Chapter 1: Tour of Cypress Bluetooth

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

1.1 Tour of ModusToolbox

ModusToolbox is a set of tools that allow you to develop, program, and debug applications for Cypress MCUs. ModusToolbox IDE is an Eclipse-based development environment that is included as part of the ModusToolbox software installer, but it is not the only supported environment. There is a command line interface (CLI) that can be used interchangeably with IDE applications. In addition, users can use ModusToolbox software with their own preferred IDE, such as Visual Studio Code or IAR.

For this class, we will focus on the ModusToolbox IDE with a little information on using the CLI.

For Windows, ModusToolbox software 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*.

If you install ModusToolbox in a non-default location, you will need to set the CY_TOOLS_PATHS environment variable for your system to point to the *ModusToolbox/tools_2.0* folder or set that variable in each Makefile.

1.1.1 Make/ Build Infrastructure

ModusToolbox uses a GNU "make" based system to create, build, program, and debug projects. The installation provides a set of tools required for all the operations to work. Those tools include:

- Gnu Make 3.81 or newer
- Git 2.20 or newer
- Python3
- GCC
- OpenOCD with supports for Cypress devices
- Configurator tools
- Library manager
- New project GUI and CLI utilities
- ModusShell
- ModusToolbox IDE

These tools can be found in the ModusToolbox installation folder under *tools_<version>* except for ModusToolbox IDE which is under *ide_<version>*.

The general application development flow is as follows. All these steps can be done using an IDE or on the command line. You can even switch back and forth seamlessly between the two.

- Clone the wiced_btsdk (this is a shared library for all Bluetooth applications in a workspace – more on this later)
- Clone the application (either from a GitHub code example or a local set of files)
- Get the libraries required by the application
- Configure the device and its peripherals using configurators
- Write the application code
- Build/Program/Debug

1.2   Proxy Settings

When creating applications from inside a Cypress internal network, some proxy settings are necessary due to our security settings. This behavior will be improved in the future. The steps (for now) are:

1. Find the proxy host name and port number for your site. There is a list at:

    https://itsm.cypress.com/solutions/726105.portal

2. Make OS specific proxy and variable updates:
    a. Windows:
        i.   Set the Windows proxy settings (Settings > Network & Internet > Proxy) to "Automatically Detect Settings"
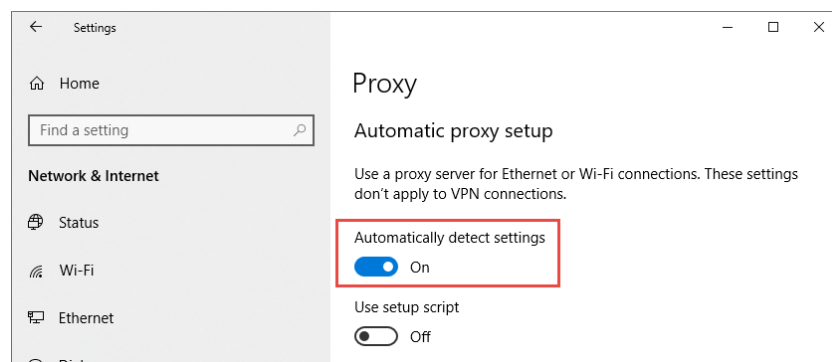
        ii. Create two Windows System variables with the proxy name and port number (Settings > Search for System Environment Variables > Environment Variables…). For example, in Oregon:

b. <u>MacOS</u>:
  i. Set "System Preferences > Network > Advanced > Proxies > Auto Proxy Discovery"
  ii. Open a terminal and check if the variables (http_proxy, https_proxy) are set in the terminal.

```
$ export | grep proxy
declare -x http_proxy="bigbro.ore.cypress.com:74/"
declare -x https_proxy="bigbro.ore.cypress.com:74/"
```
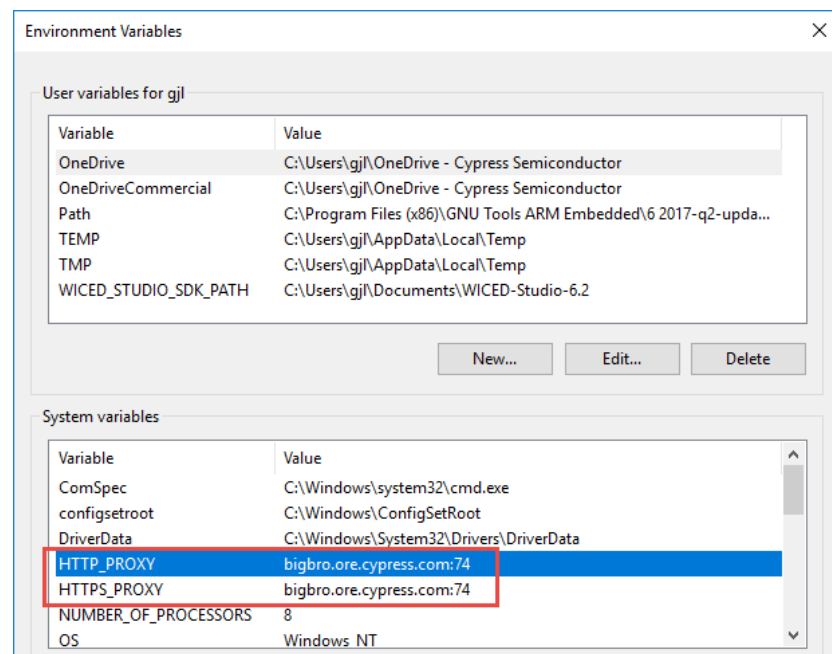
c. <u>Linux</u>:
  i. Set the "Network Proxy" to "Manual"
  ii. Open a eterminal and check if the variables are set in the terminal.

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

  iii. Setup CA Certification (Development environment#SetupCypressCACertificates)
3. Start the ModusToolbox IDE
  a. Open menu item "Window > Preferences > General > Network Connections"
   i. Set "Active Provider" to "Manual" and set HTTP and HTTPS to the correct proxy host and port for your location.
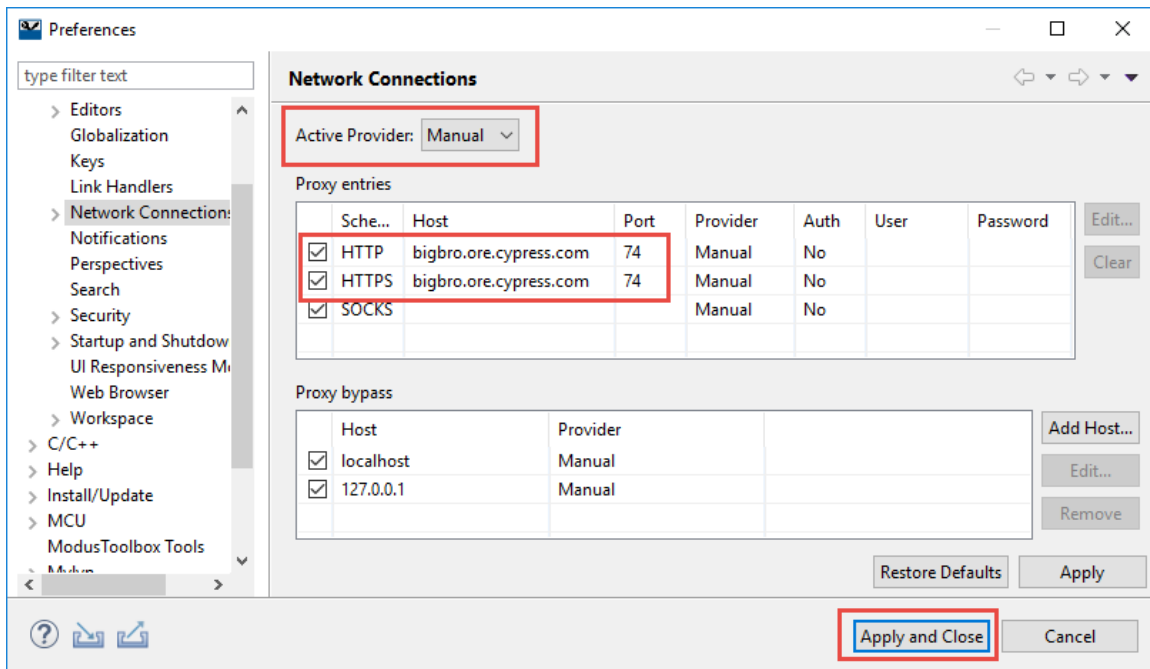
1.3   Tour of ModusToolbox IDE

1.3.1 First Look

When you first run the ModusToolbox IDE, you will create a Workspace to hold your applications. The default Workspace location is *C:/Users/<UserName>/mtw*, but you can have as many Workspaces as you want, and you can put them anywhere you want. I usually put each workspace under a folder inside *C:/Users/<UserName>/mtw* such as *wbt101* for exercise for this class.

Each time you open ModusToolbox IDE you will need to provide a workspace name such as the following:
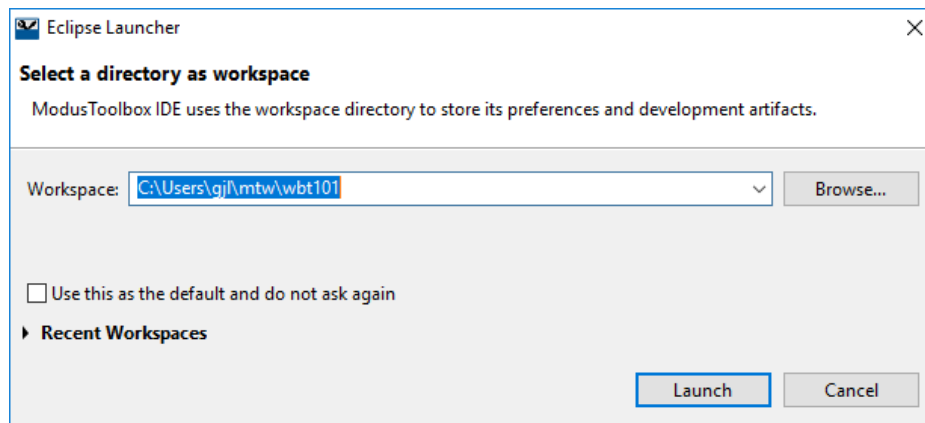
Whenever you want to switch to a different Workspace or create a new one, just use the menu item *File->Switch Workspace* and either select one from the list or select *Other…* to specify the name of an existing or new Workspace.

