

Chapter 1: Tour of ModusToolbox™

After completing Chapter 1 (this chapter) you will understand a top-level view of the components of the ModusToolbox™ Bluetooth ecosystem including the chips, modules, software, documentation, support infrastructure and development kits. You will have ModusToolbox™ installed and working on your computer and will understand how to program an existing project into a kit.

Note: This chapter was written for ModusToolbox 1.1. It has been updated; but NOT verified for ModusToolbox 2.0. Some major changes to the structure and intent of ModusToolbox were implemented in ModusToolbox 2.0 that position it more as a guide and coordinator than a required tool. ModusToolbox 2.0 is described:

ModusToolbox 2.x is a set of Reference Flows, Products and SDKs that enable an immersive development experience for customers creating converged MCU and Wireless systems. ModusToolbox 2.x leverages popular third-party networking solutions such as Mbed OS, Amazon FreeRTOS, AliOS Things and Zephyr

Updates to this section will be forthcoming. Please check back with your Workshop host to receive a current version.

TABLE OF CONTENTS

1.1 TOUR OF MODUSTOOLBOX IDE	3
1.1-1 FIRST LOOK	3
1.1-2 CUSTOMIZATION	5
1.1-3 SDK UPDATE	5
1.1-4 NEW APPLICATION WIZARD.....	5
1.1-5 QUICK PANEL	8
1.1-6 APPLICATIONS AND PROJECTS.....	8
1.1-6a Project Explorer.....	9
1.1-6b Readme	9
1.1-6c Configuration database file – design.modus.....	10
1.1-6d GeneratedSource Folder	11
1.1-6e Command-line makefile – modus.mk.....	11
1.1-6f Platform Files (wiced_platform.*)	11
1.1-6g Stack Configuration Files.....	12
1.1-6h Middleware	12

1.1-6i Application C file	13
1.1-6j Application Settings	16
1.1-6k Project Build Settings	17
1.1-7 SHARING APPLICATIONS.....	18
1.1-7a Export to Archive.....	18
1.1-7b Import from Archive.....	19
1.1-7c Import from Filesystem	21
1.2 CODE EXAMPLES	23
1.2-1 BUILT-IN CODE EXAMPLES	23
1.2-2 GITHUB CODE EXAMPLES	23
1.2-2a Method 1: Clone and Import all Eclipse Projects directly from GitHub.....	25
1.2-2b Method 2: Clone from GitHub, then Import all Eclipse Projects (if desired).....	29
1.2-2c Method 3: Download Zip from GitHub then Import all Eclipse Projects (if desired).....	33
1.2-2d Importing all Examples as General Eclipse Projects from a local Git Repository to another Workspace.....	36
1.2-2e Importing a Single ModusToolbox Example Application.....	39
1.3 TOUR OF DOCUMENTATION	40
1.3-1 IN MODUSTOOLBOX IDE	40
1.3-2 ON THE WEB	41
1.4 REPORTING ISSUES	44



1.1 TOUR OF MODUSTOOLBOX™ IDE

1.1-1 FIRST LOOK

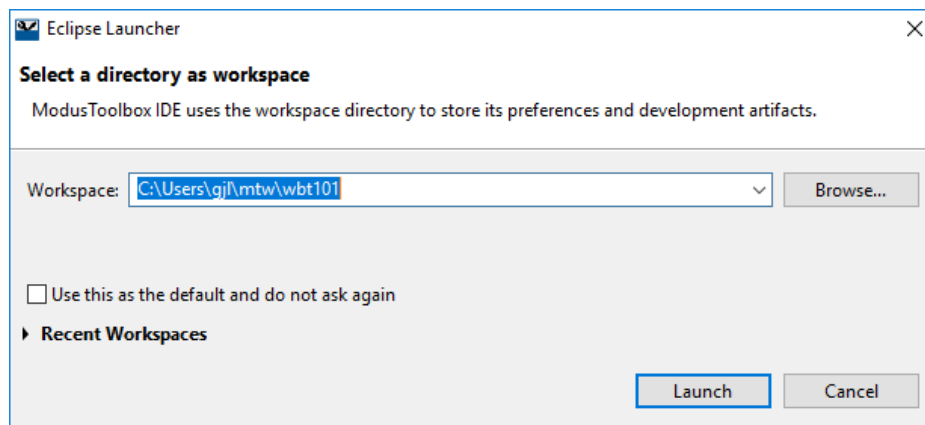
The software tool we will use is called "ModusToolbox™ IDE" and it is based on Eclipse.

For Windows users, ModusToolbox™ is installed, by default, in *C:/Users/<UserName>/ModusToolbox*. Once installed, the IDE will show up in Windows under *Start->All Programs->ModusToolbox 2.0*.

(Comments will be added on how the tool installs into Apple iOS and Linux environments when users running those platforms share their experience)

When you first run ModusToolbox™ IDE, you will create a Workspace to hold your applications. The default Workspace location is *C:/Users/<UserName>/mtw*. Create as many Workspaces as you want. Locate Workspaces anywhere that's accessible from a program on your PC. I usually put each workspace under a folder inside *C:/Users/<UserName>/mtw*.

Each time you open ModusToolbox™ IDE you can provide a unique workspace name such as the following:



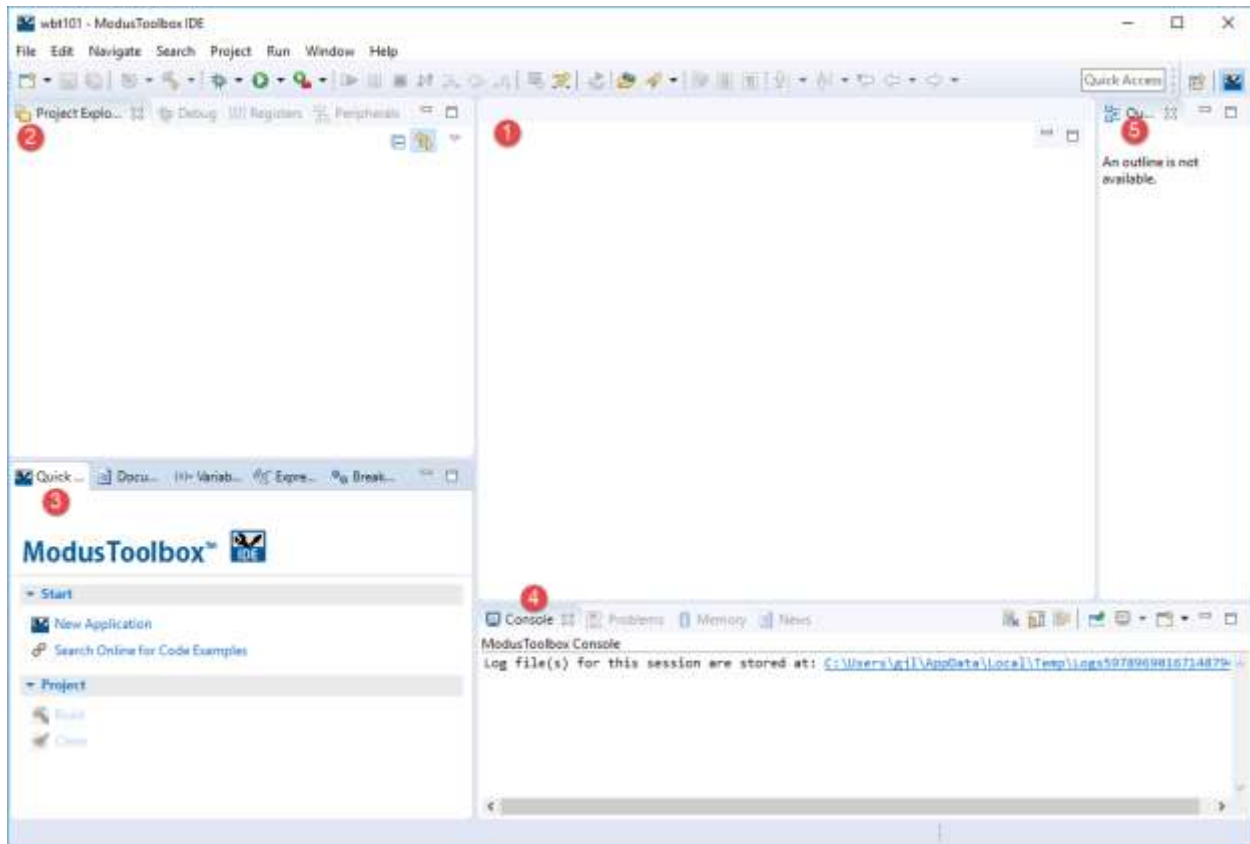
Whenever you want to switch to a different Workspace or create a new one, just use the menu item *File->Switch Workspace* and either select one from the list or select *Other...* to specify the name of an existing or new Workspace.

After clicking Launch, the IDE will start.

An (empty) ModusToolbox™ perspective (looking like Eclipse) will appear.

A perspective in Eclipse is a collection of views. The ModusToolbox™ perspective combines editing and debugging features. You can also create your own custom perspectives if you want a different set or arrangement of windows. You can always get back to the ModusToolbox™ perspective by selecting it

from the button in the upper right corner of the IDE, clicking the Open Perspective button and choosing ModusToolbox, or from *Window->Perspective->Open Perspective->Other->ModusToolbox*.



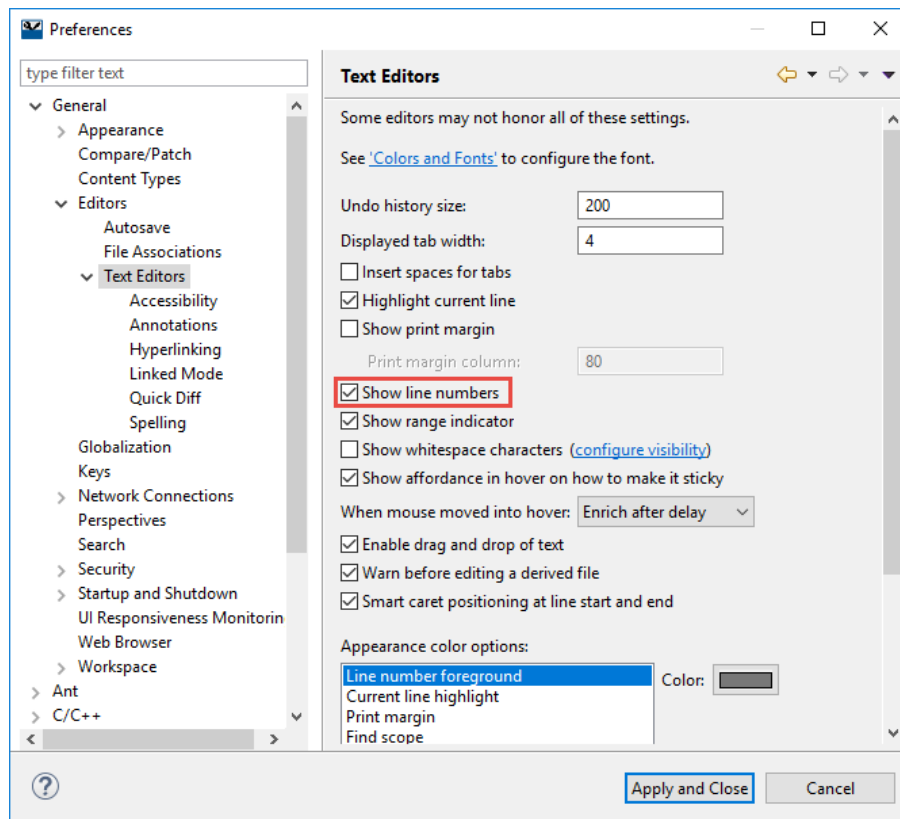
The major views are:

1. File Editor
2. Project Explorer
3. Quick Panel / Documents
4. Console / Problems
5. Outline

If you close a view unintentionally, you can reopen it from the menu *Window->Show View*. Some of the views are under *Window->Show View->Other...* You can drag and drop windows and resize them as you desire.

1.1-2 CUSTOMIZATION

Eclipse is extremely flexible – you can customize almost anything if you know where to look. A good place to start for general Eclipse settings is *Window->Preferences*. One that I always turn on is *General->Editors->Text Editors->Show line numbers*. Most of these settings are at the workspace level.

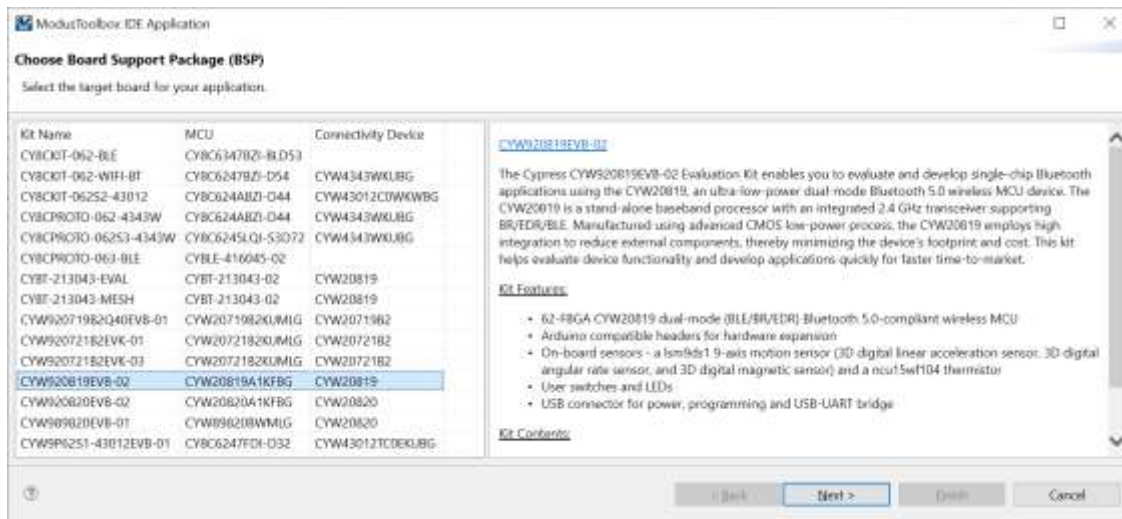


There are also project build settings which we will explore later.

