

Getting Started with CYW20819

Author: Vivek Shankar Kannan

Associated Part Family: **CYW20819**

Software Version: **ModusToolbox™ 1.1 or later**

Associated Application Notes and Code Examples: [Click here.](#)

More code examples? We heard you.

To access an ever-growing list of CYW20819 code examples using ModusToolbox, please visit the [CYW20819 Code Examples Repository on Cypress GitHub portal.](#)

AN225684 introduces you to the CYW20819 Bluetooth (BT) MCU, a BT 5.0-compliant, Arm® Cortex®-M4 CPU-based ultra-low-power MCU that supports both Classic BT and Bluetooth Low Energy (BLE). This application note helps you get started on the CYW20819 device with an overview of the device architecture, development kits and software development tools. It shows how to create a simple BLE application using the ModusToolbox IDE and the Bluetooth Software Design Kit (SDK). It also guides you to more resources available online to accelerate your learning about CYW20819.

Contents

1	Introduction.....	1	4.2	About the Design	9
2	CYW20819 Overview	2	4.3	Part 1: Create a New Application	10
2.1	Device Features	3	4.4	Part 2: Configure Design Resources.....	15
2.2	Target Applications	3	4.5	Part 3: Write the Application Code.....	23
3	Development Ecosystem.....	4	4.6	Firmware Description.....	24
3.1	Software Ecosystem - ModusToolbox.....	4	4.7	Part 4: Build, Program, and Test Your Design ..	29
3.2	Development Kits.....	7	5	Summary	31
4	My First CYW20819 BLE Application	8	6	Technical Resources.....	32
4.1	Prerequisites	8		Worldwide Sales and Design Support.....	34

1 Introduction

Applications featuring Bluetooth connectivity are trending towards lower power, more application-level features, and smaller BoM cost and board space. The CYW20819 Bluetooth Wireless MCU enables you to meet all these critical requirements. In terms of the core Bluetooth functionality, the device is Bluetooth 5.0-compliant with support for dual-mode Bluetooth operation, and it supports BLE (1 Mbps & 2 Mbps) and (EDR 2 Mbps & 3 Mbps) data rates. CYW20819 also fully implements the Bluetooth Mesh 1.0 specification. CYW20819 also supports value-add application features by having a powerful Arm Cortex-M4 CPU and a host of peripheral blocks like ADC, SPI, UART, and I²C that aid in interfacing with external on-board sensors. The presence of these peripheral blocks, on-chip flash memory, and integrated buck and LDO regulators also enables reduced BoM cost and PCB footprint. Cypress' advanced CMOS manufacturing process and the support for various system power modes enable you to design battery-operated, low-power applications using CYW20819.

The device is intended for use in audio (source), sensors (medical, home, security), HID and remote-control functionality, as well as a host of other IoT applications.

This application note will help you get started with the CYW20819 device by covering the following:

- Overview of the [CYW20819 device features and architecture](#)
- Overview of the [Development Ecosystem](#) support for the CYW20819 device that includes the software development platforms, hardware evaluation platforms, code examples, application notes, and other related technical documents.

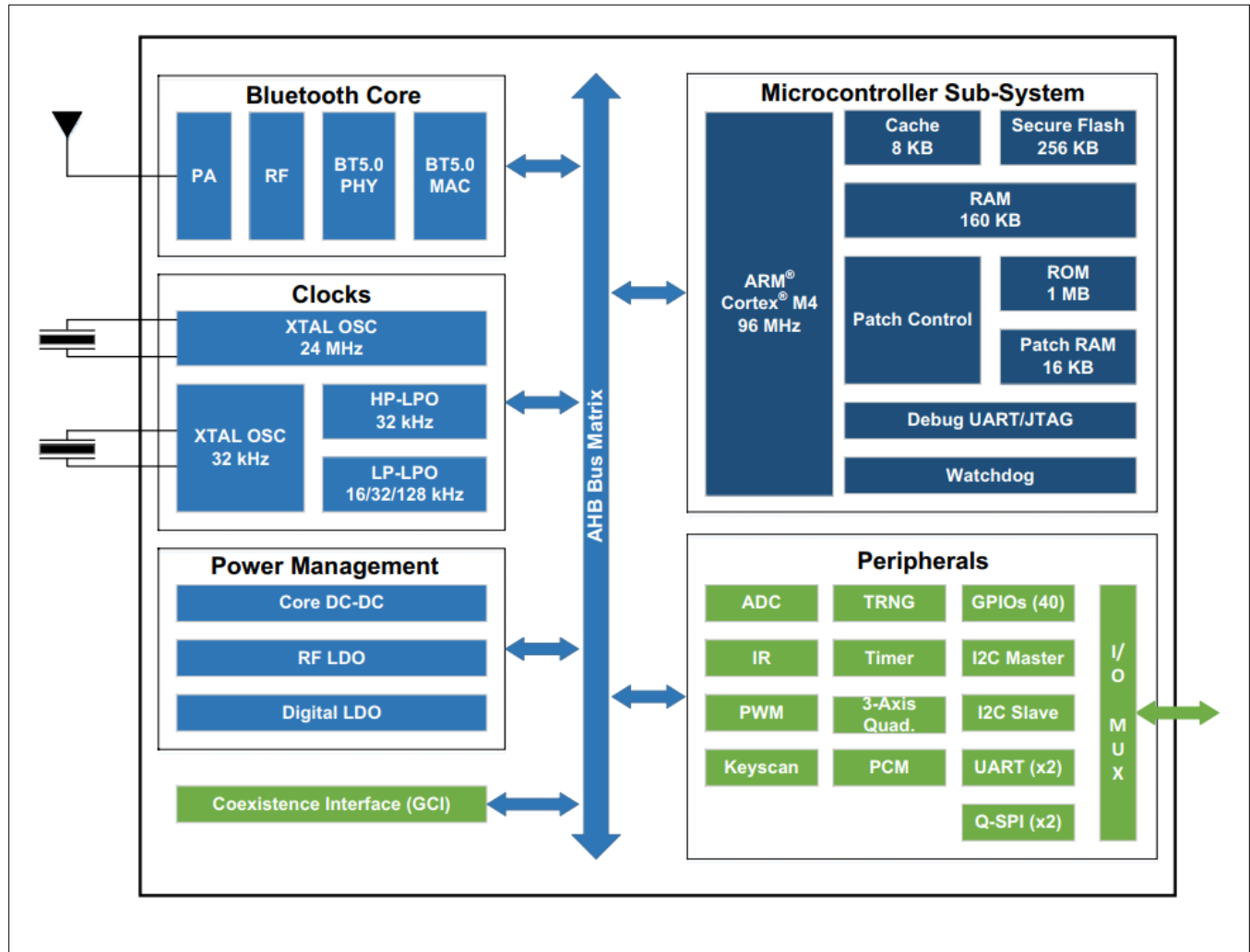
- Tutorial on how to [develop applications](#) using the CYW20819 device in ModusToolbox™ IDE software by going through the step-by-step process of creating a simple BLE-based application from scratch.

This application note does not cover the BLE or classic Bluetooth protocol details. It is assumed that the reader is already familiar with the basics of these protocols.

2 CYW20819 Overview

CYW20819 is a Bluetooth (BT) 5.0-compliant wireless MCU with an integrated 2.4 GHz transceiver with support for both BLE and Classic Bluetooth (BR, EDR). [Figure 1](#) shows the high-level architectural block diagram of the device.

