ModusToolbox™ IDE

User Guide

Document Number: 002-24375 Rev *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
http://www.cypress.com

Contents

1. Introduction

Overview

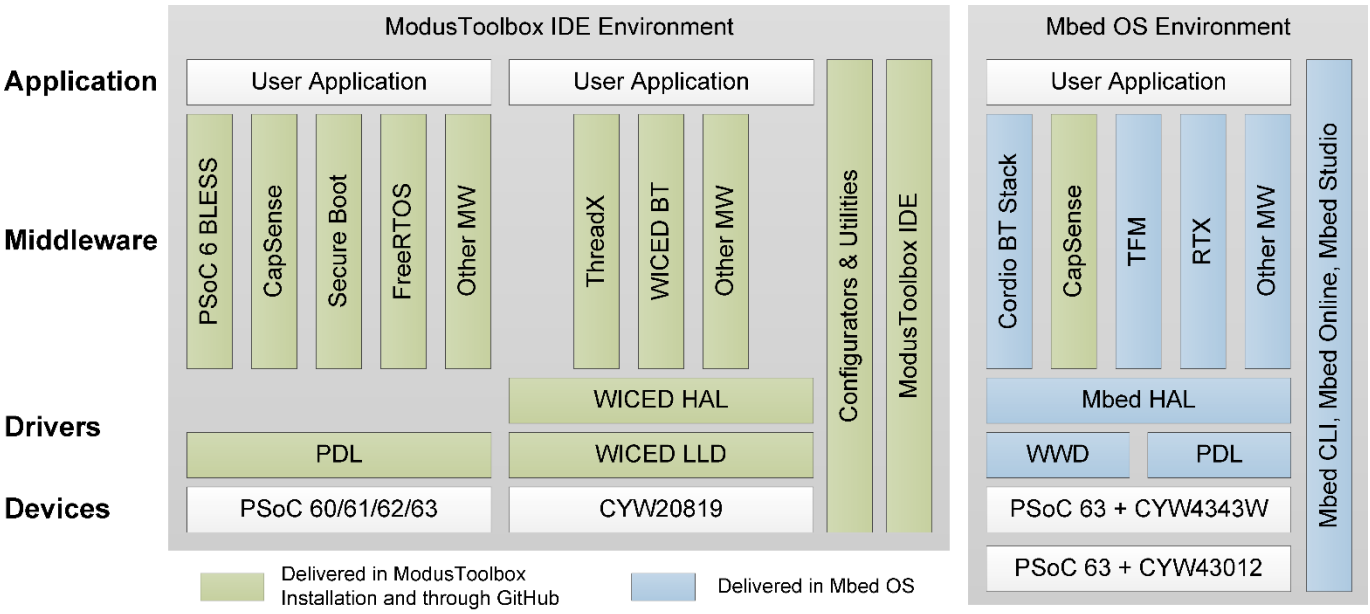
ModusToolbox™ software is a set of tools that enable you to integrate Cypress devices into your existing development methodology. One of the tools is a multi-platform, Eclipse-based Integrated Development Environment (IDE) called the ModusToolbox IDE that supports application configuration and development.

In the IDE, the top-level entity that you ultimately program to a device is called an application. The application usually consists of one Eclipse project, but it may include more than one. The IDE handles all of the dependencies between projects automatically. It also provides hooks for launching various tools provided by the software development kits (SDKs).

The SDKs provide the central core of the ModusToolbox software. They contain Configurators, drivers, libraries, middleware, as well as various utilities, Makefiles, and scripts. You may use any or all of these tools in any environment you prefer.

Another aspect of ModusToolbox includes Cypress kits that are enabled for the Mbed OS ecosystem. For more information, refer to the Cypress webpage on the Mbed OS website: https://os.mbed.com/teams/Cypress/.

The following shows a high-level view of the tools included in the ModusToolbox software.



This document provides information about creating applications as well as building, programming, and debugging them. This guide primarily covers the ModusToolbox IDE aspects of these concepts. It also covers various aspects of the SDKs.

References

The following documents should be referenced as needed for more information about a particular topic, For more information, see Documentation.

- ModusToolbox Installation Guide
- Configurator Documentation (open from a particular Configurator)
- Eclipse Documentation (available from Eclipse)
- Eclipse Survival Guide

Document Conventions

The following table lists the conventions used throughout this guide:

Convention	Usage
Courier New	Displays file locations and source code: `C:\ …cd\icc\, user entered text`
Italics	Displays file names and reference documentation: *sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
File > New Application	Represents menu paths: **File > New Application > Clone**
Bold	Displays commands, menu paths and selections, and icon names in procedures: Click the **Debugger** icon, and then click **Next**.
Text in gray boxes	Displays cautions or functionality unique to ModusToolbox software.

Revision History

Document Title: ModusToolbox™ IDE User Guide **Document Number: 002-24375**		
Revision	**Date**	**Description of Change**
**	11/21/18	New document.
*A	2/22/19	Updates for ModusToolbox version 1.1.

2. Getting Started

This section provides a basic walkthrough for how to create a couple starter applications using the IDE, selecting either a kit or custom hardware. It also covers how to build and program them using the IDE and basic launch configurations supplied for the applications.

- Launch ModusToolbox IDE
- Create Starter Applications
- Build Starter Application
- Program Starter Application
- Download/Import Code Examples

Launch ModusToolbox IDE

The ModusToolbox IDE is installed in the following directory in Windows, by default:

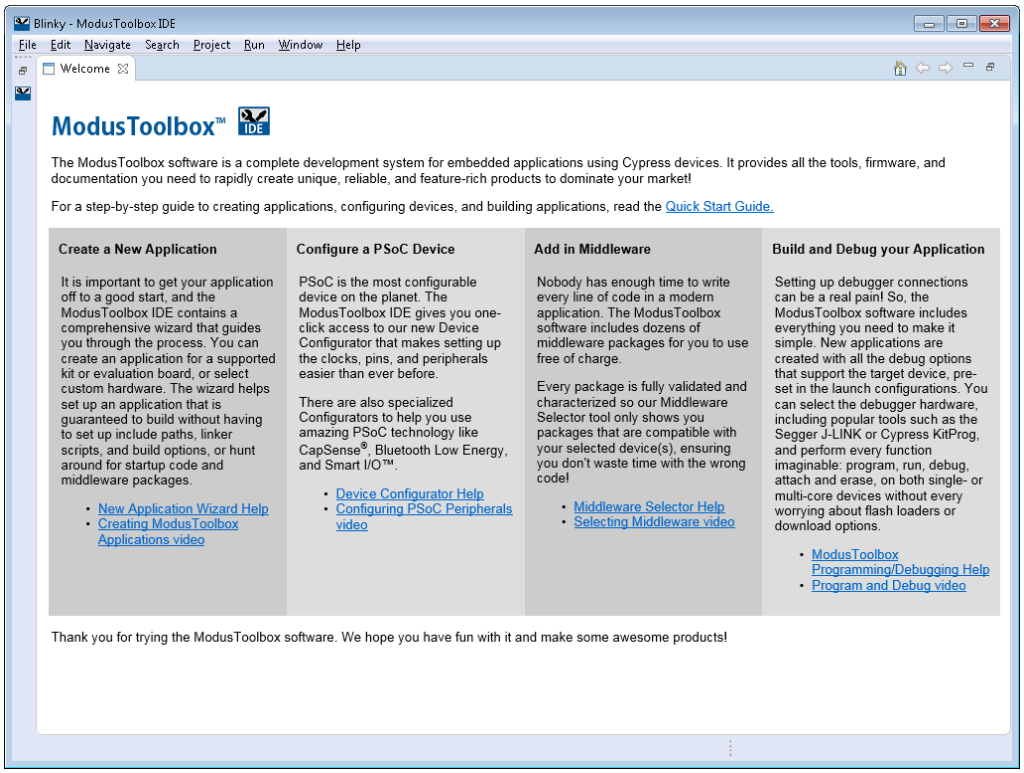
<user_home>\ModusToolbox_1.1\eclipse

For other operating systems, the installation directory will vary, based on where the software was installed.

Run the “ModusToolbox” executable file to launch the IDE as applicable for your operating system. Refer to the *Modus Toolbox Installation Guide* for more information.

When launching the ModusToolbox IDE, it provides an option to select the workspace location on your machine. This location is used by the IDE for creating and storing the files as part of application creation for a particular platform. The default workspace location is a folder called "mtw" in the user's home directory. The user is free to add additional folders under the mtw folder or to choose any other location for each workspace.

When the IDE opens the first time (or for any new workspace), it shows the Welcome Page. See Welcome Page for more information.



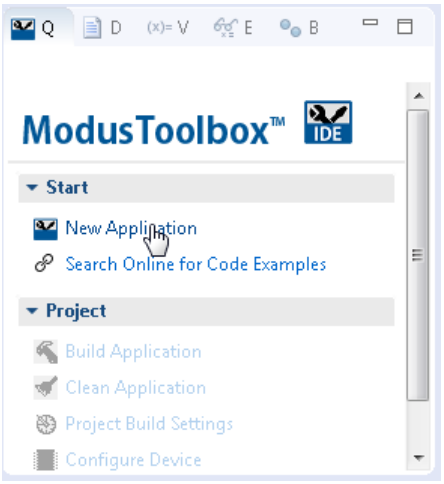
Click the **X** on the Welcome Page tab to close it, or click the **Restore** button to move it to the side.

Create Starter Applications

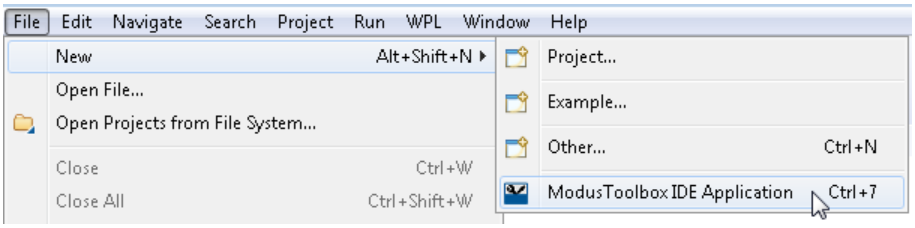
The ModusToolbox IDE provides several starter applications for use with different development kits. Refer to the *ModusToolbox IDE Release Notes* for the supported platforms. This section provides a walkthrough for creating an application by selecting a kit, and another application by selecting a chip.

Open New Application Dialog

Click the **New Application** link in the ModusToolbox IDE Quick Panel.



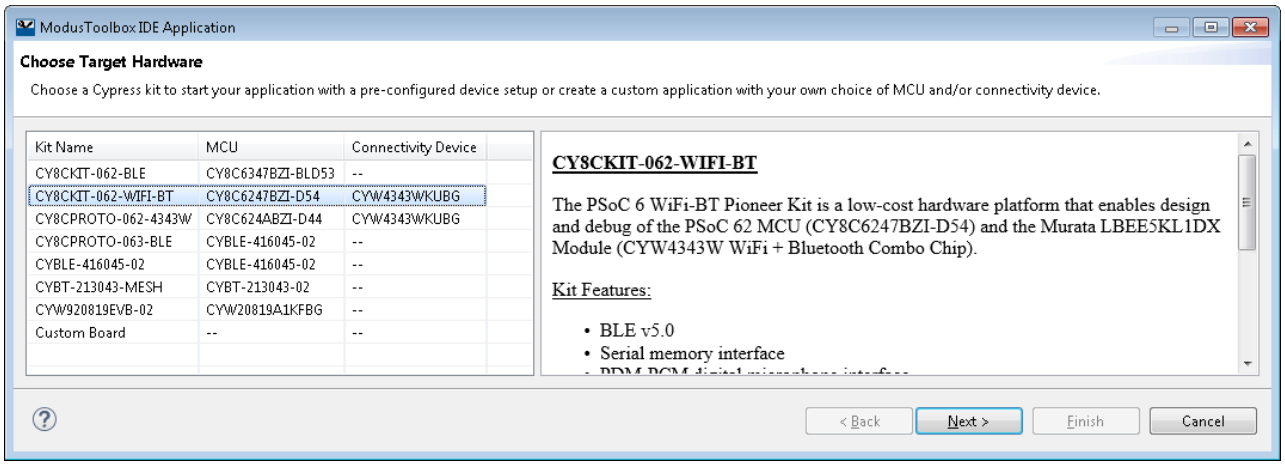
You can also select **File > New > ModusToolbox IDE Application**.



Select Kit

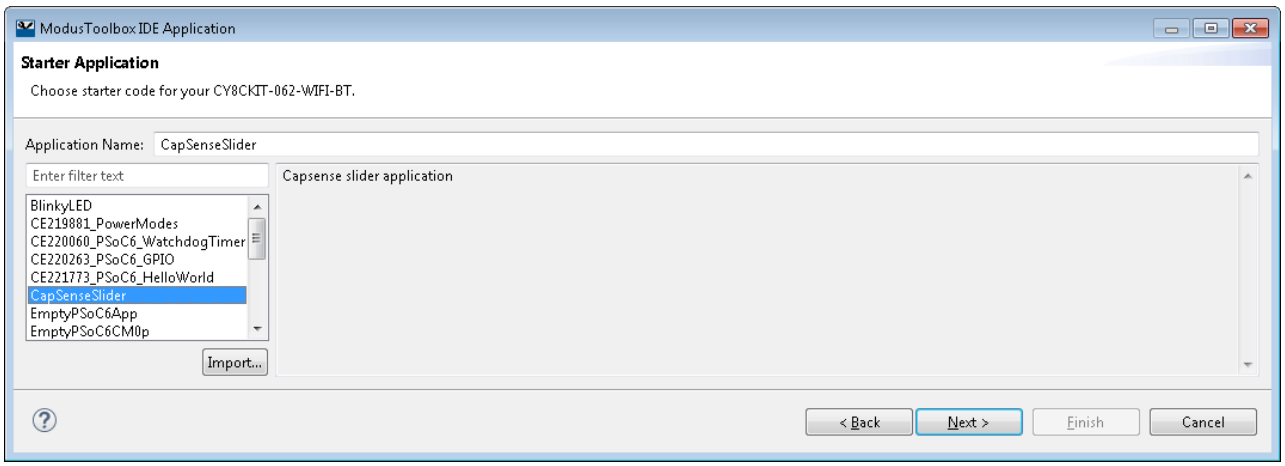
The ModusToolbox IDE Application dialog displays a list of kits, showing the Kit Name, MCU, and Connectivity Device (if applicable). As you select each of the kits shown, the description for that kit displays on the right.

You can also select **Custom Board**, which will allow you to pick a specific device to use for your application. See Custom Board.



For this example, select the CY8CKIT-062-WIFI-BT kit.

Click **Next >** to open the Starter Application page. This page lists various starter applications available for the selected kit. As you select a starter application, a description displays on the right.

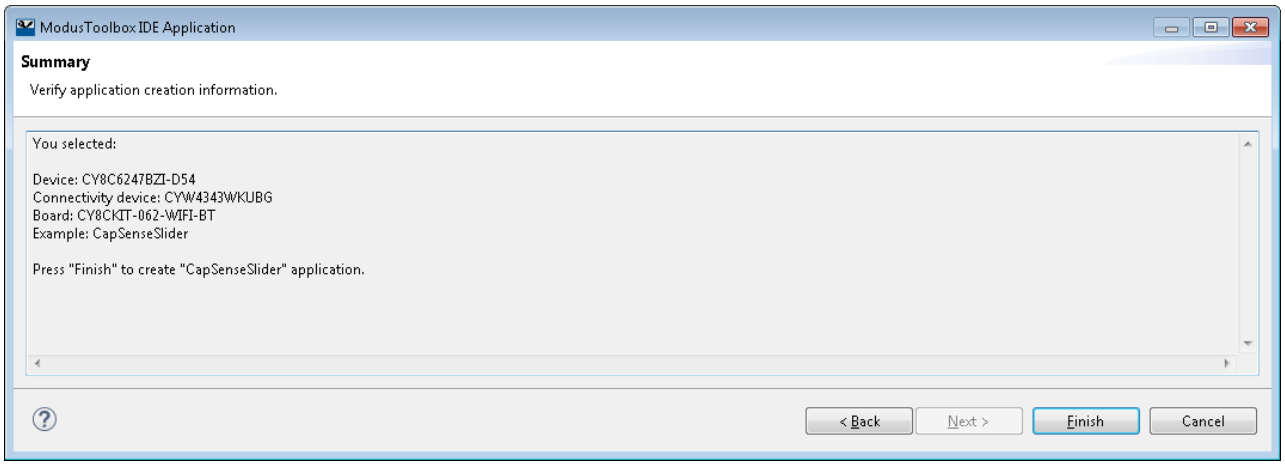


For this example:

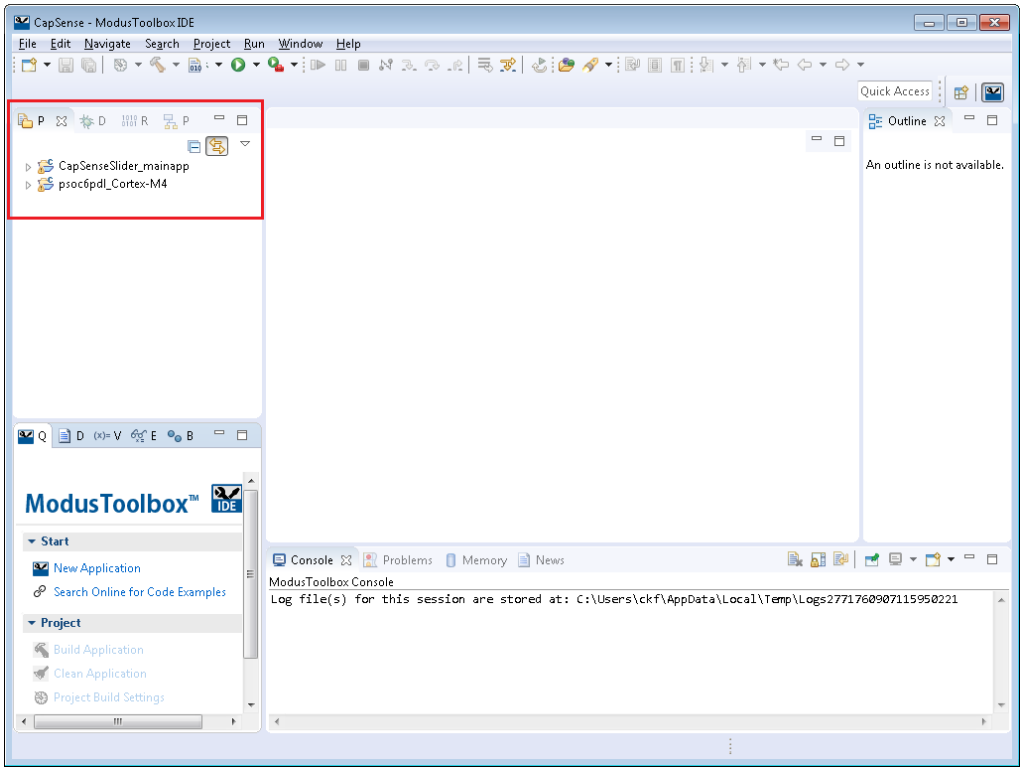
- Select the ”CapSenseSlider” application from the list.
- Type a name for your application. Do not use spaces in the application name. In this case, “CapSenseSlider” is the default name.

Note You can use the **Import…** button to select other examples that are not listed as starter applications. Many examples are available for download from the web, or you may have received an example from a colleague. In the Open dialog, select only examples that are supported by the hardware you selected for this application. Then, the example will be shown in the New Application dialog with all the other Starter Applications. See also Import Code Examples for additional details.

