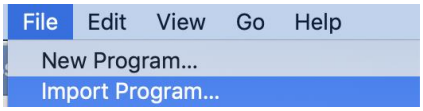


6. This project is hard coded to Eastern Daylight Savings time. Can you fix the project to be in a different time zone?
7. Where did I get the NTP library?

4.12.5 mbed-os-example-CapSense

This project will demonstrate Cypress CapSense with our libraries integrated into Mbed OS. For this example, we will use Mbed Studio.

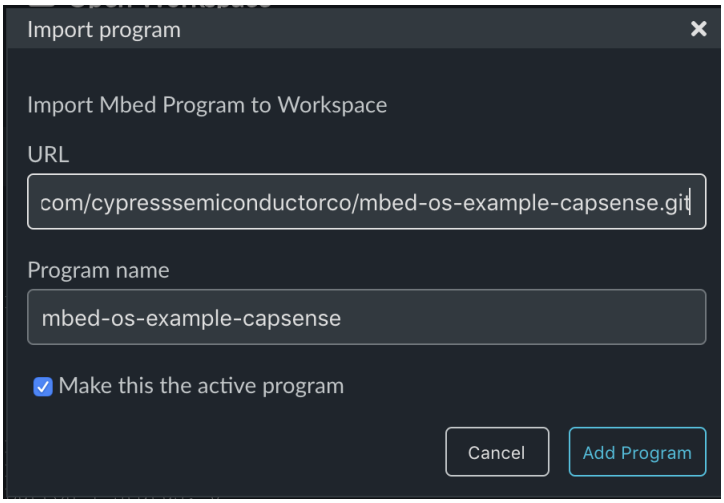
1. Start with **File > Import Program…**



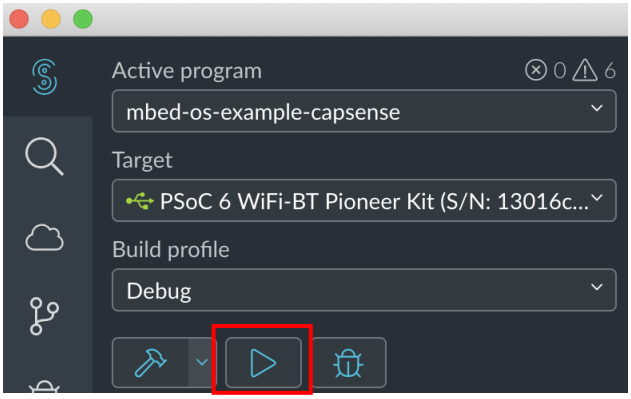
2. Type the following URL (or copy it from GitHub):

   https://github.com/cypresssemiconductorco/mbed-os-example-capsense.git

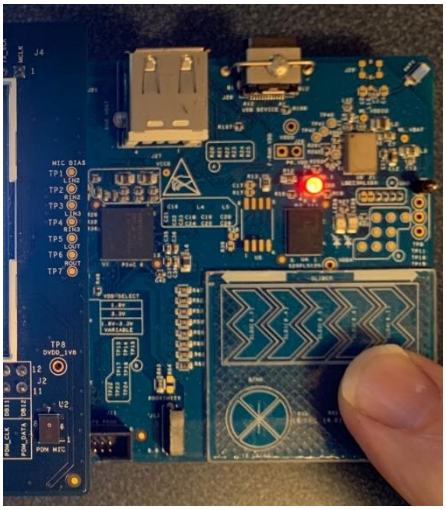
3. Then click **Add Program**.



4. Click the **Play** button to build and program the development kit.



5. When you touch the CapSense buttons, you should see the red LED turn on.



6. On the serial viewer, you should see the button presses and the slider position.

7. Modify the program to turn on the green LED instead of the red one (one line of code).
8. Modify the program to have the slider position set the intensity of an LED using a PWM.
9. Modify the program to display the slider position on the TFT screen.
10. Run this program on the CY8CPROTO_062_4343W.