1.1-4 NEW APPLICATION WIZARD

ModusToolbox™ IDE includes a wizard that sets up new applications with the required target hardware support, Bluetooth configuration code, middleware libraries, build, program and debug settings, and a "starter" application. Launch the wizard from the "New Application" button in the Quick Panel or use "New ModusToolbox™ Application" item in *the File->New menu*.

It is a multi-step wizard that first asks you to select the "Target Hardware". Note that it is possible for users to create custom Bluetooth SoC board support packages (BSPs) but that is beyond the scope of this course. The kits used in this course are the [CYBT-213043-MESH](#) and [CYW920819EVB-02](#).

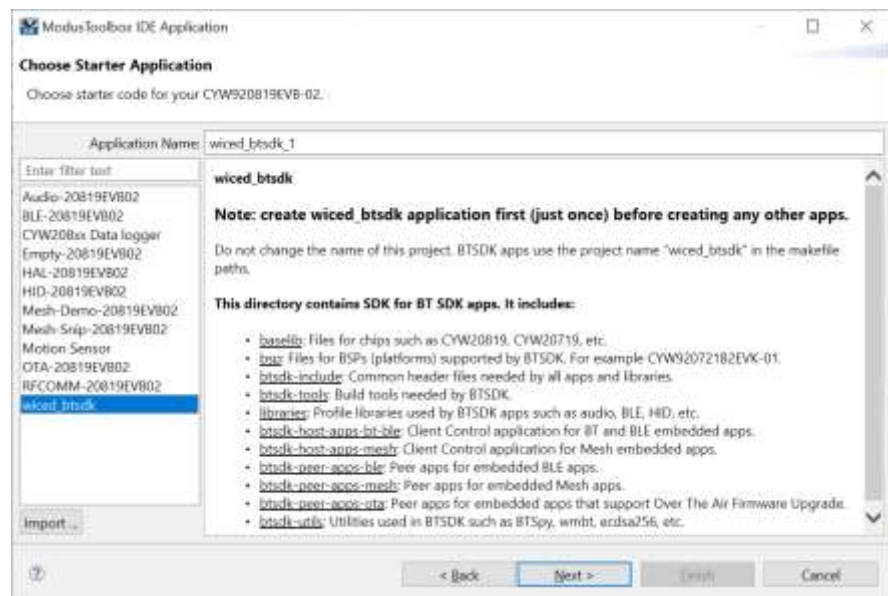


Clicking "Next" lets you choose a single or group starter application from a list of applications that support the selected hardware. You always begin with a starter application. These range in complexity from mostly empty to small snips to fully functional demos. Some starter applications are built into the IDE, others are available as code examples from GitHub or other locations.

Note that there is no real equivalent to the PSoC Creator empty project, but there is an EmptyWicedBluetooth application that gives you just enough to get started. Unlike the PSoC Creator empty project, using this as a starting point is more of an expert-level approach since it doesn't provide much in the way of code. In most cases, we recommend starting with one of the code examples and modifying it as needed for your application.

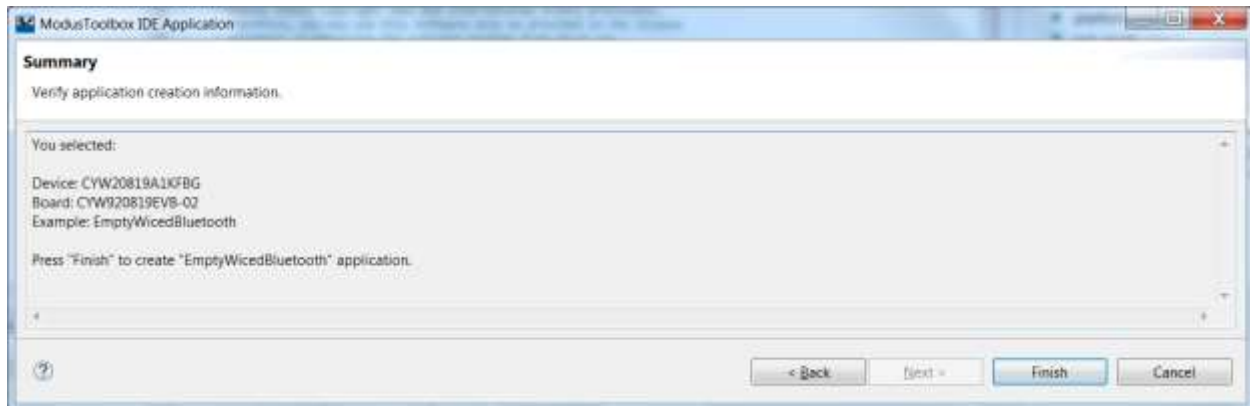
Before creating any project in a new workspace, start with `wiced_btstack`

Each starter application includes a description and a default name, which you can modify if you wish (if you create multiple copies of the same template without changing the name the tool adds an incrementing number to each created project name).



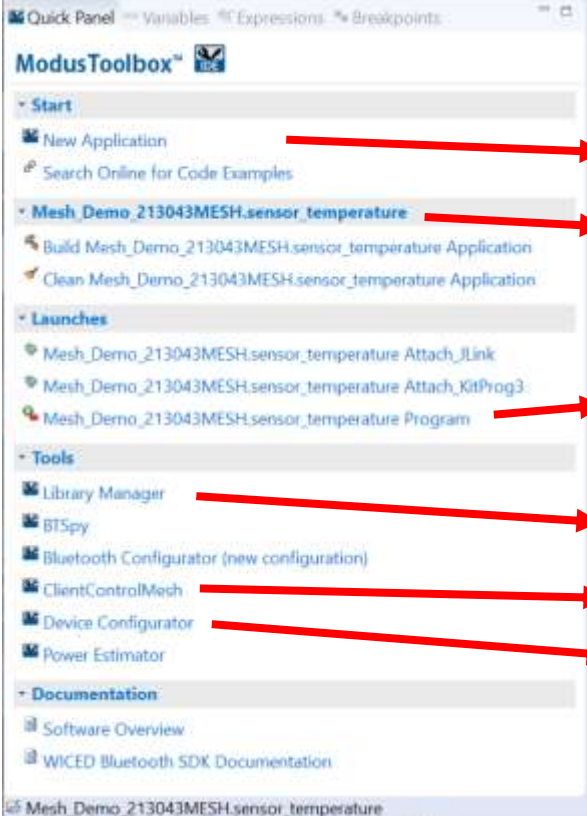
Pressing "Next" takes you to the final step, where you verify your choices and press "Finish" to create the application.

Note that the wizard does not prompt for a directory. ModusToolbox™ IDE uses Eclipse workspaces as containers for projects. All projects within a workspace reside at the top level of the workspace folder. You can choose any legal folder/file name for the project, but you cannot change its location on disk.



1.1-5 QUICK PANEL

Once you have created a new application the Quick Panel is populated with common commands, Configurators and Tools.



Equivalent Menu Path

- File->New->New ModusToolbox IDE Application
- Selected project in Project Explorer, will show up in Quick Panel
- One click build and program
- Select Board Support Packages and associated files
- Start Tools relevant to the selected project from Quick Panel
- Open Configurators for the selected project

1.1-6 APPLICATIONS AND PROJECTS

At this point, we are ready to start developing the application. In the world of ModusToolbox™, an application is "deployable firmware designed for the target hardware" while a project is "a compilation unit, organized into an Eclipse project". An application in ModusToolbox™ IDE may have more than one Eclipse project.

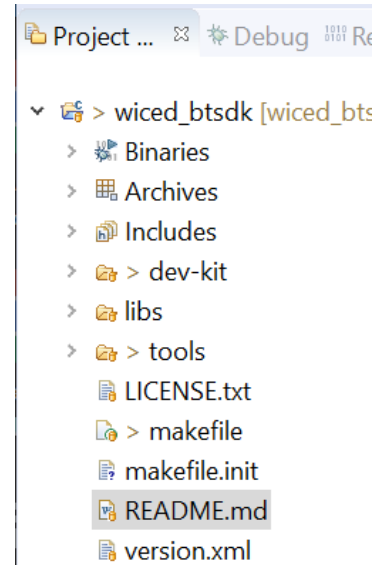
We will go through the details on the different ways to create a new application later in the next chapter, but for now let's look at what an application will look like in the IDE once it is created.

1.1-6A PROJECT EXPLORER – WICED-BTSDK

The WICED Bluetooth Software Development Kit must be the first application in any Workspace utilizing WICED Bluetooth enabled devices. In the project explorer window, you will see the project and all its associated files. WICED Bluetooth applications will consist of a single project.

The key parts of wiced_btSDK are:

- A folder for wiced-btSDK
- Common Middleware, Application and Build settings in Binaries, Archives and Includes
- Board Support platform, Configurator settings and Stack configuration items in dev-kit
- Link to documentation in libs
- Tool settings in tools
- Housekeeping items
- Common makefiles
- Readme file(s)

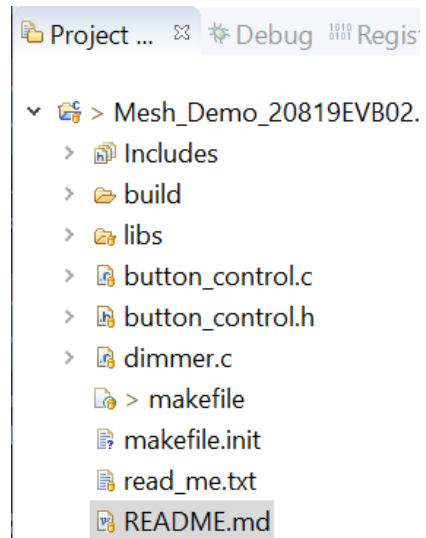


1.1-6A PROJECT EXPLORER

In the project explorer window, you will see the project and all its associated files. WICED Bluetooth applications will consist of a single project.

The key parts of a project are:

- A folder with the name of the project
- Readme file(s)
- Command-line makefile – modus.mk
- Middleware
- Application C source files
- Application settings
- Project Build settings



1.1-6B README

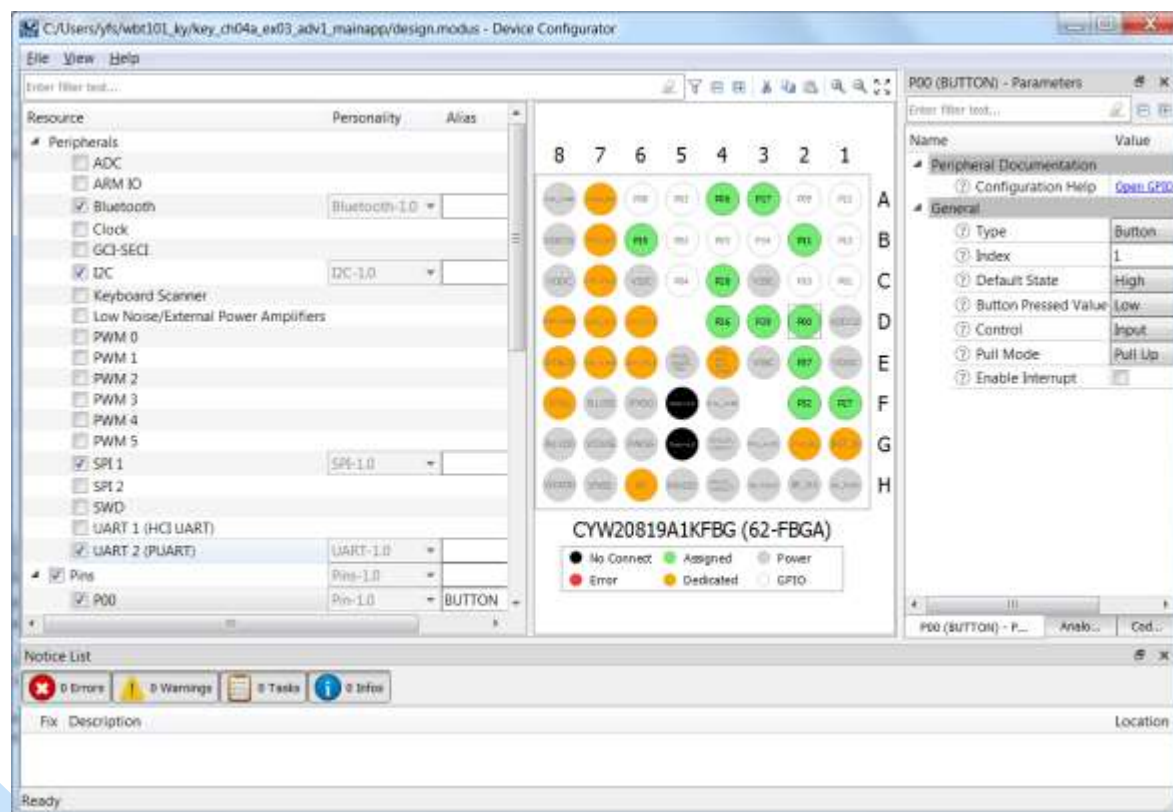
The first file to open in the file editor window will be the readme file included with the application, if there is one. This gives general information about the platform or the application that you started with.

1.1-6C CONFIGURATION DATABASE FILE – DESIGN.MODUS

The design.modus file is the database of configuration choices for the device. It is the source document for the Device Configurator tool which presents a list of pins and peripherals on the device that you can set up.

When you create a new application it includes a generic design.modus file from the BSP or a custom file from the starter template. This pre-configured design.modus file sets up the usual hardware, such as buttons and LEDs, so you do not need to make the same edits every time you create a new application. You can, of course, modify these settings, for example to drive an LED with a PWM instead of firmware. This is discussed later, in the Board Support Package section.

To launch the Device Configurator click on the Configure Device link in the Quick Panel or double-click on the design.modus file. Within Device Configurator are sections for Peripherals and Pins on the left. When you enable a peripheral or pin, the upper right-hand panel allows you to select configuration options and to open the documentation for that element. In some cases, you can launch a secondary configurator from that window (Bluetooth is one of those cases).