Click **Next >** to open the Summary page. This page shows the various options chosen for this application. Review them to ensure they are correct.



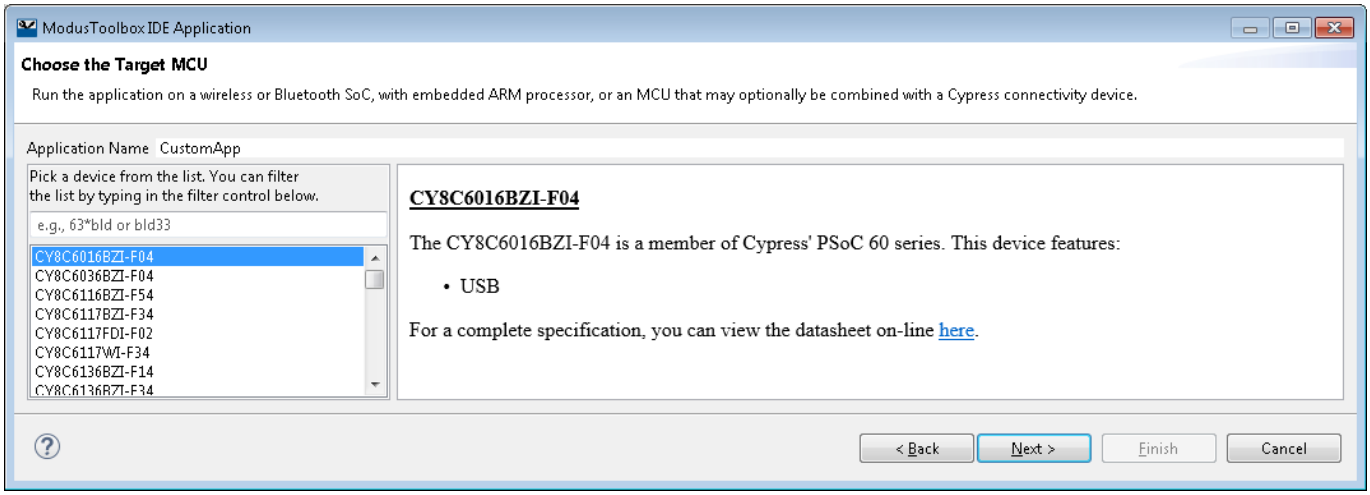
Click **Finish** to create the application. After a few moments, the ModusToolbox IDE displays progress information. When complete, the application's project folders display in the Project Explorer.



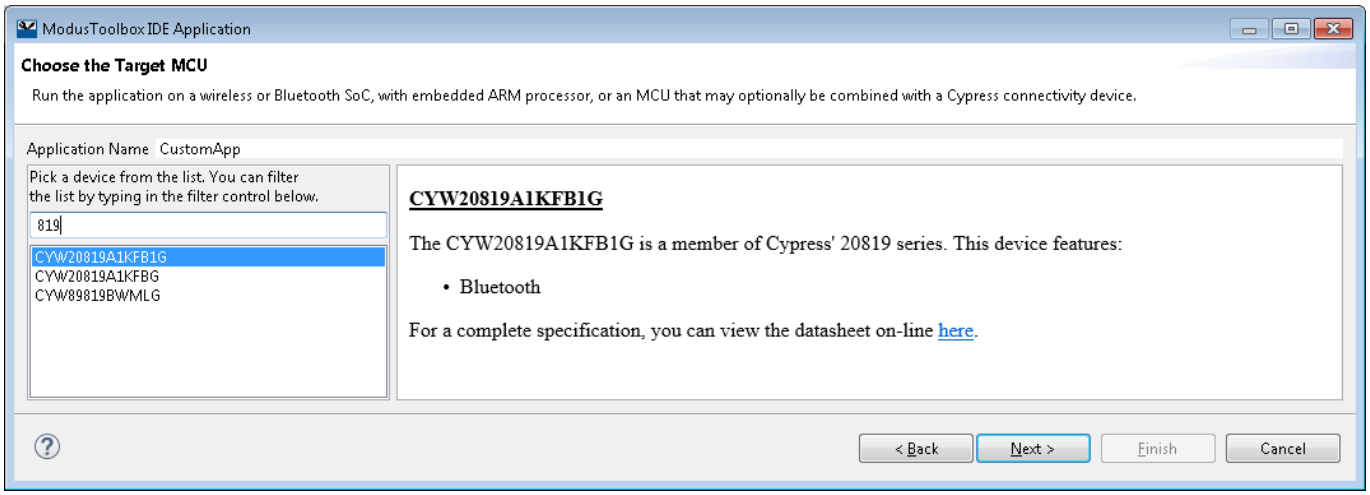
To learn how to build the application, go to the Build Starter Application section.

Custom Board

If you choose the “Custom Board” option in the Select Kit step, the next step of the dialog displays a list of devices by part numbers. When you select a part, the description displays on the right.



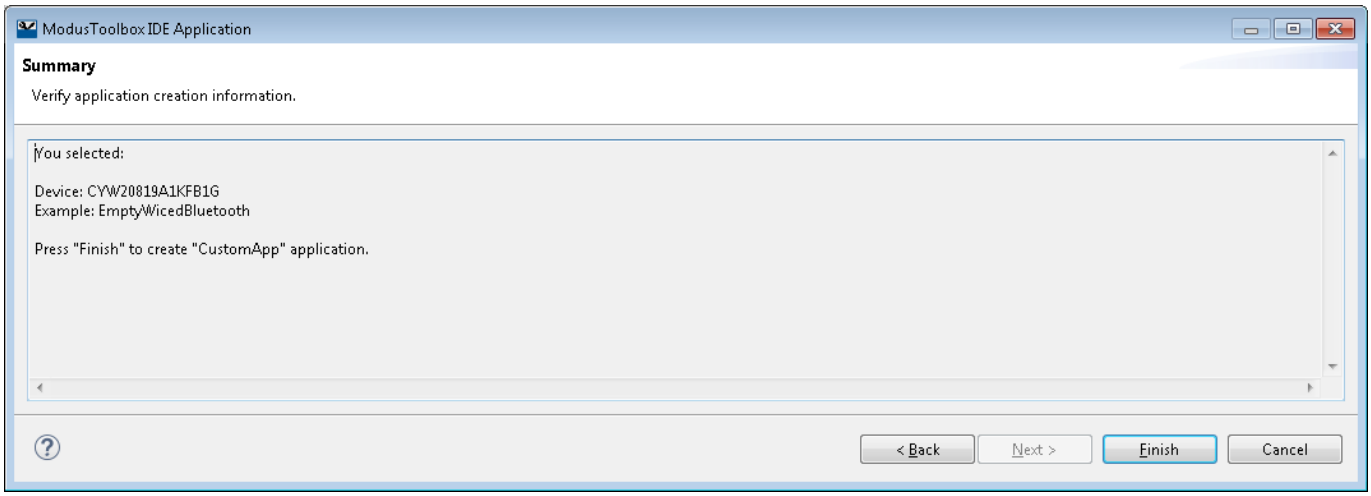
You can type in the **Filter** box to limit the number of devices displayed.



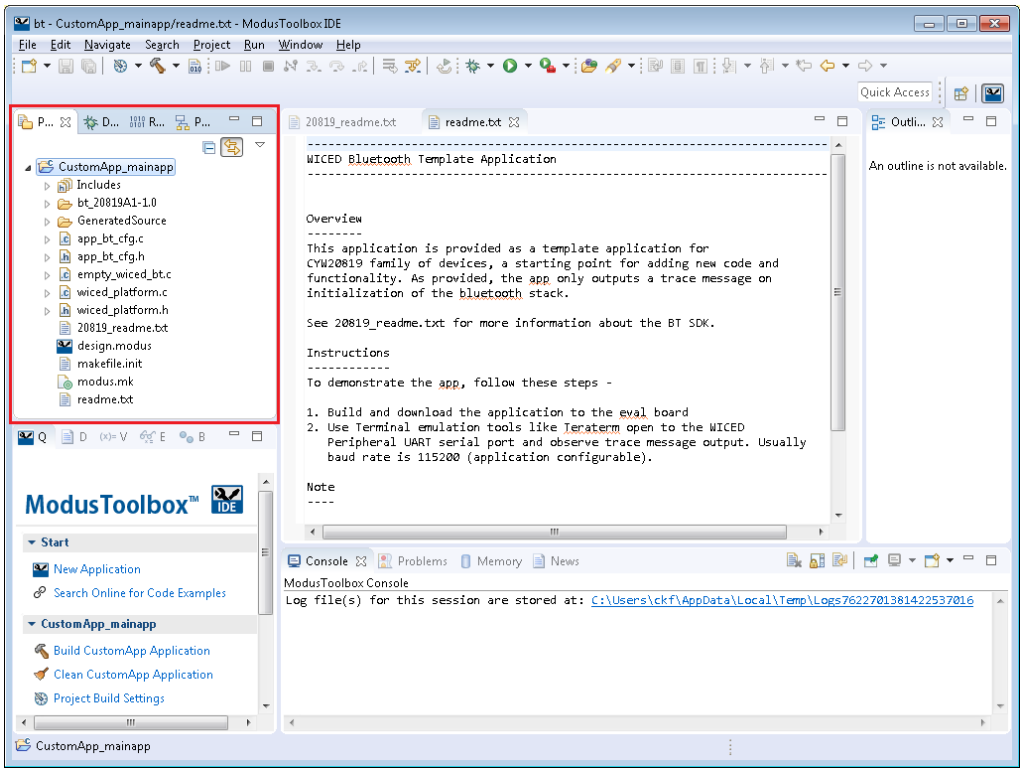
For this example, select the “CYW20819A1KFB1G” device.

Type a name for your application. Do not use spaces in the application name. In this case, “CustomApp” is the default name.

Click **Next >** to open the Summary page. This page shows the various options chosen for this application. Review them to ensure they are correct.



Click **Finish** to create the application. After a few moments, the ModusToolbox IDE will display progress information. When complete, the application's project folders will display in the Project Explorer.

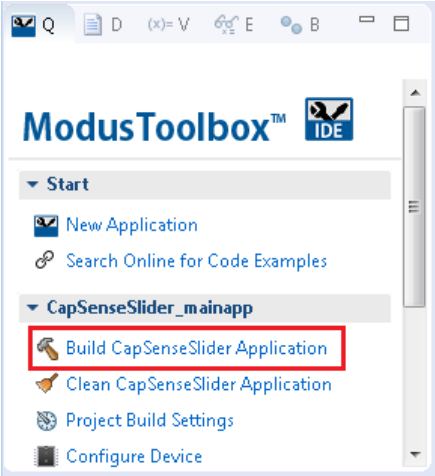


Note Applications that include a *readme.txt* file will open that file in the code editor when the application is created.

To learn how to build the application, go to the Build Starter Application section.

Build Starter Application

After loading an application, build it to generate the necessary files. Select the *<app-name>_mainapp* project in the application. Then, in the **Quick Panel**, click the “Build <app-name> Application” link.



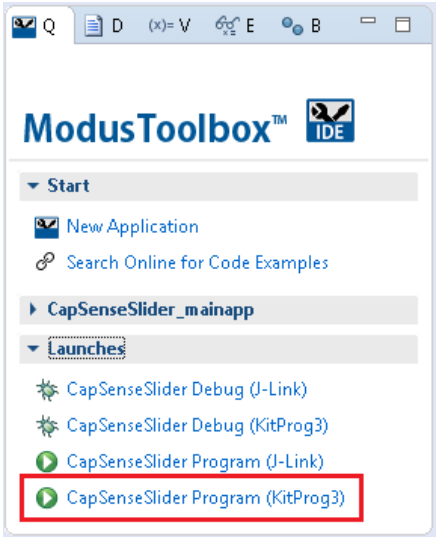
Messages display in the Console, indicating whether the build was successful or not. For more details about building applications and the various Consoles available, see the Build Applications chapter.

Program Starter Application

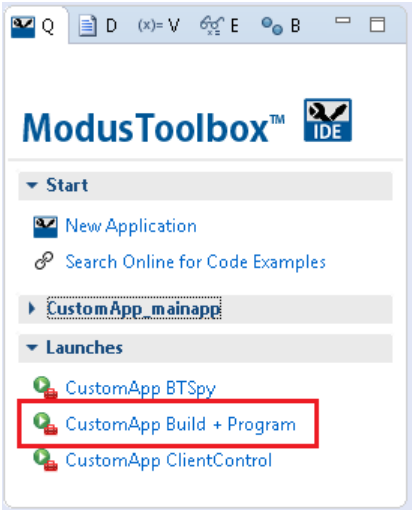
There are many more details about programming an application. This section only covers it briefly. For more detailed information, see the Program and Debug chapter.

In the Project Explorer, select the *<app-name>_mainapp* project. Then, in the **Quick Panel**, click the “<app-name> Program KitProg3” link for a PSoC 6 application, and “<app-name> Build + Program” for a CYW20819 Bluetooth application.

PSoC 6 Application



CYW20819 Bluetooth Application

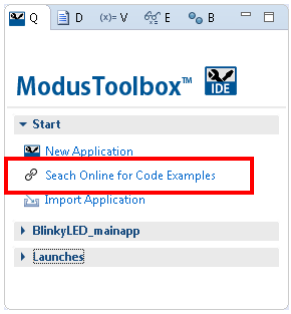


Download/Import Code Examples

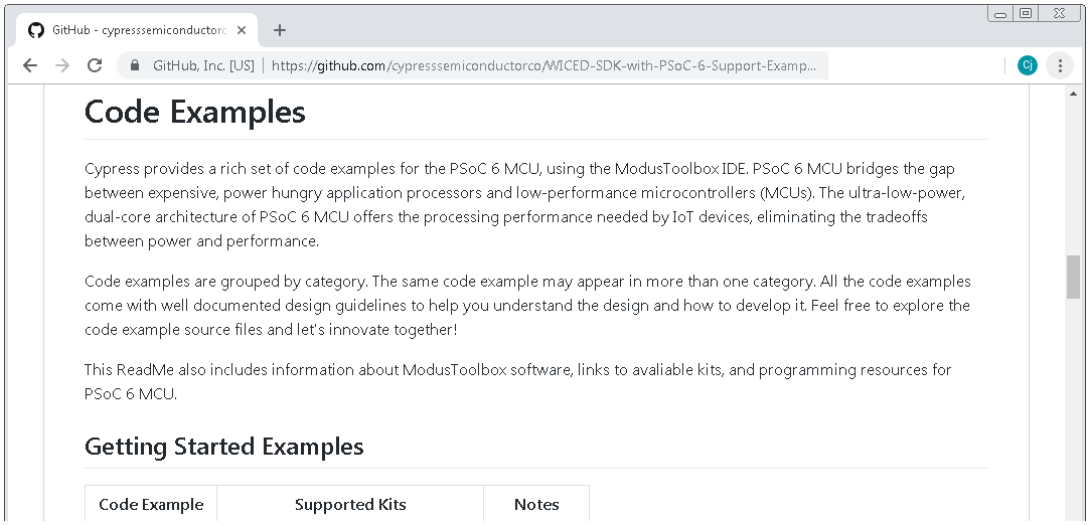
Cypress provides many code examples. These examples allow you to explore the capabilities provided by the SDK, create applications based on them, examine the source code demonstrated in them, and read their associated documentation.

Download from GitHub

The **Quick Panel** provides a link to access online code examples. Click the “Search Online for Code Examples” link.



This opens a web browser to the GitHub repository to select and download the appropriate examples.



Examples are collected into repositories on the Cypress GitHub site. You can clone or download a repository to your computer using GitHub mechanisms. You can also import a GitHub repository directly to your workspace. Use **File > Import > Git > Projects from Git** and follow the instructions in the wizard. This is standard Eclipse functionality.

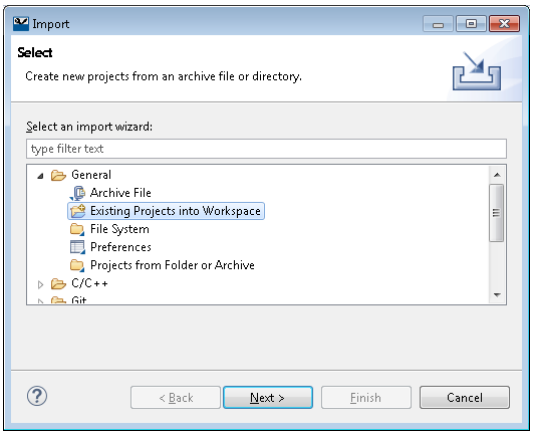
Importing the repository in this way does not create an application you can build. It does bring in all the source code for all the examples in the repository for you to examine. If you instead want an application that you can build, follow the instructions in the next section to import a code example as a ModusToolbox application.

Import Code Examples

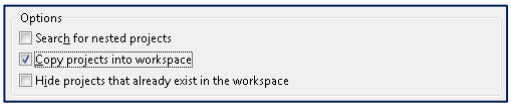
Whether you’ve downloaded an example, or received one from a colleague, Cypress recommends one of the following methods to import the example into the Modus Toolbox IDE (the method you choose depends on the format of the code example):

- If you’ve downloaded an example that includes a Makefile (*modus.mk*), use the New Application wizard to create a new application, and select the **Import…** button to open the Makefile to use as the starter application. All examples on the Cypress GitHub site follow this format.

- If you have an Eclipse-ready code example (for example, a project exported from Eclipse) that you want to import into the ModusToolbox IDE, use the standard **File > Import** process. Note the following:
  - On the Import dialog, select **General > Existing Projects into Workspace**.



  - On the next page, enable the **Copy projects into Workspace** check box.



Note There are various ways to import examples into Eclipse. If you prefer a different method, make sure that all of the project files are copied into the workspace directory.

3. Mbed OS to ModusToolbox Flow

For ModusToolbox 1.1, Cypress has enabled several kits to be used in the Mbed ecosystem. For more information, refer to the Cypress webpage on the Mbed OS website: https://os.mbed.com/teams/Cypress/.

These kits support various connectivity applications that you cannot create from the ModusToolbox IDE. You must use the Mbed OS flow for these types of applications. This section provides the necessary steps to import, build, program, and debug Mbed OS applications in the ModusToolbox IDE. The main steps include:

- Install Software for ModusToolbox
- Install and Configure Mbed CLI
- Switch Kit to DAPLink Mode
- Create Application in Mbed CLI
- Compile Application in Mbed CL (Optional)
- Export Application from Mbed CLI
- Import Application into ModusToolbox IDE
- Configure ModusToolbox IDE
- Build, Program, Debug Application

Install Software for ModusToolbox

In addition to installing ModusToolbox 1.1, you need to install other software for the Mbed OS flow. This is only required once.

Note ModusToolbox 1.0 doesn't include the GNU MCU Eclipse plugin for pyOCD debugging; you must use ModusToolbox 1.1.

Cygwin Make (Windows)

For Windows, you must install Cygwin make for the Mbed flow to work in Eclipse. Refer to the Cygwin website for instructions to download and install Cygwin. Do **not** add Cygwin to your Windows PATH system environment variable. Cygwin is only required for Eclipse.

SRecord

For Cypress PSoC 6 devices, the SRecord tool (https://sourceforge.net/projects/srecord/) is used by the ModusToolbox IDE build process to produce a combined HEX file (*mbed-os-example-blinky-combined.hex*). This combined HEX file includes the required application core from the CM0+ core, which is not included in the ELF file produced by the makefile build process (*BUILD/mbed-os-example-blinky.elf*).

You must install the SRecord tool to produce the combined HEX file in order to debug the CM4 application.