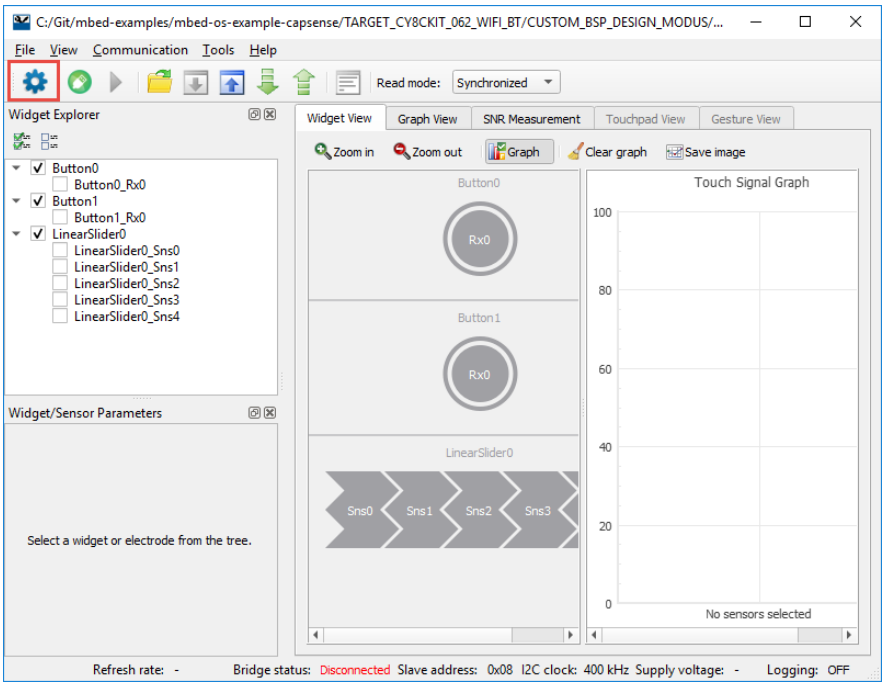
4.12.6 Run the CapSense Tuner

Use the previous example to run the CapSense Tuner.

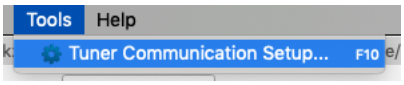
1. Switch the kit from DAPLink mode to BULK or HID mode either by pressing the mode button or by using the firmware loader tool. See the Firmware-loader section in Chapter 1 for details.
2. Launch the CapSense Tuner (<user_home>/ModusToolbox/tools_2.1/capsense-configurator/capsense-tuner.exe).
3. In the Tuner, select **File > Open**, and select the design.cycapsense file at:

   <project_root>/COMPONENT_CUSTOM_DESIGN_MODUS/TARGET_CY8CKIT_062_WIFI_BT

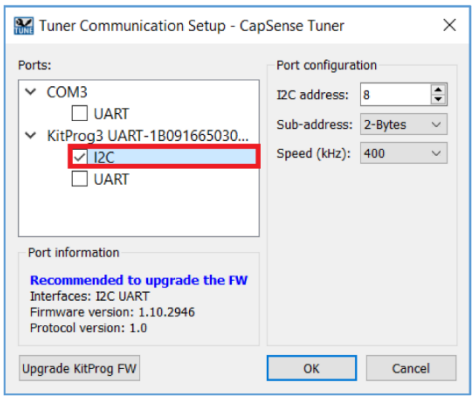
   This loads the button and slider example code configuration information.