Once you save the configurator information (*File->Save*) it creates/updates the Generated Source files for the project.

1.1-6D GENERATEDSOURCE FOLDER

When you save the device configuration, the tool generates firmware in the GeneratedSource folder. The following are the most interesting/useful files.

`cycfg_bt.h`, `cycfg_gatt_db.c` and `cycfg_gatt_db.h` (BLE GATT database setup)

`cycfg_pins.c` and `cycfg_pins.h` (definitions and setup code for GPIO, LEDs and Buttons)

`cycfg_notices.h` (enables error reporting in the ModusToolbox™ IDE Console view)

1.1-6E COMMAND-LINE MAKEFILE – MODUS.MK

The `modus.mk` file is used in the application creation process – it defines everything that ModusToolbox™ needs to know in order to create the application. If you are developing inside ModusToolbox™ IDE then `modus.mk` is not used after the application is first created. If, however, you are using a different (make-based) IDE or building on the command line, you use `modus.mk` (and `makefile.init`) to build the application.

When you choose a template in the New Application wizard you are really just selecting a `modus.mk` file, which specifies the BSP (with the `CYSDK` environment variable) and the files to include in the new application. You can use this mechanism to create your own templates.

Note: remember that the `modus.mk` is NOT automatically updated when you make changes to an application, so you may need to update it manually. Also note that if you need custom device configuration settings, you must specify the path to your custom `design.modus` and `wiced_platform.h` files in the list of source files in `modus.mk`. Otherwise, application creation will use the default files from the board support package (which we will discuss in the next chapter).

1.1-6F PLATFORM FILES (WICED_PLATFORM.*)

These files enable the startup code to configure all the GPIOs by calling the `wiced_platform_init()` function. This function loops through all the GPIOs, LEDs and Buttons on the board, setting them up according to the `design.modus` settings in the `cycfg_pins.c` file.

The `wiced_platform.h` file defines useful labels for the configured pins that you can use in your code to create reliable, portable applications. Here are the defines you will be using for buttons and pins on the CYW920819EVB-02 kit.

```
/*! pin for button 1 */
#define WICED_GPIO_PIN_BUTTON_1    WICED_P00
```

```

/*! pin for LED 1 */
#define WICED_GPIO_PIN_LED_1    WICED_P27

/*! pin for LED 2 */
#define WICED_GPIO_PIN_LED_2    WICED_P26

```

Note that in future versions of ModusToolbox™ (2.0 and beyond) these files may no longer exist. The pertinent information may instead be incorporated into the design.modus file so that everything exists in one consistent place.

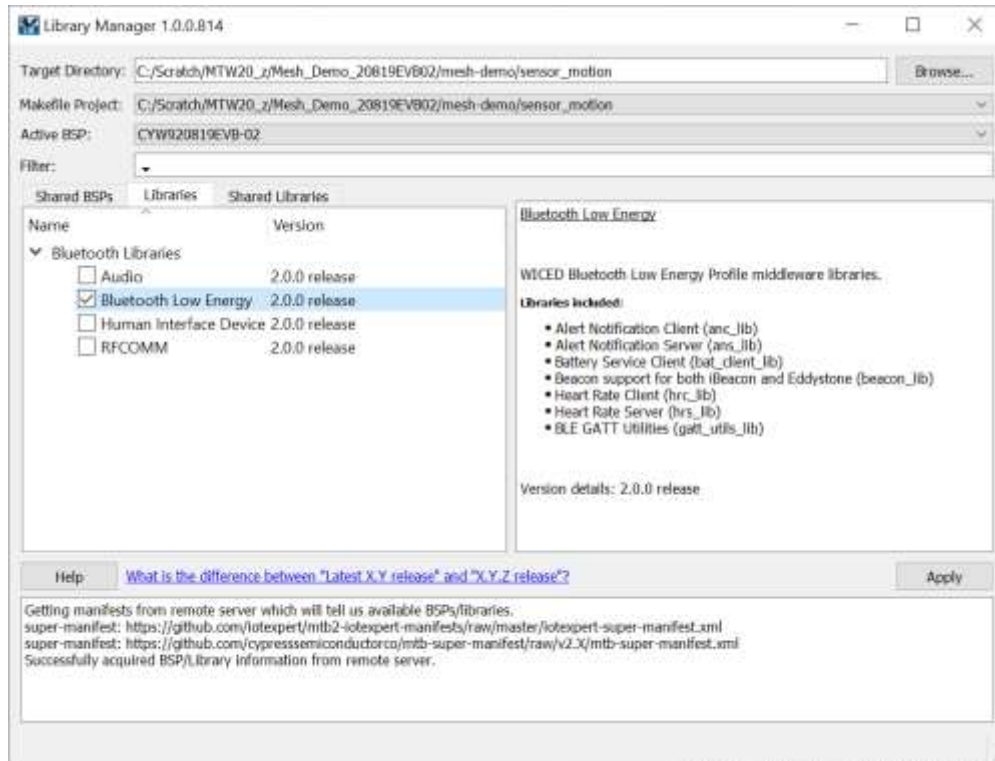
1.1-6G STACK CONFIGURATION FILES

Most templates include `app_bt_cfg.c` and `app_bt_cfg.h`, which create static definitions of the stack configuration and buffer pools. You will edit the stack configuration, for example, to change the name of your device or to optimize the scanning and advertising parameters. The buffer pools determine the availability of various sizes of memory blocks for the stack, and you might edit those to optimize performance and RAM usage.

Note: The actual file names may vary in some code examples, but the definitions of the `wiced_bt_cfg_settings` struct and `wiced_bt_cfg_buf_pools` array are required.

1.1-6H LIBRARY MANAGER

ModusToolbox provides a GUI for helping you manage library files. You can run this from the Quick panel link “Library Manager”. It is also available outside the IDE from `ModusToolbox/tools_2.0/library-manager`.



When you run the library manager GUI it will create “dot lib” files, then it will run “make getlibs”. You can always do this for yourself. You can also remove directories in the libs directory, and they can be recreated by running “make getlibs”. This means that you can check-in just the “dot libs” into your code repository and “make getlibs” will do the rest of the work.

Libraries can be hierarchical, meaning that a library can have “dot libs” inside of it and “make getlibs” will bring in all of the libraries through the hierarchy.

1.1-6I APPLICATION C FILE

Most templates include an application C file that is expected to be edited by the user to (at least) start up the application (from the `application_start()` function). The actual name of this file varies according to the template used to create the application. It is usually the same name as the template but there are exceptions to that rule. For example, in the templates provided for this class this top-level file is called `app.c`.

Note: MESH examples are implemented with a library called mesh_app_lib. The library includes the application_start() function in mesh_application.c inside the library. This is because the application itself is very regimented, and the developer does not really "own" the way devices interact. The user part of the application is restricted to activity supported by the device's capabilities, such as dimming the light with a PWM or controlling a door lock solenoid.

Inclusion of .h files

The application C file begins with various #include lines depending on the resources used in your application. These files can be found in the SDK under ModusToolbox_1.1\libraries\bt_20819A1-1.0\components\BT-SDK. A few examples are shown below. The first 4 are usually required in any project. The "hal" includes are only needed if the specific peripheral are used in the project, e.g. wiced_hal_i2c.h is required if you are using I2C.

```
#include "wiced.h"           // Basic formats like stdint, wiced_result_t, WICED_FALSE, WICED_TRUE
#include "wiced_platform.h"   // Platform file for the kit
#include "wiced_hal_gpio.h"    // GPIO drivers
#include "wiced_hal_pspi.h"    // Peripheral SPI drivers
#include "sparcommon.h"        // Common application definitions
#include "wiced_bt_stack.h"    // Bluetooth Stack
#include "wiced_bt_dev.h"      // Bluetooth Management
#include "wiced_bt_ble.h"      // BLE
#include "wiced_bt_gatt.h"     // BLE GATT database
#include "wiced_bt_uuid.h"     // BLE standard UUIDs
#include "wiced_rtos.h"        // RTOS
#include "wiced_bt_app_common.h" // Miscellaneous helper functions including wiced_bt_app_init
#include "wiced_transport.h"   // HCI UART drivers
#include "wiced_bt_trace.h"    // Trace message utilities
#include "wiced_timer.h"       // Built-in timer drivers
#include "wiced_hal_i2c.h"     // I2C drivers
#include "wiced_hal_adc.h"     // ADC drivers
#include "wiced_hal_pwm.h"     // PWM drivers
#include "wiced_hal_puart.h"   // PUART drivers
#include "wiced_hal_nvrाम.h"   // NVRAM drivers
```

```
#include "wiced_hal_wdog.h"    // Watchdog
#include "wiced_spar_utils.h"  // Required for stdio functions such as snprintf and sprintf
#include <stdio.h>              // Stdio C functions such as snprintf and sprintf
```

After the includes list you will find the `application_start()` function, which is the main entry point into the application. That function typically does a minimal amount of initialization then it starts the Bluetooth stack and registers a stack callback function by calling `wiced_bt_stack_init()`. Note that the configuration parameters from `app_bt_cfg.c` are provided to the stack here. The callback function is called by the stack whenever it has an event that the user's application might need to know about. It typically controls the rest of the application based on Bluetooth events.

Most application initialization is done once the Bluetooth stack has been enabled. That event is called `BTM_ENABLED_EVT` in the callback function. The full list of events from the Bluetooth stack can be found in the file `ModusToolbox_1.1/libraries/bt_20819A1-1.0/components/BT-SDK/20819-A1_Bluetooth/include/20819/wiced_bt_dev.h`.

A minimal C file for an application will look something like this:

```
#include "sparcommon.h"
#include "wiced_platform.h"
#include "wiced_bt_dev.h"
#include "wiced_bt_stack.h"
#include "app_bt_cfg.h"

wiced_bt_dev_status_t app_bt_management_callback( wiced_bt_management_evt_t
event, wiced_bt_management_evt_data_t *p_event_data );

void application_start(void)
{
    wiced_bt_stack_init( app_bt_management_callback, &wiced_bt_cfg_settings,
                        wiced_bt_cfg_buf_pools );
}

wiced_result_t app_bt_management_callback( wiced_bt_management_evt_t event,
                                           wiced_bt_management_evt_data_t
*p_event_data )
{
    wiced_result_t status = WICED_BT_SUCCESS;

    switch( event )
    {
```

```

case BTM_ENABLED_EVT:           // Bluetooth Controller and Host Stack Enabled

    if( WICED_BT_SUCCESS == p_event_data->enabled.status )
    {
        /* Initialize and start your application here once the BT stack is
        running */
    }

    break;

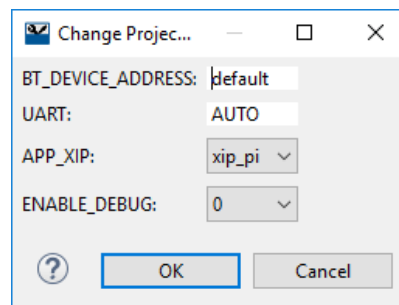
default:
    break;
}

return status;
}

```

1.1-6J APPLICATION SETTINGS

Unlike WICED Studio, ModusToolbox™ does not require the user to create and maintain a make target for every application. Source file compilation is handled by the IDE automatically. Useful application-level settings, such as debugging, are provided by the SDK and can be modified by right-clicking on the project name in the Explorer view and selecting "Change Application Settings".



The settings entered here are typically used in the code to enable or disable some feature or may be used by the build process (such as specifying the UART to use for programming). ENABLE_DEBUG, for example, causes the firmware to open a time window for the Segger debugger probe to connect to the core over SWD and take control of the application.

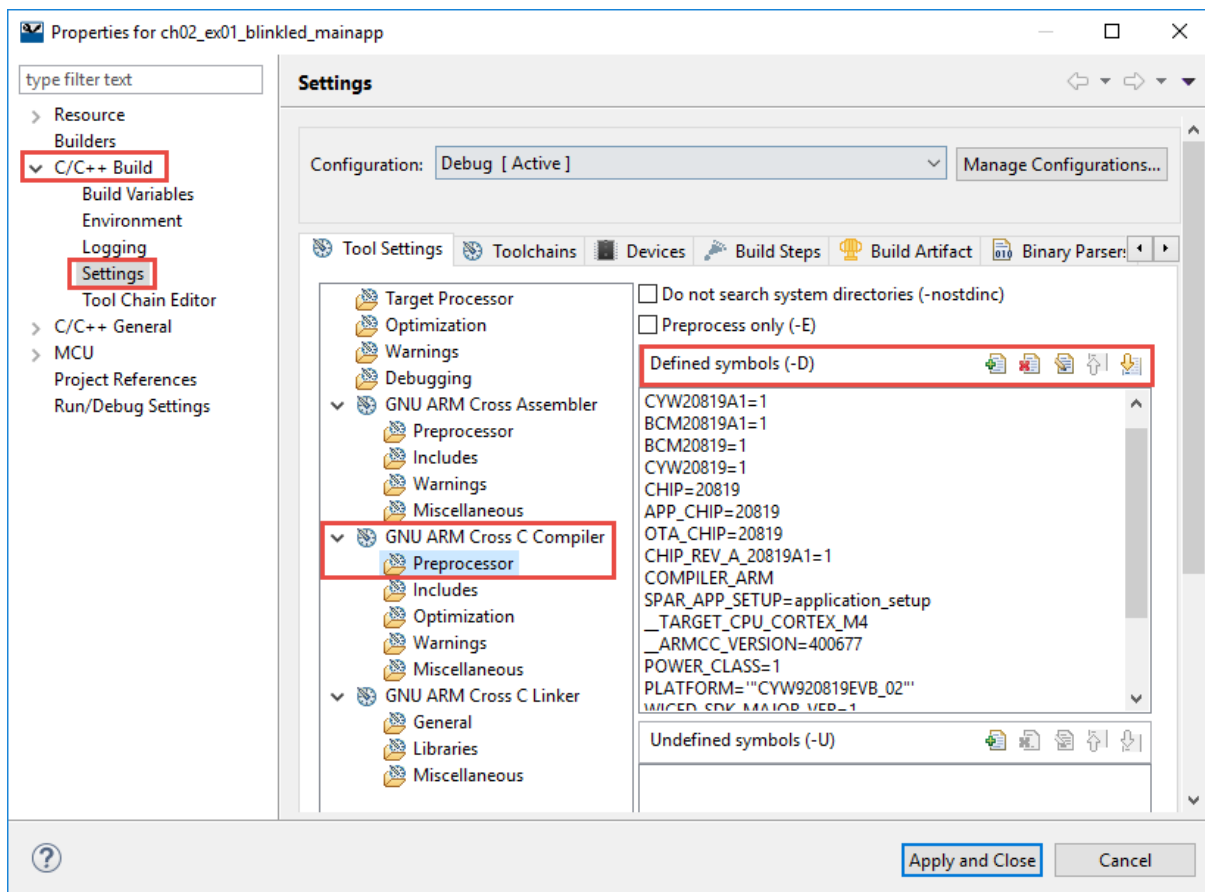
Under the hood, these values are defined in various platform make files as PLATFORM_FEATURES lines and in the application's modus.mk file as APP_FEATURES lines. Once the application is created in the IDE, the concatenation of all these settings and their values is stored at the top of modus.mk in a line labeled FEATURE_VALUES.