- Windows: download srecord-1.64-win32.zip and extract *srec_cat.exe* file to *C:\cygwin64\bin*.
- MacOS: install Homebrew, then use the command: `brew install srecord`
- Ubuntu Linux: use the command: `sudo apt install srecord`

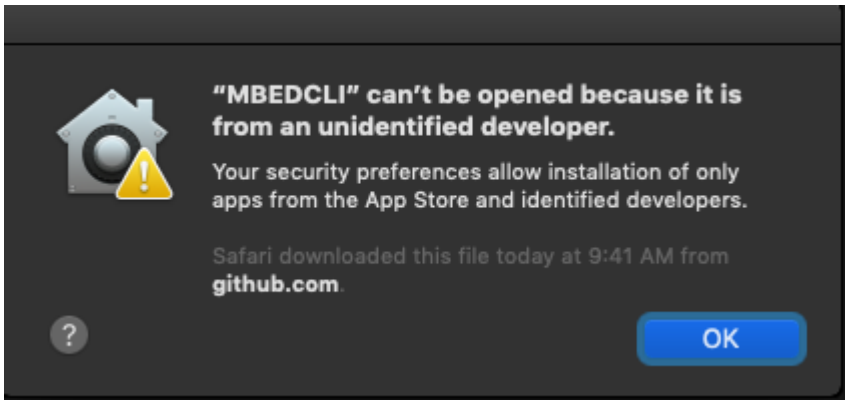
Install and Configure Mbed CLI

You should be familiar with Arm Mbed OS and the command line interface (CLI). Refer to the official Mbed OS instructions:

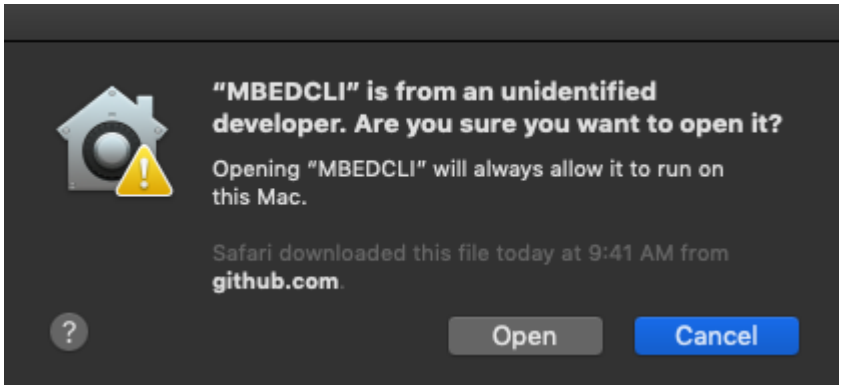
https://os.mbed.com/docs/mbed-os/v5.11/tools/installation-and-setup.html

For Windows and macOS, download and use the installer rather than manually installing it. The “see documentation” link on the Mbed webpage is for manual installation.

For macOS, you will get an error that it is from an unidentified developer.



If you get this message, right-click on the MBEDCLI application and select **Open**. Then, the message will look like the following:



Click **Open**.

Note You only have to do this once; macOS will remember the decision, and the app will open normally from then on

Update PyOCD

In order for Mbed applications to work with ModusToolbox, make sure the Mbed Python packages are updated to versions compatible with Cypress kits.

For macOS, run the MBECLI application to start an mbed-capable Terminal before installing pyocd. This ensures pyocd installs in the self-contained mbed environment. PyOCD is directly supported by the Python bundled with the MBEDCLI application.

Run the following command from a terminal window:

```
pyocd --version
```

The reply should be (or later):

```
0.16.1
```

If you don't have the correct version, run:

```
pip install -U pyocd
```

Configure Compilers

Install and configure the appropriate compilers for Mbed CLI. Refer to the Mbed OS for website supported compilers:

https://os.mbed.com/docs/mbed-os/latest/tools/after-installation-configuring-mbed-cli.html

To use the GCC 7.2.1 distribution bundled with ModusToolbox, run the `mbed config` command as appropriate for you operating system:

For Windows:

```
mbed config -G GCC_ARM_PATH %USERPROFILE%\ModusToolbox_1.1\tools\gcc-7.2.1-1.0\bin
```

For macOS:

```
mbed config -G GCC_ARM_PATH /Applications/ModusToolbox_1.1/tools/gcc-7.2.1-1.0/bin
```

For Linux:

```
mbed config -G GCC_ARM_PATH ~/ModusToolbox_1.1/tools/gcc-7.2.1-1.0/bin
```

To use another compiler, run the `mbed config` command and enter the appropriate path. The following are a few typical paths for compilers on Windows:

```
mbed config -G GCC_ARM_PATH "C:\Program Files (x86)\GNU Tools ARM Embedded\6 2017-q2-
update\bin"
mbed config -G ARM_PATH "C:\Program Files (x86)\ARM_Compiler_5.06u6"
mbed config -G ARMC6_PATH "C:\Program Files\ARMCompiler6.11\bin"
mbed config -G IAR_PATH "C:\Program Files (x86)\IAR Systems\Embedded Workbench 7.5\arm"
```

Note IAR 8 is not compatible with Mbed OS 5.11.

To use the ARM v5/v6 compilers bundled with Keil MDK v5, use one of these commands as appropriate:

```
mbed config -G ARM_PATH "C:\Keil_v5\ARM\ARMCC"
mbed config -G ARMC6_PATH "C:\Keil_v5\ARM\ARMCLANG\bin"
```

Switch Kit to DAPLink Mode

Use the fw-loader tool included with ModusToolbox 1.1 to upgrade the kit firmware using the following instructions. See also KitProg Firmware Loader for more information. The tool is located in the following directory, by default:

<install_dir>/tools/fw-loader-2.1/bin/

Note For Windows 7 only, the Mbed serial port driver must be installed on your system with the connected kit. This driver should be installed with the Mbed CLI. If not, the driver can be found at: https://os.mbed.com/handbook/Windows-serial-configuration. Do not install this on Windows 10.

- Press and hold button **SW3** while reconnecting the kit to the computer; the kit enters KitProg Bootloader mode. If successful, the amber LED2 will blink at ~2 Hz rate.
- Run the fw-loader tool from the command line to update the KitProg3 firmware:

  ```
  fw-loader --update-kp3
  ```

- After updating the firmware, switch the kit to DAPLink mode:
  - For single-button kits (CY8CPROTO_062_4343W) press button **SW3** for >2 seconds and release.
  - For two-button kits (CY8CKIT-062-BLE, CY8CKIT_062_WIFI_BT, CYW943012P6EVB-01, and CY8CKIT-062-4343W) press and release button **SW4** (custom app).

- Run the following from the command line:

  ```
  mbedls
  ```

  The reply should indicate that a valid device is connected.

Create Application in Mbed CLI

Open an Mbed CLI prompt and create the directory for Mbed examples. For example:

```
mkdir mbed-examples
cd mbed-examples
```

Note On Windows, do **not** use a Cygwin terminal window, and make sure Cygwin is not in your Windows PATH system environment variable.

Import the mbed-os-example-blinky example:

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

Switch to the example directory:

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

Compile Application in Mbed CLI (Optional)

Note This step is optional. It may be useful to confirm that the application builds successfully and produces a HEX file.

Compile the Mbed application for one of the supported target boards with one of the supported toolchains

```
mbed compile --target TARGET --toolchain TOOLCHAIN
```

where TARGET is one of:

CY8CKIT_062_WIFI_BT
CY8CPROTO_062_4343W
CY8CKIT_062_BLE
CY8CKIT_062_4343W

and TOOLCHAIN is one of:

GCC_ARM
ARM
IAR

For example:

```
mbed compile --target CY8CKIT_062_WIFI_BT --toolchain GCC_ARM
```

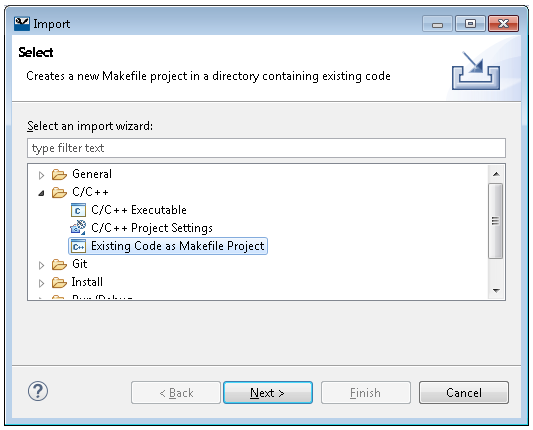
Export Application from Mbed CLI

To export the Mbed application for use in the ModusToolbox IDE, use the Mbed CLI `export` command. For example:

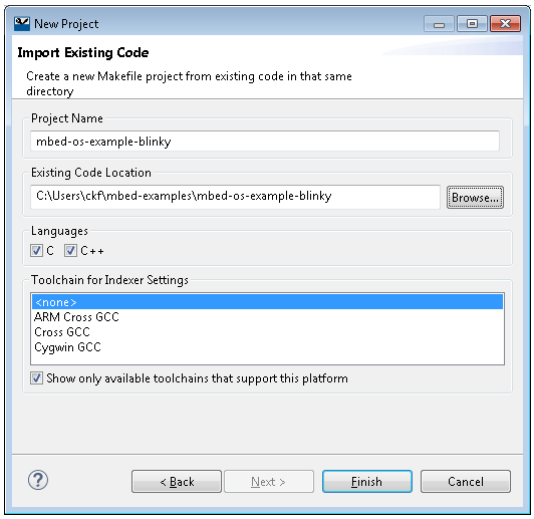
```
mbed export -i eclipse_gcc_arm -m CY8CKIT_062_WIFI_BT
```

Import Application into ModusToolbox IDE

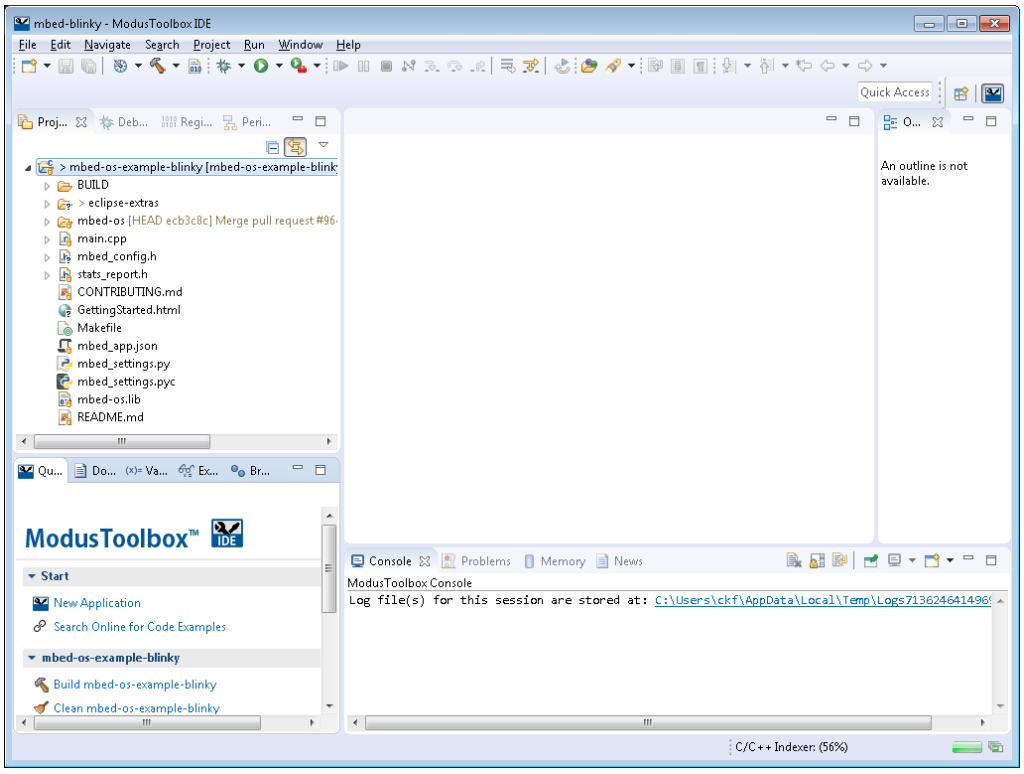
Use the standard Eclipse **Import** function to import the Mbed application. Expand the **C/C++** folder and select the **Existing Code as Makefile Project** option. Click **Next >**.



Click **Browse…** and navigate to the directory where you exported the application. Ensure the **Toolchain for Indexer Settings** is set to **<none>**. Do not change any other settings. Click **Finish**.



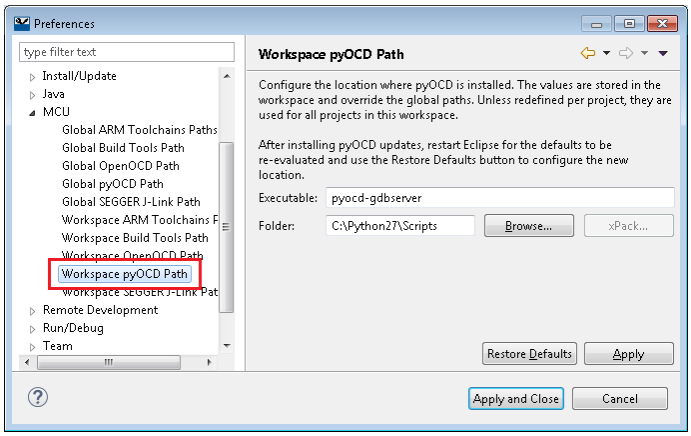
When the import finishes, the Mbed OS application will be shown in the Project Explorer.



Configure ModusToolbox IDE

Define the Workspace pyOCD Path

Open the Preferences dialog, select **MCU > Workspace pyOCD Path**, and set the following workspace paths (adjust the path to the Scripts directory for your python/pyocd installation):



Windows:

- Workspace pyOCD Path > Executable = pyocd-gdbserver
- Workspace pyOCD Path > Folder = C:\Python27\Scripts

macOS:

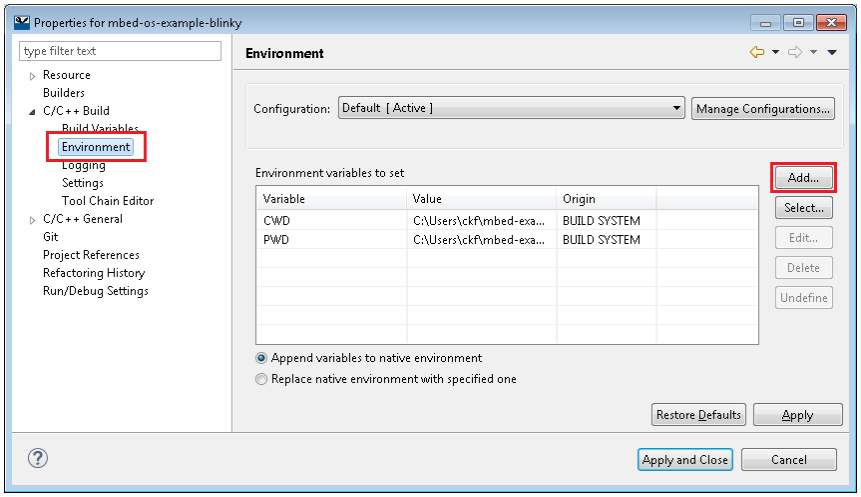
- Workspace pyOCD Path > Executable = pyocd-gdbserver
- Workspace pyOCD Path > Folder = /Applications/MBEDCLI.app/Contents/Resources/miniconda/bin

Linux:

- Workspace pyOCD Path > Executable = pyocd-gdbserver
- Workspace pyOCD Path > Folder = ~/.local/bin

Create the PATH variable

Right-click on the project and select **Properties**. Navigate to **C/C++ Build > Environment**.



Click **Add** to add a new variable with name PATH, and select "Add to all configurations."

Click **OK** to close the New Variable dialog. This step is recommended for Windows to clean up any existing value of the system PATH variable in the current project (this ensures that the project configuration is isolated from the host environment).

Then, click **Edit** to open the dialog again. Enter the appropriate value:

- Windows: C:/cygwin64/bin;${cy_tools_path:gcc-7.2.1}/bin
- macOS: ${cy_sdk_install_dir}/tools/gcc-7.2.1-1.0/bin:/usr/bin:/bin:/Applications/MBEDCLI.app/Contents/Resources/miniconda/bin:/Applications/MBEDCLI.app/Contents/Resources/bin:/Applications/MBEDCLI.app/Contents/Resources/git/bin:/usr/local/bin

  Also add /usr/local/Cellar/srecord/1.64/bin to /etc/paths
- Linux: Check the gcc directory path: /usr/bin:/bin:${cy_sdk_install_dir}/tools/gcc-7.2.1-1.0/bin

Note It is possible to target a custom Arm Toolchain instead of GCC 7.2.1 provided by ModusToolbox. To use the official GNU Arm Embedded Toolchain, install the following to the default location:

https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads

Then prepend the following to the Eclipse PATH instead of ${cy_tools_path:gcc-7.2.1}/bin:

<install_dir>/GNU Tools ARM Embedded/7 2018-q2-update/bin

Build, Program, Debug Application

Build

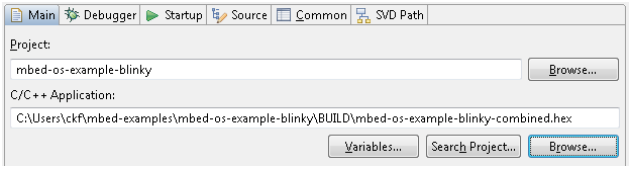
Select **Project > Build** to build the project and generate the combined HEX file discussed previously. When complete, the Console displays a message similar to the following:

```
hex file ready to flash: BUILD/mbed-os-example-blinky-combined.hex
```

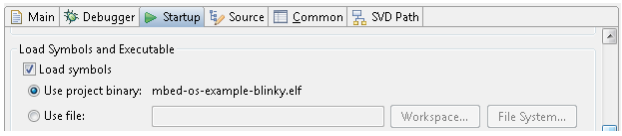
Program

Open the Run Configurations dialog. Select the “program” configuration under **GDB PyOCD Debugging**.

On the **Main** tab, under **C/C++ Application**, click **Browse…** to change the specified target from the ELF file to the combined HEX file. Click **Apply**



On the **Startup** tab, make sure the Load symbols option is using the ELF file. If not, select the elf file, and then click **Run**.



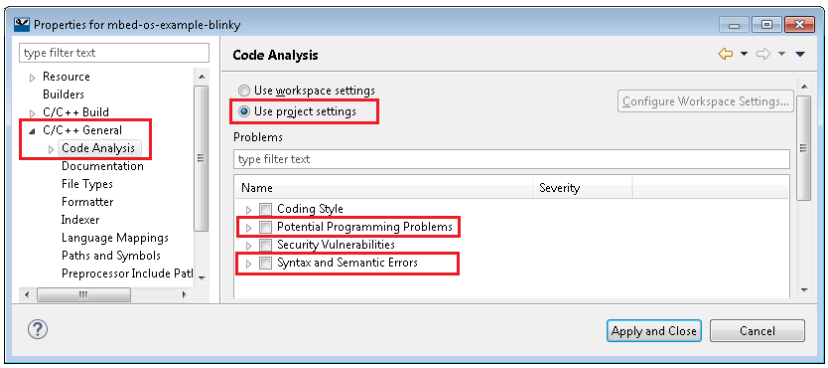
When programming is complete, the LED should blink red on the board. You may need to press the **Reset** button.

Debug

Open the Debug Configurations dialog and select the “debug” configuration under **GDB PyOCD Debugging**. Repeat the same process for changing the configuration as shown under programming. Then click **Debug**.

The IDE will switch to debugging mode and will halt at the break, ready for debugging.