After clicking Launch, the IDE will start, and you will see the (empty) ModusToolbox perspective.

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*.
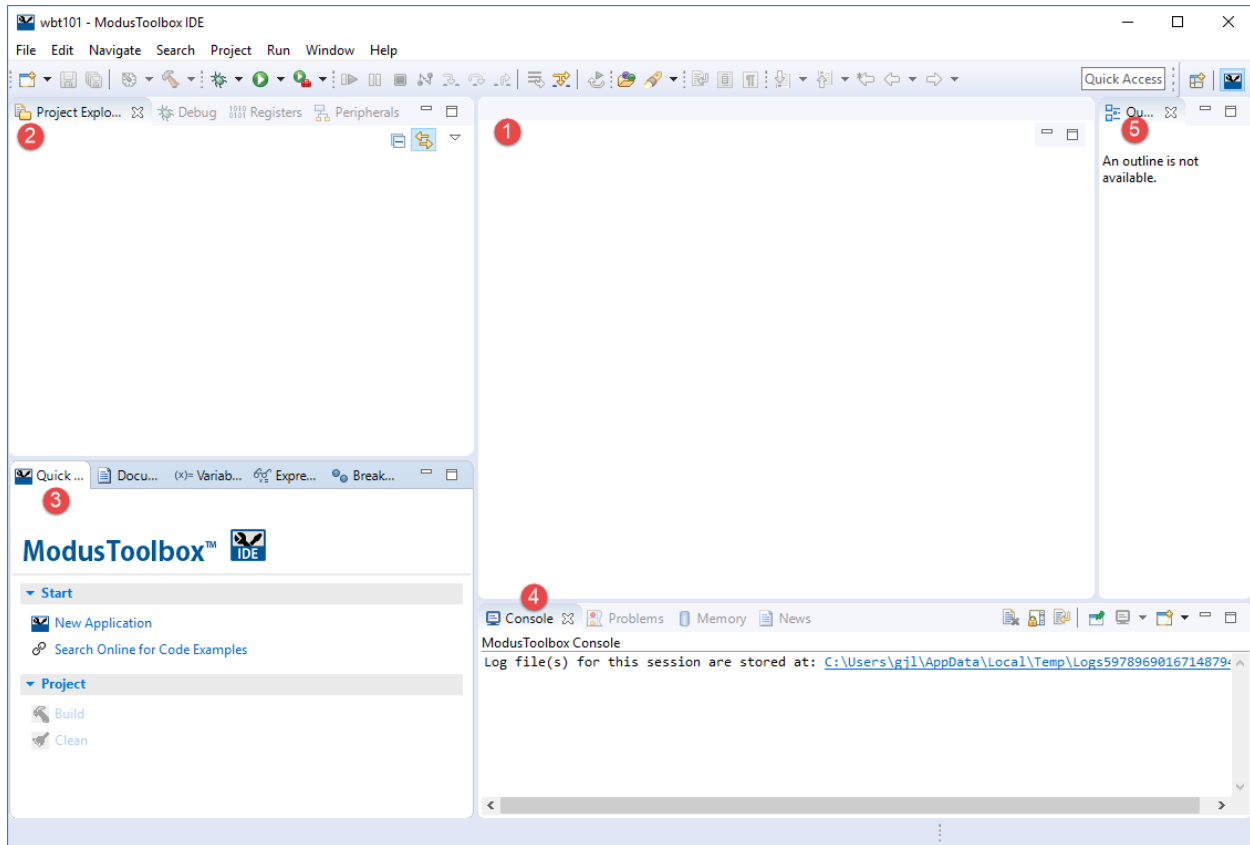
The major views are:

1. File Viewer/Editor
2. Project Explorer
3. Quick Panel
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.

1.3.2 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.
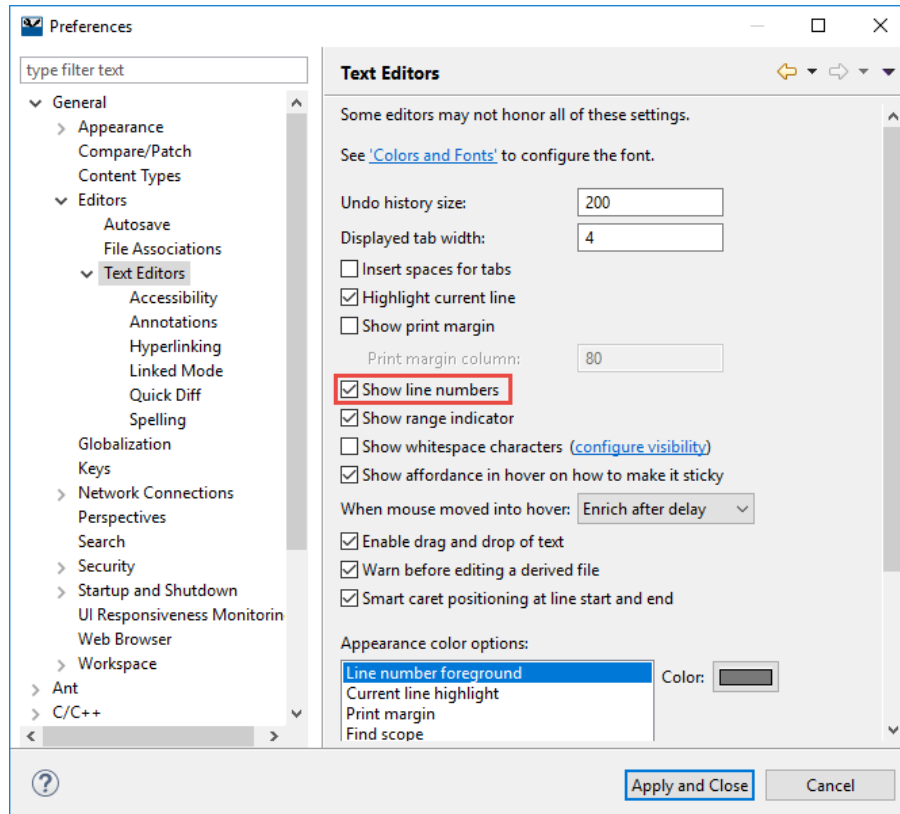
There are also project build settings which we will explore later.

1.3.3 Bluetooth SDK Architecture

The Bluetooth SDK is downloaded from GitHub just like the other project dependencies. In the case of the SDK, it is contained in a separate Eclipse project called wiced_btsdk. This project referenced by all the applications in your workspace. That is, the wiced_btsdk is shared among all applications in a single workspace.

The best practice is to create the wiced_btsdk project first, but in future versions of the IDE and CLI the project will be created automatically for you if it isn't found. It doesn't matter which hardware you select for the wiced_sdk project – one copy will work for all boards – but the name must NOT be changed.

The SDK contains the base libraries for all the supported devices (wiced_btsdk/dev-kit/baselib), board support packages (BSPs) for all of the supported boards (wiced_btsdk/dev-kit/bsp), and Bluetooth tools such as ClientControl and BTSpy which you will use in later chapters (wiced_btsdk/tools).
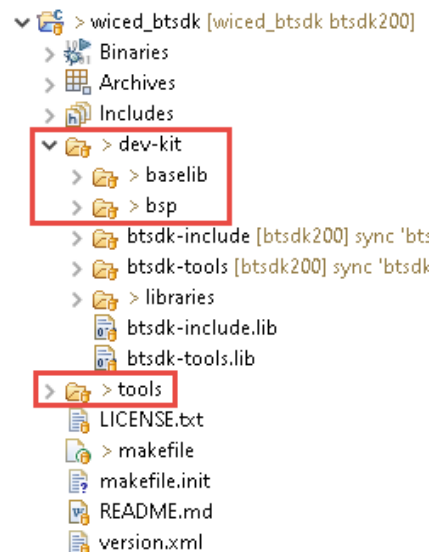
Since the SDK is shared among all applications in a workspace, any changes to files contained in the SDK will affect all applications in that workspace. This is especially important for the BSP since it contains the device configurator files.

Therefore, if you want to use the device configurator to change settings for an application without affecting other applications, you can do one of three things: (1) override the device configurator files for a single application; (2) create a custom BSP inside the wiced_btsdk; or (3) create a new workspace with a new copy of the wiced_btsdk that you can modify. The first is good when a single application requires a different configuration while 2 and 3 are better if you want a single custom configuration for multiple applications.

The steps to override the device configurator files for a single application are:

1. Copy the *wiced_btsdk/dev-kit/bsp/TARGET_<bsp_name>/COMPONENT_bsp_design_modus* folder and its contents to the application's project folder (the folder with the top-level source code). Change the name to something unique that still starts with COMPONENT_. For example:

   COMPONENT_my_design_modus

2. Disable the configurator files from the BSP and include your custom configurator files by adding the following to the makefile (the makefile already has a blank line that says COMPONENTS += so you can add your component name to that line). Replace "my_design_modus" with whatever name you used for the folder (excluding COMPONENTS_).

   ```
   COMPONENTS += my_design_modus
   DISABLE_COMPONENTS += bsp_design_modus
   ```

3. Make sure your project is selected and open the device configurator from the Quick Panel. Make the required changes and save/exit the configurator.

The steps to create a custom BSP are:

1. Copy an existing BSP and paste it with a new name in the same folder, which is wiced_btsdk/dev-kit/bsp.
2. Edit the makefile for the application to include the new BSP in the TARGET, SUPPORTED_TARGETS, and TARGET_DEVICE_MAP variables.
3. Make sure your project is selected and open the device configurator from the Quick Panel. Make the required changes and save/exit the configurator.

1.3.4 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*.

When you create a new application, the appropriate files for the application and its libraries are cloned from GitHub (you can use local application files as you will see later, but even in that case the libraries are pulled from a repository). This means that you can always get the latest version of a code example or library (or you can get older versions if you prefer). The downside is that you need an internet connection when creating an application.

There is a super-manifest file that contains a list of manifest files. The manifest files in turn, list the locations/versions of repos where the tools should search for applications and libraries. The default super-manifest and manifest files allow the tool to find all the SDK libraries, middleware libraries, code examples, and board support packages that are supported by Cypress. Both the super-manifest and manifest are XML files.

The default super-manifest can be found on GitHub in this repo:

https://github.com/cypresssemiconductorco/mtb-super-manifest

View that file to see the location, contents, and format of the default manifest files.