4. Click **Tools > Tuner Communication Setup** or press [F10] or click the gear button.



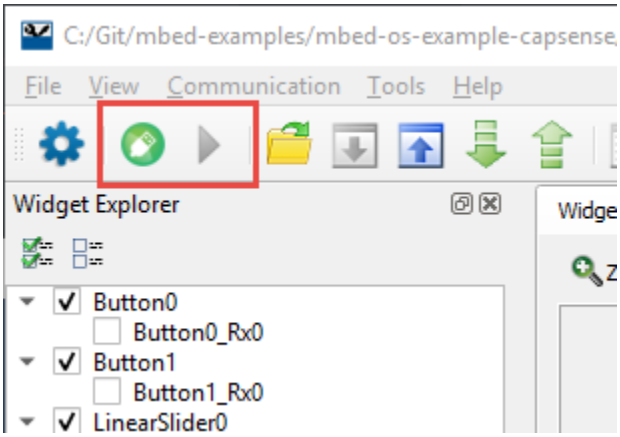
5. In the dialog, select I2C under KitProg and configure as follows:



6. Check the parameters and close the dialog.

   - I2C Address: 8
   - Sub-address: 2-Bytes
   - Speed (kHz): 400

7. On the Tuner, click the **Connect** button or press [F4].



8. Click the **Start** button or press [F5].

9. Under **Widget View** tab on the right side, you can see the corresponding widgets turning blue when you touch the button or slider.

10. You can also view the sensor data in the **Graph View** tab. For example, to view sensor data for Button 0, you need to select Button0_Rx0 under Button0.

4.12.7 AWS IoT Subscriber

This project connects to a Wi-Fi network as a STA(client), discovers Greengrass core devices, establishes a secure TCP connection over MQTT to the AWS Greengrass endpoint, subscribes to a topic and waits in a loop to receive messages that are published on the topic.

1. Start by cloning the examples:

   ```
   git clone https://github.com/cypresssemiconductorco/mbed-os-example-aws-iot-client.git
   ```

2. Change directory to the Subscriber example directory:

   ```
   cd mbed-os-example-aws-iot-client
   cd subscriber
   ```

3. Prepare the cloned working directory for mbed:

   ```
   mbed config root .
   ```

4. Open the file aws_config.h and configure the AWS parameters such as Thing name, certificates, private key etc. per the user's AWS account. Refer to the AWS IoT Core chapter for instructions.

5. Pull the necessary libraries and dependencies. This will pull mbed-os, AWS_Iot_Client library and its internal 3rd party dependencies:

   ```
   mbed deploy
   ```

6. Build the subscriber app:

   ```
   mbed compile -t GCC_ARM -m CY8CPROTO_062_4343W -f
   ```

7. Connect to the kit via serial port.

   You should observe console logs that indicate connection to network, AWS endpoint, subscription to the topic and waiting for messages.

8. Using the AWS console, publish messages to the subscribed topic.

   You would observe messages are being received by the subscriber and printed on the kit's console

4.12.8 AWS IoT Publisher

This project connects to a Wi-Fi network as a STA(client), discovers Greengrass core devices, establishes a secure TCP connection over MQTT to the AWS Greengrass endpoint and publishes messages periodically to a topic.

1. Use the same cloned project from the Subscriber example.

2. Change directory to the Publisher example directory:

   ```
   cd mbed-os-example-aws- iot-client
   cd publisher
   ```

3. Prepare the cloned working directory for mbed:

   ```
   mbed config root .
   ```

4. Pull the necessary libraries and dependencies. This will pull mbed-os, AWS_Iot_Client library and its internal 3rd party dependencies:

   ```
   mbed deploy
   ```

5. Configure the SSID and passphrase of the desired network in the accompanying mbed_app.json.

6. Open the file aws_config.h and configure the AWS parameters such as Thing name, certificates, private key etc. per the user's AWS account. Refer to the AWS IoT Core Chapter for instructions.

7. Build the publisher app:

   ```
   mbed compile -t GCC_ARM -m CY8CPROTO_062_4343W
   ```

8. Connect to the kit via serial port.

   You should observe console logs that indicate connection to network, AWS endpoint, subscription to the topic and waiting for messages.

4.12.9 Adding CapSense

This project shows how to enable CapSense on an existing project. You will use the Mbed CLI for this project.

1. Create an empty PSoC 6 project:

   ```
   mbed new --program empty_psoc6_project
   cd empty_psoc6_project
   ```

2. Generate a CapSense configuration from the configuration in the target into your project.

   The design.modus for each Mbed OS target includes all the pins and associated configuration for enabling CapSense on that board, but it is missing the firmware to initialize the CapSense subsystem. This is because the CapSense middleware is a closed source binary that can't be included in Mbed OS and therefore must be acquired after project creation (see step #3).