Note While debugging, you may see various errors in your code (*main.cpp* file), such as “function ‘printf’ could not be resolved.” These are likely not real errors. They are caused by the import of the make file into Eclipse. To turn these errors off, go to **Project > Properties > C/C++ > Code Analysis**, and disable “Potential Programming Problems” and “Syntax and Semantic Errors.”

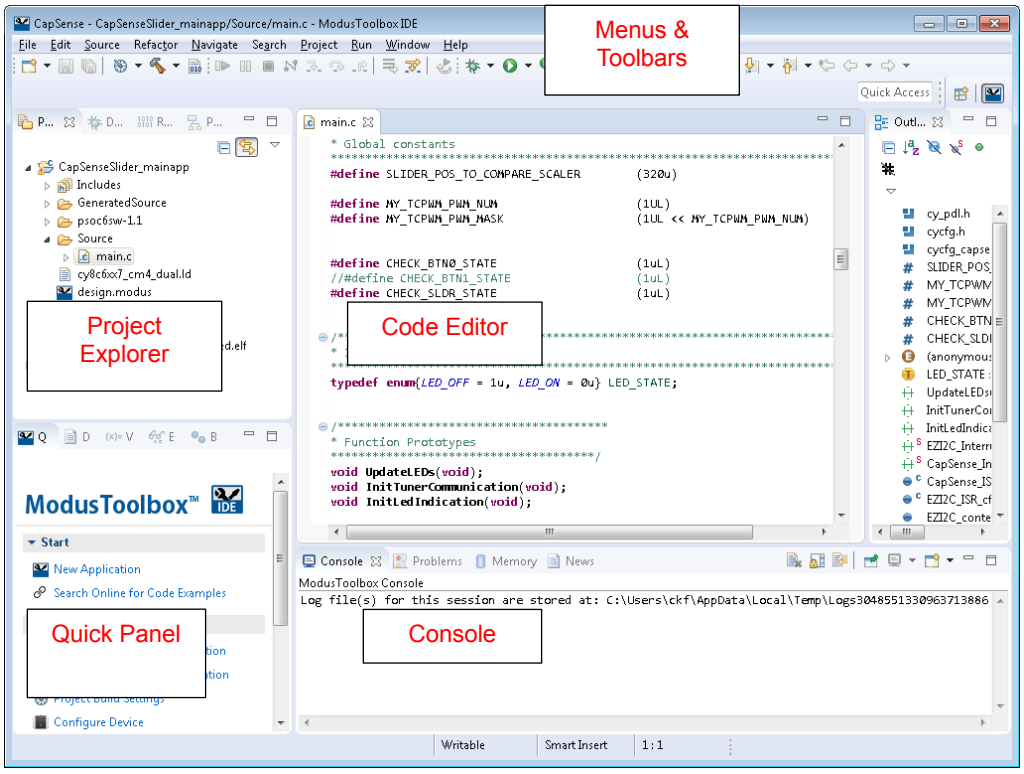


4. IDE Description

Overview

The ModusToolbox IDE is based on the Eclipse IDE “Oxygen” version. It uses several plugins, including the Eclipse C/C++ Development Tools (CDT) plugin. For more information about Eclipse, refer to the *Eclipse Workbench User Guide*. Cypress also provides a document called the *Eclipse Survival Guide*, which provides tips and hints for how to use the ModusToolbox IDE.

The IDE contains Eclipse standard menus and toolbars, plus various panes such as the Project Explorer, Code Editor, and Console. One difference from the standard Eclipse IDE is the “ModusToolbox Perspective.” This perspective provides the “Quick Panel,” a “News View,” and adds tabs to the Project Explorer. “Perspective” is an Eclipse term for the initial set and layout of views in the IDE. The ModusToolbox IDE also provides a Welcome Page, which displays on first launch of the IDE for a given workspace.



Note If you switch to a different perspective, you can restore the ModusToolbox Perspective by clicking the ModusToolbox icon button in the upper right corner. You can also select **Perspective > Open Perspective > ModusToolbox** from the **Window** menu. To restore the ModusToolbox perspective to the original layout, select **Perspective > Reset Perspective** from the **Window** menu.

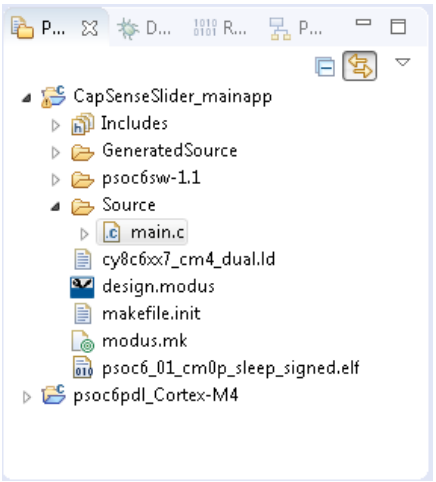
The following describe different parts of the IDE:

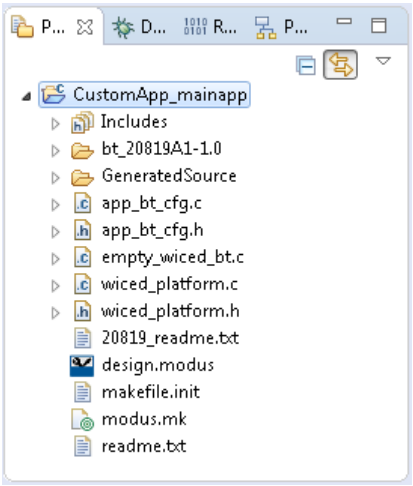
- Menus and Toolbars – Use the various menus and toolbars to access build/program/debug commands for your application. Many of these are covered in the *Eclipse Workbench User Guide*.
- Project Explorer – Use the Project Explorer to find and open files in your application. See Project Explorer for more information.
- Quick Panel Tab – Use this tab to access appropriate commands, based on what you select in the Project Explorer.
- Documents Tab – Use this tab to access various documents. This tab is next to the Quick Panel tab. You can also open documentation from the **Help** menu.
- News View – This is an Eclipse view in the ModusToolbox Perspective that displays a page of blog articles from cypress.com. This is located in the same panel as the Console and Problems tabs.

- Code Editor – Use the Code Editor to edit various source files in your application.
- Welcome Page – This is an HTML document. It contains links to Help topics and videos. You can open it it any time from the **Help** menu.

Project Explorer

In the ModusToolbox IDE, after creating an application, the Project Explorer contains one or related project folders. The following images show a PSoC 6 application and a CYW20819 Bluetooth application. For applications imported from Mbed OS, see Mbed OS to ModusToolbox Flow.



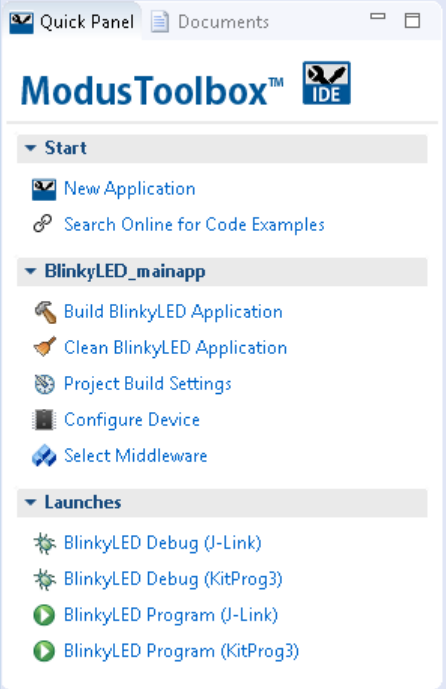


Both types of applications contain a similar project structure. In most cases, focus on the *<app-name>_mainapp* project. It contains the main application source code, includes, generated source, *design.modus* file, and the Makefile.

Note PSoC 6 applications also include a *psoc6pdl_Cortex-M4* project folder, which makes available all of the source code necessary to interact with the peripherals.

Quick Panel

As stated previously, the Quick Panel is part of the ModusToolbox Perspective. It provides quick access to commands and documentation based on what you have selected in the Project Explorer.



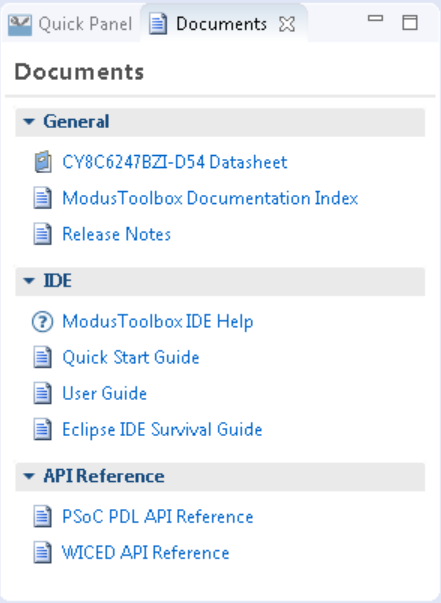
Quick Panel Tab

The first tab of the Quick Panel contains links to various commands, organized as follows:

- **Start** – This contains the New Application link to create new applications, and a link to find Code Examples.
- **Selected <app-name>_project** – This contains different project-related links based on the project that is selected in the Project Explorer, as well as the type of application. Links here include: Build and Clean the application, access Build Settings, Configure Device, and Select Middleware.
- **Launches** – This contains various Launch Configurations, based on the selected application project and device, which can be used to program the device and launch the debugger. This area is only populated if you have the top project in your application selected (*<app-name>_mainapp*). For more information, see Launch Configurations.

Documents Tab

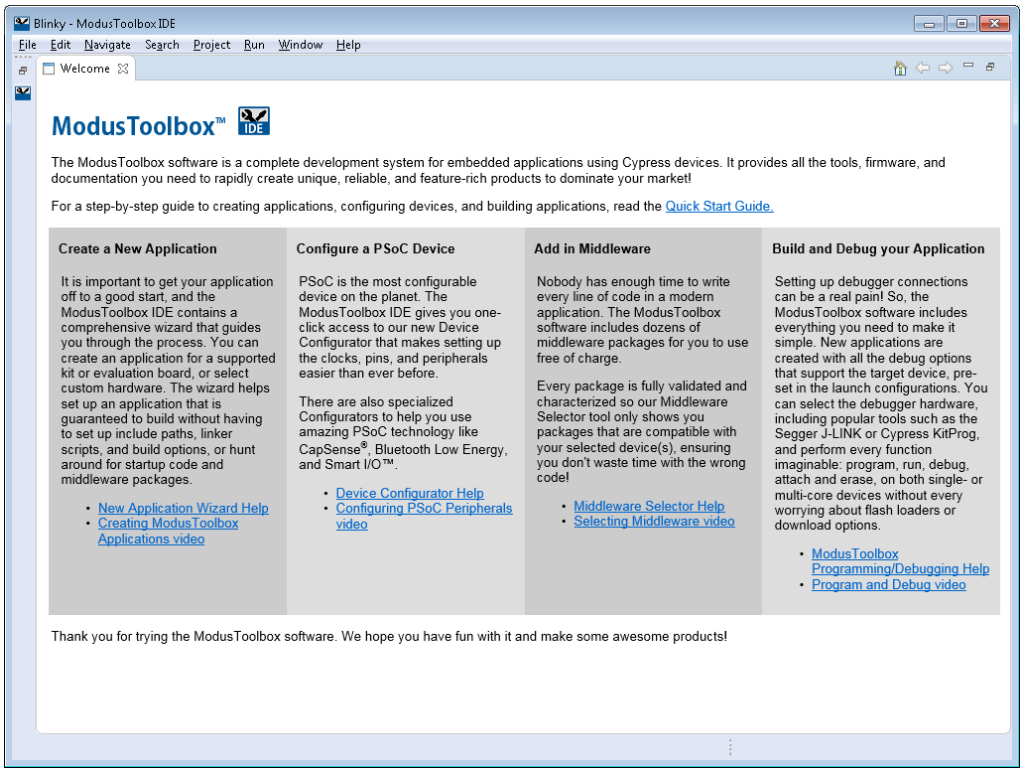
This tab contains links to various documents, such as the applicable device datasheet, IDE documentation, and API documentation.



Most of these documents are also available from the **Help** menu, along with other useful documentation and links to the web.

Welcome Page

The Welcome page is a static HTML file that provides an overview of the ModusToolbox software, as well as links to various Help topics and videos. This opens by default for a new installation and any new workspace.



Click the **X** on the Welcome Page tab to close it, or click the **Restore**  button to move it to the side.

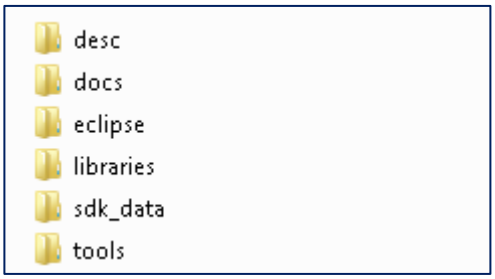
You can access this page at any time from **Help > Welcome**.

5. SDK Description

The SDK provides the central core of ModusToolbox software. It contains configuration tools, drivers, libraries, middleware, as well as various utilities, Makefiles, and scripts. You may use one or a few of these tools in any environment you prefer.

Directory Structure

Refer to the *ModusToolbox Installation Guide* for information about installing ModusToolbox. Once it is installed, the various ModusToolbox top-level directories are organized as follows:



These directories contain the following files and folders:

- **desc –** This contains xml files used with information about installed SDKs. You do **not** need to use anything in this folder.
- **docs –** This is the top-level documentation directory. It contains various top-level documents and an html file with links to documents provided as part of ModusToolbox. See Documentation for more information.
- **eclipse (or ModusToolbox.app on macOS) –** This contains the IDE. See IDE Description.
- **libraries –** This contains firmware and resources files, including:
  - bt_20819A1-<version>– This contains Bluetooth support drivers, middleware, and documentation.
  - platforms-<version> – This contains internal / helper Makefiles used with the IDE and command-line builds.
  - psoc6sw-<version> – This contains PSoC 6 support drivers, middleware, and documentation.
  - udd-<version> – This contains internal device data files. You do **not** need to use anything in this folder.
- **sdk_data –** This contains files used to track installed SDKs. You do **not** need to use anything in this folder.
- **tools –** This contains all the various tools and scripts provided as part of ModusToolbox. See Tools for more information.

Documentation

The */docs* directory contains top-level documents and an html document with links to all the documents included in the installation and on the web.

Release Notes

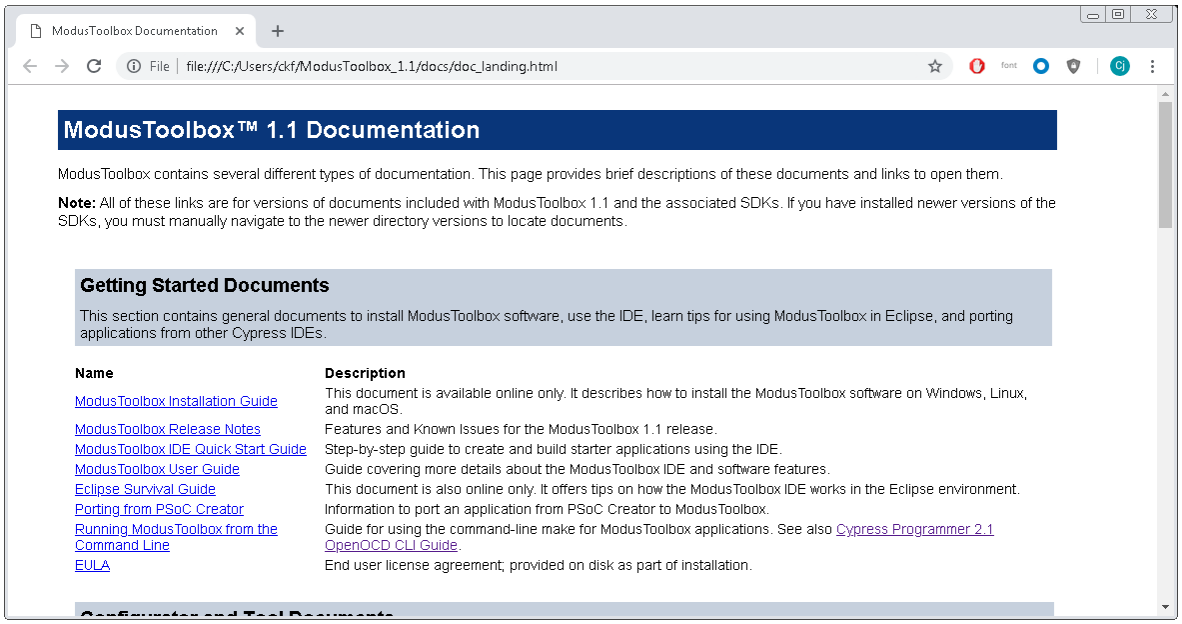
For the 1.1 release, the release notes document is for all of the ModusToolbox software included in the installation. In the future, there may be separate SDK-only and IDE-only versions of the release notes, depending on what updates you install.

Top-Level Documents

This folder contains the ModusToolbox IDE Quick Start Guide and this user guide. These guides cover different aspects of using the IDE.

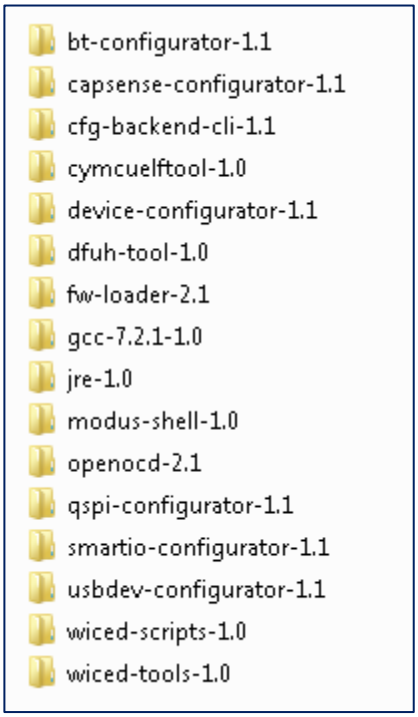
Document Index Page

The *doc_landing.html* file provides links to all the documents included in the installation and on the web. This file is available from the IDE **Help** menu and the **Documents** tab in the **Quick Panel**.



Tools

The */tools* folder contains the following tools (version numbers may be different in your installation):



- Configurators – There are several Configurators used to update various settings for different peripherals. See Use Configurators.
- cfg-backend-cli – This contains backend support files used by the system. You do **not** need to interact with this folder.
- cymcuelftool – This tool is used to manipulate Elf files. Refer to the *CyMCUElfTool User Guide* located in the tool's doc folder.
- dfuh-tool – This tool is used to communicate with a PSoC 6 MCU that has already been programmed with an application that includes device firmware update capability.
- fw-loader – This is the Firmware Loader tool used to update firmware on the programmer/debugger device on PSoC 6 kits. See KitProg Firmware Loader.
- GCC – ModusToolbox software includes GCC version 7.2.1 as the preferred toolchain. See https://www.gnu.org/software/gcc/ for information.
- JRE – This folder contains the Java Runtime Environment version provided as part of the tool. This is used by the IDE and the backend. See https://www.java.com for more information.
- Modus-Shell – This folder contains various helper utilities used by the system. You do **not** need to interact with this folder.
- Open OCD – This contains the version of the Open On-Chip Debugger used by ModusToolbox to program various boards. For more information, refer to the *Cypress Programmer 2.0 User Guide*.
- WICED Scripts – This contains scripts used to build WICED applications You do **not** need to use anything in this folder.
- WICED Tools – This folder contains different scripts and tools for programming and debugging the Bluetooth-based platforms. Refer to the Bluetooth documentation as needed.