The `modus.mk` also may include `#defines` for the application using lines of the form `CY_APP_DEFINES` but these are not shown in the Change Application Settings dialog. Note that these are used during application creation and command line builds but changing them in `modus.mk` after the application has been created will not affect builds in the IDE. To change `#defines` in the IDE after an application has been created, use the project build settings as described next.

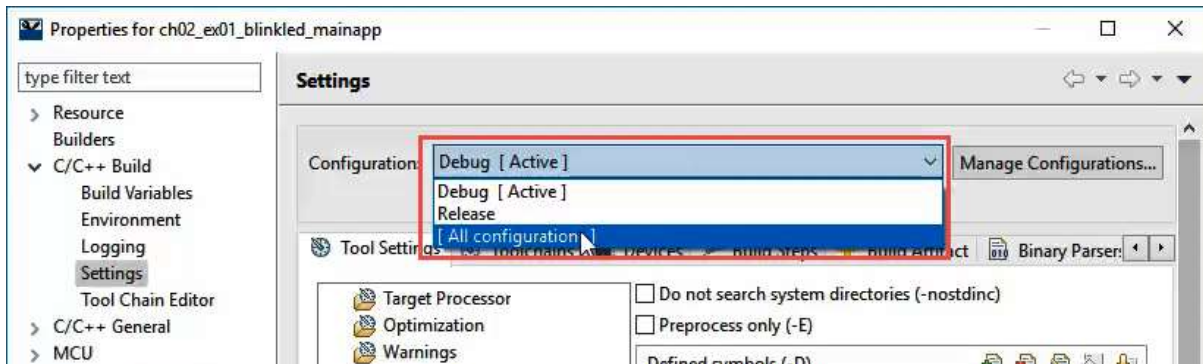
1.1-6K PROJECT BUILD SETTINGS

Build settings are project specific. You can access them from the Quick Panel, by right-clicking on the project and selecting *Properties*, or from the menu item *Project->Properties*. You will notice that there are a LOT of properties. Most of them are fine with default values but just know that just about anything can be customized.

For build settings, you will find them under the left window hierarchy of *C/C++ Build->Settings*. Inside the Settings window there are tabs for *Tool Settings*, *Toolchains*, etc. Under *Tool Settings* there is additional hierarchy containing settings for *Optimization*, *Assembler*, *Compiler*, etc. For example, you can add `#define` processor directives to your project under *GNU ARM Cross C Compiler->Preprocessor*.



The build settings are specific to a build configuration (e.g. Debug vs. Release). If you want to make changes to all configurations at once, you can choose [All Configurations] from the drop-down menu.



To select the active configuration, right-click on the project and select *Build Configurations->Set Active* or use the menu item *Project->Build Configurations->Set Active*. Again, remember that build configurations are specific to a project, NOT an application. If your application has multiple projects and you want to set the active configuration for all of them, select them all in the project explorer window first (use Shift-Click on Control-Click) and then set the active configuration.

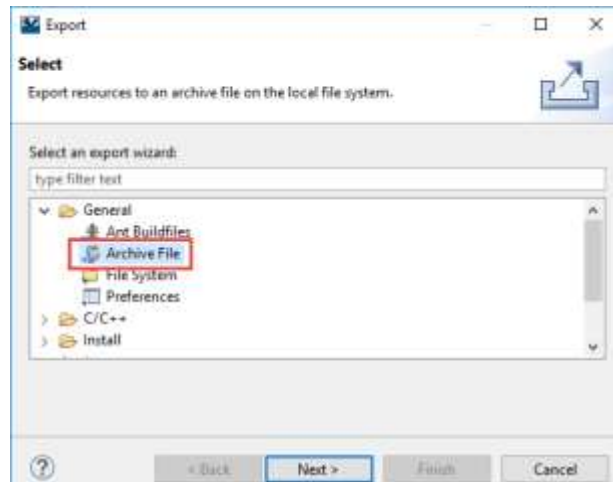
1.1-7 SHARING APPLICATIONS

As discussed above, creation of a new application relies on a starter application (i.e. template). The starter application includes a `modus.mk` file and the source files. If the application uses custom device configuration settings, it will also include `design.modus` and `wiced_platform.h` files.

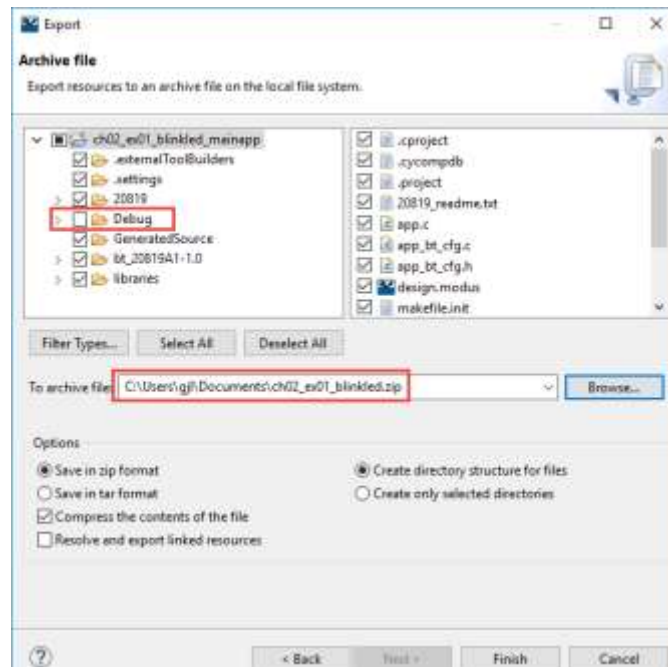
Another way to share an application is to export it from ModusToolbox™ IDE into an archive file. This captures a full copy of the application that can then be imported as an existing project into another workspace. The steps are:

1.1-7A EXPORT TO ARCHIVE

1. Select the project or projects to be exported
2. Choose *File->Export->General->Archive File* and click Next



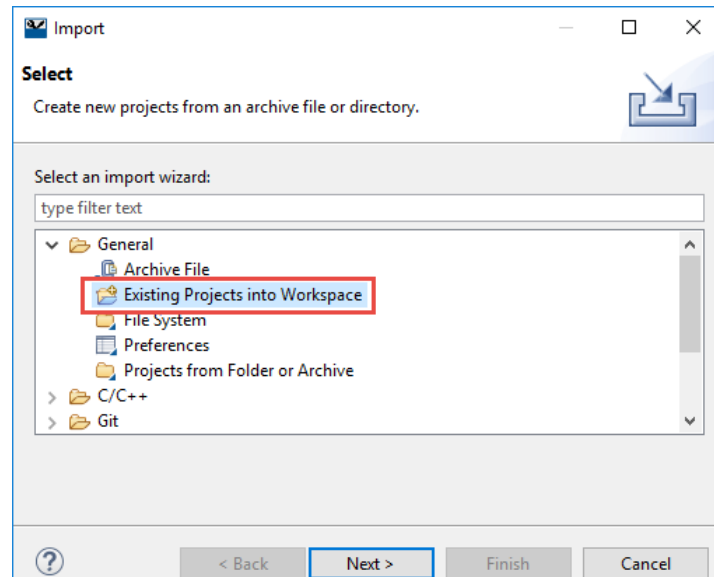
3. Uncheck the Debug and Release folders if they exist to save space (these are build output files)
4. Enter the desired file path/name



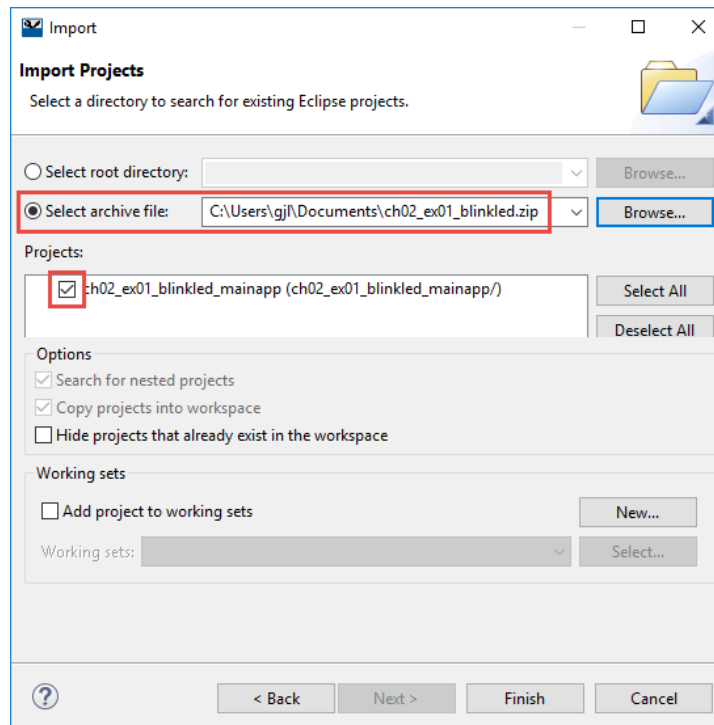
5. Click Finish

1.1-7B IMPORT FROM ARCHIVE

1. In the new workspace choose *File->Import->General->Existing Projects into Workspace*
 - a. DO NOT choose *File->Import->General->Archive File!*



2. Choose *Select archive file* and specify the path to the zip file.
3. Check the box next to the project(s) you want to import.

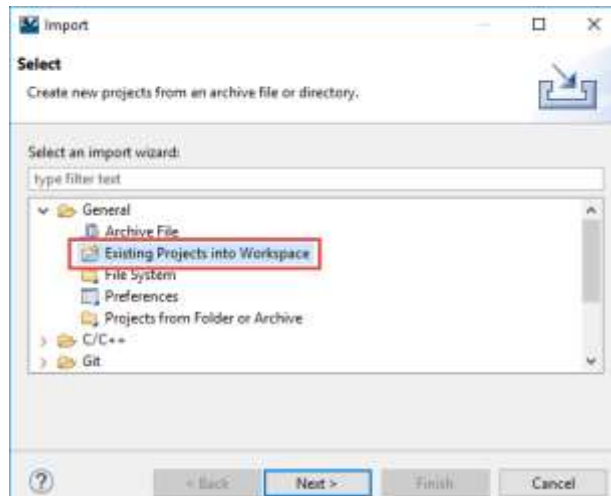


4. Click Finish

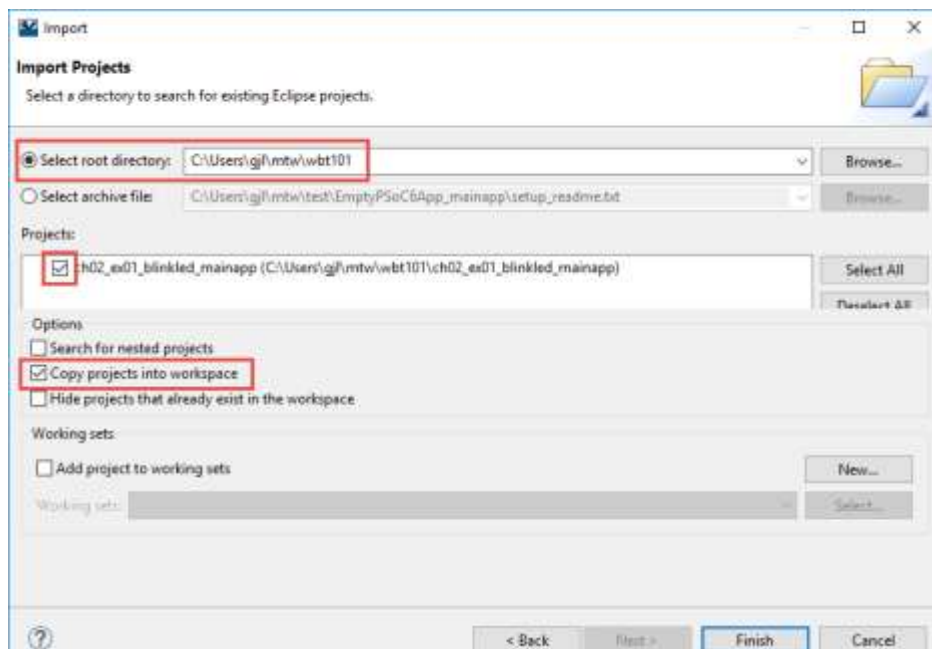
Yet another way to share a full application is to just use the filesystem path to the application when importing existing projects. This works the same as the previous option except that an archive is not created first. The steps are:

1.1-7C IMPORT FROM FILESYSTEM

1. In the new workspace choose *File->Import->General->Existing Projects into Workspace* and click Next.



2. Choose the path to the projects to be imported.
 - a. This can either be the path to a workspace or an individual project.
3. Check the box next to the project for projects you want to import.
4. Check the box *Copy projects into workspace*.