Figure 1. CYW20819 Block Diagram



2.1 Device Features

- Bluetooth Subsystem
 - Complies with Bluetooth Core Specification version 5.0
 - Includes support for BLE (1 Mbps and 2 Mbps), Basic Rate (BR), Extended Data Rate (EDR) 2 Mbps and 3 Mbps, eSCO.
 - Programmable TX Output Power up to +4 dBm
 - Excellent receiver sensitivity (-95.5 dBm for BLE 1 Mbps)
- Microcontroller
 - Powerful Arm Cortex-M4 core with a maximum speed of 96 MHz
 - 256 KB on-chip flash, 176 KB on-chip RAM
 - Bluetooth stack, Peripheral drivers, and Security functions built into ROM (1 MB) allowing applications to efficiently use on-chip flash for standalone operation without the need for an external MCU
 - AES-128 and True Random Number Generator (TRNG)
 - Security functions in ROM including Elliptic Curve Digital Signature Algorithm (ECDSA) signature verification
 - Over-the-air (OTA) firmware update support
- Peripherals
 - Up to 40 GPIOs
 - I²C, I²S, UART, PCM, and two SPI/Quad-SPI interfaces
 - Auxiliary ADC with up to 28 analog channels
 - Programmable key scan 20 x 8 matrix, three-axis quadrature signal decoder
 - General-purpose timers, PWM, real-time clock (RTC), watchdog timers (WDT)
- Power Management and Clocking
 - On-chip power-on reset (POR), Integrated Buck (DC-DC), and LDO regulators
 - On-chip software-controlled power management unit (PMU)
 - Multiple system power modes to optimize power consumption
 - 24-MHz external crystal oscillator for device operation (±20 ppm accuracy)
 - On-chip 32-kHz Low Power Oscillator (LPO) with optional external 32-kHz crystal oscillator support
- Wi-Fi Coexistence
 - Global Coexistence Interface (GCI) for use with Cypress Wi-Fi parts
 - Serial Enhanced Coexistence Interface (SECI) for use with SECI-compatible Wi-Fi parts

For more information on the device, including the electrical specifications, see the [device datasheet](#).

2.2 Target Applications

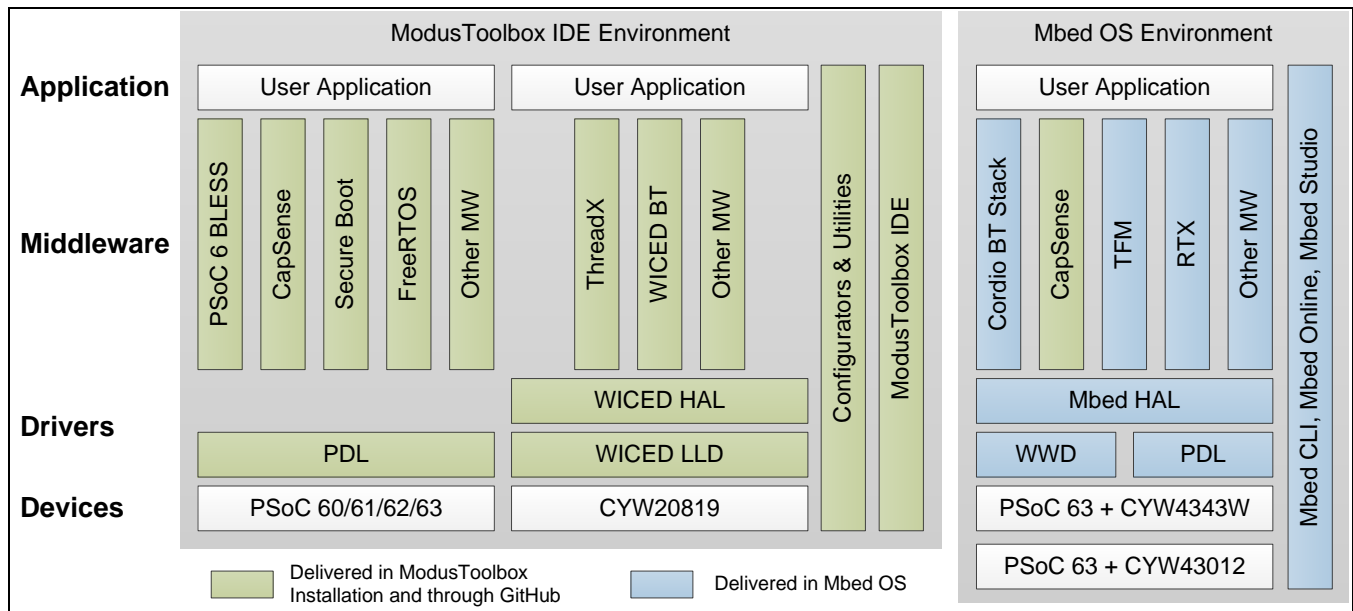
- | | |
|--|--------------------------------|
| ▪ Wearables and fitness bands | ▪ Proximity sensors |
| ▪ Audio source applications | ▪ Key fobs |
| ▪ BLE Mesh Home and Industrial automation | ▪ Thermostats and thermometers |
| ▪ Blood pressure monitors and other medical applications | ▪ Toys |
| | ▪ Remotes |

3 Development Ecosystem

3.1 Software Ecosystem - ModusToolbox

Cypress provides ModusToolbox as the software development platform for CYW20819 applications. ModusToolbox software is a set of tools that enable you to integrate Cypress devices into your existing development methodology. One of the tools is a multi-platform, Eclipse-based Integrated Development Environment (IDE) called the ModusToolbox IDE that supports application configuration and development. [Figure 2](#) shows the high-level view of the tools included in the ModusToolbox software.

Figure 2. ModusToolbox Software Overview



3.1.1 Getting Started with ModusToolbox

Visit the [ModusToolbox](#) home page to download and install the latest version of ModusToolbox. Refer to the *ModusToolbox Installation Guide* document in the *Documentation* tab of [ModusToolbox](#) home page for information on installing the ModusToolbox software. After installing, launch ModusToolbox and navigate to the following items:

- **Quick Start Guide:** Choose **Help > ModusToolbox IDE Documentation > Quick Start Guide**. This guide gives you the basics for using ModusToolbox.
- **User Guide:** The detailed user guide at **Help > ModusToolbox IDE Documentation > User Guide**.

These documents are also available in the *Documentation* tab of the [ModusToolbox](#) home page.

3.1.2 ModusToolbox IDE

The ModusToolbox IDE is based on the Eclipse IDE “Oxygen” version. It uses several plugins, including the Eclipse C/C++ Development Tools (CDT) plugin. Cypress provides an [Eclipse Survival Guide](#), which provides tips and hints for how to use the ModusToolbox IDE.