6. Configure Applications

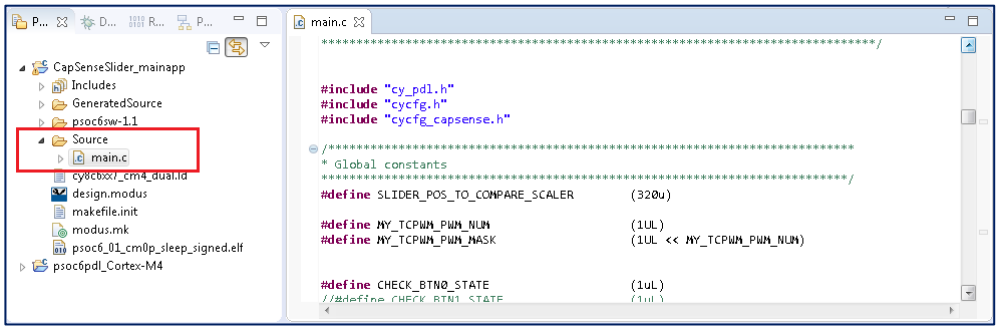
This chapter covers how to make various changes to your application. It includes:

- Modify Code
- Use Configurators
- Change ModusToolbox Settings
- Select Middleware
- Rename Application

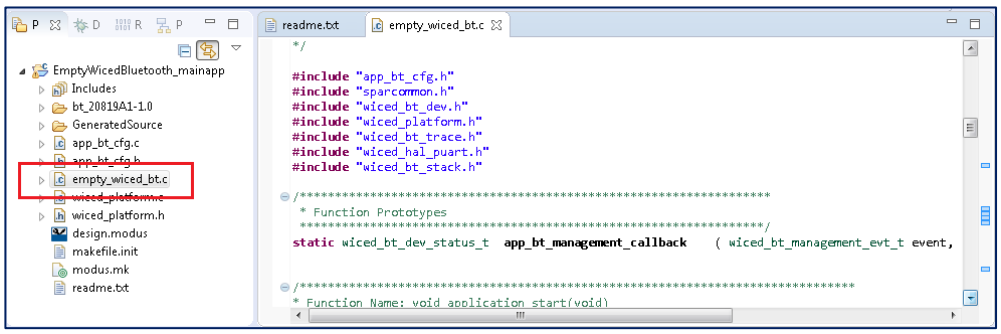
Modify Code

Most starter applications work as they are, and there is no need to add or modify code. However, if you want to update and change the starter application to do something else, or if you are developing your own application, open the appropriate file in the code editor.

- **PSoC 6:** In the Project Explorer, expand the *<app-name>_mainapp\Source* project folder and double-click the *main.c* file.



- **CYW20819 Bluetooth:** In the Project Explorer, expand the *<app-name>_mainapp* project folder and double-click the application *<app-name>.c* file.



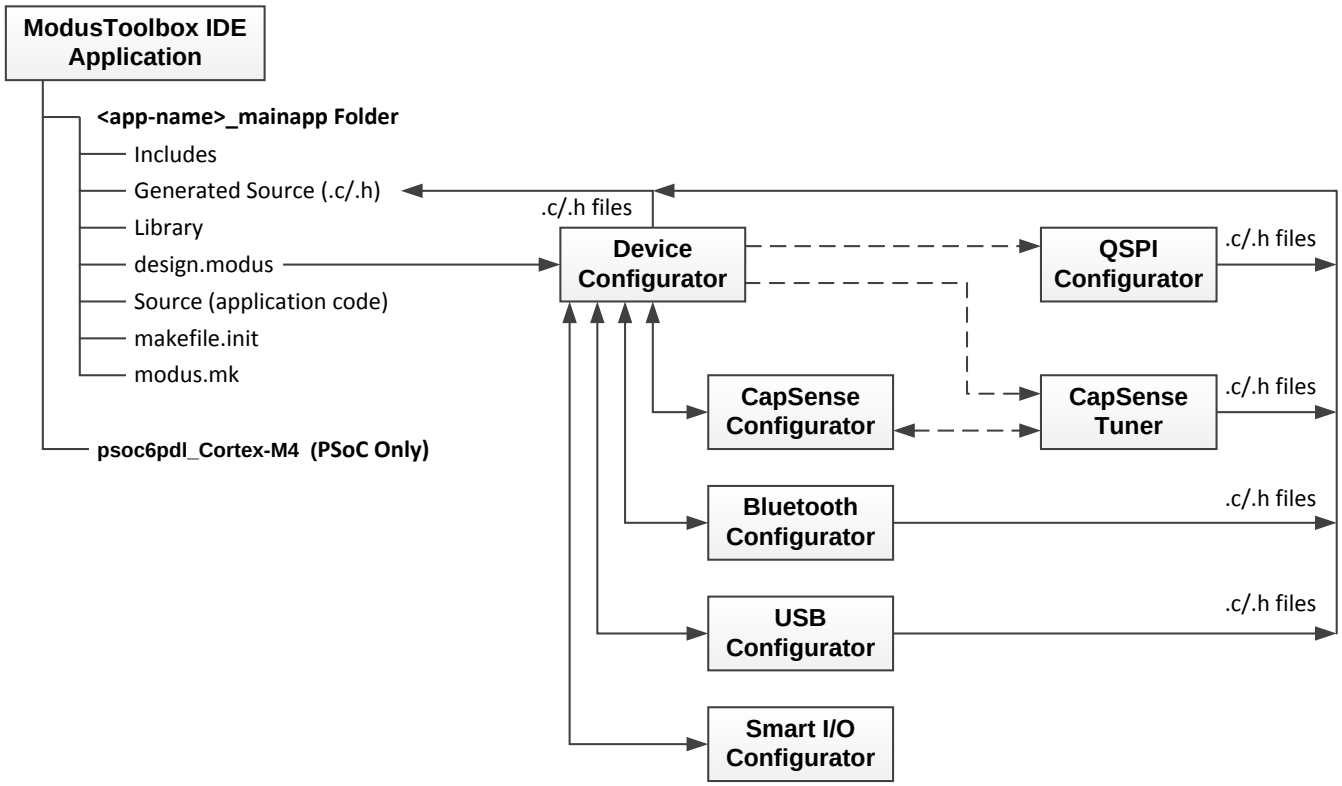
As you type into the file, an asterisk (*) will appear in the file's tab to indicate changes were made. The **Save/Save As** commands will also become available to select.

Use Configurators

ModusToolbox software provides graphical applications called configurators that make it easier to configure a hardware block. For example, instead of having to search through all the documentation to configure a serial communication block as a UART with a desired configuration, open the appropriate configurator to set the baud rate, parity, stop bits, etc. Upon saving the hardware configuration, the tool generates the C code to initialize the hardware with the desired configuration.

Overview

The ModusToolbox IDE manages configuration of resources in a target application. The IDE stores configuration settings in the *design.modus* file. This file is used by the graphical configurators, which generate firmware. This firmware is stored in the application's "GeneratedSource" folder. The following diagram provides a high-level view of this interaction between the IDE and the configurators.



The *design.modus* file is responsible for holding all of the hardware configuration information. It contains the following:

- Selected device
- Resource parameters
- Configurator parameters
- Constraints
- Referenced kits

Configurators are independent of each other, but they can be used together to provide flexible configuration options. If intended to be used together, the generated source needs to be saved in the same location. They can be used stand alone, in conjunction with other tools, or within a complete IDE. Everything is bundled together as part of the unified SDK for distribution purposes. Configurators are used for:

- Displaying a user interface for editing parameters
- Setting up connections such as pins and clocks for a peripheral
- Generating code to configure middleware

Available Configurators

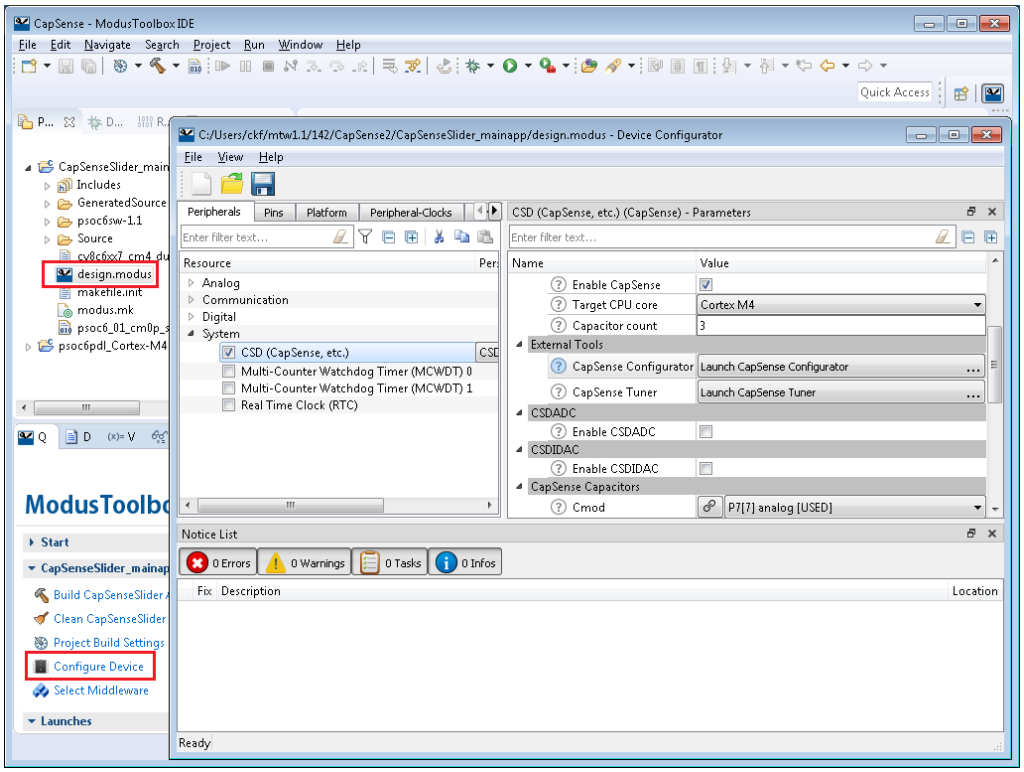
The available Configurators include:

- Device Configurator: Set up the system (platform) functions such as pins, interrupts, clocks, and DMA, as well as the basic peripherals, including UART, Timer, etc.
- CapSense Configurator and Tuner: Configure CapSense, test it, and generate the required firmware.
- USB Configurator: Configure USB settings and generate the required firmware.
- QSPI Configurator: Configure external memory and generate the required firmware.
- Smart I/O™ Configurator: Configure the Smart I/O.
- Bluetooth Configurator: Configure the Bluetooth settings.

Refer to the applicable Configurator guide, available from that tool's **Help** menu.

Launching the Device Configurator

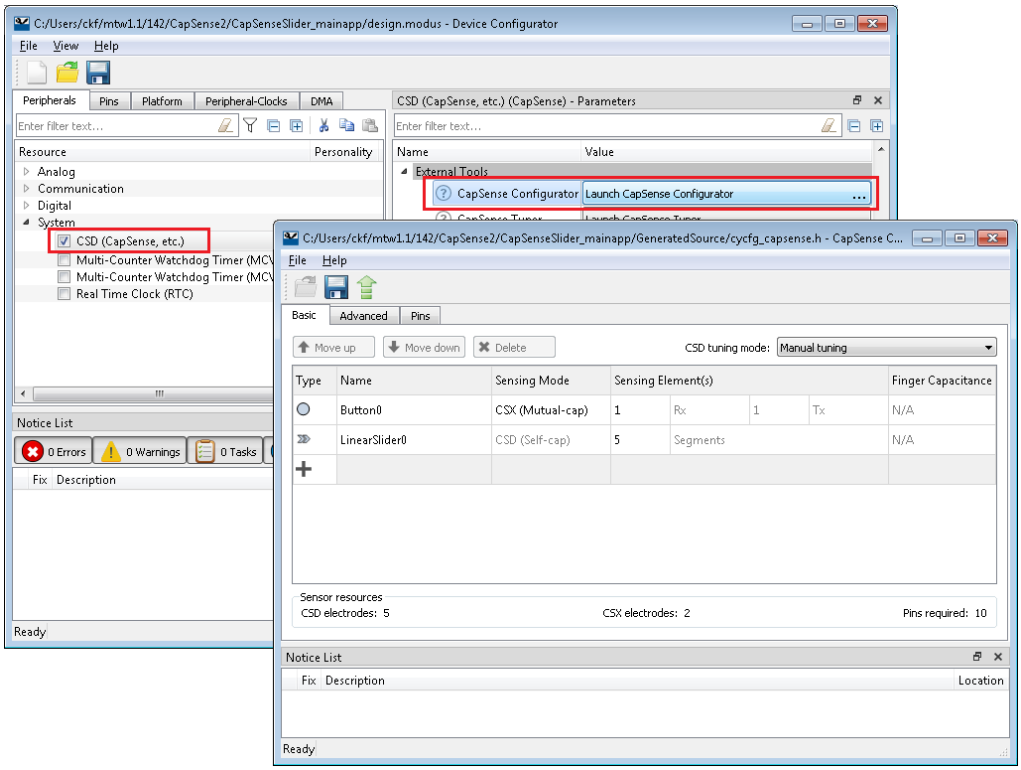
If your application includes a *design.modus* file, double-click on it in the Project Explorer to launch the Device Configurator from the ModusToolbox IDE. You can also click the “Configure Device” link in the **Quick Panel**.



As you select and enable different resources in the Device Configurator, the **Parameters** pane displays various configuration settings that can be updated. For more details, refer to the *ModusToolbox Device Configurator Guide* available from the **Help** menu.

Launching Separate Configurators

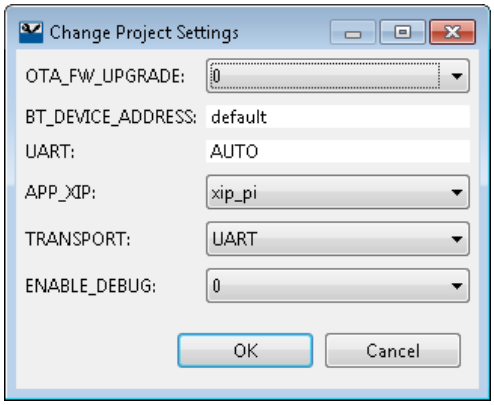
To launch a separate configurator from the Device Configurator, enable the desired resource under **Peripherals > Resource**, then click the **[Launch . . .]** button under **Parameters**.



In this example, the CapSense Configurator opens in a separate window. Refer to the individual configurator guides for more information. These are available from the configurators' **Help** menu.

Change Application Settings

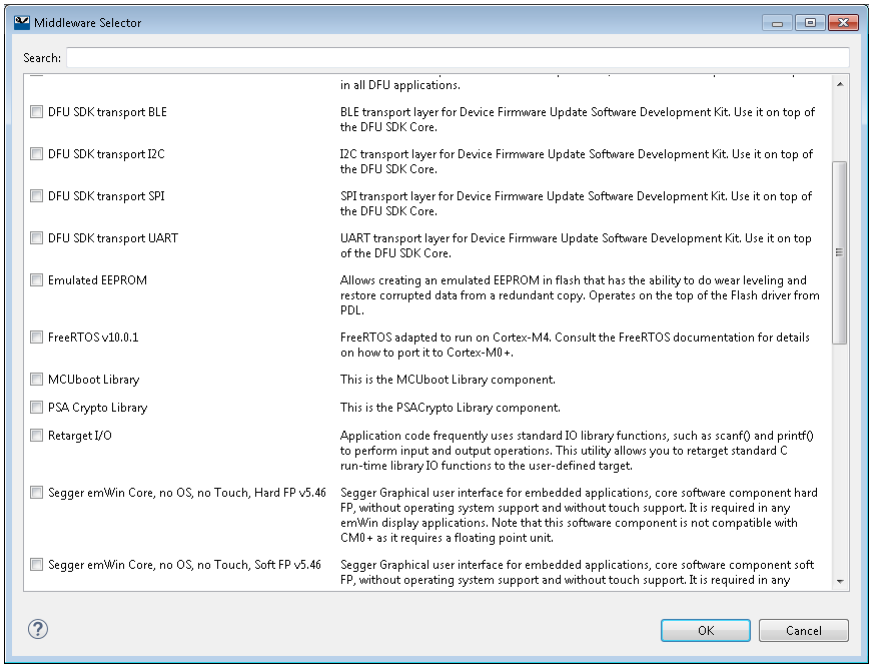
For CYW20819 Bluetooth applications only, use the Change Project Settings dialog to change various application parameters.



To open the dialog, right-click on a project and select **Change Application Settings**. For more information about the various parameters, refer to the *20819_readme.txt* file.

Select Middleware

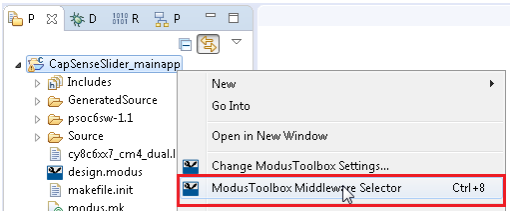
The Middleware Selector allows you to select what firmware packages to add or remove from the application. The tool displays only middleware that is valid for the current platform. After adding or removing middleware from an application, the necessary compiler options, such as include paths, will be added or removed.



A middleware component can be distributed as a pre-compiled library. In this case, there are library variants compiled using soft-float and hard-float application binary interfaces (ABIs). The ABI used to compile a library is specified in the middleware component name. You must compile your entire application with the same ABI and link with a compatible set of libraries. Therefore, use the same floating-point ABI for middleware libraries and the toolchain settings for your application. To change the Float ABI toolchain settings, open **Project > Properties**. Then select the **C/C++ Build > Settings > Target processor** and specify the Float ABI option. You must also specify the same Float ABI for all dependent projects.

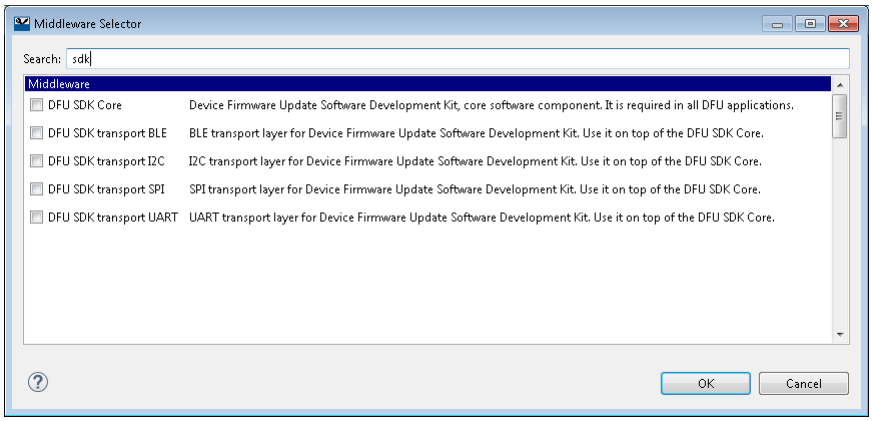
Launch the Dialog

Right-click on the main project and select **ModusToobox Middleware**. You can also click the “Select Middleware” link in the **Quick Panel**



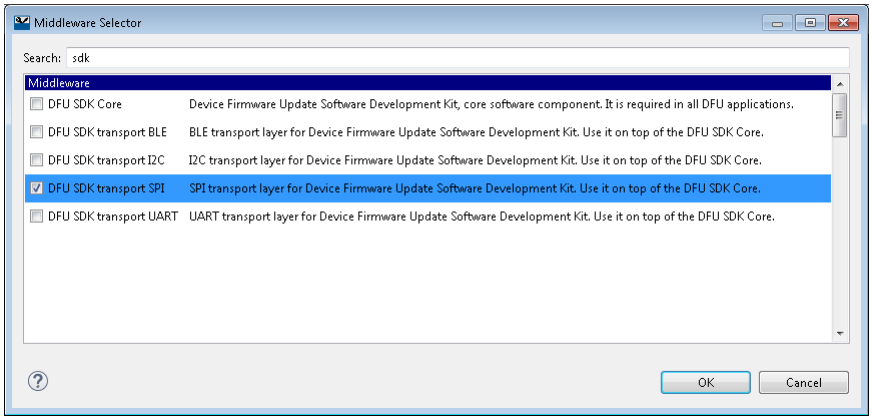
Search

The Search box helps you find various components in the dialog by filtering the matching string. This follows the Java pattern matching syntax (https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html). For example, a period (.) will show everything.