5. Click Finish

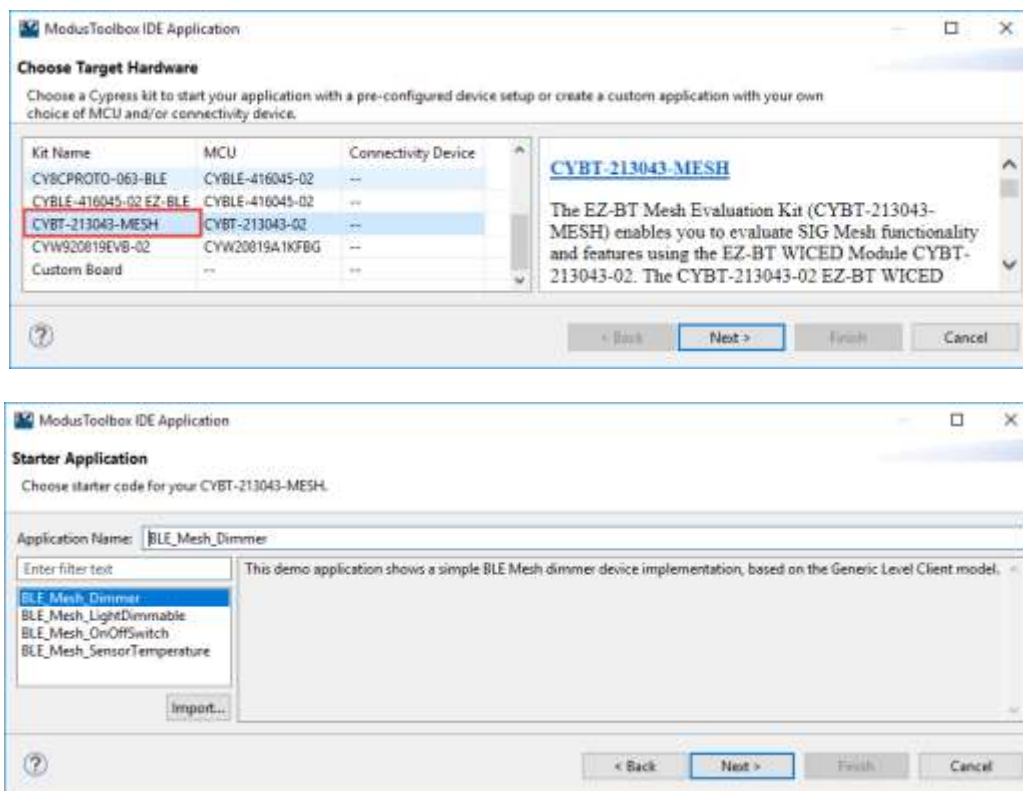
V | **Five Years Out**

1.2 CODE EXAMPLES

ModusToolbox™ contains a wealth of code examples. Some code examples are built into the Bluetooth SDK while others are available for download from GitHub. The online code examples are generally categorized in one of two types: demo (fully featured applications) and snip (examples that show one or a few concepts).

1.2-1 BUILT-IN CODE EXAMPLES

For the built-in examples, you can just choose "New Application" from the Quick Panel in the ModusToolbox™ IDE, select one of the 20819 kits (CYW920819EVB-02 or CYBT-213043-MESH), and then pick one of the available starter applications:



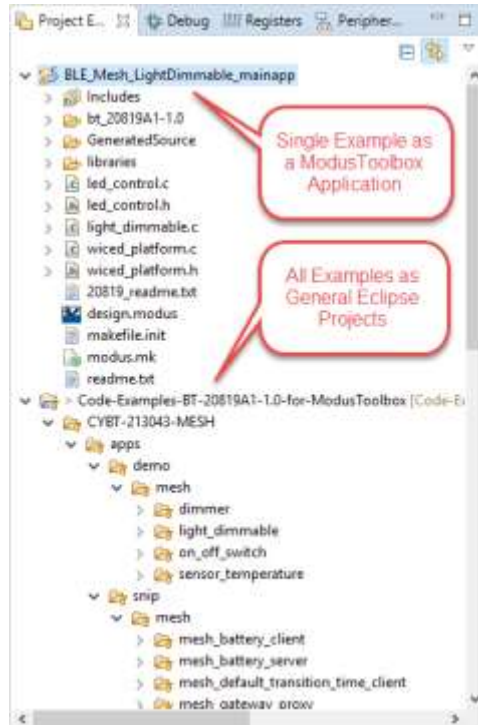
1.2-2 GITHUB CODE EXAMPLES

To search for code examples online, click on "Search Online for Code Examples" from the Quick Panel. This opens a web browser to the GitHub repository. From the web browser, choose the *20189A1 Bluetooth Examples* repository. This will present you with a repository containing code examples for the CYW920819EVB-02 and CYBT-213043-MESH kits. Under each kit, you can browse through the demo and snip applications under the various folders.

A direct link to the 20819 Bluetooth code example repository is:

<https://github.com/cypresssemiconductorco/Code-Examples-BT-20819A1-1.0-for-ModusToolbox™>

There are two basic ways to use the code example repository. First, you can import all the examples at once into ModusToolbox™ IDE as general Eclipse projects. This allows you to view (and copy from) the source code of all examples, but the examples cannot be built as ModusToolbox™ applications. Second, you can create a new ModusToolbox™ application from any of the code examples. This gives you a full application that you can build and download to a device. Note that you can use both methods in a single workspace if you so desire.

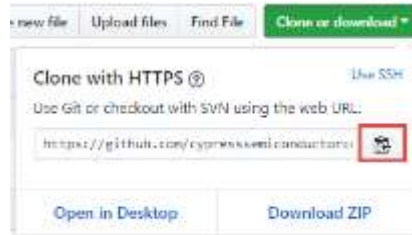


There are three ways to get a copy of the entire repository. They are:

1. Clone the repository and import projects as general Eclipse projects directly from GitHub as a single step in ModusToolbox™ IDE. Then create individual ModusToolbox™ applications in IDE as desired.
2. Use Git Clone to get a copy of the repository on your local machine. Then import all projects as general Eclipse projects and/or create individual ModusToolbox™ applications as desired.
3. Download a Zip file from GitHub and unzip on your local machine. Then import all projects as general Eclipse projects and/or create individual ModusToolbox™ applications as desired.

Note: You don't need to clone or download a zip of the repository for every workspace – just one copy is enough. After that you can import all the projects as existing Eclipse projects or into any workspace or create new ModusToolbox™ applications based on any individual examples. The methods for all these operations are detailed below.

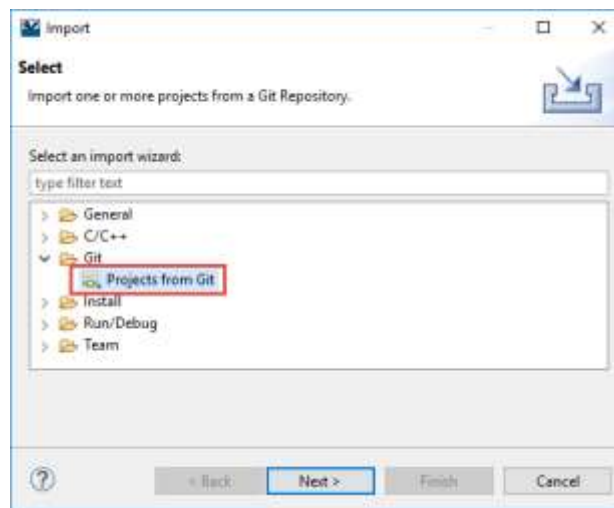
Note: From the GitHub page, you can click the button "Clone or download" to open a window that will allow you to download a Zip file or to copy the URI for the repository to the clipboard to use in cloning (the copy button is shown with a red box around it in the figure below).



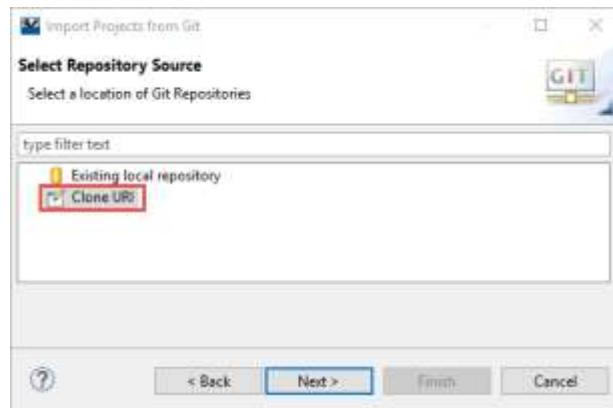
1.2-2A METHOD 1: CLONE AND IMPORT ALL ECLIPSE PROJECTS DIRECTLY FROM GITHUB

The steps to clone a repository from GitHub and import all the projects in that repository as general Eclipse projects from within ModusToolbox™ IDE are shown below. Note that this should be done only if you have not previously downloaded the repository.

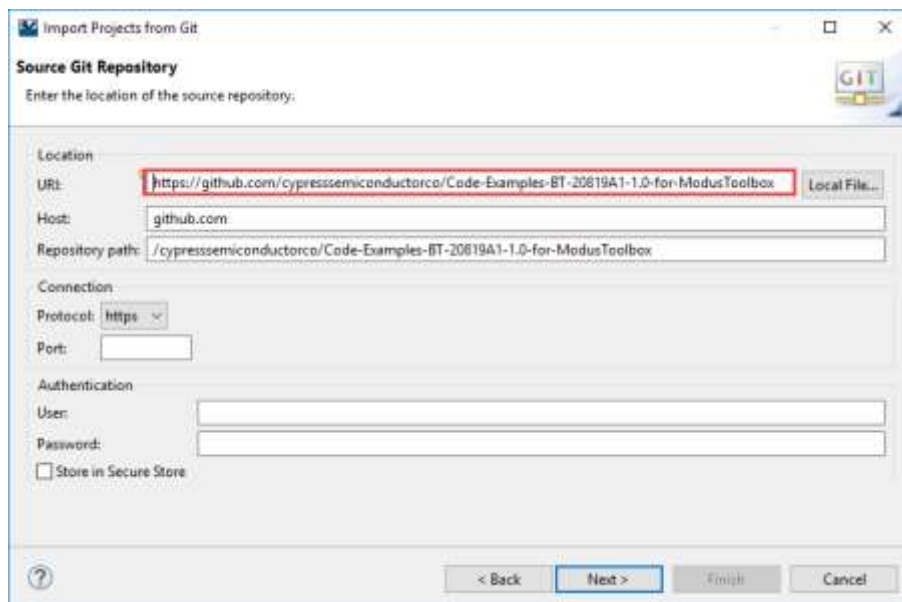
1. Choose File -> Import -> Git -> Projects from Git and click Next.



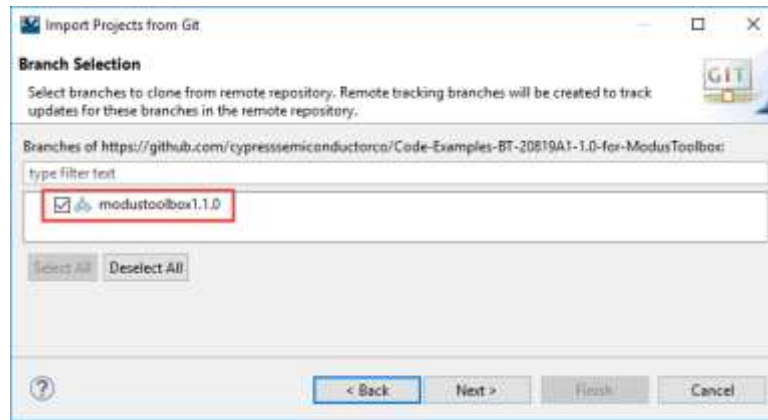
2. Choose Clone URI and click Next.



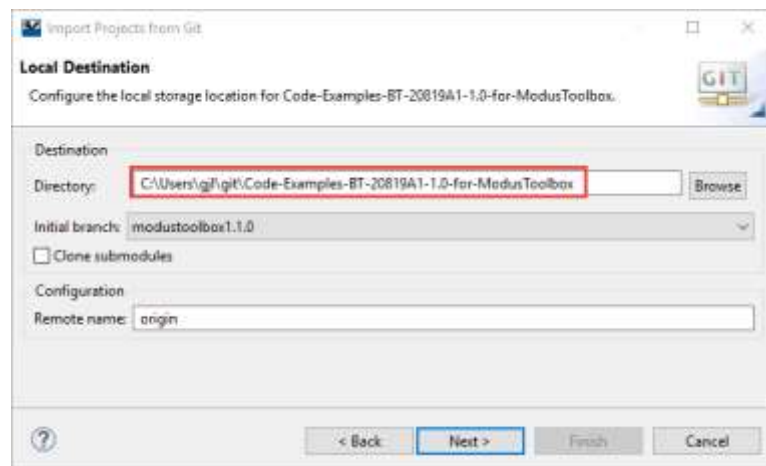
3. Enter the URI. The host and repository path will be filled in automatically. Click Next.
 - a. Hint: Remember that you can copy the URI from the "Clone or download" window from GitHub so you don't need to type it in manually.



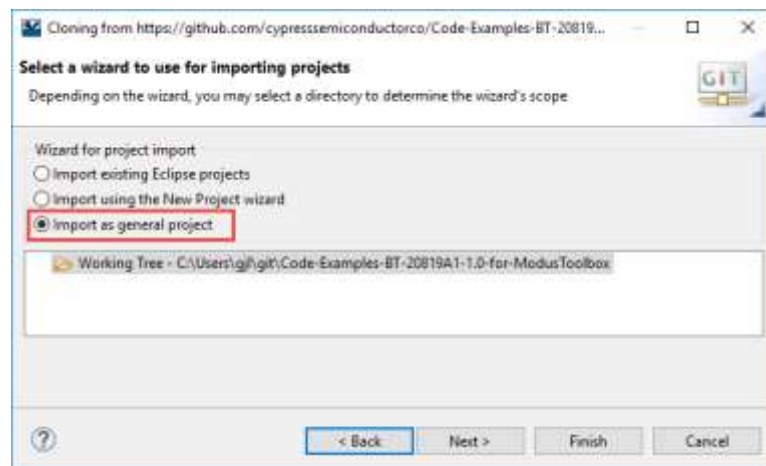
4. Select the branch to download and click Next. The branch you choose will depend on the version of ModusToolbox™ that you are using.