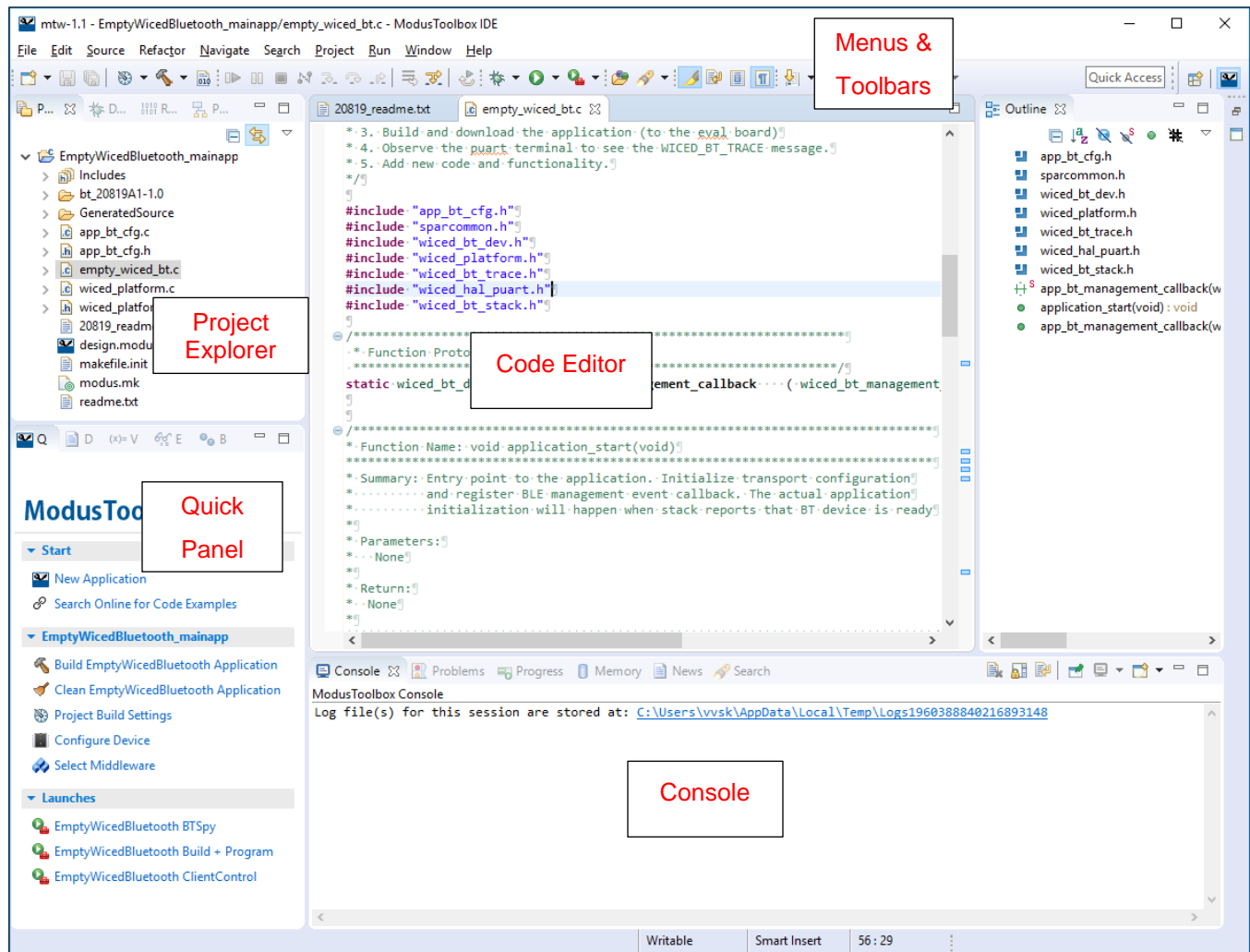
The IDE contains Eclipse-standard menus and toolbars, plus various panes such as the Project Explorer, Code Editor, and Console as shown in [Figure 3](#). One difference from the standard Eclipse IDE is the “ModusToolbox Perspective.” This perspective provides the “Quick Panel,” a “News View,” and adds tabs to the Project Explorer. In the IDE, the top-level entity that you ultimately program to a device is called an application. The application consists of one or more Eclipse projects. The IDE handles all dependencies between projects automatically. It also provides hooks for launching various tools provided by the software development kits (SDKs).

With the ModusToolbox IDE, you can:

1. Create a new application based on a list of starter applications, filtered by kit or device, or browse the collection of code examples online.
2. Configure device resources to build your hardware system design in the workspace.

3. Add software components or middleware.
4. Develop your application firmware.

Figure 3. ModusToolbox IDE Overview



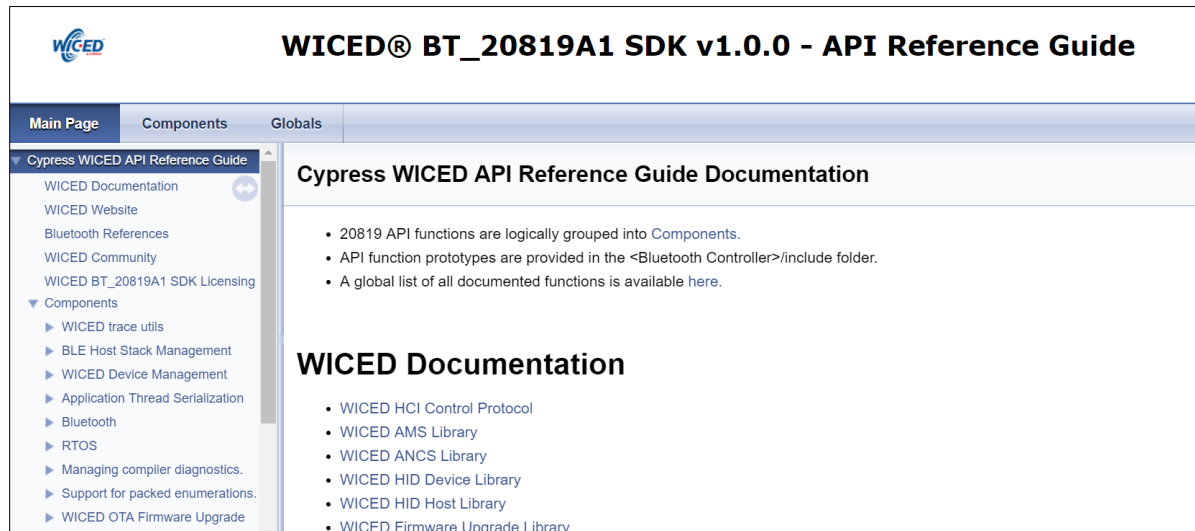
3.1.3 SDK for CYW20819

SDKs provide the central core of the ModusToolbox software. SDKs make it easier to develop firmware for supported devices without the need to understand all the intricacies of the device resources. An SDK contains configuration tools, drivers, libraries, and middleware as well as various utilities, Makefiles, and scripts. CYW20819 application development is enabled by the 20819A1 SDK within ModusToolbox. Important features of the 20819A1 SDK are listed below:

- Dual-mode Bluetooth (BT) stack included in the device ROM (BR/EDR and BLE)
- BT stack and profile level APIs for embedded BT application development (Host and Controller on the same device)
- WICED™ HCI protocol to simplify developing applications that require interfacing with more powerful processors
- APIs and drivers to access on-chip peripheral blocks such as SPI, UART, and ADC.
- Bluetooth protocols supported include Generic Access Profile (GAP), Generic Attribute Profile (GATT), Security Manager Protocol (SMP), Radio Frequency Communication protocol (RFCOMM), Service Discovery Protocol (SDP), and BLE Mesh protocol.
- BLE and BR/EDR profile APIs, libraries, and sample applications

- Support for Over-The-Air (OTA) upgrade
- **SDK API Reference:** Choose **Help > ModusToolbox API Reference > WICED API Reference**. This HTML format guide gives you information on the SDK APIs related to the CYW20819 device. The API functions are logically grouped into components, as shown in [Figure 4](#), for easy navigation and reference.

Figure 4. Bluetooth SDK API Reference



Refer to the [20819A1 SDK Technical Brief document](#) to know the detailed list of features supported in the SDK.