If you want to include your own repos in the search path (e.g. to create/share your own BSPs, libraries, projects, etc.), you can create a file in ~/.modustoolbox/manifest.loc that lists your own super-manifest file(s). Your super-manifest files can then point to your own manifest file(s). The paths in your super-manifest and manifest(s) must point to an HTTP URL (which can, for example, be a GitHub repo or file). Eventually, it will be possible to specify a local file as well as an HTTP path.

Board Support Package

First you must choose the "Board Support Package (BSP)". 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.
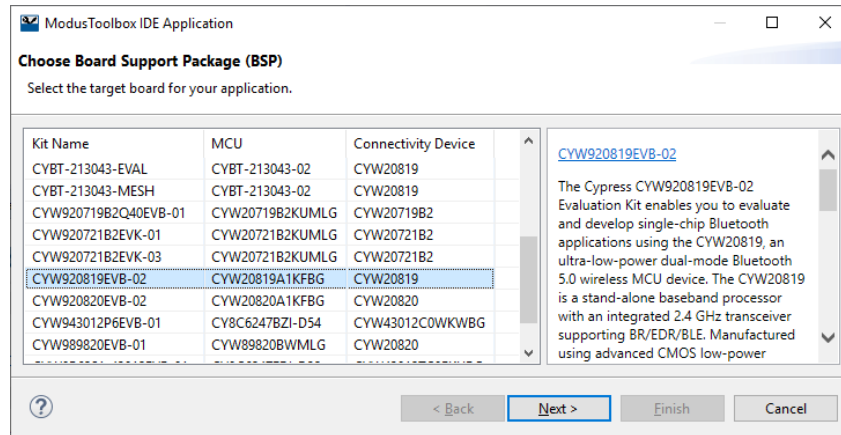
Starter Application

Clicking "Next" lets you choose a 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 ta fully functional demos. For this class we will provide you with template starter applications for many of the exercises. You will access these using the "Import…" button in new application wizard.
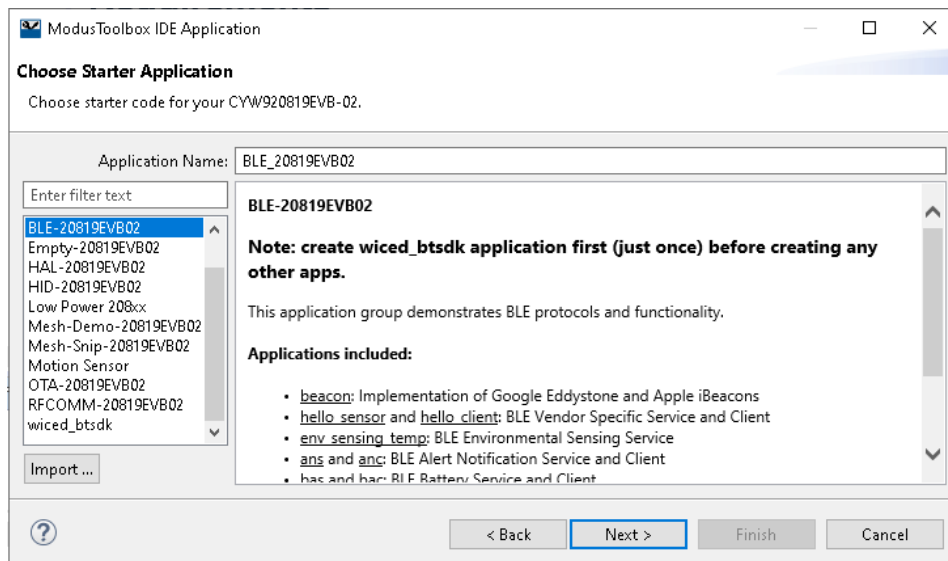
There is no real equivalent to the PSoC Creator empty project, but there is an Empty application that gives you just enough to get started. Unlike the PSoC Creator empty project, using this as a starting point is more of an expert-level approach since it doesn't provide much in the way of code. In most cases, we recommend starting with one of the code examples and modifying it as needed for your application.

Each starter application or set of applications 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.

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.
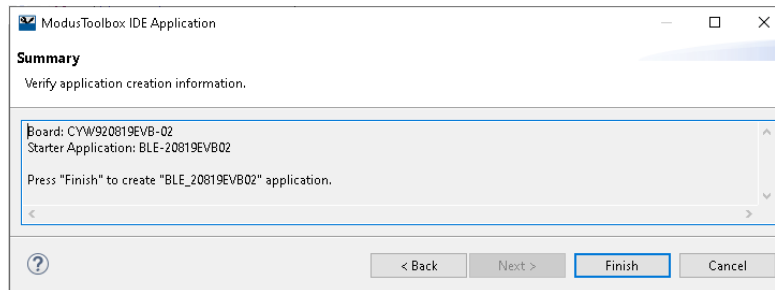
Application Sets

Most starter applications are actually a "set" of applications instead of an individual application. For example, the BLE_20819EVB02 starter application contains 12 different BLE applications that are all created when you use that set as the starter application. For that reason, the name you provide is used as the prefix of the applications for that set. For example, BLE_20819EVB02 will give you projects called:

> BLE_20819EVB02.anc
> BLE_20819EVB02.ans
> BLE_20819EVB02.bac
> BLE_20819EVB02.bas
> BLE_20819EVB02.beacon
>
> …

On disk, this is done using hierarchy. For the example described above, the directory structure is:

> BLE_20819EVB02/ble/anc/<application_files>
> BLE_20819EVB02/ble/ans/<application_files>
> BLE_20819EVB02/ble/bac/<application_files>
> BLE_20819EVB02/ble/bas/<application_files>
> BLE_20819EVB02/ble/beacon/<application_files>
>
> …
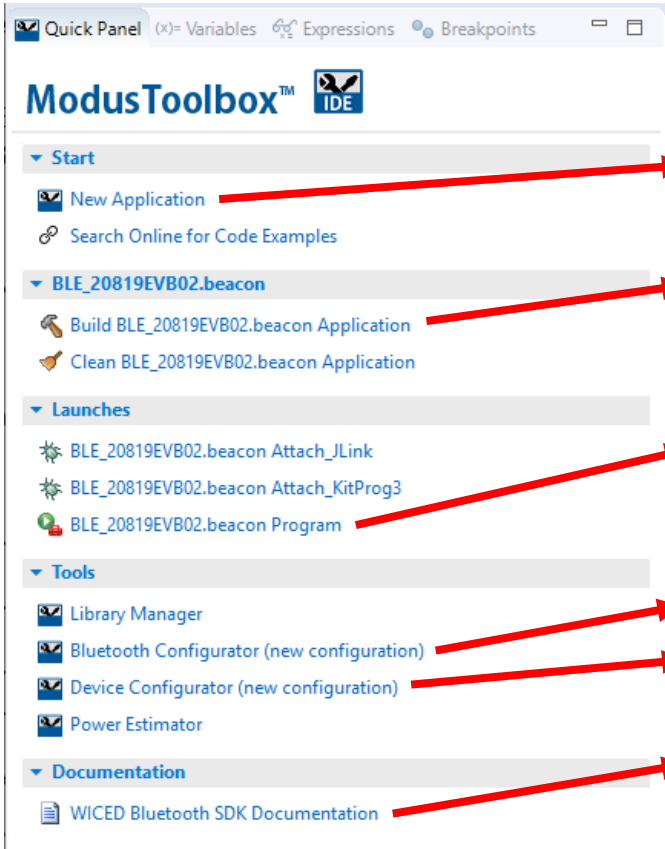
The lowest level of the hierarchy is where the individual applications are stored. Got to that directory if you want to use the command line interface. Note that the middle level of the directory hierarchy (ble in this example) is not used in the Eclipse application name.

For the exercises in this class, we will use single application templates, so we will not include the extra hierarchy. In that case, the makefile is modified slightly from the starter applications – more on that later.

1.3.5 Quick Panel

Once you have created a new application the Quick Panel is populated with common commands so that you don't have to hunt for them. There are top level commands, application level commands, links to configurators, and links to launches (for program and debug). There is also a section with links to documentation.

Equivalent Menu Path

File -> New -> New ModusToolbox IDE Application

Select application project in Project Explorer, then Project -> Build

Run->External Tools -> External Tools Configurations -> Program   -> <appname>

Double click *.cybt in the application project

Double click design.modus in the BSP

Help -> ModusToolbox General Documentation
Help -> ModusToolbox IDE Documentation

1.3.6 Applications

At this point, we are ready to start developing the application. As mentioned before, a WICED Bluetooth application is the application project and the wiced_btsdk project. We looked briefly at wiced_btsdk earlier, so let's look at the application project now.

Project Explorer

In the project explorer window, you will see your application project and all its associated files.
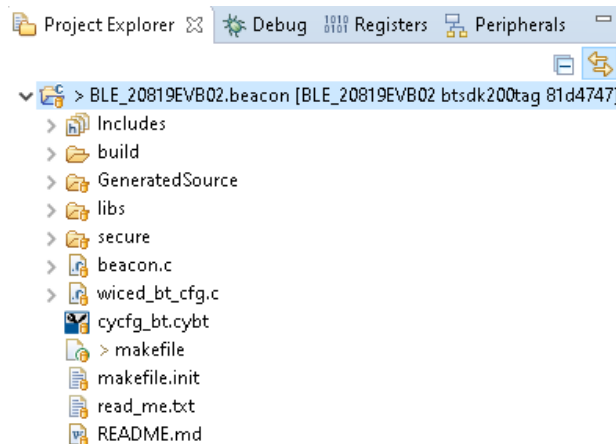
The key parts of a project are:
- A folder with the name of the project
- Readme files (read_me.txt and/or README.md)
- Application configurator files (e.g. cycfg_bt.cybt for Bluetooth)
- A GeneratedSource folder with the output files from the application level configurators
- A libs folder
- makefile
- Stack configuration files (e.g. wiced_bt_cfg.c or app_bt_cfg.c)
- Application C source files

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.

Application Configurator Files

Files for configurators are distinguished by their file extension. For example, any file that ends with *.cybt* is a Bluetooth configurator file.

GeneratedSource Folder

When you save from a configurator, the tool generates firmware in the GeneratedSource folder. The following are the most interesting/useful files (e.g. cycfg_gatt_db.c and cycfg_gatt_db.h for BLE).

libs

For Bluetooth starter applications, this folder just contains a file named index.html. It is used to populate the links in the Documents tab in the quick panel.

makefile

The makefile is used in the application creation process – it defines everything that ModusToolbox needs to know to create/build/program the application. This file is interchangeable between ModusToolbox IDE and the Command Line Interface so once you create an application, you can go back and forth between the IDE and CLI at will.