Enable/Disable Middleware Component

Click the check box next to the desired middleware component to enable it or disable it as applicable.



When you click on a middleware component, it is highlighted.

- If there are any dependencies, they will be highlighted in a lighter color.
- If an enabled component has dependencies, all dependencies will also be enabled.
- Disabling a component will **not** disable dependent components. You must disable dependent components one at a time.

Click **OK** to close the dialog and update your application.

Rename Application

Due to various project dependencies, you cannot use the Eclipse Rename function for ModusToolbox 1.1 applications. If you want to rename an application, you must create a new application with the desired name. Then, copy files from the old application to the new one.

If you attempt to use the Eclipse Rename dialog, a message will display when you click **OK**.

7. Build Applications

This chapter covers various aspects of building applications. Building applications is not specifically required, because building is performed as part of the programming and debugging process. However, if you are running the ModusToolbox IDE without any hardware attached, you may wish to build your application to ensure all the code is correct. If you changed code in one of your projects, you may wish to build just that project.

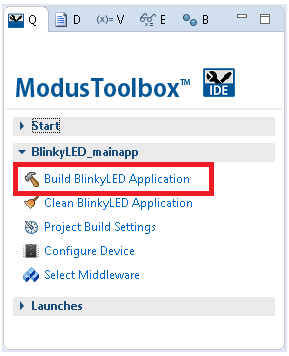
- Build with Make
- Build with IDE
- Generated ELF Files
- Enable HEX File Generation

Build with Make

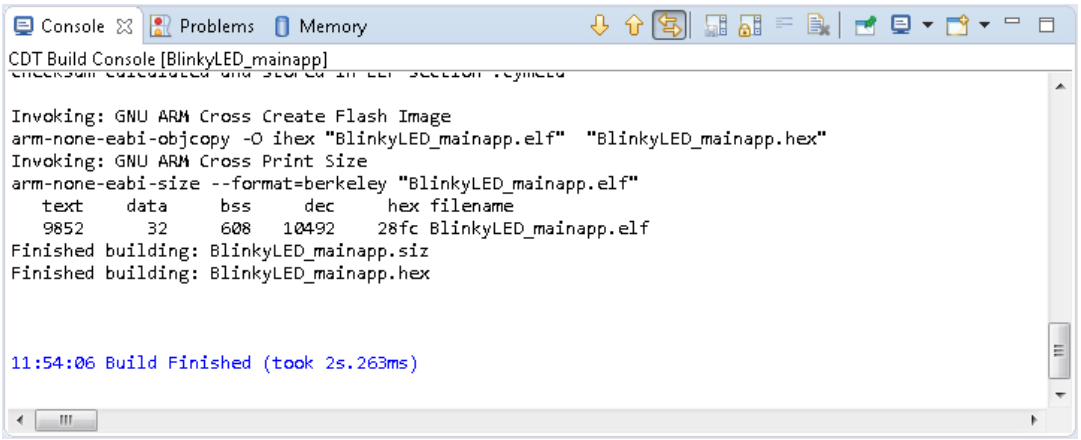
You can build applications from the command line using your preferred version of Make. Refer to the *Running ModusToolbox from the Command Line* document located in the *<install>/docs directory* for more information.

Build with IDE

After loading an application, it is best to first build everything to generate the necessary files. Click on an application's "mainapp" project in the Project Explorer. Then in the **Quick Panel**, click the "Build <app-name> Application" link.



Messages display in the Console, indicating whether the build was successful or not.



Note Be aware that there are several Console views available.

You can access the different views using the **Display Selected Console** button (or the pull-down arrow). The Global Build Console is the only way to see all of the results in one place. If you just have the standard Console open, it resets every time a new project in the application starts building. You won't see any errors if they are not on the final project that gets built.

For subsequent updates, you can build one or more projects using the right-click menu options. Any projects that are dependent on the project being built will also be built. The ModusToolbox IDE supports all the usual application and build options available for the native Eclipse IDE.

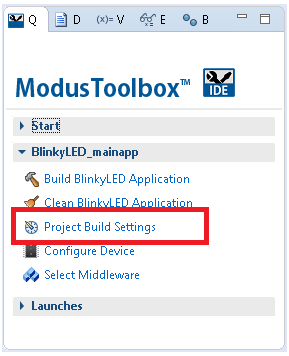
Note When the active build configuration is changed it affects only the selected project. The active build configuration for any dependent projects is specified separately for each project. Therefore, if you want to use the same configuration for all projects (for example, Debug or Release), it must be set for each project. It is possible to select multiple projects at once from the Project Explorer and then select the active configuration.

Build Settings

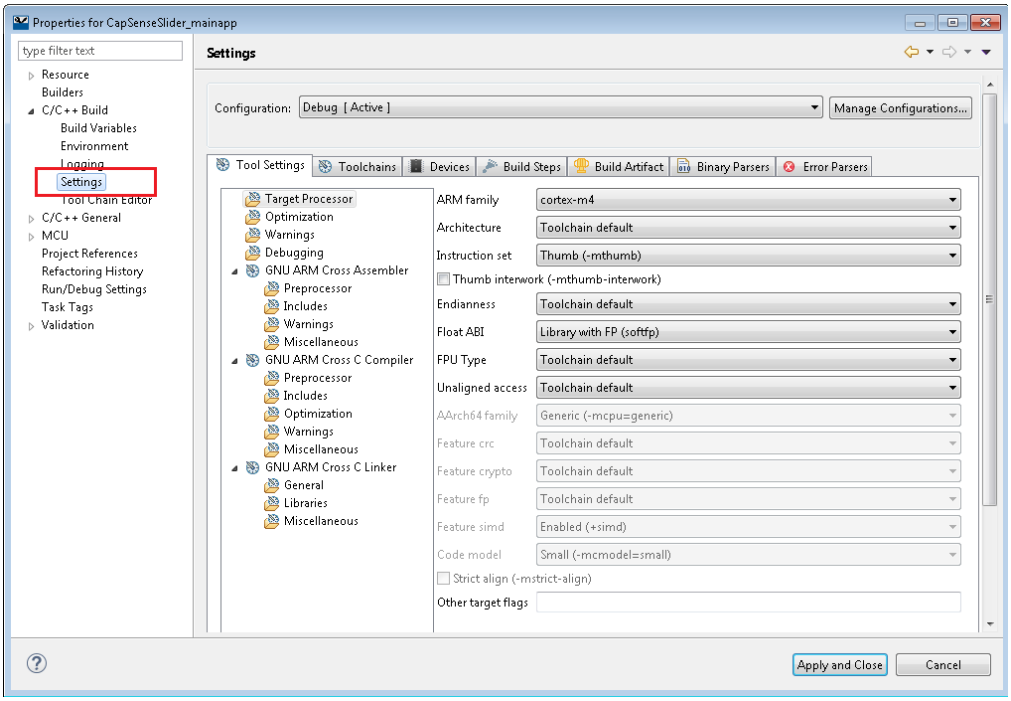
The ModusToolbox IDE uses standard Eclipse CDT build settings for things such as:

- Custom defines
- Include paths
- General options to the compiler/linker

To change build settings, first select the appropriate project and then click the “Project Build Settings” link in the **Quick Panel**. Note that build settings are project specific, NOT application specific. So, they must be set in the project or projects in which they are to be used.



This link opens the Eclipse Properties dialog on the **C/C++ Build > Settings** page. This dialog has many pages with a lot of settings. The following sections show a few that you may wish to use for your application and environment.



Note Any changes made to C/C++ build settings in the IDE are not written back to external Makefiles, such as *modus.mk*.

The Tool Settings tab contains the following settings:

- Target Processor – This section allows for configuring settings related to the target device and what optional hardware features it provides.
- Optimization – This section allows for configuring optimization options that apply to both the assembler and compiler.
- Warnings – Provides controls for configuring common warning options for the assembler and compiler.
- Debugging – Provides controls for configuring common debug related information for the assembler and compiler.
- GNU ARM Cross Assembler – Provides a summary of the assembler command line options that were configured in any of the other tabs.
  - Preprocessor – Provides options for setting up additional command line specified preprocessor defines and undefines for the assembler.
  - Includes – Provides options for setting up include paths for the assembler.
  - Warnings – Provides general input for specifying command line arguments for controlling how different warnings are handled for the assembler.
  - Miscellaneous – Provides general input for any additional assembler command line options to specify.
- GNU ARM Cross C Compiler – Provides a summary of the compiler command line options that were configured in any of the other pages.
  - Preprocessor - Provides options for setting up additional command line specified preprocessor defines and undefines for the compiler.
  - Includes – Provides options for setting up include paths for the compiler.
  - Optimization – This allows for compiler specific optimization settings which where not specified in the top level Optimization section.
  - Warnings – Provides general input for specifying command line arguments for controlling how different warnings are handled for the compiler
  - Miscellaneous – Provides general input for any additional assembler command line options.
- GNU ARM Cross C Linker – Provides a summary of the linker command line options that were configured in any of the other pages.
  - General – Provides options for specifying basic linker command line options.
  - Libraries – Provides options for specifying what libraries to link against and where those libraries are located.
  - Miscellaneous – Provides options for specifying additional linker command line arguments.

For details about what options are available, and what they do, refer to the GCC option summary:

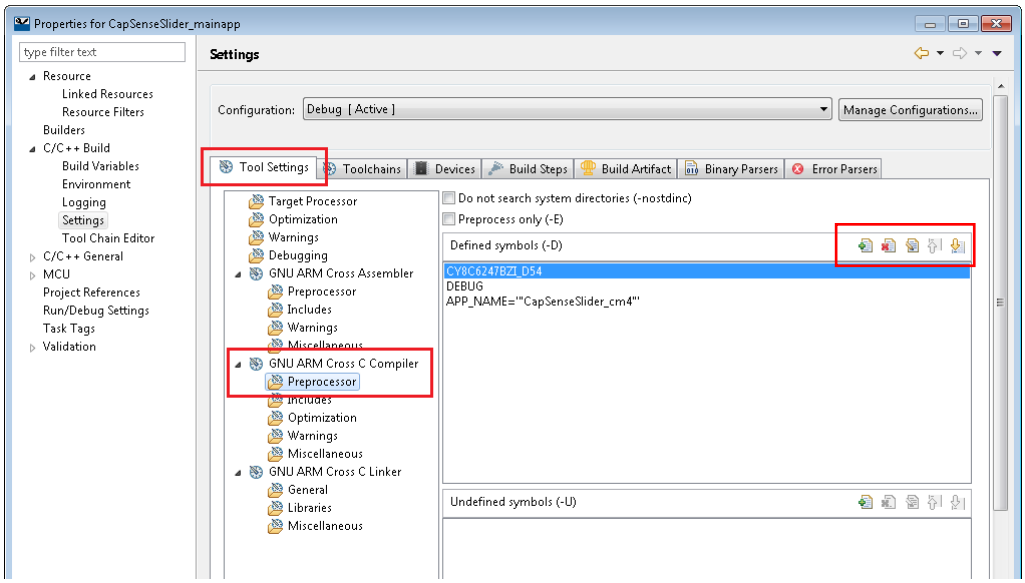
https://gcc.gnu.org/onlinedocs/gcc-7.2.0/gcc/Option-Summary.html

More details about the Eclipse build settings can be found here:

https://help.eclipse.org/oxygen/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Freference%2Fcdt_u_properties.htm

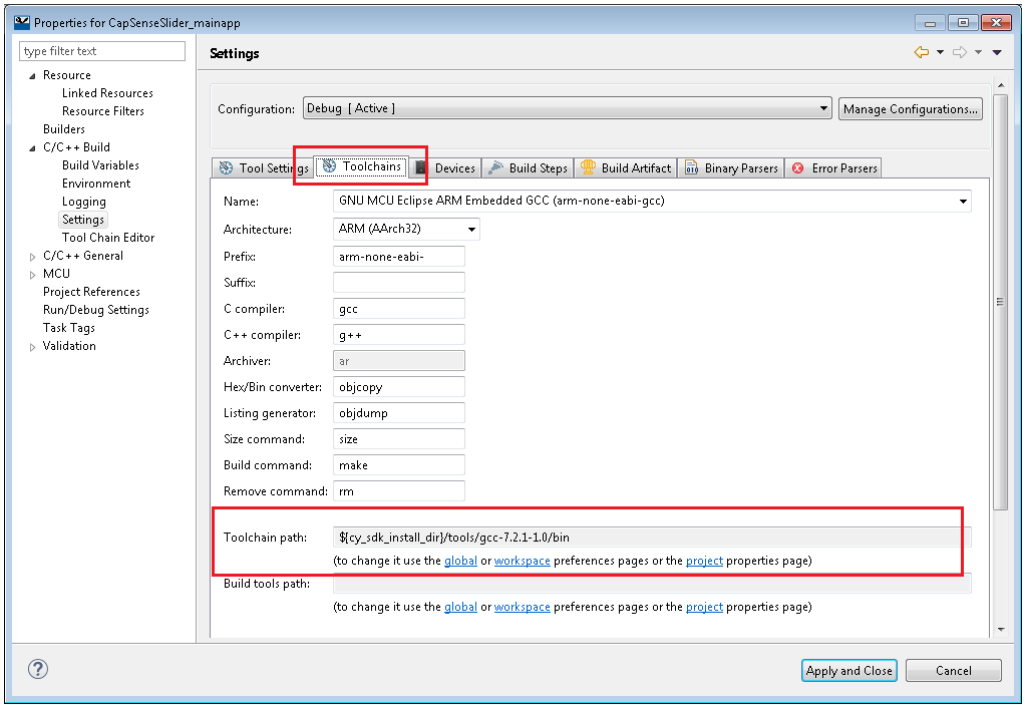
Macros

To define a macro, click the **Tool Settings** tab, and select **GNU ARM Cross C Compiler > Preprocessor**. Use the icons to add, delete, edit, or reorder macros.



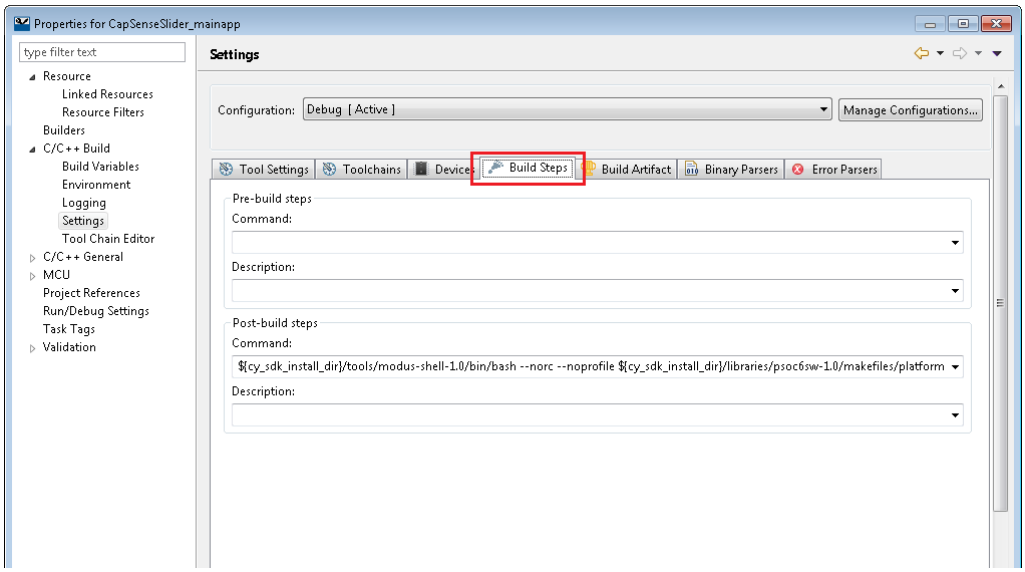
GCC Version

ModusToolbox software includes GCC version 7.2.1 as the preferred toolchain to use with the ModusToolbox IDE. If you have a different version of GCC you prefer, update the project properties to point to the appropriate toolchain folder. You must do this for all projects in an application. Click the **Toolchains** tab, and then click the appropriate link to update global or workspace preferences, or project properties.



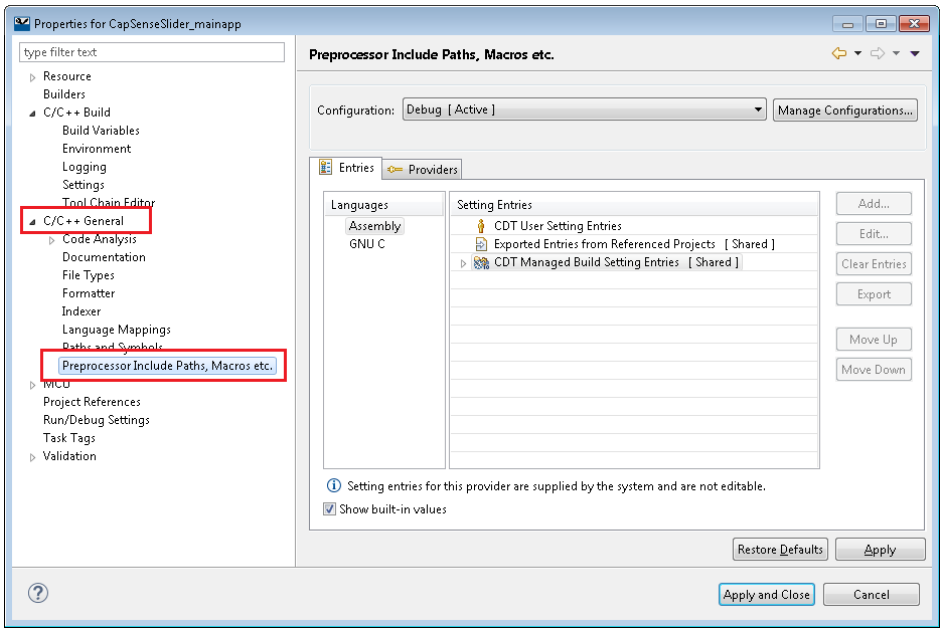
Pre-Build/Post-Build Commands

Click the **Build Steps** tab to access pre-build and post-build commands.



Include Paths

To add/update include paths, go to **C/C++ General > Processor Include Paths …**

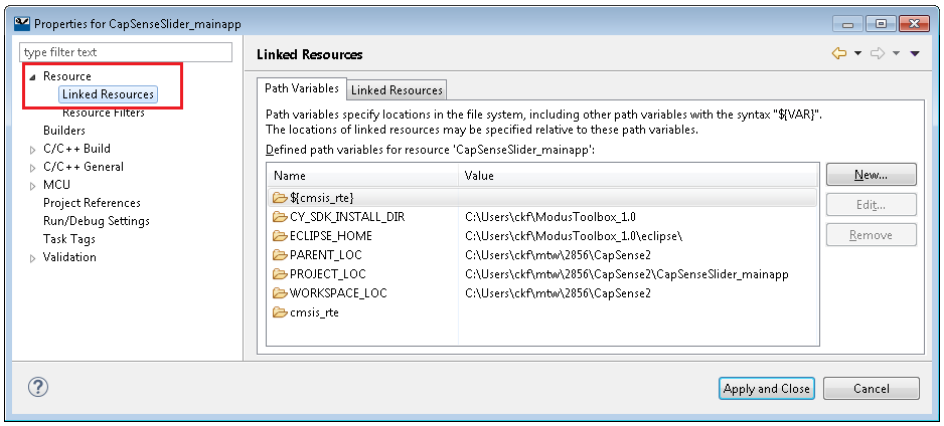


Path/Build Variables

As shown in previous screen captures, there are path variables used in various settings. Some are defined by the underlying Eclipse framework, others are added for use with the ModusToolbox IDE. The following sections show where these are defined.