3.1.4 Configurators

ModusToolbox software provides graphical applications called Configurators that make it easier to configure hardware resources. The configurators applicable for CYW20819 are listed below.

- The Device Configurator is used to enable/configure the peripherals and the pins used in the application.
- The Bluetooth Configurator is used to generate the BLE GATT database for the application.

In the next section, we will take a detailed look at how to use these configurators as part of a BLE application creation exercise.

4 My First CYW20819 BLE Application

This section provides step-by-step instructions to build a simple BLE-based application for the CYW20819 device using the ModusToolbox IDE. A Bluetooth SIG-defined standard profile called [Find Me Profile \(FMP\)](#) is implemented in the design. The steps covered in this section are:

- [Part 1: Create a New Application](#)
- [Part 2: Configure Design Resources](#)
- [Part 3: Write Application Code](#)
- [Part 4: Build, Program, and Test your Design](#)

These instructions require that you use a particular code example (*BLE_FindMe* in this case). However, the extent to which you use the code example (CE) depends on the path you choose to follow through these instructions. Note that the terms Code Example (CE) and application mean the same thing in the context of this document. A Code Example (CE) is simply an existing ModusToolbox application that serves a specific purpose or functionality.

We have defined two paths through these instructions depending on what you want to learn:

Path	<u>“Using CE directly” path</u> (Evaluate existing Code Example (CE) directly)	<u>“Working from Scratch” path</u> (Use existing Code Example (CE) as a reference only)
Best For	Someone who is new to the tool or device, and wants to quickly see how it all works.	Someone who wants hands-on experience to learn to develop CYW20819-based Bluetooth applications in ModusToolbox.

What you need to do for each path is clearly defined at the start of each part of the instructions.

If you start from scratch and follow all instructions in this application note, you use the code example as a reference while following the instructions. Working from scratch helps you learn the design process but also takes more time. Alternatively, you can just evaluate the existing code example directly to get acquainted with the CYW20819 application development flow in a short time.

In both cases, you should start by reading [Prerequisites](#) and [About the Design](#).

4.1 Prerequisites

Ensure that you have the following items for this exercise.

- [ModusToolbox](#) 1.1 or later version installed on your PC
- [CYW920819EVB-02](#) evaluation board
- [CySmart™ iOS/Android](#) app or any other Android or iOS app that supports the Immediate Alert Service (IAS). You can also use the [CySmart Host Emulation Tool](#) Windows PC application if you have access to the [CY5677 CySmart BLE 4.2 USB Dongle](#).

Scan the following QR codes from your mobile phone to download the CySmart app.

iOS



Android

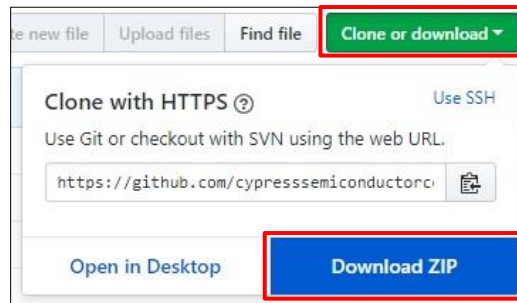


4.1.1 Download Code Examples

Code Examples are collected into repositories on the Cypress GitHub site. Clone or Download the CYW20819 Code Examples repository from [GitHub](#) to a location on your PC. There are three ways you can do this.

1. Download the code examples repository as a zip file by going to the [CYW20819 Code Examples repository on GitHub](#), and using the **Clone or download** option as shown in [Figure 6](#). Unzip the downloaded file to a location on your hard drive. This approach is especially useful if you are not familiar with the Git version control system.

Figure 6. Download CYW20819 Code Examples Repository from GitHub

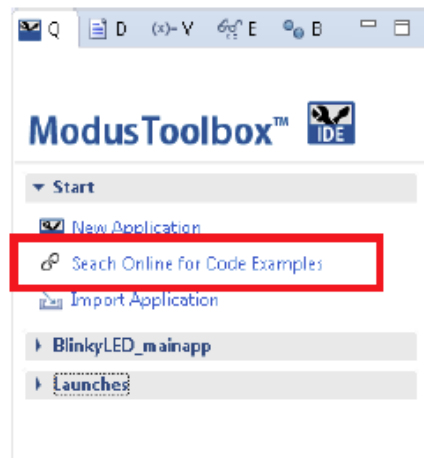


2. In the ModusToolbox IDE, select **File > Import > Git > Projects from Git** and follow the instructions in the wizard. This is standard Eclipse functionality.
3. Use **git clone** or the GitHub desktop client.

Once you have the repository on your PC, the BLE Find Me profile code example is located inside the repository folder at `Code-Examples-BT-20819A1-1.0-for-ModusToolbox\CYW920819EVB-02\apps\demo\BLE_FindMe`. You will be using this code example either as reference or use it directly depending on which development flow you choose to follow in the next sections.

Note that the ModusToolbox IDE also provides a link to access the online code examples in the **Quick Panel** as shown in Figure 7. Click the **Search Online for Code Examples** link. This opens a web browser to the GitHub repository to select and download the appropriate repository. You can then use the above-mentioned steps to download the required repository.

Figure 7. Quick Launch Option to Search Online for Code Examples

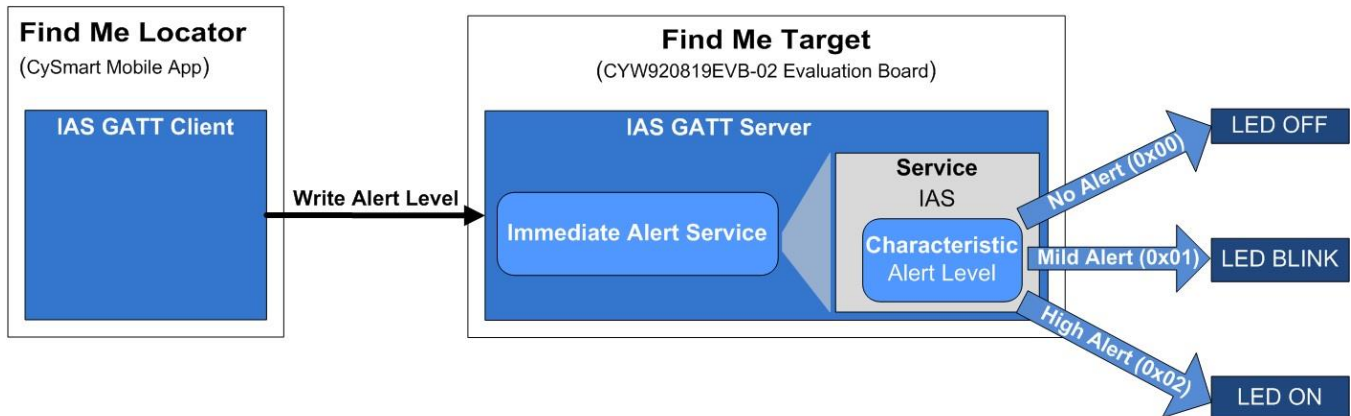


4.2 About the Design

This design implements a BLE **Find Me Profile (FMP)** that consists of an Immediate Alert Service (**IAS**). FMP and IAS are a BLE standard Profile and Service respectively, as defined by the **Bluetooth SIG**. The design uses the two LEDs (red LED, yellow LED) on the CYW920819EVB-02 kit. The red LED (LED2) displays the IAS alert level – no alert (LED OFF), mild alert (LED blinking), high alert (LED ON). The yellow LED (LED1) indicates whether the Peripheral device (CYW20819) is advertising (LED blinking), connected (LED ON), or disconnected (LED OFF). In addition, a debug UART interface is used for sending the Bluetooth stack and application trace messages.