Various build settings can be set in the makefile by the user to change the build system behavior. These can be "make" settings or they can be settings that are passed to the compiler. Some examples are:

- Target Device (`TARGET`=CYW920819EVB-02)
- Relative path to the wiced_btsdk (`CY_SHARED_PATH`=`$(CY_APP_PATH)`/../../../wiced_btsdk)
- Method to determine the Bluetooth Device Address (`BT_DEVICE_ADDRESS?`=default)
- Enable OTA Firmware Upgrade (`OTA_FW_UPGRADE?`=1)
- Compiler Setting to Enabling Debug Trace Printing (`CY_APP_DEFINES+`= -DWICED_BT_TRACE_ENABLE)

You will get to experiment with some of these settings in later chapters.

In particular note the ../../../ in the CY_SHARED_PATH. That accounts for the hierarchy in the Bluetooth starter applications. In the templates we use in later chapters, the hierarchy will be removed so that will be changed to just ../

Stack Configuration Files

Many of the templates you will use in this class 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 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.

Application C file

All starter applications include at least one C source file that starts up the application (from the application_start() function) and then implements other application functionality. The actual name of this file varies according to the starter application 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 always 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.

The application C file begins with various #include lines depending on the resources used in your application. These header files can be found in the SDK under wiced_btsdk/dev-kit/baselib/20819A1/include. 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 "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_rtos.h"               // RTOS functions
#include "wiced_hal_nvram.h"          // NVRAM drivers
#include "wiced_hal_wdog.h"           // Watchdog
#include "wiced_spar_utils.h"         // Required for stdio functions such as snprint and snpritf
#include <stdio.h>                    // Stdio C functions such as snprint and snpritf
```

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, 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 wiced_btsdk/dev-kit/baselib/20819A1/include/wiced_bt_dev.h.

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

```c
#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;
}
```

1.3.7 Device Configuration

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. As mentioned earlier, the design.modus file is shared by all applications in a workspace that use a given BSP. Keep that in mind if you chose to edit the design.modus file.

Because of that, we will not use the Device Configurator in the exercises, but we will show you how it can be used.

To launch the Device Configurator click on the link in the Quick Panel or double-click on the design.modus file in the BSP for your application (wiced_btsdk/dev-kit/bsp/<BSP Name>. As you can see there 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).
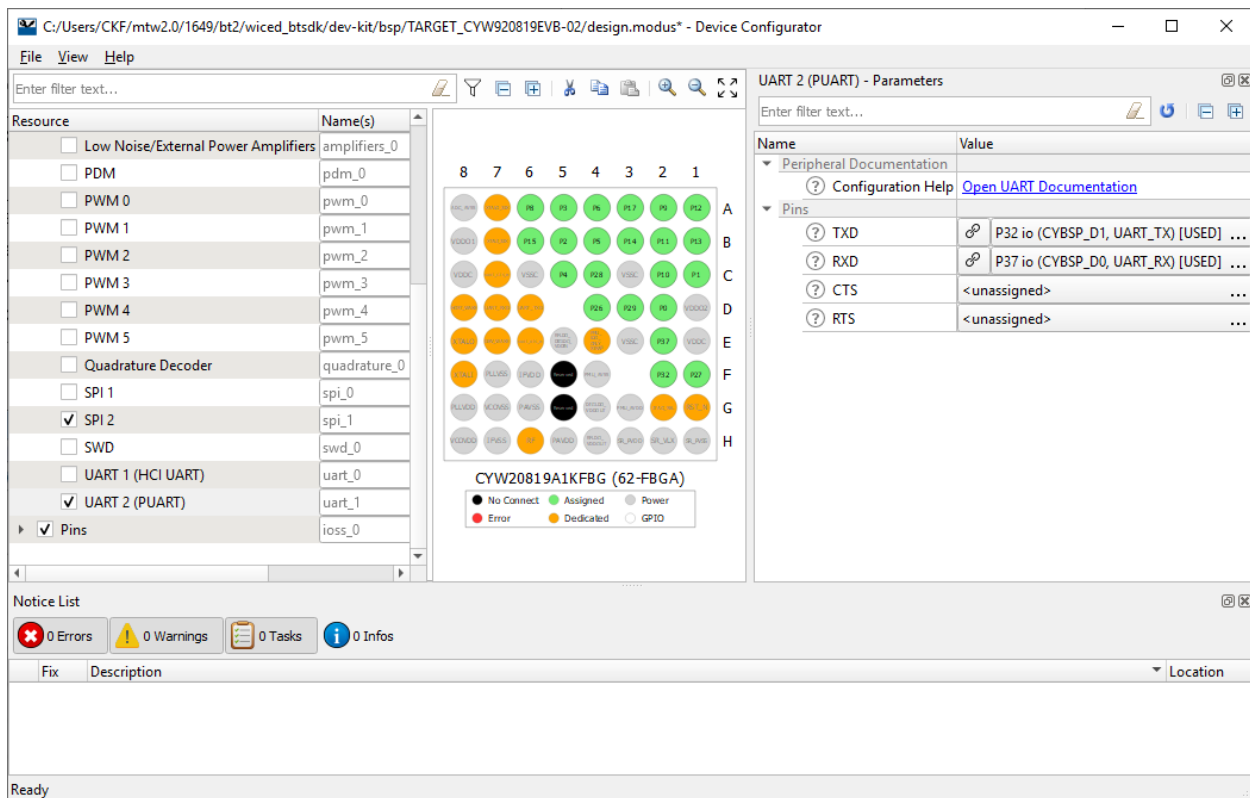
Once you save the configurator information (*File->Save*) it creates/updates the Generated Source files in the project BSP.

1.3.8 Library Manager

There is a Library Manager that is intended to add/remove individual libraries from an application. However, for BT applications, the libraries are all currently shared (from wiced_btsdk) so the library manager isn't useful (at least not yet).

Instead, the way to add a new library to an application is as follows (you will get to try this out in later chapters):

1. Add the appropriate include in your source code.
2. Add the library name to the COMPONENTS variable in the makefile.
3. Add the path to the library to the SEARCH_LIBS_AND_INCLUDES variable in the makefile.

For example, to add the over-the-air (OTA) firmware upgrade library, you would add:

1. source code:

```
#include "wiced_bt_ota_firmware_upgrade.h"
```

2. makefile:

```
COMPONENTS+=fw_upgrade_lib
.
.
.
SEARCH_LIBS_AND_INCLUDES+=$(CY_SHARED_PATH)/dev-kit/libraries/btsdk-ota
```

Note: for future reference, the required makefile edits are also shown at the bottom of the README.md file in the wiced_btsdk project or BT starter applications.

1.3.9 Renaming / Copying Projects

You can rename a project from inside the Eclipse IDE by right clicking on the project in the project explorer and selecting "Rename" (don't rename wiced_btsdk though – that will break things). This renames the project as it shows up in Eclipse, but it does NOT change the name of the folders in the filesystem. The Eclipse project name can be found in the *.project* file inside the project folder.

You can also copy/paste but you must keep the relative path to the wiced_btsdk the same (or else change the path in the makefile to match the new relative path). This is difficult to get just right inside the IDE so it is easier to copy/paste from a file window (Windows explorer, etc.) and then update the project name in the *.project* file. Once you do that, you can import the project into Eclipse using *File > Import > General > Existing Projects into Workspace*.

After either a rename or a copy/paste, the Launches section for the new project in the Quick Panel will be empty. The launches can be recreated from the command line (more on the CLI in a minute) by running the command:

    make eclipse CY_IDE_PRJNAME=<IDE project name>

where <IDE project name> is the name of the new project.

1.3.10 Sharing Applications

Applications can be shared by checking them into revision control systems such as Git. Another way is to export 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. However, once you import from the archive file, the hierarchy will be flattened. In order for the resulting project to be built, a change is required in the makefile to update the relative path to the wiced_btsdk. You'll see that step in the next section. The steps to export into an archive file are:

Export to Archive

1. Select the project or projects to be exported. Note that you may or may not want to export the wiced_btsdk project. It is often easier to re-create it in a new workspace.
2. Choose *File->Export->General->Archive File* and click Next.
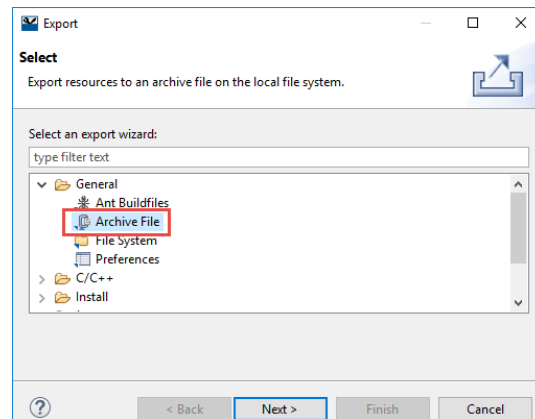
3. Uncheck the build folder if it exists to save space (this contains build output files).
4. Enter the desired file path/name for the archive file.
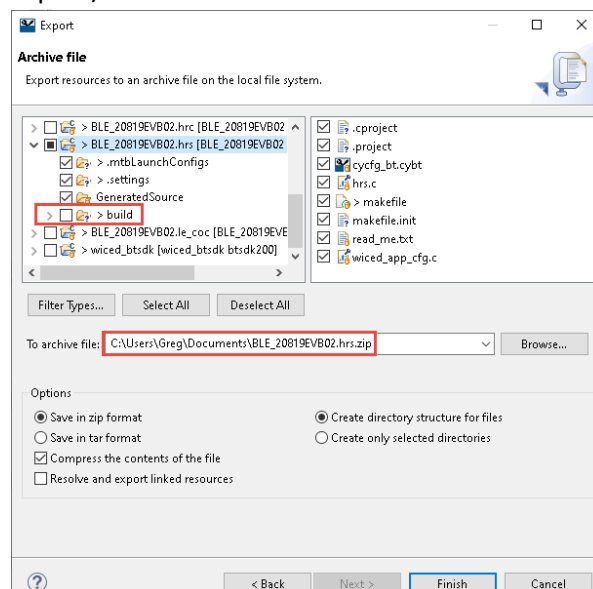
5. Click Finish.

Import from Archive

1. Create a new workspace if you don't have one already.
2. Create the wiced_btsdk project using the normal method in the new workspace unless you are importing it from the archive.
3. In the new workspace choose *File->Import->General->Existing Projects into Workspace.*
   a. DO NOT choose ~~*File->Import->General->Archive File*~~!
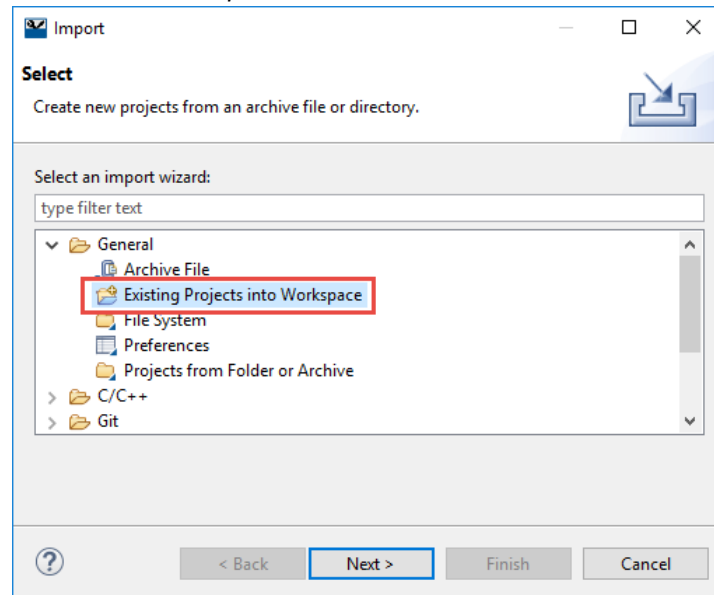
4. Choose *Select archive file* and specify the path to the zip file.
5. Check the box next to the project(s) you want to import.
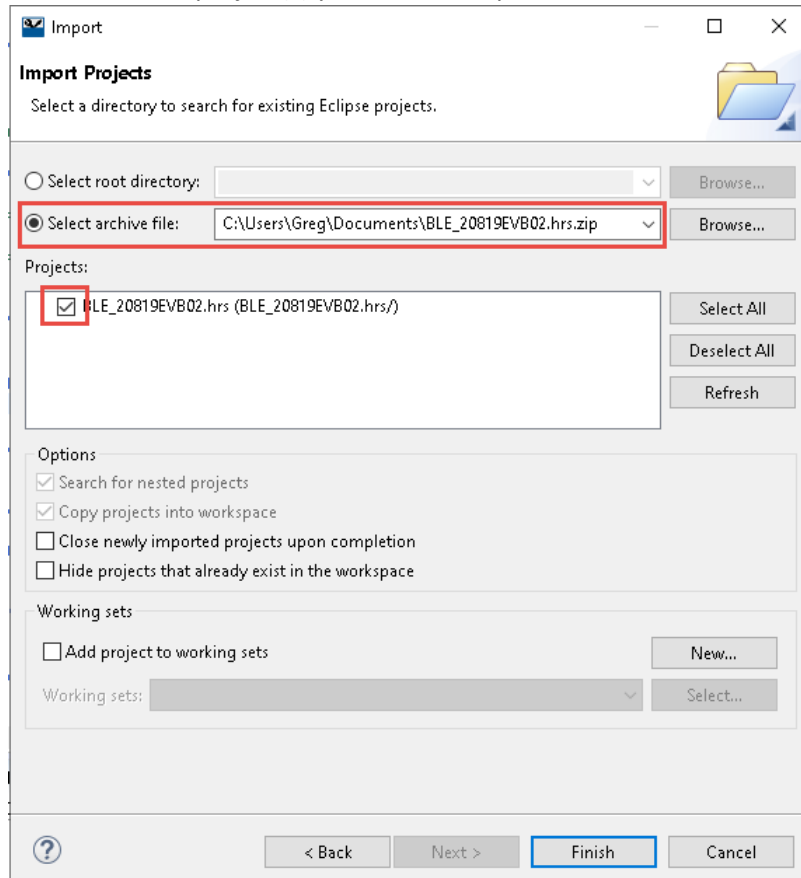
6. Click Finish.
7. <u>While the project name will be the same, the hierarchy will be different from the starting project</u>. For the example shown above, the hierarchy of the starting project is:

           <workspace>/BLE_20819EVB02/ble/hrs/<project files>

After importing using this method, the hierarchy of the new project will be:

           <workspace>/BLE_20819EVB02.hrs/<project files>

Due to this change, the wiced_btsdk path is now ../wiced_btsdk instead of ../../../wiced_btsdk. Therefore, to allow the new project to be built, open the makefile and change the CY_SHARED_PATH as follows:

   CY_SHARED_PATH=$(CY_APP_PATH)/../wiced_btsdk

Import from Filesystem

Yet another way to share a full application is to copy the project in a file window (e.g. windows explorer) to the new workspace and then import from inside Eclipse without copying the project to the workspace. This method will keep the existing folder hierarchy so that the makefile need not be changed.

1. Create a new workspace if you don't have one already.
2. Create the wiced_btsdk project using the normal method in the new workspace unless you are importing it from the old workspace.
3. Copy the desired project or projects and their associated hierarchy to the new workspace using the filesystem (e.g Windows explorer). You can change the names of folders if you wish during this step. Remember that the project name in Eclipse will be taken from the .project file in the project directory.
4. In the new workspace choose *File->Import->General->Existing Projects into Workspace* and click Next.
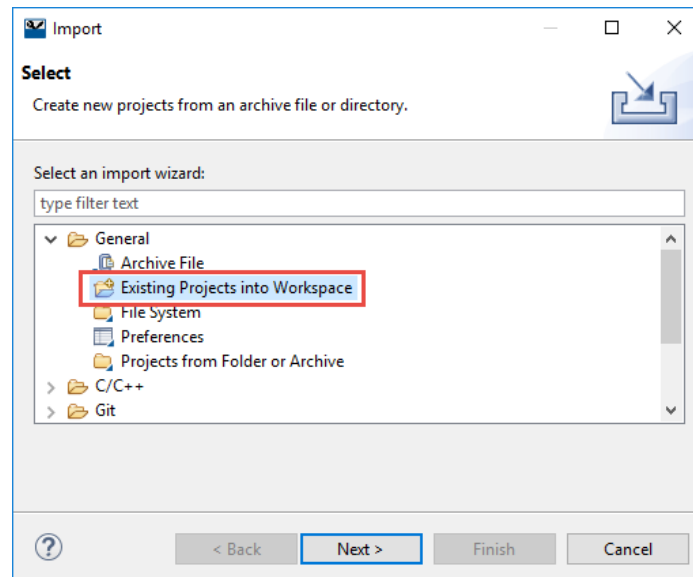
5. Choose the path to the project or projects to be imported. This should be the path to where you copied the project(s) in the NEW workspace.
6. Check the box next to the project or projects you want to import.
7. <u>Uncheck the box</u> *Copy projects into workspace* since you already copied the to the workspace using the filesystem.
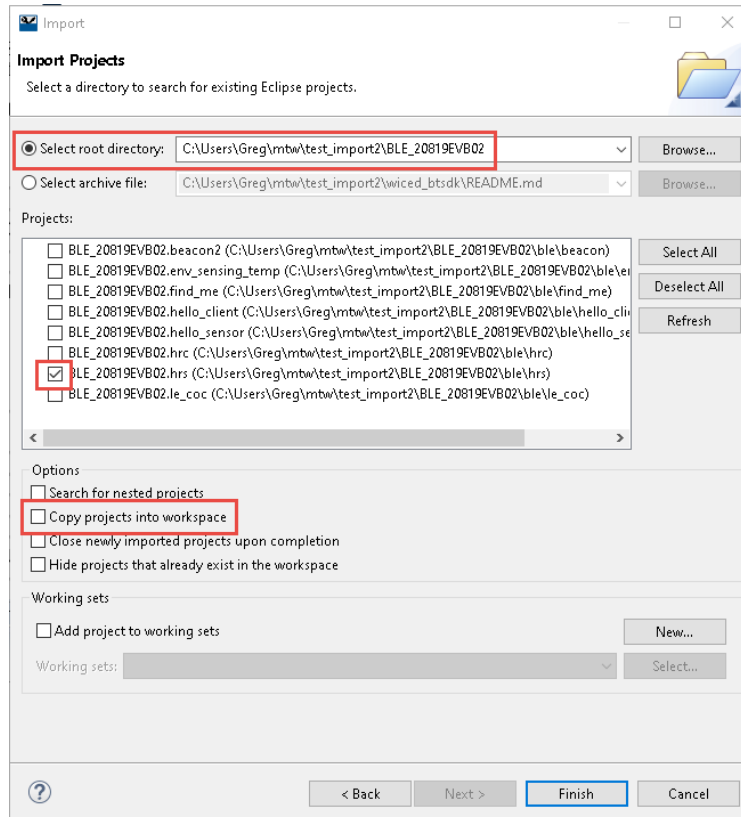
8. Click Finish.
9. This will maintain the hierarchy of the original project(s) so you can build without any updates to the makefile.

1.4 Command Line Interface (CLI)

In addition to the ModusToolbox IDE, there is a command line interface that can be used for most, if not all, operations including downloading applications, running configurators, building, programming, and even debugging.