   To generate the code for your board's CapSense configuration, you need to run the CapSense configurator. There is a command line version of the tool that can put the output code anywhere you want. In this example we add it to the root of the project. This avoids leaving the mbed-os directory in a 'dirty' state that would cause failures when updating the library later.

   ```
   <user_home>/ModusToolbox/tools_2.1/capsense-configurator/capsense-
   configurator-cli.exe -c ./mbed-
   os/targets/TARGET_Cypress/TARGET_PSOC6/TARGET_CY8CKIT_062_WIFI_BT/COMPONENT_
   BSP_DESIGN_MODUS/design.cycapsense -o
   C:/Users/<initials>/mbed/empty_psoc6_project/ -g
   ```

Note: you can run capsense-configurarot-cli.exe -h to see a list of available options.

3. Add the Cypress CapSense middleware library to your project and select the latest v2.X tag:

```
mbed add https://github.com/cypresssemiconductorco/capsense
cd capsense
mbed update latest-v2.X
cd ../
```

4. Copy the main.cpp file from the mbed-os-example-CapSense into your application.

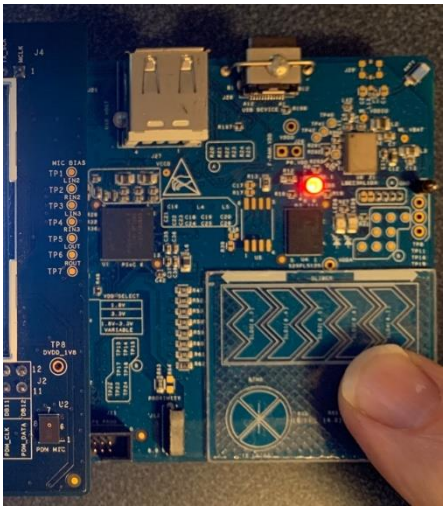
5. Set your target and toolchain:

```
mbed toolchain GCC_ARM
mbed target CY8CKIT_062_WIFI_BT
```

6. Build and program the application:

```
mbed compile -f
```

When you touch the CapSense buttons, you should see the red LED turn on.



4.12.10 SD Card (FAT Filesystem)

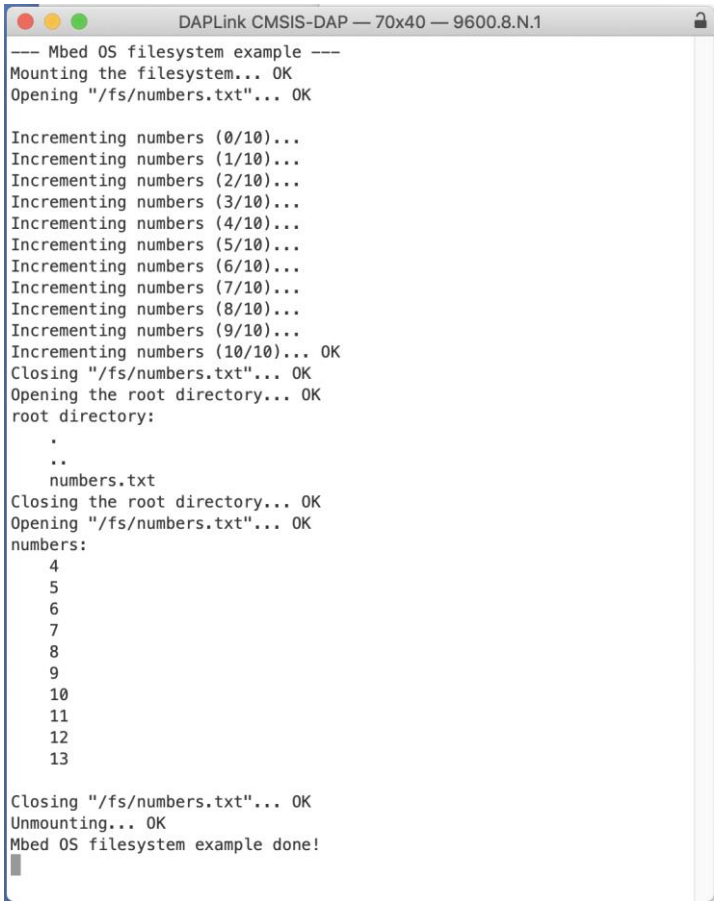
Not yet supported. Will be available in a future version of Mbed OS version.