An iOS/Android mobile device or a PC can act as the BLE Central device, which connects to the Peripheral device.

Figure 8. Find Me Profile (FMP) Application using CYW20819



4.3 Part 1: Create a New Application

This part takes you step-by-step through the process of creating a new ModusToolbox application. Before performing the steps in this section, decide whether you want to import and run the code example as-is or you would rather learn how to create an application from scratch. Depending on your choice the steps you need to follow are as shown below:

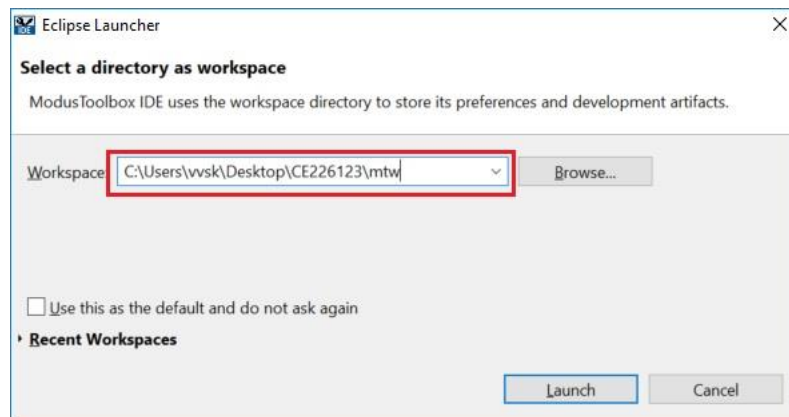
Path	<u>“Using CE directly”</u> path (Evaluate existing Code Example (CE) directly)	<u>“Working from Scratch”</u> path (Use existing Code Example (CE) as reference only)
Actions	Do Steps 1, 2, 3, 4. Ignore Step 5.	Do Steps 1, 2, 3, 5. Ignore Step 4.

Launch ModusToolbox and get started.

1. Select a new workspace.

At launch, ModusToolbox presents a dialog to choose a directory for use as the workspace directory. The workspace directory is used to store workspace preferences and development artifacts such as device configuration and application source code. You can choose an existing empty directory by clicking the **Browse** button as Figure 9 shows. Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path, and ModusToolbox will create the directory for you.

Figure 9. Select a Directory as Workspace

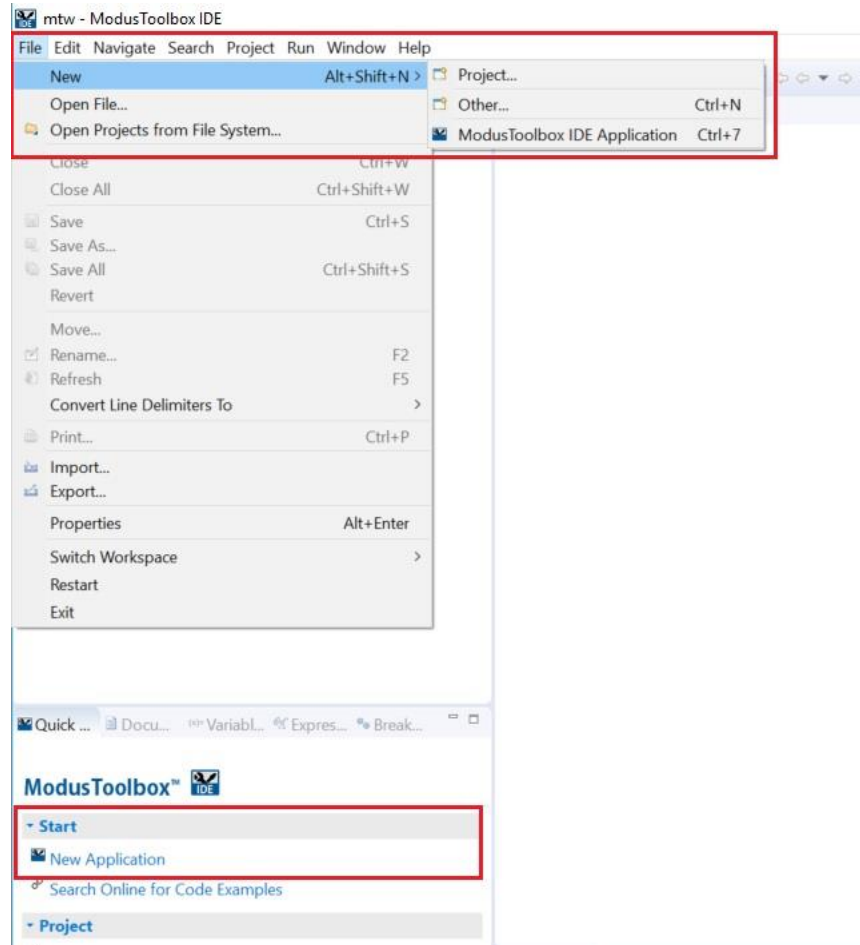


2. Create a new ModusToolbox Application.

Click **New Application** in the Start group of the Quick Panel. Alternatively, you can choose File > New > ModusToolbox IDE Application (Figure 10).

The ModusToolbox IDE Application window appears.

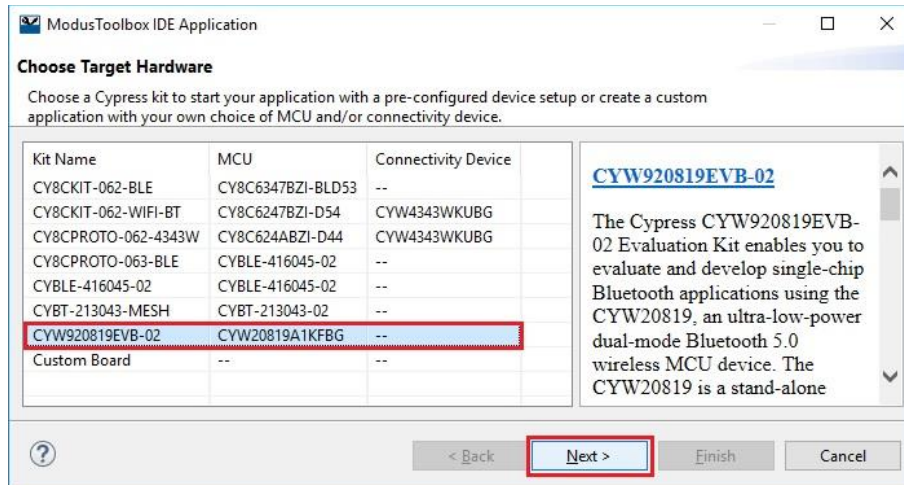
Figure 10. Create a New ModusToolbox IDE Application



3. Select CYW20819-based Target Hardware.

ModusToolbox presents the list of Cypress kits to start your application development. In this case, we want to develop an application on the CYW920819EVB-02 Evaluation Board that utilizes the CYW20819 device. Select **CYW920819EVB-02** and click **Next** (Figure 11).

Figure 11. Choose Target Hardware



Notes on Custom Board Option: If you have a custom board based on the CYW20819 device, select **Custom Board**, and click **Next**. Select the applicable CYW20819 device from the list and proceed further. Even though this application note does not use the “Custom Board” development flow, the steps that follow can be used as a guide for the “Custom Board” development flow as well.