The make/build system works the same in both environments so once you have an application setup in the IDE, you can go back and forth between the command line and IDE at will.

If you start from the IDE, the application is automatically capable of command line operations. If you start from the command line, you can run an operation to allow the application to be imported into the IDE. In either case, once you have it setup, you can use both methods interchangeably depending on what you want to do.

1.4.1 Modus Shell

Modus Shell is a Cygwin environment that is setup for ModusToolbox. If you already have Cygwin installed and it has make and git, you can use Cygwin directly. If not, Modus Shell is a very convenient option.

To run Modus Shell, go to the tools directory (<install>/ModusToolbox/tools_<version>/modus-shell) and double click on Cygwin.bat. This will launch the shell. It is Linux based, so you can use any Linux commands.
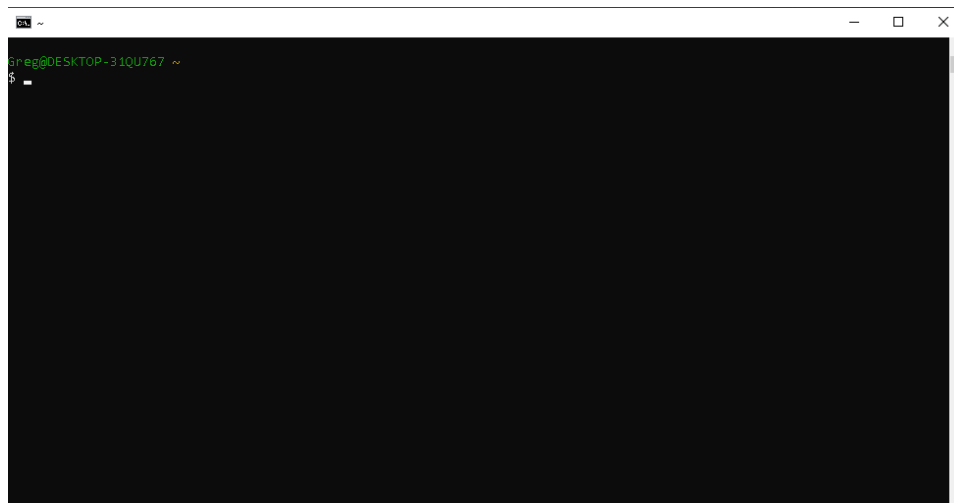
1.4.2 Downloading an Application

There is a tool called project-creator that can be used to setup an application to use on the command line. In fact, there is both a GUI version of this tool (project-creator.exe) and a command-line version (project-creator-cli.exe). The GUI version works just like the new application wizard and is a very easy way to setup an application which can then be used in the command line interface. The command line version requires a few arguments, but if you run it with no arguments it will print help information.

Both tools can be found in the ModusToolbox installation at:
<install_folder>/ModusToolbox/tools_<version>/project-creator

If you want to setup and application manually without using one of the project creator tools, you can use Git and make commands to download an application and its dependencies from their repositories (typically on GitHub). You can find all of the available code examples at:
https://github.com/cypresssemiconductorco/Code-Examples-BT-SDK-for-ModusToolbox.

For example, you could run these commands from Modus Shell to download and build the Empty Bluetooth code example (the make commands will be explained in the next section):

Make a Directory to Hold the Applications (i.e. a Workspace)

> *mkdir my_workspace*
>
> *cd my_workspace*

Download the BTSDK (only need to do this once per workspace)

> *git clone https://github.com/cypresssemiconductorco/wiced_btsdk*
>
> *cd wiced_btsdk*
>
> *make getlibs*
>
> *cd ..*

Download the application (or application group)

> *git clone https://github.com/cypresssemiconductorco/mtb-examples-CYW920819EVB-02-btsdk-empty*
>
> *cd Empty_20819EVB02/template/empty_wiced_bt*
>
> *make getlibs*

Build the Application

> *make build*

1.4.3 Modus CLI Commands

To run ModusToolbox CLI commands, you must first change to the directory with your application's project (the directory with the makefile). For example:

cd C:/Users/Greg/mtw/wbt101/BLE_20819EVB02/ble/beacon

Once in the application's project folder, you can run make with various arguments such as:

make help
>>Print top level information and options of the make system.

make *help CY_HELP=build*
>>Print help information about the build make target.

make build
>>Build the application.

make build VERBOSE=true
>>Build the application with verbose output messages.

make program
>>Build the application and then program it to the target device.

make qprogram
>>Program the application using the existing build output files.

make clean
>>Clean the application.

make getlibs
>>Search for library files in the project (*.lib) and download the associated libraries.

make config_bt
>>Run the Bluetooth configurator. If an existing .cybt file exists it will open it. If not, it will allow you to create a new one once the tool launches.

make config_bt design
>>Run the device configurator. This will open the configurator file from the BSP for your target device. Remember that the BSP is in a location that may be shared between applications so be careful when editing this file.

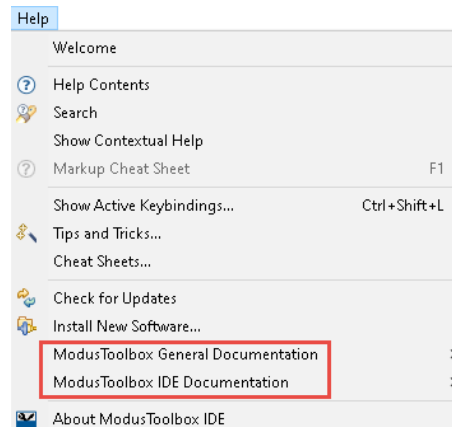*make eclipse CY_IDE_PRJNAME=my_project*
>>Create Eclipse launch targets to allow importing to the IDE.

There are many other options and variables available. Explore the make help output to find out what else is possible.
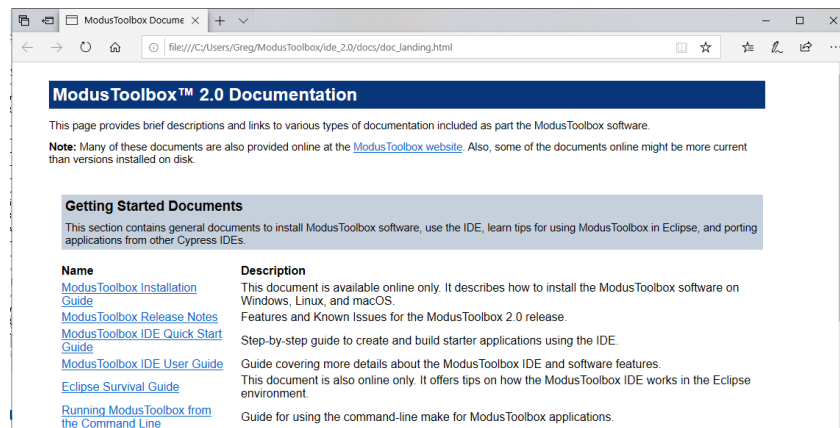
1.5 Tour of Documentation

1.5.1 In ModusToolbox IDE

In the ModusToolbox IDE Help menu there are 2 items of particular interest:
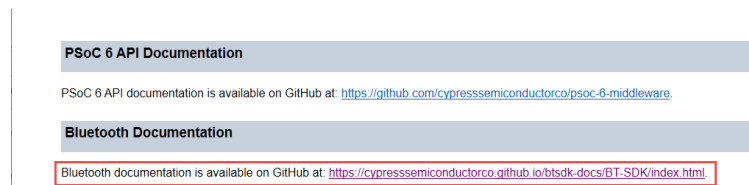

ModusToolbox General Documentation

The first item of interest has a link to the ModusToolbox Documentation Index which is a web page that looks like this:
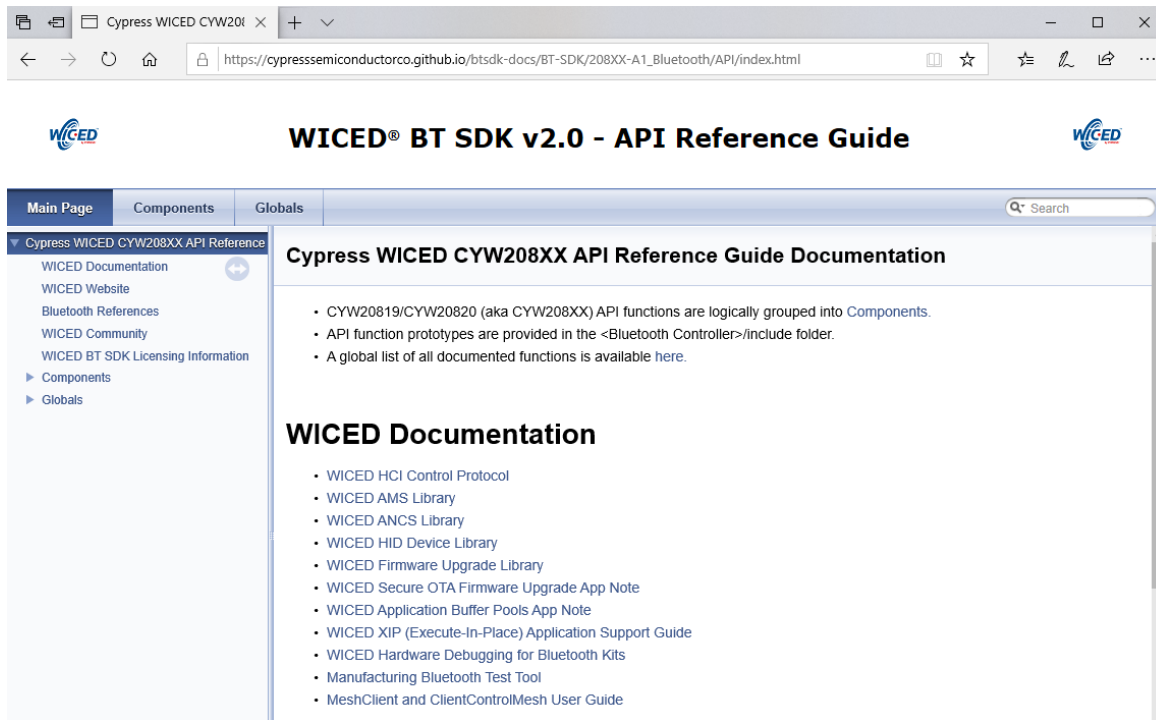

At the bottom of that page you will find a link to the BT SDK API guide.

The Bluetooth Documentation link takes you to a page where you select from a list of Bluetooth devices. This page is also available directly from a link in the Quick Panel. Once you select your device, you will get to this page:

The documentation on the right is useful for specific topics while the section on the left is invaluable for understanding the API functions, data types, enumerations, macros, etc.

ModusToolbox IDE Documentation

The second item on the Help Menu if interest contains documents for:

- Quick Start Guide
- User Guide
- Eclipse IDE Survival Guide

The third item contains tips and tricks on using the Eclipse IDE with Cypress devices intended for those who have relatively little experience with Eclipse.