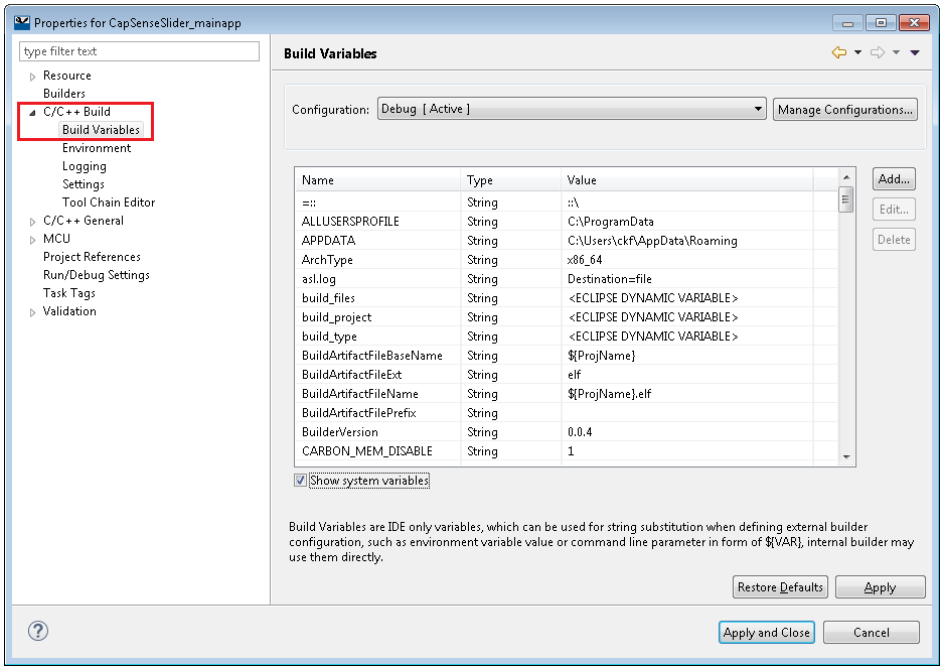
Eclipse Path Variables

These are defined on the **Resource > Linked Resources** Properties page.



ModusToolbox IDE Build Variables

These are defined on the **C/C++ Build > Build Variables** page.



Generated ELF Files

By default, the IDE only generates ELF files as part of the build. To enable generating HEX files, see Enable HEX File Generation. Each type of application generates ELF files as follows:

PSoC 6 Devices

For PSoC 6 MCU devices, the following ELF files are generated. After a build, the ELF files are located in <app-name_mainapp>/Debug/

- <app-name>_mainapp_final.elf: This is used by the Run / Debug Configurations in the IDE
- <app-name>_mainapp_signed.elf: This is used by the command-line interface (CLI) to program the image via OpenOCD.
- <app-name>_mainapp.elf: This is a standard ELF file created by the linkers in the PSoC 6 MCU build process.

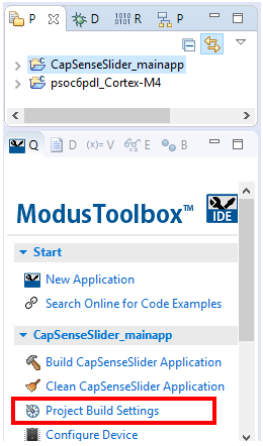
CYW20819 Bluetooth Devices

For CYW20819 Bluetooth devices, only the <app-name>_mainapp.elf file is generated.

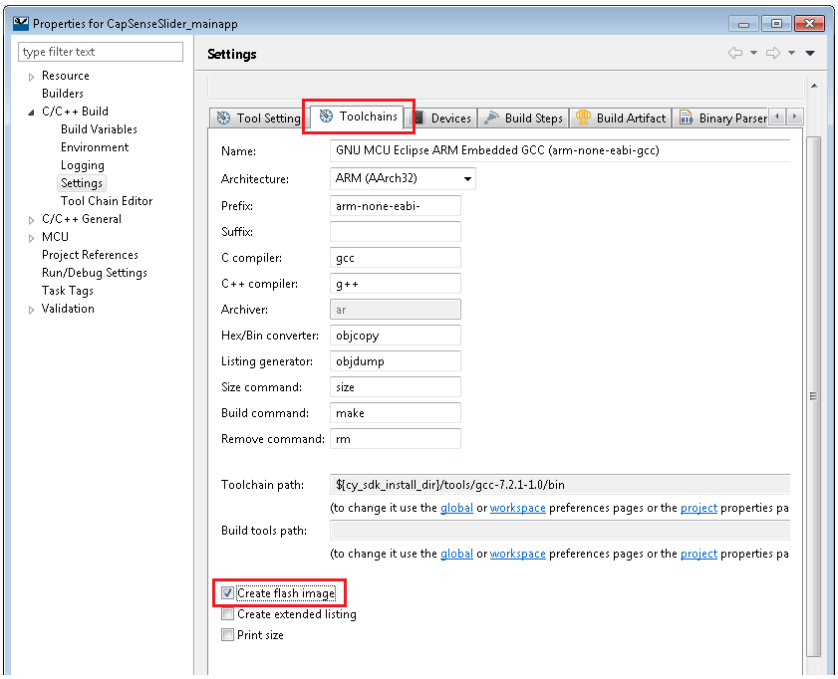
Enable HEX File Generation (PSoC 6)

Follow these instructions to enable HEX file generation in the ModusToolbox IDE for a PSoC 6 application. These steps include using the final ELF file as an input. This is not applicable to CYW20819 Bluetooth devices.

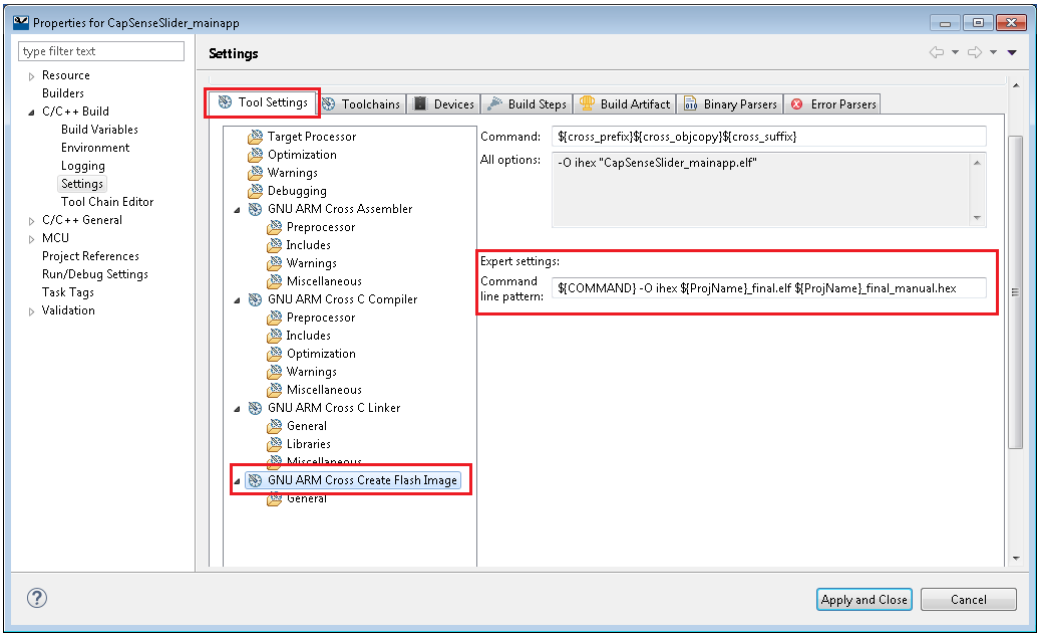
1. Click on the <app-name>_mainapp project and select the “Build Settings: link in the the **Quick Panel**.



2. On the Properties dialog, select the **Toolchains** tab on the **C/C++ Build > Settings** page, select the **Create flash image** check box.



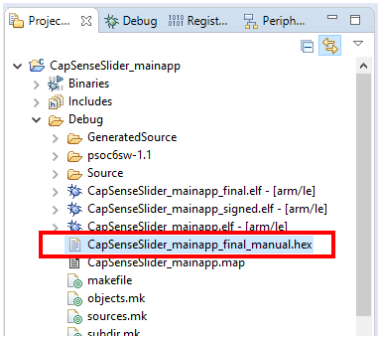
3. Select the **Tool Settings** tab Then, select **GNU ARM Cross Create Flash Image** in the tree and edit the **Expert Settings, Command line pattern** field.



   a. Change the existing pattern to the following:
   ```
   ${COMMAND} -O ihex ${ProjName}_final.elf ${ProjName}_final_manual.hex
   ```

   b. Click **Apply and Close**.

4. Build the application. When complete, expand the **Debug** folder and observe that the specified HEX file was generated.



8. Program and Debug

Programming and debugging is native to your chosen development environment. Cypress devices are supported in the major program and development solutions. Primarily, this means J-Link and OpenOCD. These solutions provide for programming flash within a device and provide a GDB server for debugging.

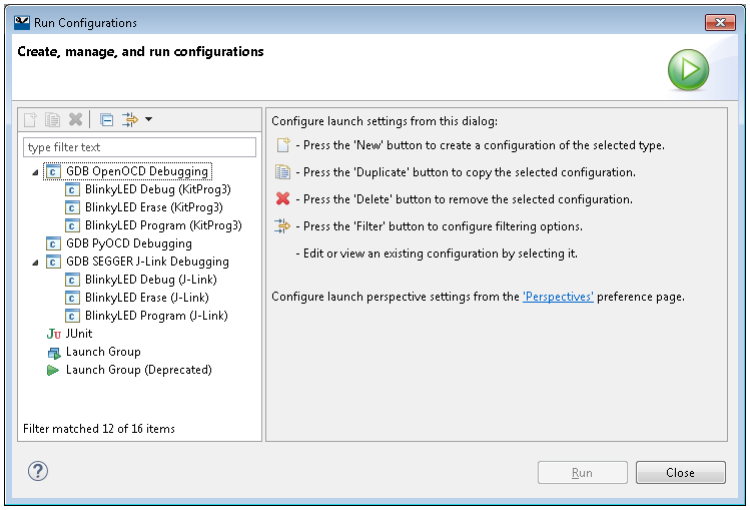
This chapter covers various topics related to building and debugging using the ModusToolbox IDE for PSoC 6 devices and CyW20819 Bluetooth devices.

PSoC 6 Programming/Debugging

Launch Configurations

The flow for programming and debugging is similar for all devices. The ModusToolbox IDE contains several Launch Configurations that control various settings for programming the devices and launching the debugger. Depending on the kit and type of applications you are using, there are various Launch Configurations available.

There are two sets of configurations: one for KitProg3 (included on-board on most Cypress PSoC 6 based kits) and another for J-Link. These are shown in the Run/Debug Configurations dialog, similar to the following.



You can open these dialogs from the **Run** menu or by selecting the down arrow next to the **Run** and **Debug** commands.

These configurations include the application name and protocol, for example:

CapSenseSlider Program (KitProg3)

CapSenseSlider Debug (J-Link)

Note KitProg3 configurations may not work if a J-Link probe is attached to the kit.

When an application is created, the tool generates the following launch configurations for both KitProg3 and J-Link. Some items display in the **Quick Panel**, and some are in the Run/Debug Configurations dialog only.

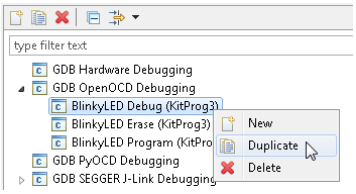
- **Debug:** This launch configuration builds the entire application on both cores, programs all the device's memories, and then starts a Cortex-M4 debugging session.
- **Erase:** This launch configuration erases all internal memories.
- **Program:** This launch configuration builds the entire application on both cores, programs all the device's memories, and then runs the program.

Attach to Running PSoC 6 Target

There are some cases where you may wish to attach to a running PSoC 6 target as part of a debug session. This is accomplished by copying and modifying existing debug configurations.

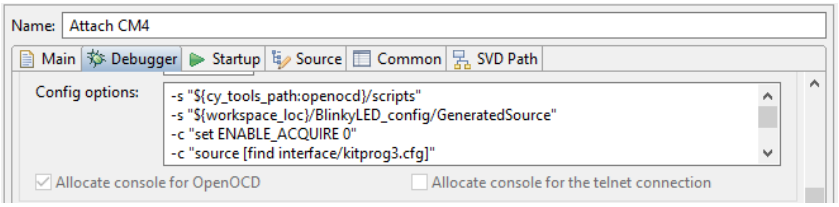
KitProg3 over OpenOCD

1. Open the Debug Configurations dialog, and duplicate the appropriate debug configuration for the application; for example, "BlinkyLED Debug (KitProg3)."



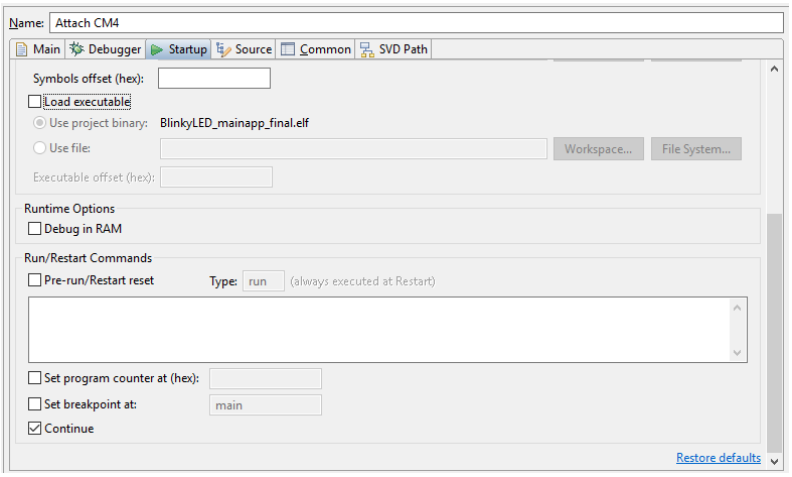
2. Rename the new Debug configuration to something meaningful, such as "Attach CM4."

3. Go to the **Debugger** tab.



   a. In the **Config options** field, insert the following as the first command:
   `-c "set ENABLE_ACQUIRE 0"`

4. Go to the **Startup** tab.



   a. Unselect **Initial Reset**.

   b. Unselect **Load executable**.

   c. Unselect **Pre-run/Restart reset**.

   d. Delete everything in the text field under the **Pre-run/Restart reset** check box.

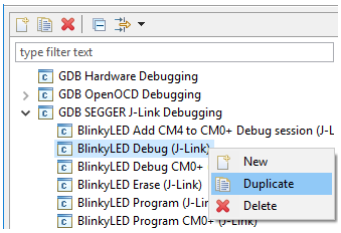
e. Unselect **Set breakpoint at**.

f. Select **Continue**.

5. Program your device, and attempt to attach to the running target using the new launch configuration.

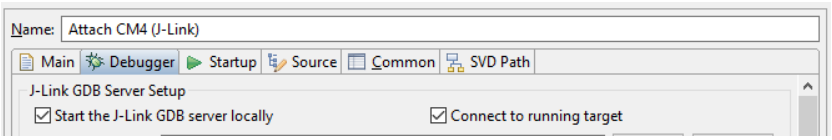
J-Link

6. Duplicate the appropriate debug configuration for the target core; for example, "Debug (J-Link)."



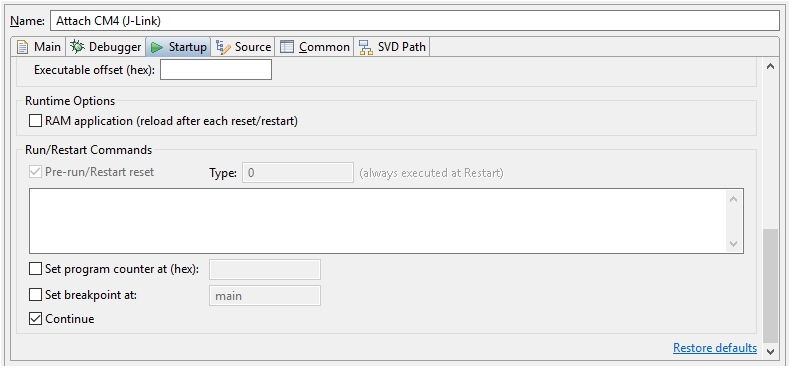
7. Rename the new Debug configuration to something meaningful, such as "Attach (J-Link)."

8. Go to the **Debugger** tab.



a. Select **Connect to running target**.

9. Go to the **Startup** tab.



a. Delete everything in the text field under the **Pre-run/Restart reset** check box.

b. Unselect **Set breakpoint at**.

c. Select **Continue**.

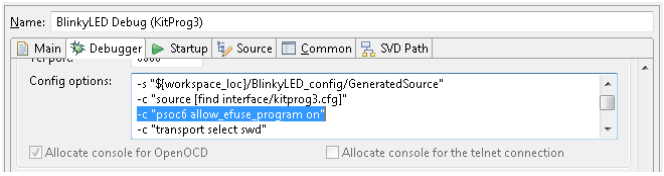
10. Program your device, and attempt to attach to the running target using the new launch configuration.

Programming eFuse

PSoC 6 MCUs contain electronic fuses (eFuses), which are one-time programmable. They store device specific information, protection settings and customer data.

By default eFuses are not programmed even if they are present in the programming file. To enable eFuse programming, add the following command for the appropriate Debug Configuration, under the **Debugger** tab in the **Config options** field (after `-c "source [find target/psoc6.cfg]"`):

```
-c "psoc6 allow_efuse_program on"
```



Debug Connection Options

By default, a typical PSoC 6 application created in the ModusToolbox IDE includes launch configurations set up for two probes: Cypress KitProg3 (built into Cypress kits) and Segger J-Link.

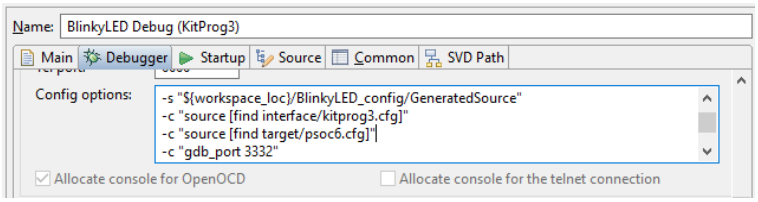
Communication Firmware	Debug Connection	More Information
Cypress-provided KitProg3	OpenOCD via CMSIS-DAP	https://www.cypress.com/products/psoc-programming-solutions
SEGGER-provided J-Link DLL	SEGGER J-Link	SEGGER J-Link

Select Specific CMSIS-DAP Device

If there are two or more CMSIS-DAP devices connected to your computer, the first detected device will be used by default. KitProg3 supports both CMSIS-DAP modes – HID and BULK. BULK devices are selected first, then HID devices. You can specify a CMSIS-DAP device by:

- VID and PID of the USB device
- Serial Number of the USB device
- VID, PID, and Serial Number of the USB device

To do this, you must add a specific command for the appropriate Debug Configuration, under the **Debugger** tab in the **Config options** field (after -c "source [find interface/kitprog3.cfg]")



Selecting by VID and PID

Use OS-specific tools to determine the VID and PID of connected devices. For example, on Windows, use the Device Manager. Use the “cmsis_dap_vid_pid” command to select a CMSIS-DAP device with a specific VID and PID. If there are two or more devices with the same specified VID/PID pair, OpenOCD uses the first detected device from the passed list.