4. Import the BLE_FindMe Code Example (Applicable only for the “Using CE directly” flow)

The **Import** option is useful to import any code example that you downloaded from a repository or received from a colleague into the ModusToolbox IDE. You will be using this feature to import the **BLE_FindMe** code example for the *Using CE directly* flow. In the **Starter Application** dialog shown in Figure 12, click **Import** and navigate to the **BLE_FindMe** code example location in the CE repository you downloaded earlier (*Code-Examples-BT-20819A1-1.0-for-ModusToolbox\CYW920819EVB-02\apps\demo\BLE_FindMe*). Select *modus.mk* inside the code example folder. The **BLE_FindMe** example name and description appear in the **Starter Application** dialog. Modify the application name if required – *CE226123* is the name used in Figure 12. Click **Next** and then **Finish** (Figure 13) to complete the application creation process.

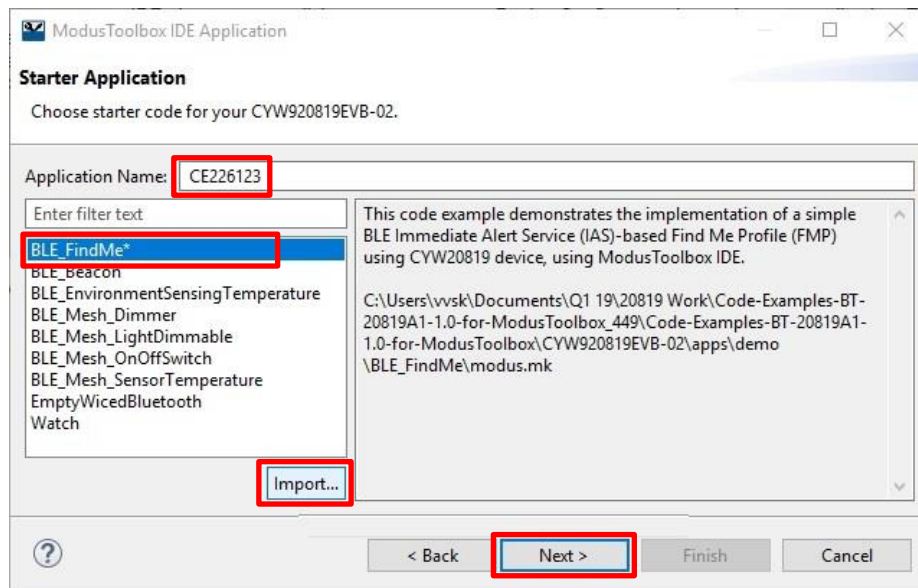
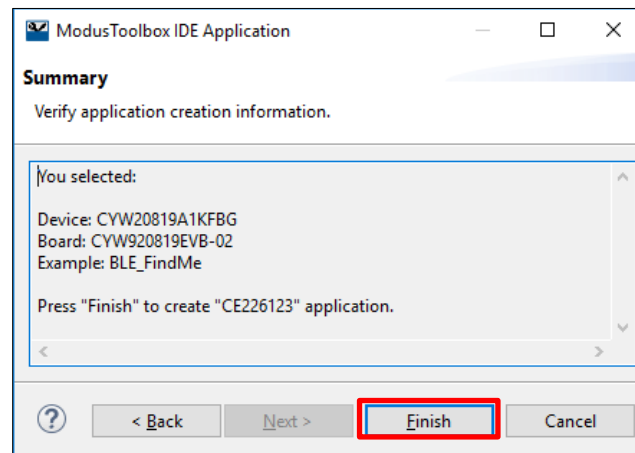
Figure 12. Import *BLE_FindMe* Code Example


Figure 13. Application Summary



You have successfully created a new ModusToolbox application for CYW20819.

5. Select a Starter Application and Create the Application (Applicable only for “Working from Scratch” flow)

You will be using an existing template application as the starting point for the *Working from Scratch* development flow. In the **Starter Application** dialog shown in Figure 14, select **EmptyWicedBluetooth**. In the **Name** field, type in a name for the application (**CE226123** in this case) and click **Next**; the application summary dialog appears. Click **Finish** (Figure 15) to let ModusToolbox create the application for you.

Figure 14. Starter Application Window

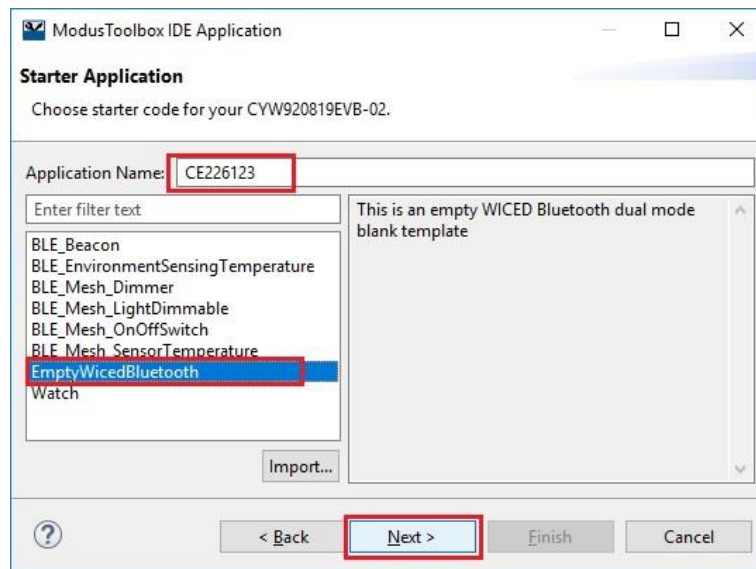
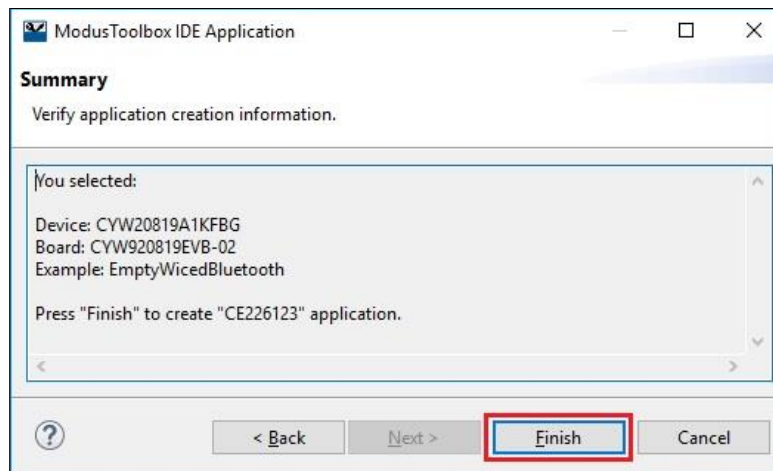


Figure 15. Application Summary



You have successfully created a new ModusToolbox application for CYW20819.

4.4 Part 2: Configure Design Resources

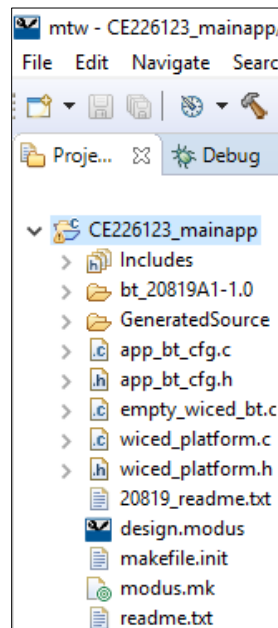
In this step, you will configure the design resources for your application and generate the configuration code.