1.5.2 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):
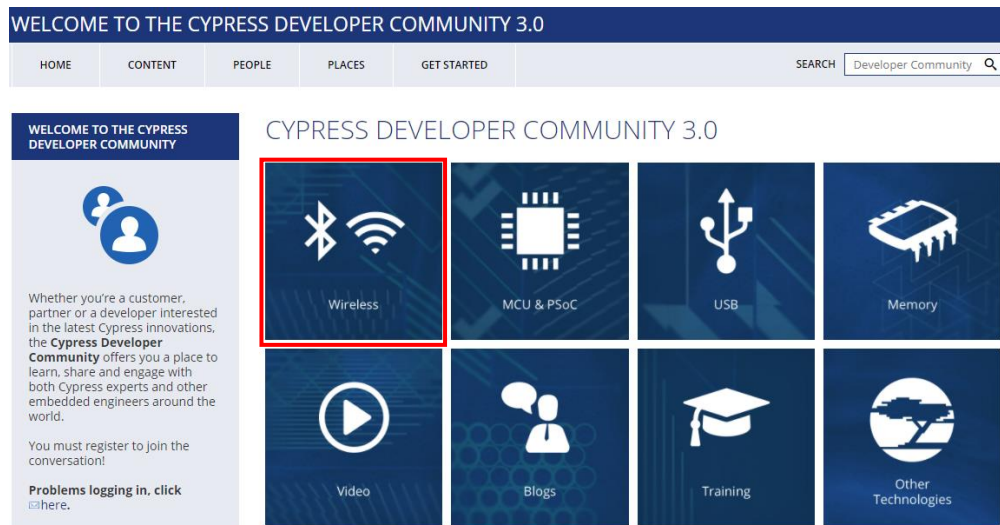
Clicking on the *Wireless* icon will take you to the page as shown below. From this page, you will find links to pages that allow you to download ModusToolbox, search for answers, ask questions, etc.

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

[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

1.6   Reporting Issues

If you are a Cypress employee and you find an issue in ModusToolbox (bug, missing or confusing documentation, enhancement request), please use a "JIRA" to report it:

jira.cypress.com

Click on Create to start submitting a JIRA. Use the project type of "MTBIDE" or "BTSDK", depending on the issue and then fill in as many details as you can to report the issue.

Non-Cypress employees can ask questions and report issues in the developer community.

1.7   Tour of Bluetooth

Bluetooth is a short-range wireless standard that runs on the 2.4 GHz ISM (Industrial, Scientific, and Medical) band modulation. It is controlled by the Bluetooth Special Interest Group (SIG).

Discussions about Bluetooth are typically divided into *Classic Bluetooth* and *Bluetooth Low Energy*.

1.7.1 The Bluetooth Special Interest Group (SIG)

The Bluetooth Special Interest Group is an industry consortium that owns the specifications for Bluetooth. All the Bluetooth documentation is available at www.bluetooth.org. You can register for an account on that website.
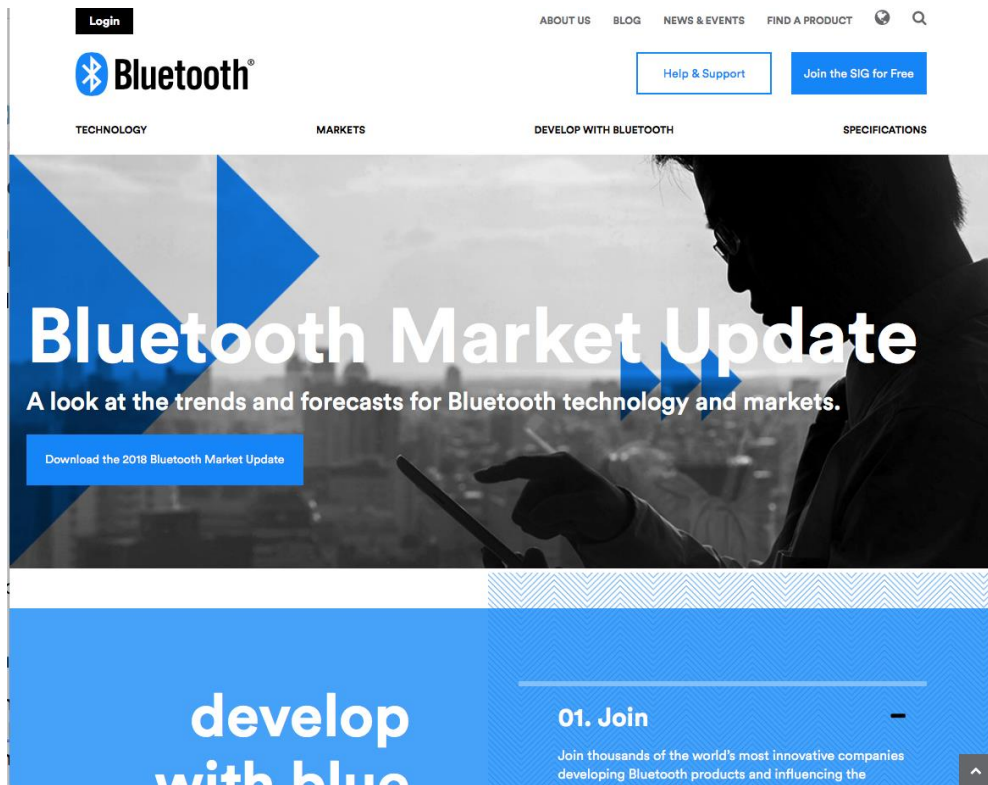
The current Bluetooth Specification is Version 5.1 is a ~3000 page long document that can be downloaded from the Bluetooth SIG website at https://www.bluetooth.com/specifications/bluetooth-core-specification

1.7.2 Classic Bluetooth

Classic Bluetooth uses 79 channels with a channel spacing of 1 MHz. It has three main speeds – Basic Rate (BR) and two Extended Data Rates (EDR). Each of these uses a different modulation scheme.

Mode	Speed	Modulation
Basic Rate	1 Mbps	GFSK (Gaussian Frequency Shift Keying)
Extended Data Rate	2 Mbps	π/4 DQPSK (Differential Quadrature Phase Shift Keying)
Extended Data Rate	3 Mbps	8DPSK (Octal Differential Phase Shift Keying)

The range is dependent on the transmission power which is divided into four classes:

Class	Max Permitted Power		Typical Range
	(mW)	(dBm)	(m)
1	100	20	100
2	2.5	4	10
3	1	0	1
4	0.5	-3	0.5

1.7.3 Bluetooth Low Energy

Bluetooth Low Energy (BLE) uses 40 channels with a channel spacing of 2 MHz (and so it shares the same range of frequencies with Bluetooth Classic). It provides much lower power consumption than Classic Bluetooth. Lower power is not achieved by reducing range (i.e. transmission power) but rather by staying actively connected for short bursts and being idle most of the time. This requires devices to agree on a connection interval. This connection interval can be varied to trade off the frequency of data transmitted vs. power. Therefore, BLE is excellent for data that can be sent in occasional bursts such as sensor states (i.e. temperature, state of a door, state of a light, etc.) but is not good for continuous streaming of data such as audio. BLE typically transmits data up to 1 Mbps, but 2 Mbps can be achieved in Bluetooth version 5 with shorter range.

1.7.4 Bluetooth History

| Bluetooth Spec	Year	Major Features
1.0	1999	Initial standard.
1.1	2002	Many bug fixes. Addition of RSSI and non-encrypted channels.
1.2	2003	Faster connection and discovery. Adaptive Frequency Hopping (AFH) Host Control Interface (HCI) Addition of flow control and retransmission.
2.0 + EDR	2004	Addition of EDR (up to 3 Mbps).
2.1 + EDR	2007	Addition of Secure Simple Pairing (SPP) and enhanced security. Extended Inquiry Response (EIR).
3.0 + HS	2009	Addition of HS which uses Bluetooth for negotiation and establishment, then uses an 802.11 link for up to 24 Mbps. This is called Alternative MAC/PHY (AMP). Addition of Enhanced Retransmission Mode (ERTM) and Streaming Mode (SM) for reliable and unreliable channels.
4.0 + LE	2010	Addition of BLE. Addition of Generic Attribute Profile (GATT). Addition of Security Manager (SM) with AES encryption.
4.1	2013	Incremental software update.
4.2	2014	LE secure connections with data packet length extension. Link Layer privacy. Internet Protocol Support Profile (SPP) version 6.
5	2016	LE up to 2 Mbps for shorter range, or 4x range with lower data rate. LE increased packet lengths to achieve 8x data broadcasting capacity.
5.1	2019	Mesh-based model hierarchy Angle of Arrival (AoA) and Angle of Departure (AoD) fo rtracking Advertising Channel Index GATT Cacheing

1.8 Tour of Chips

Device	Key Features	Notes
CYW20706	• Bluetooth BR, EDR and LE 5.x • ARM Cortex-M3 • 848 kB ROM • 352 kB RAM (data and patches) • 2 kB NVRAM	WICED Studio
CYW20719	• Bluetooth BR, EDR and LE 5.x • 2 Mbps LE v5 • 96 MHz ARM Cortex-M4 • Single Precision FPU • 2 MB ROM • 1 MB On-Chip Flash • 512 kB RAM	WICED Studio
CYW20819	• Bluetooth BR, EDR and LE 5.x • BR/EDR 2 Mbps and 3Mbps • LE 2Mbps • Ultra-low power • 96 MHz ARM Cortex-M4 • 1 MB ROM • 256 kB On-Chip Flash • 176 kB RAM	ModusToolbox
PSoC 4 BLE	• BLE 4.2 • 48 MHz ARM Cortex-M0 • 256 kB On-Chip Flash • 32 kB RAM	PSoC Creator
PSoC 6 BLE	• BLE 5.x • 150 MHz ARM Cortex-M4 & M0+ • 1MB or 2 MB On-Chip Flash • 288 KB RAM	ModusToolbox PSoC Creator

This class covers only the WICED Bluetooth SoC devices, not the PSoC BLE devices.