4.12.11 SPI Flash (Little Filesystem)

For this example, we will use the following example code:

https://github.com/iotexpert/mbed-os-example-littlefilesystem

1. Start by either running `mbed import` or by using import in Mbed Studio.
2. Build and program with either `mbed compile -f` (don't forget to set your target and toolchain) or with the Mbed Studio **Play** button.



3. How does it know which device to write the filesystem onto?

4. How does it know what pins are attached to the PSoC 6?

5. Modify the program to write something other than numbers onto the device.

6. Modify the program to print the free storage available.
7. Modify the program to list the files in the “Little Filesystem”

4.12.12 SPI Flash XIP

This example demonstrates the Execute-In-Place (XIP) feature of PSoC 6 MCU using the SMIF block. In this example, a function is placed and executed from external QSPI memory. The function increments a set of integers and then prints them to a UART terminal.

1. Import the example code:

```
mbed import https://github.com/cypresssemiconductorco/mbed-os-example-xip
cd mbed-os-example-xip
```

2. Make sure the kit is in DAPLink mode.
3. Identify your COM port, and open a serial terminal.
4. Compile and program:

```
mbed compile -m CY8CPROTO_062_4343W -t GCC_ARM -f
```

5. Observe the output in the serial terminal.
6. Find and open the map file (in the BUILD directory). Where is function_external_memory located in the XIP memory space?

4.12.13 Bluetooth

Mbed OS Bluetooth examples use the Cordio BLE stack. The Mbed GitHub site contains several examples at: https://github.com/ARMmbed/mbed-os-example-ble.

1. Start by cloning the examples:

```
git clone https://github.com/ARMmbed/mbed-os-example-ble.git
```

2. Change directory to the example directory:

```
cd mbed-os-example-ble
cd BLE_LED
```

3. Update the source tree:

```
mbed deploy
```

4. Set the target:

```
mbed config target CY8CKIT_062_WIFI_BT
```

5. Set the toolchain:

```
mbed config toolchain GCC_ARM
```

6. Compile and program:

   ```
   mbed compile -f
   ```

7. Download and install scanner software on your smart phone.
8. Open the scanner software and start a scan.

   Refer to the project README.md file for more information.

9. What are the UUID and Characteristic of the LED Service?

10. Change the color of the LED.

4.12.14 Custom Targets

Creation of custom targets to represent a customer's production board is a critical activity. Arm is developing extended documentation showing how this is done. In this exercise you will use the documentation described in the Mbed OS Targets, and the draft document from Arm to create a custom version of the CY8CKIT-062-WIFI-BT.

1. Open a web browser to view the draft documentation from Arm:

   https://github.com/ARMmbed/mbed-os-5-docs/blob/dabd87922c6bf7957c527fbb77f3aeeaafed0884/docs/porting/custom-target-porting.md

2. Import the mbed-os-example-blinky example code.

   ```
   mbed import mbed-os-example-blinky
   ```

3. Change to the example directory:

   ```
   cd mbed-os-example-blinky
   ```

4. Create a file named custom_targets.json in your application.
5. Open the file targets.json from <root>/mbed-os-example-blinky/mbed-os/targets.
6. In targets.json find the entry for CY8CKIT_062_WIFI_BT
7. Copy the whole entry for the CY8CKIT_062_WIFI_BT and paste entry into your new custom_targets.json file from step 4.
8. Add brackets before and after the pasted entry.