Path	“Using CE directly” path (Evaluate existing Code Example (CE) directly)	“Working from Scratch” path (Use existing Code Example (CE) as reference only)
Actions	Read and understand all steps. The CE already has all the resource configurations done, so you need not perform any of the steps in this section.	Perform all steps

The **EmptyWicedBluetooth** application template has all the resources available on the CYW902819EVB-02 kit pre-configured and ready for use. These resources include user LEDs, push buttons, and communication peripherals (Bluetooth, UART, I²C, and SPI). The template application also contains a default application code snippet that initializes the device and the Bluetooth stack, and prints a status message on the Peripheral UART (PUART) interface.

Before proceeding further, a quick tour of the ModusToolbox project explorer is in order. [Figure 16](#) shows the ModusToolbox project explorer view after the template application has just been created.

Figure 16. Project Explorer View



- The *Includes* folder contains the header files related to the Bluetooth stack, hardware abstraction layer (HAL), hardware platform, etc. The API function prototypes, constants, and defaults are contained in these header files. The application code you write will use these API functions and constants as required.
- The *bt_20819A1-1.0* folder contains additional source/header files. The *spar_setup.c* file in this folder tree contains the *application_start_internal()* function, which is executed as part of the device startup code. This function calls the hardware platform default initialization routine (*wiced_platform_init()*), and then the *application_start()* function. The *application_start()* function is the entry point for the user application code; in this template project, the function resides in the *empty_wiced_bt.c* file.
- The *GeneratedSource* folder contains the design configuration for this application. This includes the list of resources enabled/disabled for the application, and configuration settings for the different resources. We will discuss more about the contents of this folder in the next step of the application development flow.
- The *design.modus* file is a graphical configurator design file that contains hardware resource configuration settings of the application. This file is used by the graphical configurators, which generate/modify the configuration firmware files in the *GeneratedSource* folder. We will discuss more about this file in the next step of the application development flow.

- The *empty_wiced_bt.c* file contains the `application_start()` function, which is the entry point for the user application code execution after device startup. It also contains the Bluetooth stack management event handler function. You can rename this file if you wish as you will see later.
- The *app_bt_cfg.c* and *app_bt_cfg.h* files contain the runtime Bluetooth stack configuration parameters like device name, and advertisement/connection settings.
- The *wiced_platform.c*, *wiced_platform.h* files contain the functions and macro definitions related to the initialization of the hardware platform (CYW920819EVB-02 in this case), including the LEDs, buttons, and sensors used on the kit.
- The *modus.mk* file is the top-level Makefile that describes the design, and is created and used by the IDE. It contains the design make variables of the application that are passed to the build system. This file is modified by the IDE during certain operations like changing Application Settings or Middleware Selection. There is no need for you to edit this file. We will not be discussing in detail about this file in the scope of this document.
- The *20819_readme.txt* file contains information on the device features, available evaluation kits, SDK features for the CYW20819 device, software tools available, and details of the Application Settings.
- The *readme.txt* file contains information that explains what the application does (in this case, *EmptyWicedBluetooth*).
- *makefile.init* is used as part of the build process, and it allows a way for advanced users to specify new rules and variables in the build flow.

Now, let's get in to the details of how to configure the design resources in the template application.

1. Configure Hardware Resources

Now that you have an application set up from the **EmptyWicedBluetooth** template application, it's time to configure the hardware resources required for this application, and to generate the corresponding configuration code. The ModusToolbox IDE stores the configuration settings of the application in the *design.modus* file. This file is used by the graphical configurators, which generate the configuration firmware. This firmware is stored in the application's *GeneratedSource* folder.

The Device Configurator is used to enable/configure the peripherals and the pins used in the application. To launch the Device Configurator, double-click the *design.modus* file or click on **Configure Device** in the Quick Panel as shown in [Figure 17](#). Any time you make a change in the Device Configurator, click **File > Save** to save the updated configuration.

Figure 17. Quick Panel View

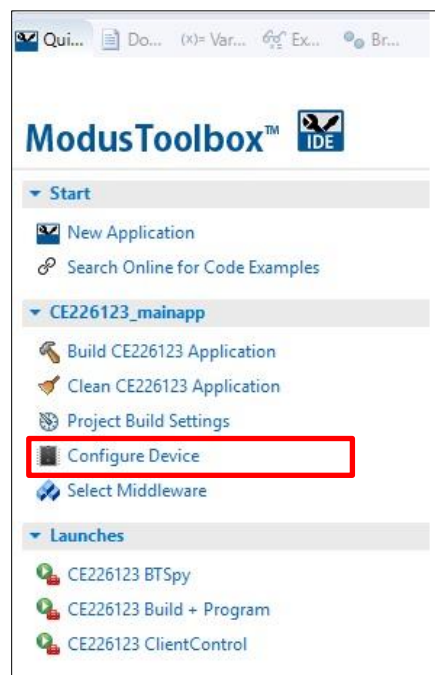
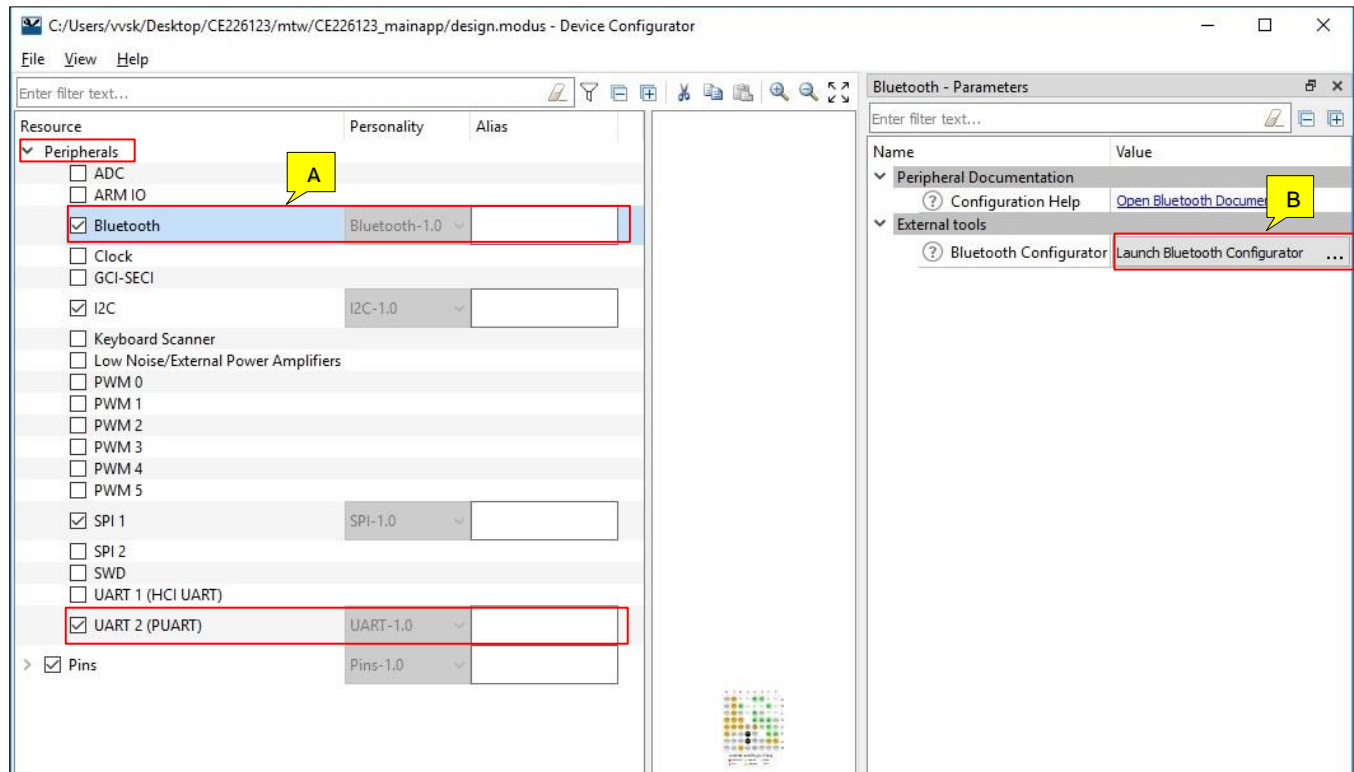


Figure 18 shows the Device Configurator view showing the Peripherals view for this application.