The Cypress CYW20819 is an ultra-low power (ULP), highly integrated, and dual-mode Bluetooth wireless MCU. By leveraging the all-inclusive development platform ModusToolbox, it allows you to implement the industry's smallest-footprint, lowest-power Bluetooth Low Energy (BLE) and dual mode Bluetooth applications quickly. CYW20819 is a Bluetooth 5.x compliant SoC with support for Bluetooth Basic Rate (BR), Enhanced Data Rate (EDR), and BLE.

The CYW20819 employs the highest level of integration to eliminate all critical external components, thereby minimizing the device's footprint and the costs associated with implementing Bluetooth solutions. A 96 MHz CM4 CPU coupled with 256 kB on-chip flash and 1 MB ROM for stack and profiles offers significant processing power and flash space to customers for their applications. CYW20819 is the optimal solution for a range of battery-powered single/dual mode Bluetooth internet of things applications such as home automation, HID, wearables, audio, asset tracking, and so on.

1.9 Tour of Partners
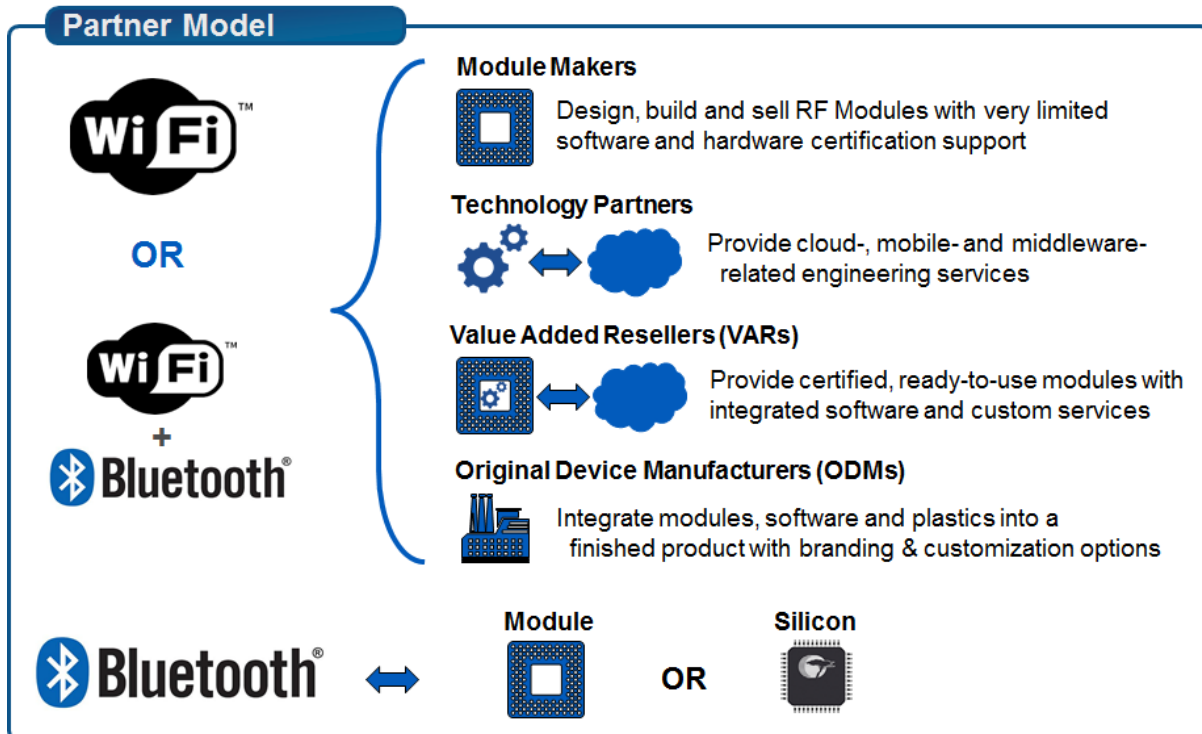
A global partner ecosystem enables you to get the level of support you need for your IoT application

An IoT Selector Guide including partner modules available can be found in the Community at:

https://community.cypress.com/docs/DOC-3021

1.10 Tour of Development Kits

1.10.1 Cypress CYW920819EVB-02

- Bluetooth 5.x plus 2 Mbps LE from v5
- 96 MHz ARM Cortex-M4
- Integrated transceiver
- 1 MB ROM, 256 kB On-Chip Flash, 176 kB SRAM
- 1 User Button, 2 User LEDs
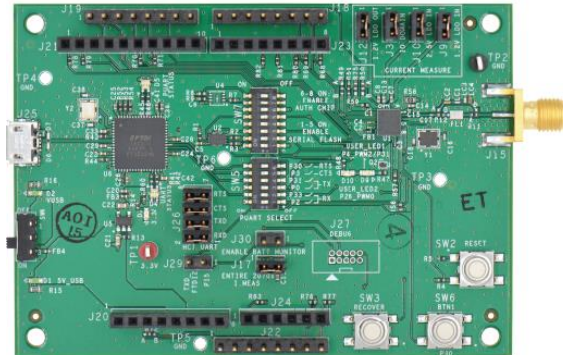- USB JTAG Programmer/Debugger

1.10.2 Cypress CYBT-213043-MESH

- Bluetooth Mesh kit with 20819 module
- Each kit contains 4 boards to evaluate mesh networks
- 1 User Button, RGB LED, ambient light sensor, PIR motion sensor

1.10.3 Cypress CYW920706WCDEVAL

- Monolithic, Single-chip, Bluetooth 5.x + HS
- ARM Cortex-M3 processor
- Integrated transceiver
- 848 kB ROM, 352 kB SRAM (data and patches), 2 kB NVRAM, 512 kB External Serial Flash
- 1 User Button, 2 User LEDs
- USB JTAG Programmer/Debugger

1.10.4 Cypress CYW920719Q40EVB-01

- Bluetooth 5.x plus 2 Mbps LE from v5
- 96 MHz ARM Cortex-M4
- Integrated transceiver
- 2 MB ROM, 1 MB On-Chip Flash, 512 kB SRAM
- 1 User Button, 2 User LEDs
- USB JTAG Programmer/Debugger

1.11 Exercise(s)

Exercise - 1.1 Create a Forum Account and Download the Latest BT SDK

1. Go to https://community.cypress.com/welcome
2. Click "Log in" from the top right corner of the page and login to your Cypress account. If you do not have an account, you will need to create one first.
3. Once you are logged in, click the "Wireless" icon and then explore.
4. Go to "Wireless", "ModusToolbox Bluetooth" and download the latest ModusToolbox BT SDK from the Downloads table.
5. Unzip the BT SDK file once it has been downloaded.

Exercise - 1.2 Start ModusToolbox IDE, and Explore the Documentation

1. If you are on a Cypress internal network, set the environment variables before starting the IDE. See section 1.2
2. Run ModusToolbox IDE and create a new workspace.
3. If you are on a Cypress internal network, set the required settings in ModusToolbox IDE. See section 1.2
4. Explore the different documents available in the Help menu such as the *ModusToolbox IDE Help*, *Quick Start Guide, User Guide* and *Eclipse IDE Survival Guide*.
5. Open and explore the *ModusToolbox Documentation Index* page. Be sure to look at the *CYW920819EVB-02 Kit* link and the *Bluetooth Documentation -> CYW208XX API Reference*.
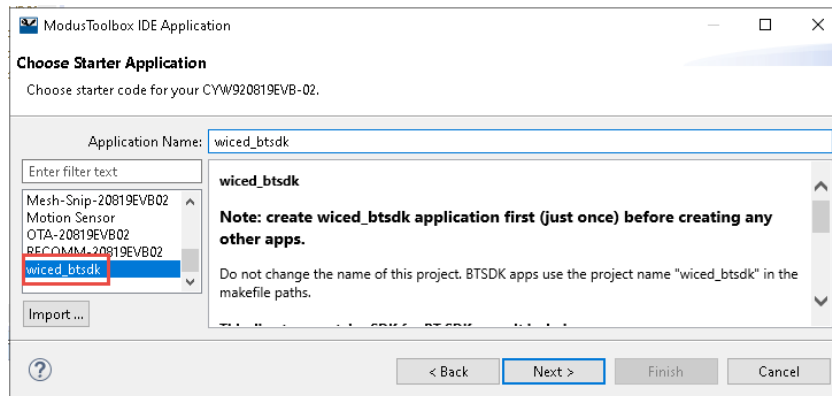
Questions to answer:
1. Where is the WICED API documentation for the PWM located?

Exercise - 1.3 Program a Simple Application

In this exercise, you will install the wiced_btsdk, create a new application, and then build/program it to a kit.

1. In the Quick Panel tab click "New Application".
2. Select the CYW920819EVB-02 kit and click "Next >".
3. Select "wiced_btsdk".

4. Click "Next >"
5. Click "Finish" (it will take a few minutes to download the SDK. Remember you only need this step once per workspace since the SDK is shared).
6. Once that's done, click "New Application" again.
7. Select the CYW920819EVB-02 kit and click "Next >".
8. Select "Empty-20819EVB02"
9. Change the application name to **ch01_ex01_led**.
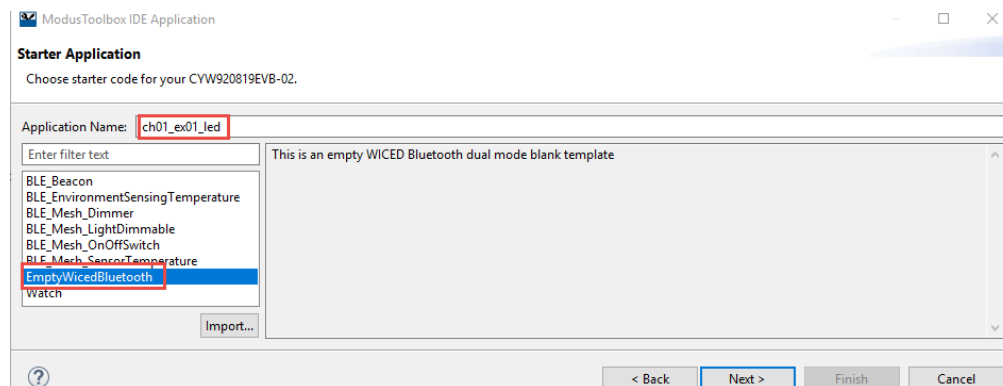
Note: you can call the application anything you like but it will really help if you maintain an alphabetically sortable naming scheme – you are going to create quite a few projects.

10. Click "Next ". Verify the presented device, board, and example, then click "Finish". When the application has been created, the Project Explorer window in Eclipse should look like the

following screenshot (ModusToolbox IDE adds ".empty_wiced_bt" to the project name because, as you learned earlier, some starter applications actually contain a set of applications):
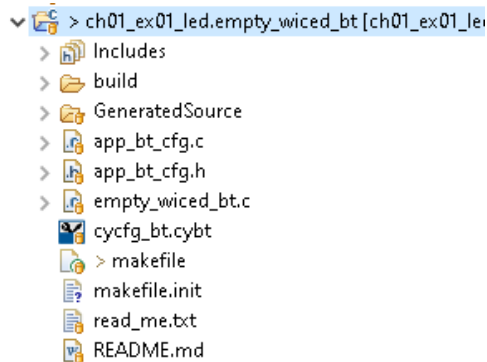
11. Open the top-level C file which in this case is called empty_wiced_bt.c.
12. Add the following line in the application_start function:

```
wiced_hal_gpio_set_pin_output(WICED_GPIO_PIN_LED_1, GPIO_PIN_OUTPUT_LOW);
```

13. Connect your CYW920819EVB-02 kit to a USB port on your computer.
14. In the Quick Panel, look in the "Launches" section and click the "ch01_ex01_led.empty_wiced_bt Program" link.
15. Once the build and program operations are done, you should see "Download succeeded" in the Console window and LED1 (the yellow user LED) should be on.