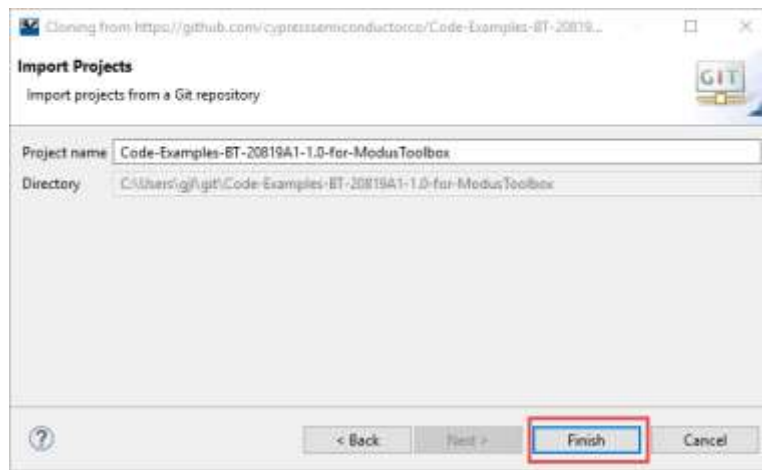
5. Choose a destination on the local machine for the repository and click Next



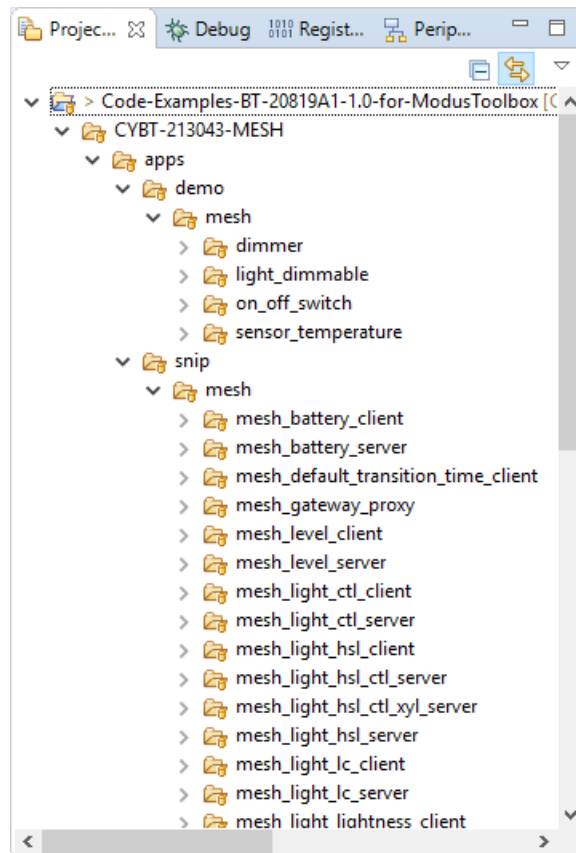
6. Choose Import as general project and click Next



7. Use the provided project name or enter another name of your choice. Click Finish to complete the import.



8. Projects will appear hierarchically in the Project Explorer Window as shown below.

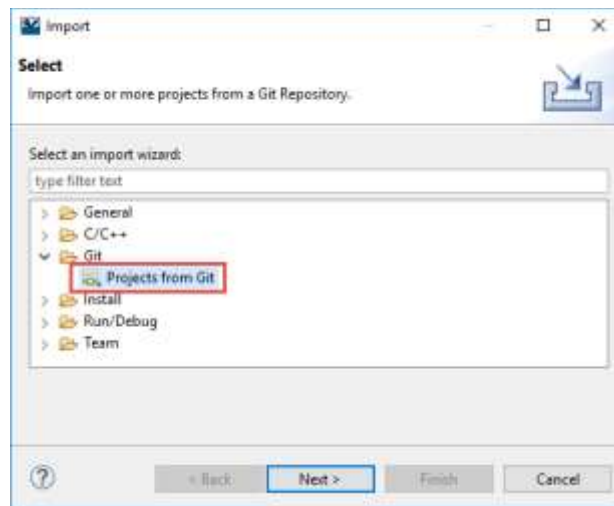


If you want to skip the other two methods for importing the entire repository, click [here](#).

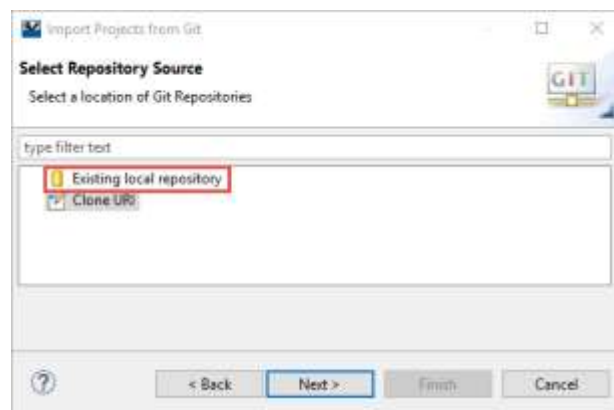
1.2-2B METHOD 2: CLONE FROM GITHUB, THEN IMPORT ALL ECLIPSE PROJECTS (IF DESIRED)

If you use Git Clone using normal Git commands to get a copy of the repository outside of ModusToolbox™ IDE, you can still import all the projects into a workspace as general Eclipse projects. This will allow you to view/copy source code of all the applications without having to import each one as a ModusToolbox™ application. The steps are as follows:

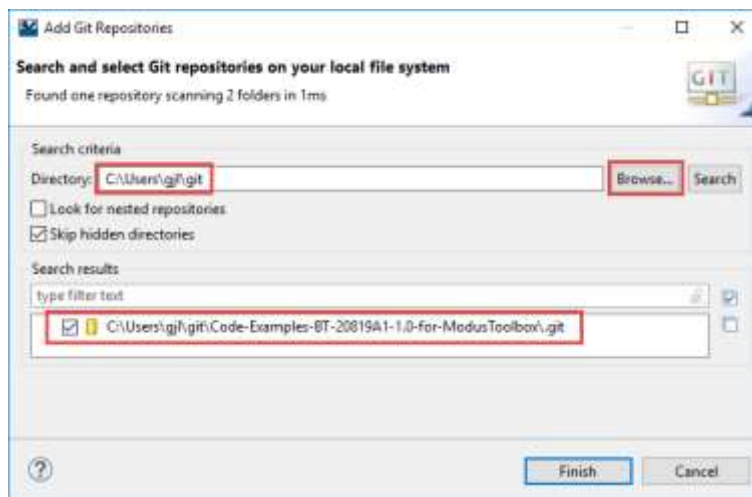
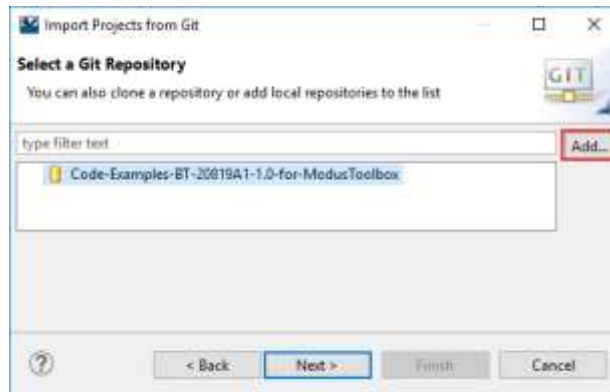
1. Choose File -> Import -> Git -> Projects from Git and click Next



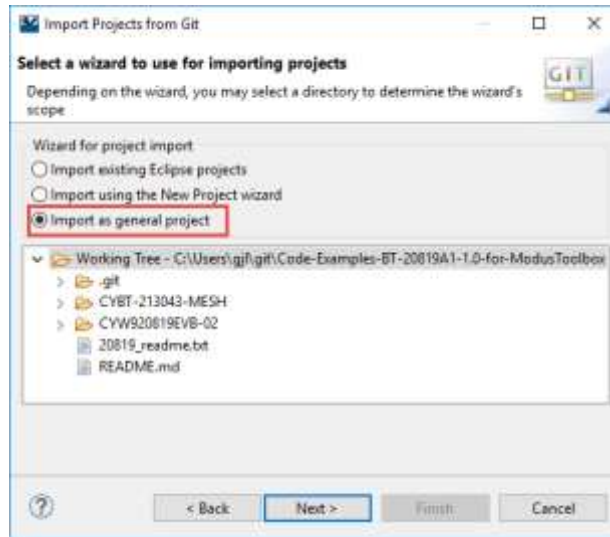
2. Choose Existing local repository and click Next.



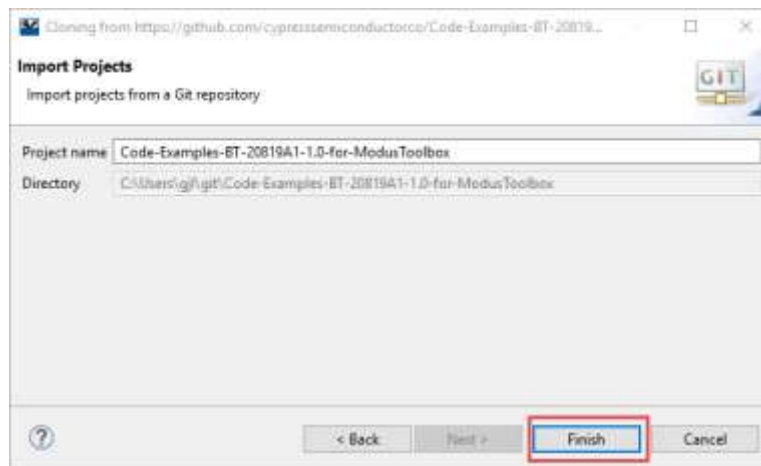
- Click Add then Browse to specify the path to the local Git repository. Check the box next to the repository desired. Click Finish, then Next.



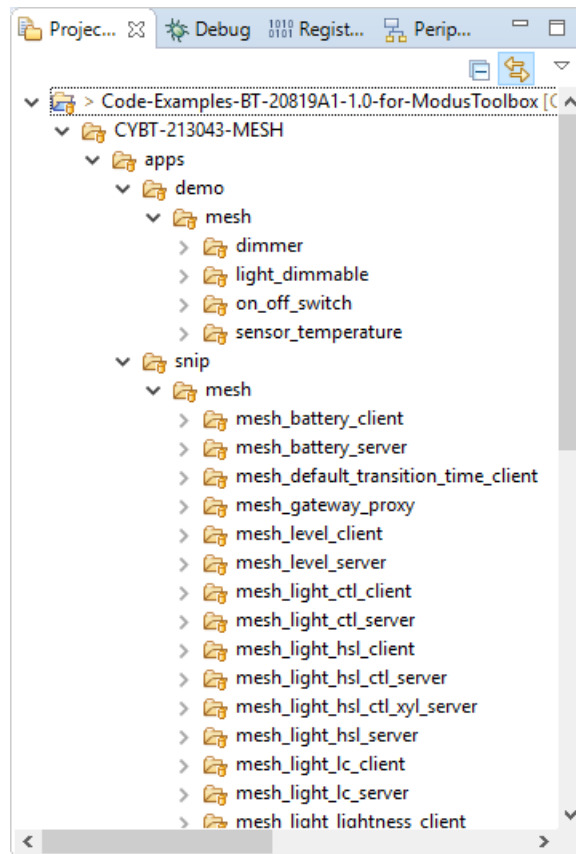
4. Select Import as general project and click Next.



5. Use the provided project name or enter another name of your choice. Click Finish to complete the import.



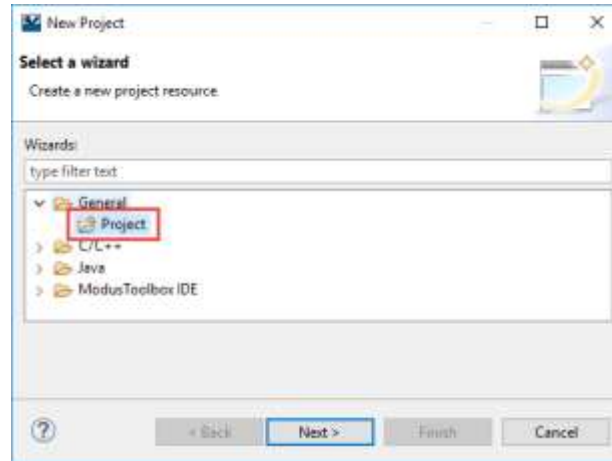
6. Projects will appear hierarchically in the Project Explorer Window as shown below.