- To specify KitProg3 in CMSIS-DAP BULK mode with VID = 0x04B4 and PID = 0xF155:

```
cmsis_dap_vid_pid 0x04B4 0xF155
```

- To specify KitProg3 in CMSIS-DAP HID mode with VID = 0x04B4 and PID = 0xF154:

```
cmsis_dap_vid_pid 0x04B4 0xF154
```

- To specify any (HID or BULK) connected KitProg3 device:

```
cmsis_dap_vid_pid 0x04B4 0xF154 0x04B4 0xF155
```

Selecting by Serial Number

There should not be more than one device with the same serial number. Use this method if you want to use only one specific device. Use OS-specific tools to determine the Serial Number of connected devices. You can also use the fw-loader utility with `--device-list` option. See KitProg Firmware Loader.

Use the “cmsis_dap_serial” command to select a CMSIS-DAP device with a specific Serial Number.

- To specify a CMSIS-DAP device with Serial Number = 0B0B0F9701047400 the following command is used:

```
cmsis_dap_serial 0B0B0F9701047400
```

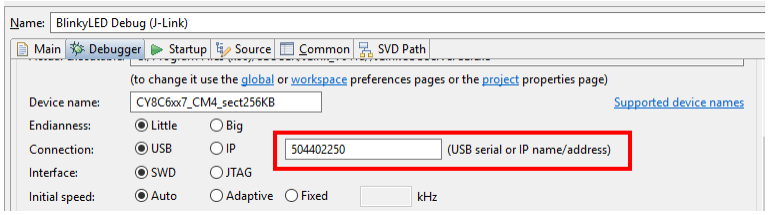
Selecting by both VID/PID and Serial Number

You can use both commands together in any order. For example:

```
cmsis_dap_vid_pid 04B4 F155 cmsis_dap_serial 0B0B0F9701047400
```

Select Specific J-Link Device

If there are two or more J-Link devices connected to your computer, the first detected device is used by default. You can select a specific J-link device by setting the serial number for the appropriate Debug Configuration, under the **Debugger** tab:



KitProg Firmware Loader

The PSoC 6 MCU kits include onboard programmer/debug firmware, called KitProg. The CY8CPROTO-062-4343W kit has KitProg3 by default. However, the CY8CKIT-062-BLE and CY8CKIT-062-WIFI-BT kits come with KitProg2 firmware installed, which does not work with the ModusToolbox software. **You must update to KitProg3**. KitProg3 provides the CMSIS-DAP (HID) protocol and the CMSIS-DAP (Bulk) protocol, which is up to ~2.5 times faster. Both modes can be used via OpenOCD.

ModusToolbox software includes a command-line tool “fw-loader” to update Cypress kits and switch the KitProg firmware from KitProg2 to KitProg3, and back. The following is the default installation directory of the tool on Windows:

<user_home>\ModusToolbox_<version>\tools\fw-loader-<version>\bin

For other operating systems, the installation directory will vary, based on where the software was installed.

Use the fw-loader tool to update the KitProg firmware as required for your needs. KitProg2 does not work with the ModusToolbox software. Likewise, if you update to KitProg3, PSoC Creator won’t work with Cypress kits until you restore KitProg2.

Note On a Linux machine, you must run the *udev_rules\install_rules.sh* script before the first run of the fw-loader.

Command-Line Options

Run the tool from the command line:

```
[install_path]>fw-loader
```

Use the following options, as needed:

- `--help` or `/?` (or no arguments) – Display a list of supported commands with their descriptions.
- `--device-list` – Display a list of connected devices.
- `--update-kp3 [device-name]` – Update the FW of the specified device name to KitProg3.
- `--update-kp2 [device-name]` – Updates the FW of the specified device name to KitProg2.

Display List of Devices

Use the following command to see if the device is recognized. For KitProg2 devices, use the **SW3 Mode Switch** button on the kit to enable Proprietary mode. Otherwise, the fw-loader tool will not recognize the device. See Mode Switch. If you have more than one KitProg attached, use the following command to display devices by name and identifier:

```
[install_path]>fw-loader --device-list
Cypress Firmware Updater, Version: 2.1.0.31
(C) Copyright 2019 by Cypress Semiconductor
All Rights Reserved

Info: Start API initialization
Info: Connected - KitProg3 CMSIS-DAP BULK-1718126C03227400
Info: Connected - KitProg3 CMSIS-DAP HID-181B025303147400
Info: Hardware initialization complete (899 ms)
Connected supported devices:
        1: KitProg3 CMSIS-DAP BULK-1718126C03227400     FW Version 1.1.157
        2: KitProg3 CMSIS-DAP HID-181B025303147400      FW Version 1.10.1289
```

Update to KitProg3

Use the following command to update to KitProg3. If you have more than one KitProg device attached, you must use the `[device-name]` option to specify the KitProg to update. If you have only one KitProg device attached, you can omit the `[device-name]` option.

```
[install_path]>tools>fw-loader-1.0>bin>fw-loader --update-kp3 "KitProg3 CMSIS-DAP BULK-
1718126C03227400"

Cypress Firmware Updater, Version: 2.1.0.31
(C) Copyright 2019 by Cypress Semiconductor
All Rights Reserved

Info: Start API initialization
Info: Connected - KitProg3 CMSIS-DAP BULK-1718126C03227400
Info: Connected - KitProg3 CMSIS-DAP HID-181B025303147400
Info: Hardware initialization complete (899 ms)
Device 'KitProg3 CMSIS-DAP BULK-1718126C03227400' opened successfully
Info: Kit FW is 'KitProg3' ver. 1.01 b157.  Upgrade file is 'KitProg2' ver. 1.05
 b000.
Info: Disconnected - KitProg3 CMSIS-DAP BULK-1718126C03227400
Info: Connected - KitProg Bootloader-1718126C03227400
Info: Bootloader Version: Major 1, Minor 1, Build 40
Info: FW Upgrade to version: 1.05 b000
Info: Bootloading of KitProg FW...
Info: Verifying of KitProg FW...
Info: FW Upgrade completed
```

```
Info: Disconnected - KitProg Bootloader-1718126C03227400
Info: Connected - KitProg2-1718126C03227400
FW update completed successfully
```

Switch Back to KitProg2

Use the following command to switch back to KitProg2. If you have more than one device attached, you must use the `[device-name]` option to specify the KitProg to update. If you have only one KitProg device attached, you can omit the `[device-name]` option.

```
[install_path]>fw-loader --update-kp2 "KitProg3 CMSIS-DAP HID-181B025303147400"

Cypress Firmware Updater, Version: 2.1.0.31
(C) Copyright 2019 by Cypress Semiconductor
All Rights Reserved

Info: Start API initialization
Info: Connected - KitProg2-1718126C03227400
Info: Connected - KitProg3 CMSIS-DAP HID-181B025303147400
Info: Hardware initialization complete (888 ms)
Device 'KitProg3 CMSIS-DAP HID-181B025303147400' opened successfully
Info: Kit FW is 'KitProg3' ver. 1.10 b1289.  Upgrade file is 'KitProg2' ver. 1.0
5 b000.
Info: Disconnected - KitProg3 CMSIS-DAP HID-181B025303147400
Info: Connected - KitProg Bootloader-181B025303147400
Info: Bootloader Version: Major 1, Minor 1, Build 40
Info: FW Upgrade to version: 1.05 b000
Info: Bootloading of KitProg FW...
Info: Verifying of KitProg FW...
Info: FW Upgrade completed
Info: Disconnected - KitProg Bootloader-181B025303147400
Warning: Wait for upgraded device is timed out
FW update completed successfully
```

Mode Switch

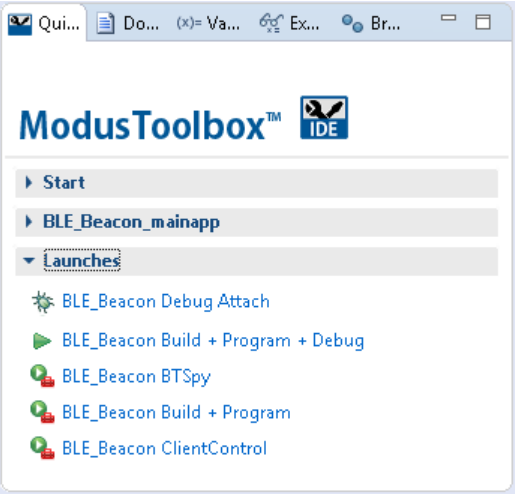
The kits have a set of LEDs and a **SW3 Mode Switch** button. To switch KitProg modes, push and release the **SW3 Mode Select** button on the kit. LED2 indicates the status, as follows:

Firmware	Switches Between	Status of LED2
KitProg2	Proprietary and CMSIS-DAP HID	On = Proprietary Off = CMSIS-DAP HID
KitProg3	CMSIS-DAP HID and CMSIS-DAP Bulk	On = CMSIS-DAP Bulk Breathing (1 Hz) = CMSIS-DAP HID

Bluetooth Programming/Debugging

CYW20819 Bluetooth applications have different launch configurations than PSoC 6 MCU applications. Bluetooth applications use External Tools configurations for programming the device, and Debug configurations for launching the debugger. The Program launch configurations are available from the Quick Panel when you select <app name>_mainapp project. The Debug Configurations are visible in the Quick Panel only after you set ENABLE_DEBUG to 1 in the Change Project Settings dialog. Each configuration is preceded by the application name.

Note For Bluetooth applications, if you use the Quick Panel Program or Debug options, the IDE builds the application. However, if you go through the IDE menu to use a launch configuration, the IDE will not build the application.



Program Configuration

This launch configuration runs a script and programs the device's memories.

- **<app-name> Build + Program:** Build + Program is used to build and program embedded applications onto the target device.

There are also additional configurations shown here for Bluetooth functionality. These configurations just launch the specified tool in a separate window:

- **<app-name> BTSpy:** BTSpy is a host app that can be used in conjunction with ClientControl to receive, decode and display protocol application and stack trace messages.
- **<app-name> Client Control:** ClientControl is a host app that can communicate with the embedded app to perform various functionalities, tests, exercising features, etc.

Debug Configurations

Make sure to enable debug to see the debug configurations. The configurations include:

- **<app-name> Debug Attach:** Debug Attach is used to start the debugger without first programming the device.
- **<app-name> Build + Program + Debug:** This configuration builds and downloads the application to the board and starts debug mode.

Debug Settings

CYW20819 Bluetooth devices use the JTAG SWD interface on a kit for debug via OpenOCD or J-Link probe. Typically, GPIOs need to be reprogrammed (via firmware) to enable SWD, so debug can't be performed directly after a device reset. The debugger is usually attached to a running device and symbols only are loaded.

The CYW920819EVB-02 and CYBT-213043-MESH kits have additional requirements in order to launch the ModusToolbox IDE debugger. In general, most debugging for these kits is done with logs and sniffers, since real time execution is usually needed for the protocols to run properly.

Refer to the *WICED Hardware Debugging* document (002-20504) for more details.

Mbed OS Programming/Debugging

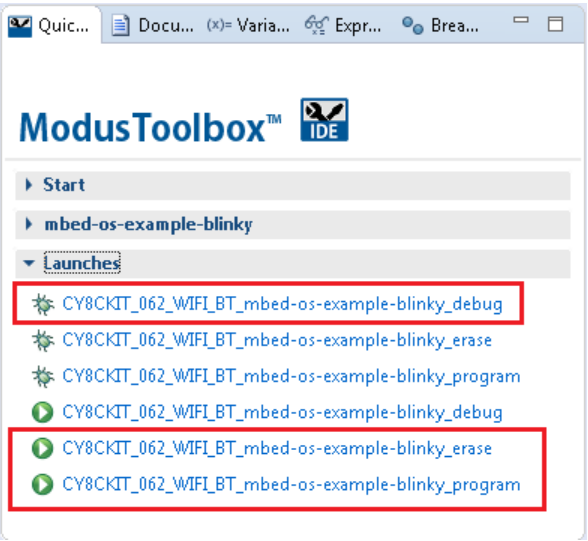
Launch Configurations

When an application is imported from Mbed OS into the ModusToolbox IDE, the tool generates the following launch configurations. There are three debug configurations and three program configurations:

<target>_<app-name>_debug – Programs the device as needed and launches the debugger.

<target>_<app-name>_erase – Erases the device.

<target>_<app-name>_program – Programs the device.



Note The three debug launches will attempt to switch to the debug perspective, while the three program launches will not. Cypress recommends using the debug launch for “debug” and the program launches for “erase” and “program.”

Change PyOCD Message Color

Messages in the console for pyOCD display in red because they are written to stderr. The Eclipse preference for stderr defaults to red. If you want to change the color of stderr:

1. Open the Preferences dialog and select **Run/Debug > Console**.
2. Click on the option for **Standard Error text color**, select the desired color, and click **OK**.
3. Then, click **Apply and Close** to close the Preferences dialog.

Enable Peripherals View

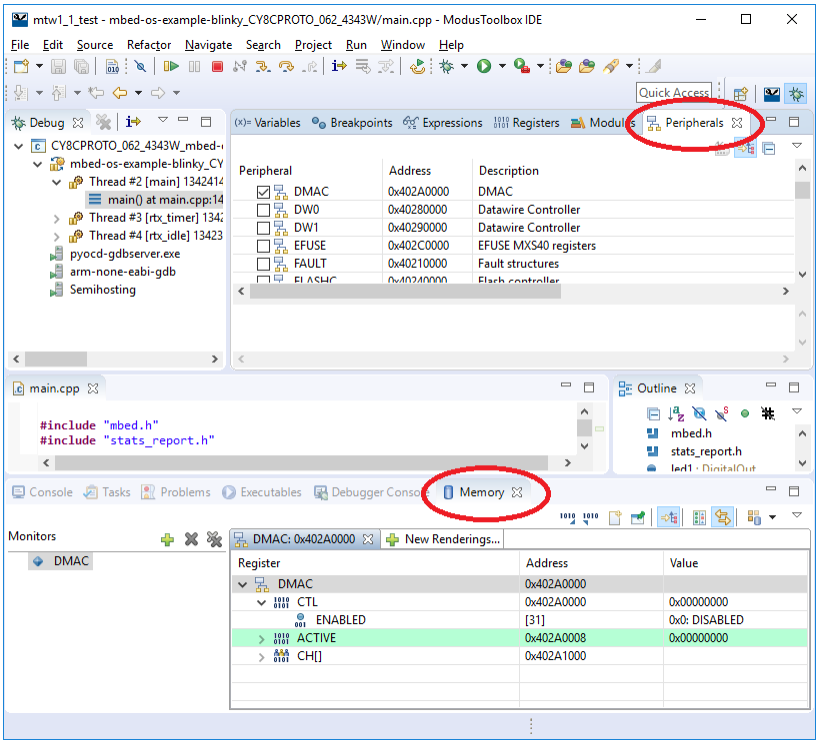
For applications imported from Mbed OS, the **Peripherals** view does not contain any peripherals because you must provide the path to the device hardware description .SVD file. To do this:

1. Open the Debug Configurations dialog and select the appropriate pyOCD debug launch configuration.
2. Select the **SVD Path** tab and provide path to the .SVD file. The following .SVD files included with ModusToolbox are available under the following paths:

   PSoC6ABLE2: <install_dir>/libraries/udd-1.1/udd/devices/MXS40/PSoC6ABLE2/studio/svd/psoc6_01.svd

   PSoC6A2M : Install_dir>/libraries/udd-1.1/udd/devices/MXS40/PSoC6A2M/studio/svd/psoc6_02.svd

3. Click **Apply** and then click **Debug**. When the debugger halts, notice that the **Peripherals** view contains various peripherals to select, depending on the application.

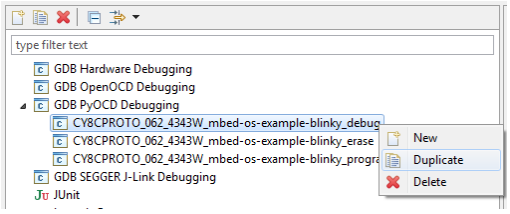


Attach to Running Target (PyOCD)

There are some cases where you may wish to attach to a running PSoC 6 target as part of a debug session. This is accomplished by copying and modifying existing debug configurations.

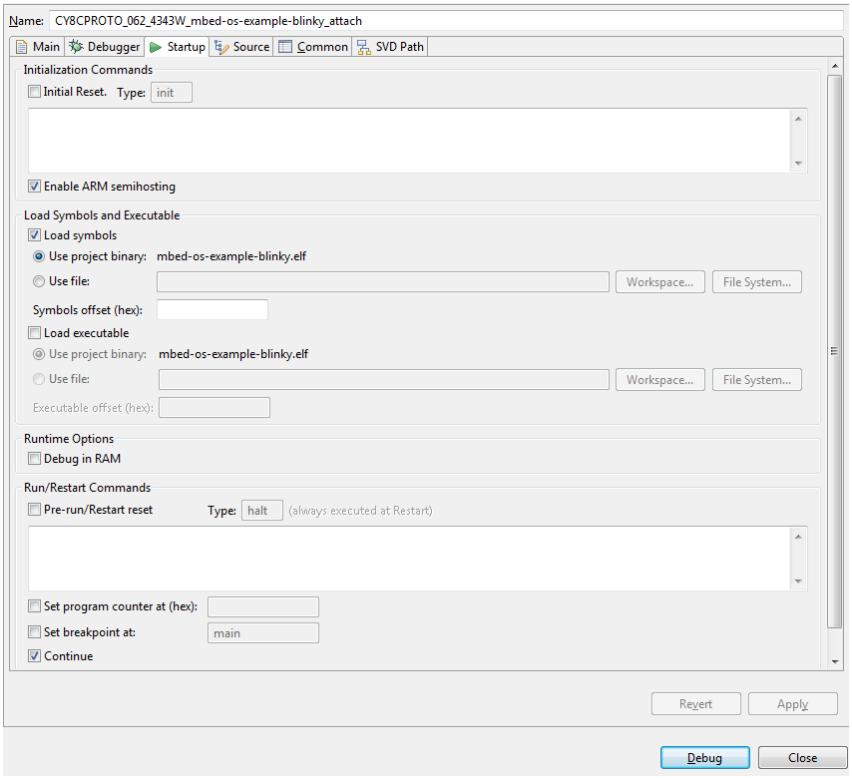
1. See Import Mbed OS Applications for instructions to import an example Mbed project into the ModusToolbox IDE workspace.

2. Open the Debug Configurations dialog, and duplicate the Debug configuration; for example, "CY8CPROTO_062_4343W_mbed-os-example-blinky_debug."



3. Rename the new Debug configuration to something meaningful, such as "CY8CPROTO_062_4343W_mbed-os-example-blinky_attach."

4. Go to the **Startup** tab.



   a. Unselect **Initial Reset**.

   b. Unselect **Load executable**.

   c. Unselect **Pre-run/Restart reset**.

   d. Unselect **Set breakpoint at**.

   e. Select **Continue**.

5. Ensure your KitProg3 is in DAPLink mode and your target is running. See KitProg Firmware Loader.

6. Click **Debug** to attach to the running target using the new launch configuration.