   See the link in step #1 for the Customizing section of the custom-target-porting.md file to see how a correctly formatted entry should look.

9. Rename the target in custom_targets.json from CY8CKIT_062_WIFI_BT to MY_BOARD.

10. Copy the entire TARGET_CY8CKIT_062_WIFI_BT directory from mbed-os into the root of your blinky application.
11. Change the name of the copied target directory in your application to TARGET_MY_BOARD.
12. Copy the wifi_nvram_image.h:
    - **from**: <root>/mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/TARGET_WHD/resources/nvram/TARGET_CY8CKIT_062_WIFI_BT
    - **to**: TARGET_MY_BOARD
13. In the Device Configurator, click **File > Open** and navigate to the design.modus file in TARGET_MY_BOARD/COMPONENT_BSP_DESIGN_MODUS.
14. If you receive a message like 'Error loading file', click **OK** on the message and in the 'Select Device Support Library location' dialog, navigate to the devicesupport.xml file found here:

    <root>/mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/psoc6pdl
15. Disable the CSD block.
16. Change your board's Cortex-M4 frequency from 100 MHz to 50 MHz.
17. Save the file and close the Device Configurator.
18. Change the color of the LED that will blink.

    Change the pin assignments for MY_BOARD by opening <root>/mbed-os-example-blinky/TARGET_MY_BOARD/PinNames.h and editing LED1 and LED2 as follows:

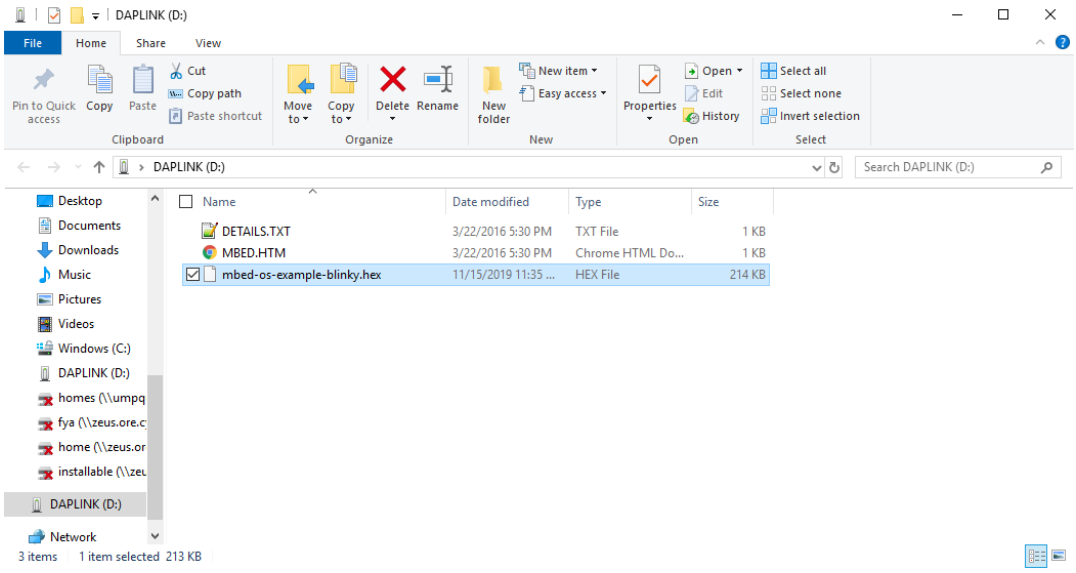
    ```
    #define LED1 P11_1
    #define LED2 P0_3
    ```
19. Assign your toolchain and target for the application:

    ```
    mbed target MY_BOARD
    mbed toolchain GCC_ARM
    ```
20. Build the project:

    ```
    mbed compile
    ```

21. We have created a custom target, so using `mbed compile -f` won't program the device.

    To program the device, use file explorer to find the mbed-os-example-blinky.hex and drag it into the DAPLINK drive:



22. How would you change the PSoC 6 MPN for your board?

23. What command can you use to make programming via `mbed compile -f` work today?