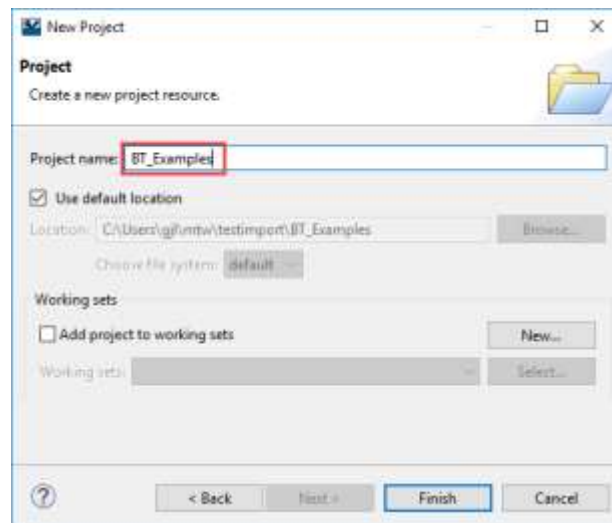
1.2-2C METHOD 3: DOWNLOAD ZIP FROM GITHUB THEN IMPORT ALL ECLIPSE PROJECTS (IF DESIRED)

If you choose to download a zip file from GitHub, save the file to a location on your local machine and unzip it. Once you have done that, if you want to import all the examples as general Eclipse projects you must first create a top-level project and then import the examples. The procedure is as follows:

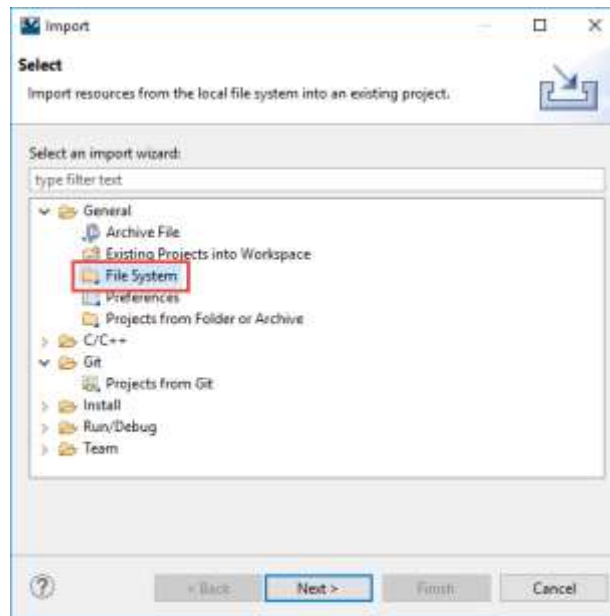
1. Choose File -> New -> Project -> General -> Project and click Next.



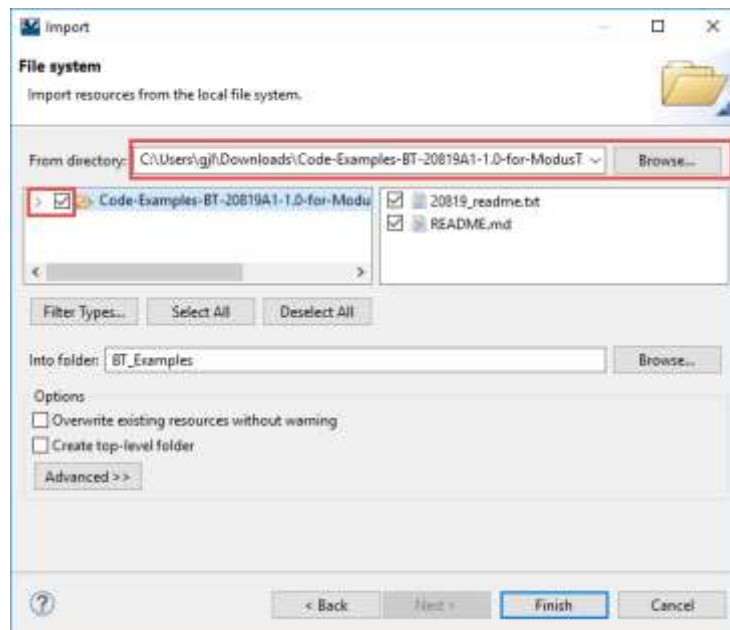
2. Give the project a name. The default location will be your current workspace. Click Finish. This will create a new general Eclipse project.



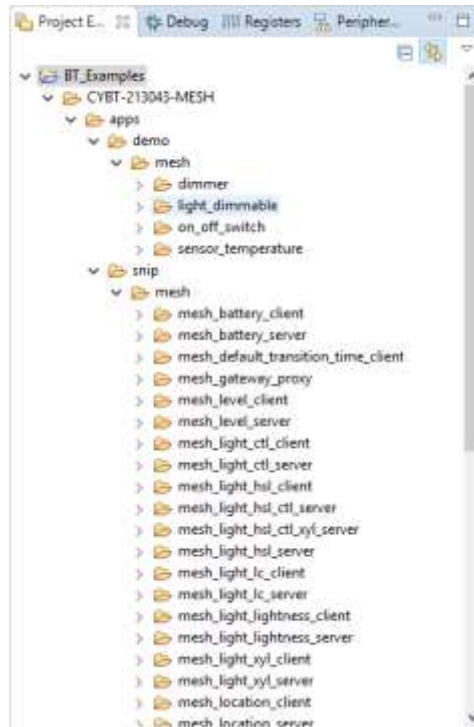
3. Choose File -> Import -> General -> File System.



4. Click Browse to choose the location where you unzipped the file. Check the box next to the folder name and then click Finish.



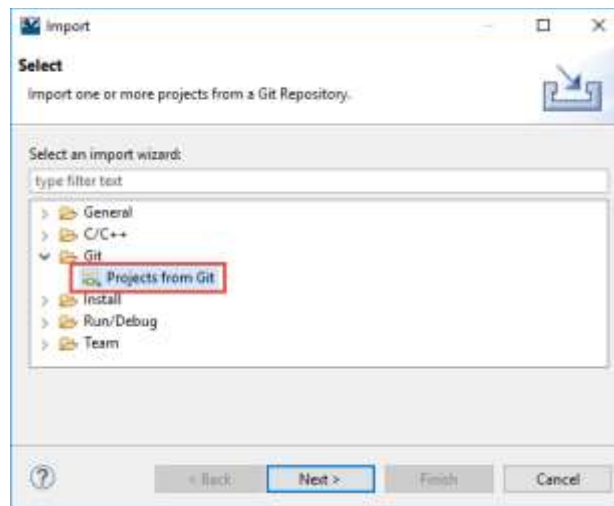
5. Projects will appear hierarchically in the Project Explorer Window as shown below.



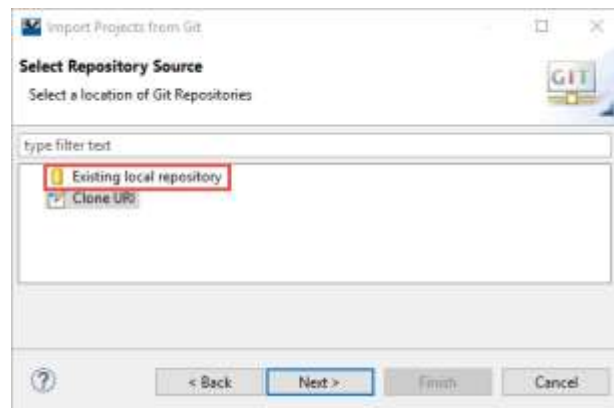
1.2-2D IMPORTING ALL EXAMPLES AS GENERAL ECLIPSE PROJECTS FROM A LOCAL GIT REPOSITORY TO ANOTHER WORKSPACE

If you already have a local Git repository (either from method 1 or method 2) and have already imported the projects as general Eclipse projects but want to import them into another workspace, the procedure is as shown below. (The only difference from what was done on the first import from a Git repository is that you must choose to Import existing Eclipse projects instead of Import as general project.)

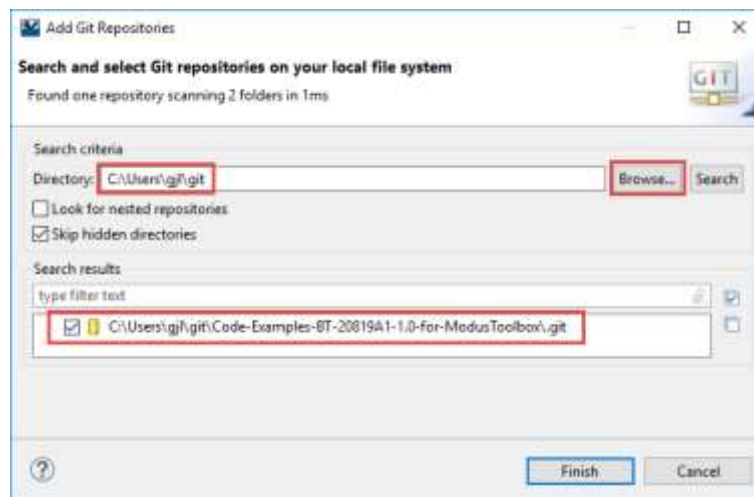
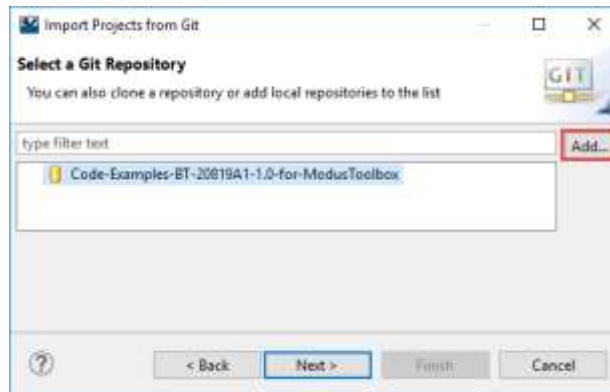
1. Choose File -> Import -> Git -> Projects from Git and click Next



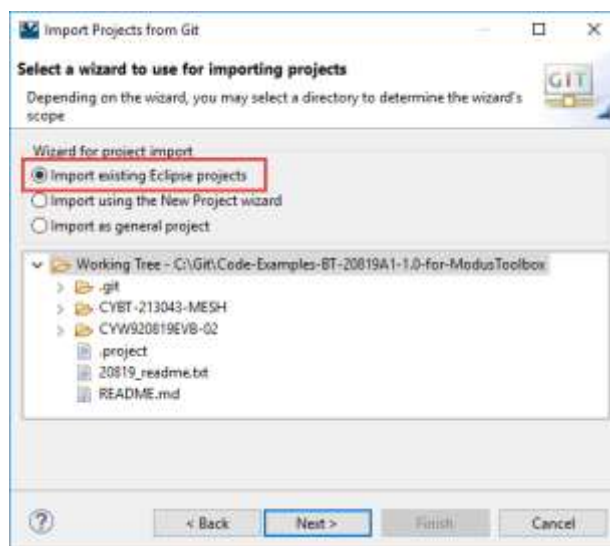
2. Choose Existing local repository and click Next.



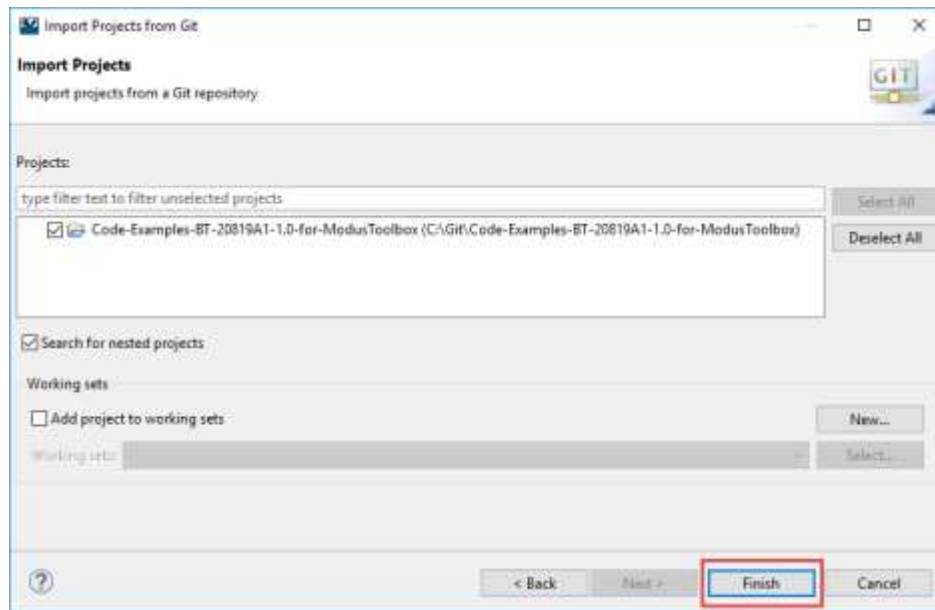
- Click Add then Browse to specify the path to the local Git repository. Check the box next to the repository desired. Click Finish, then Next.



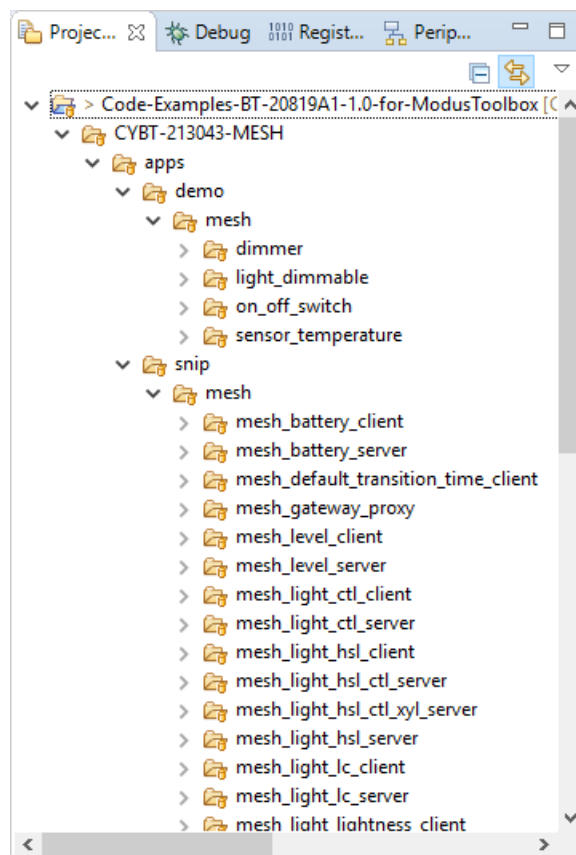
- Select Import existing Eclipse projects and click Next.



- Click Finish to complete the import.



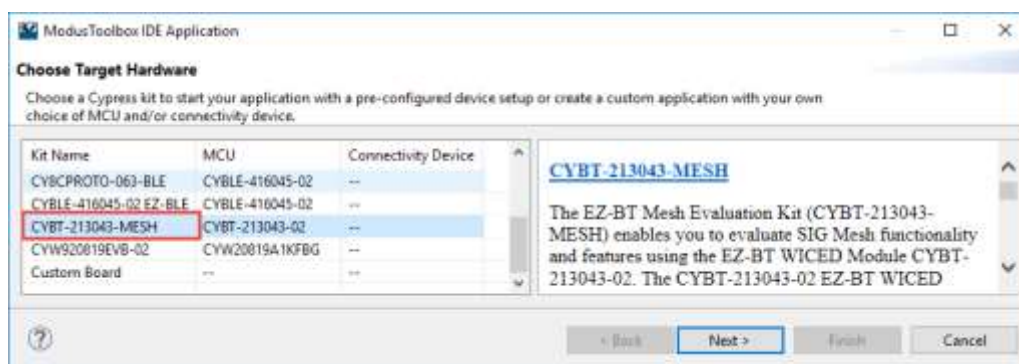
- Projects will appear hierarchically in the Project Explorer Window as shown below.



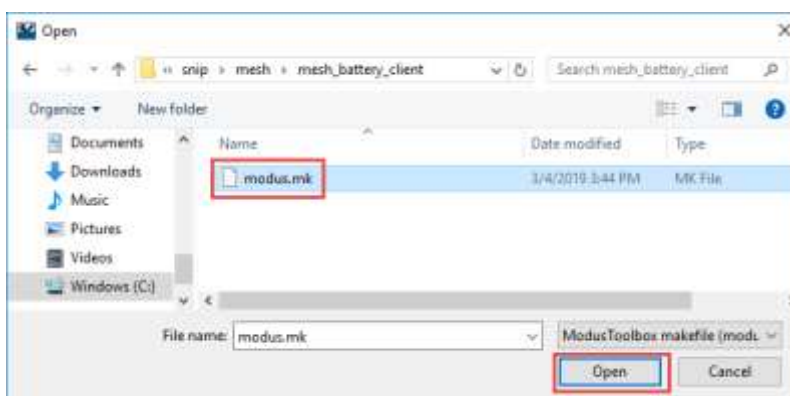
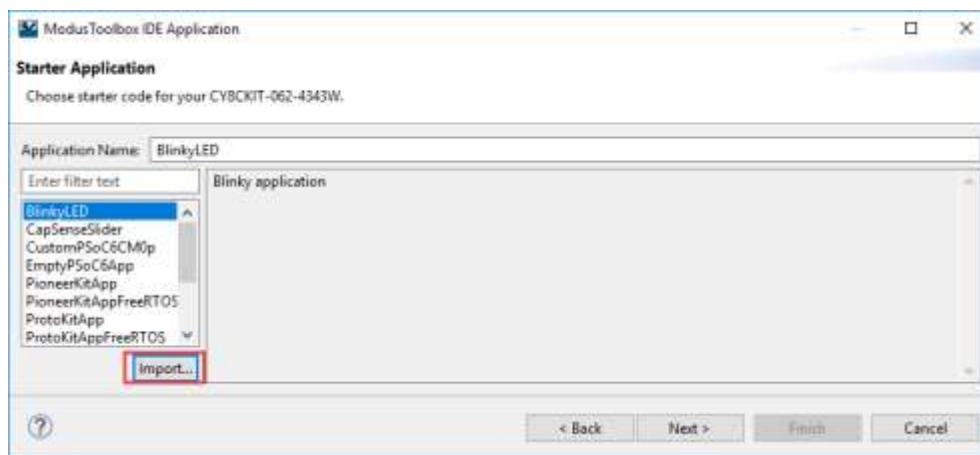
1.2-2E IMPORTING A SINGLE MODUSTOOLBOX™ EXAMPLE APPLICATION

Once you have a copy of the repository (using any of the three methods described above) you just use "New Application" from the Quick Panel or choose File -> New -> ModusToolbox™ IDE Application. Then follow the steps as shown:

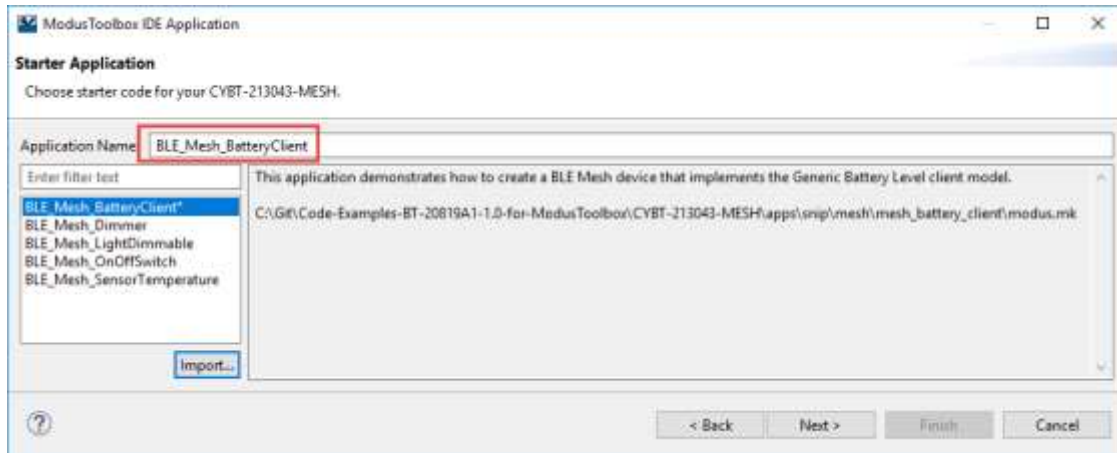
1. Select the desired target hardware and click Next.



2. From the Starter Application window, Click Import and navigate to the desired application. Select the "modus.mk" file and click Open.



3. Change the Application name if desired and click Next.

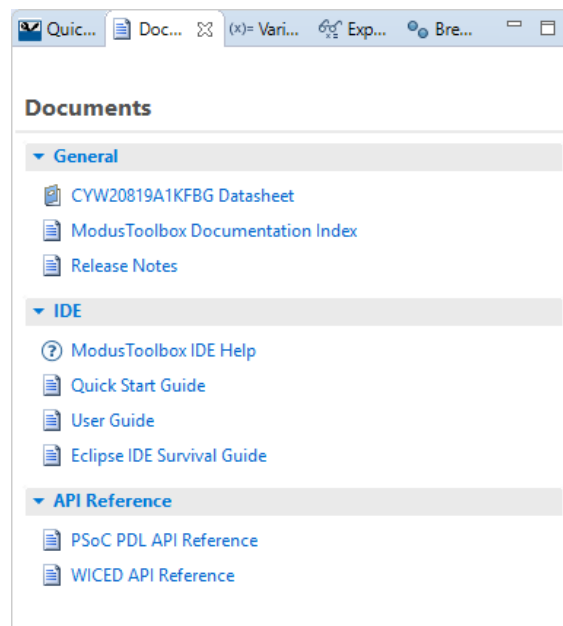


4. Click Finish. This will create a complete ModusToolbox™ application that you can build and program. This can be done as often as you like and into as many workspaces as you want.

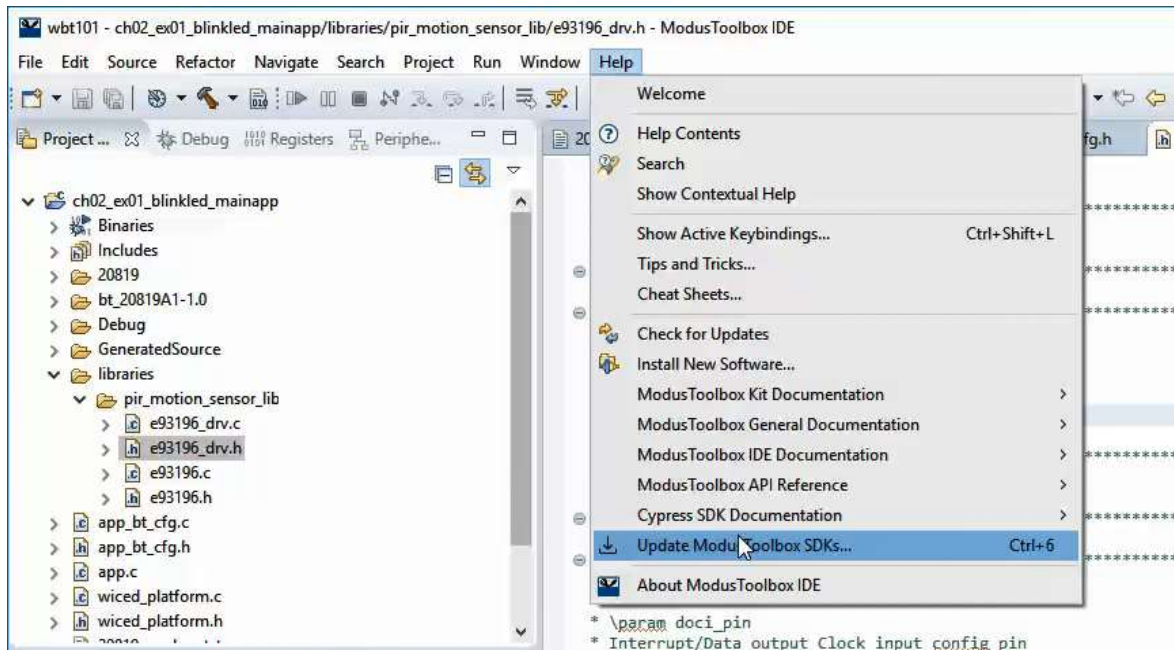
1.3 TOUR OF DOCUMENTATION

1.3-1 IN MODUSTOOLBOX™ IDE

Next to the Quick Panel tab is a tab that contains links to documentation. It includes documentation for the selected device, documentation for the IDE, and API references.

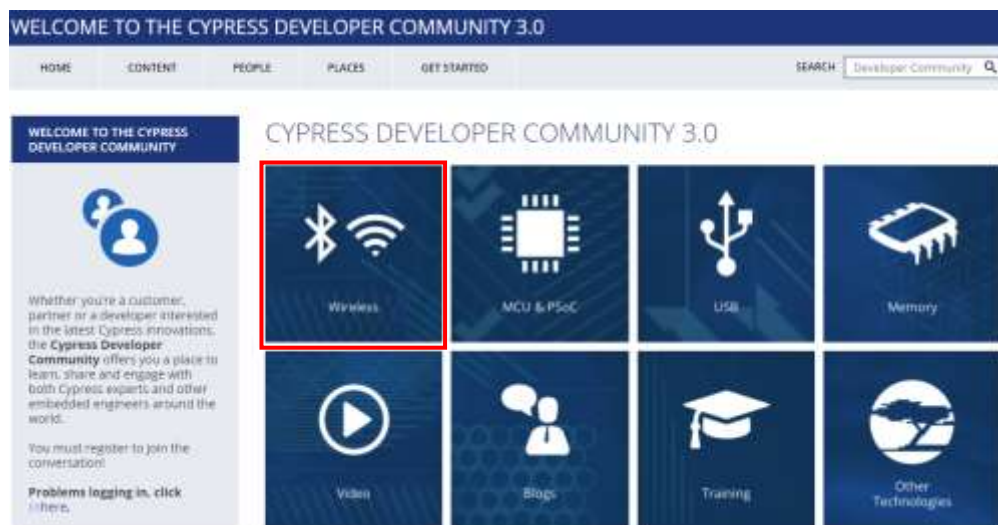


The *Help* menu has most of those links as well as links to kit and SDK documentation. The Help menu also has an item to check for updates to the installed SDKs.



1.3-2 ON THE WEB

Navigating to "www.cypress.com > Design Support > Community" will take you to the following site (the direct link is <https://community.cypress.com/welcome>):



Clicking on the *Wireless* icon will take you to the page as shown below. From this page, you will find links to pages that allow you to download ModusToolbox™, search for answers, ask questions, etc.



From the Wireless Connectivity page, click on ModusToolbox™ Bluetooth to get to the page where you can find downloads for ModusToolbox™ IDE and the latest BT SDK as well as links to product guides, code examples, videos, etc.

WELCOME TO THE CYPRESS DEVELOPER COMMUNITY 3.0

HOME

CONTENT

PEOPLE

PLACES

COMMUNITY INFORMATION

SEARCH
Developer Community

Home > All Places >

ModusToolbox BT SDK

Follow

Overview

Content

People

Subspaces

Actions

About

Share

SEARCH THIS COMMUNITY

Search

BT SDK RESOURCES

CYW20820 Product Guide

CYW20819 Product Guide

CYW20721B2 Product Guide

MODUSTOOLBOX IDE

ModusToolbox

OTHER MODUSTOOLBOX SDKS

ModusToolbox BT_20819A1 SDK

ModusToolbox PSoC 6 SDK

ModusToolbox Mbed SDK

INDIVIDUAL LEADERS

ModusToolbox simplifies embedded systems development by delivering flexibility and easy-to-use tools within a familiar integrated development environment (IDE) under Windows®, macOS®, and Linux®. In addition, it provides a sophisticated platform for system peripheral configuration as well as application-specific configurators for Bluetooth®, CapSense®, and others. For more information on ModusToolbox, go the the [ModusToolbox Community Page](#)

ModusToolbox BT SDK

The BT SDK is targeted for the CYW20819, CYW20820 and CYW20721 ultra low power Bluetooth 5.0 SoCs and the ModusToolbox IDE. ModusToolbox 1.1 with the Bluetooth SDK provides a complete development environment to allow you to quickly create Bluetooth enabled IoT solutions like smart watches, medical devices, or home automation platforms. The BT SDK includes the following:

- Bluetooth firmware
- Platform and board support packages
- Build system
- WICED HCI RPC protocol
- Local tools including BTSpY trace utility and debugger
- Various sample applications

ModusToolbox Downloads

ModusToolbox 1.1 (Windows installer)

ModusToolbox 1.1 (macOS installer)

ModusToolbox 1.1 (Linux installer)

ModusToolbox BT SDK 1.2

BT SDK 1.2 Release Notes