Figure 18. Device Configurator - Peripherals

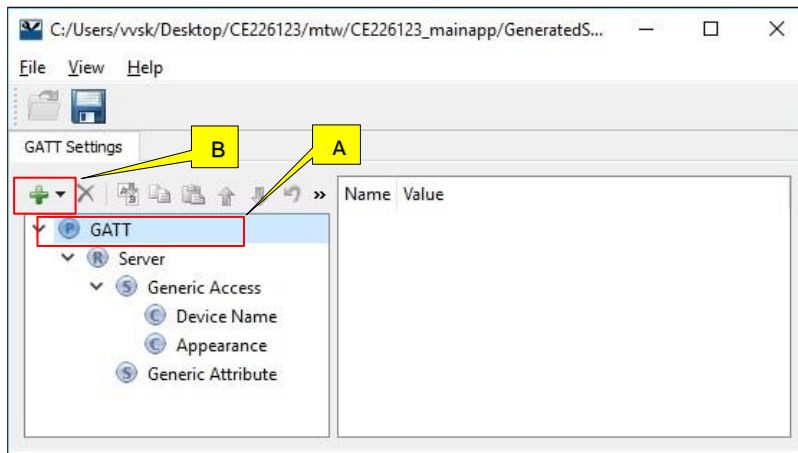


- Peripherals:** The template application has the peripherals on the kit enabled by default as shown in Figure 18 (Bluetooth, I²C, SPI 1, UART 2). For the Find Me application, only the Bluetooth and PUART (UART 2) peripherals are used. However, we can leave the default peripheral enable/disable settings as they are in the template application – the I²C and SPI will not be enabled in the code so having them enabled in the configurator doesn't matter.

To configure a peripheral, click on the peripheral name on the **Resource** pane; the **Parameters** pane shows the configuration options provided for that peripheral. For more information on the Device Configurator, select **Help** - > **View Help** to open the configurator documentation.

- Bluetooth Configurator:** The Bluetooth peripheral has an additional configurator called the Bluetooth Configurator that is used to generate the BLE GATT database for the application. For the Find Me Profile application, we need to generate a GATT database corresponding to the Find Me Target role of the CYW20819 device. To launch the Bluetooth Configurator from the Device Configurator window, click on the **Bluetooth** resource under the **Peripherals** section, and then click on the **Launch Bluetooth Configurator** button under the **Bluetooth-Parameters** window (as shown in callouts A and B in Figure 18). The default view of the Bluetooth Configurator is shown in Figure 19.

Figure 19. Bluetooth Configurator



To add the Find Me Target profile, click on **GATT** profile (A in Figure 19), and then the **+** icon (B in Figure 19). Select the **Find Me Target (GATT Server)** profile from the drop-down menu as shown in Figure 20.

Figure 20. Adding Find Me Target Profile

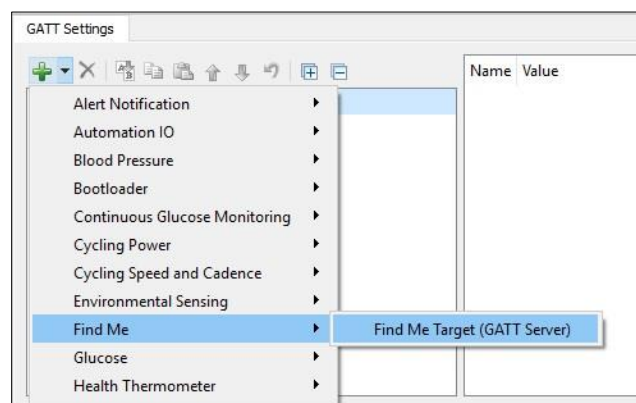
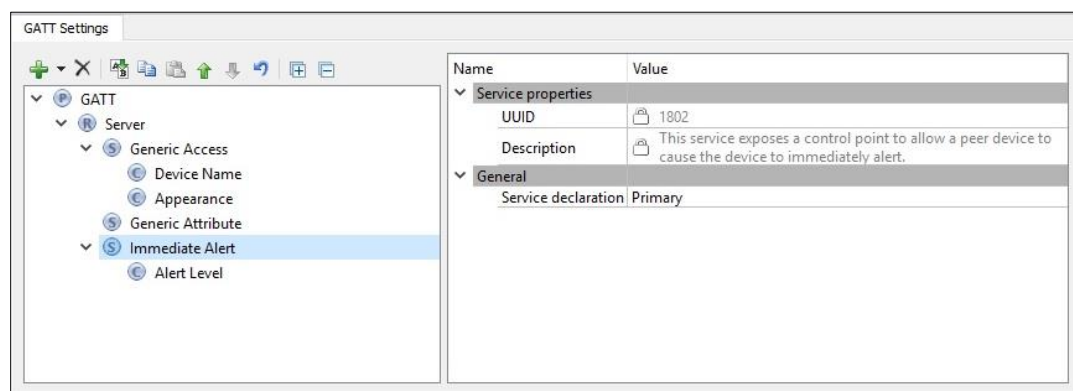


Figure 21 shows the GATT database view once the Find Me Target Server has been added. Note that the Immediate Alert Service corresponding to the Find Me Target profile has been added in Figure 21. Click **File > Save** in the Configurator window or click the Save icon. The configurator stores the GATT database in the source files *cycfg_gatt_db.h*, *cycfg_gatt_db.h* in the *GeneratedSource* folder. In the same folder, the *cycfg_bt.h* file contains the Bluetooth configurator settings in XML format, which the configurator uses to load the settings when launched again.

Figure 21. Final GATT Database View



- Pins:** The pins used in the application are enabled in the **Pins** section of the Device Configurator as shown in Figure 22. The template application already has some pins enabled and configured corresponding to the kit features, such as the pins used for LEDs, buttons, and peripherals like I²C, SPI, and UART. For the Find Me application, the pins of interest are the two LED pins (LED1, LED2), and the UART interface pins (UART_TX, UART_RX) (see Table 1). The parameter settings for these pins are left at the default settings as shown in Figure 23, Figure 24, Figure 25, and Figure 26. We will leave the SPI and I²C pins unchanged – we won't be using those pins for anything else, so it doesn't hurt to have the pins assigned to the default peripherals for the board we are using. Any time you make a change in the Device Configurator, click **File > Save** or click the Save icon to save the updated configuration.

Figure 22. Device Configurator - Pins

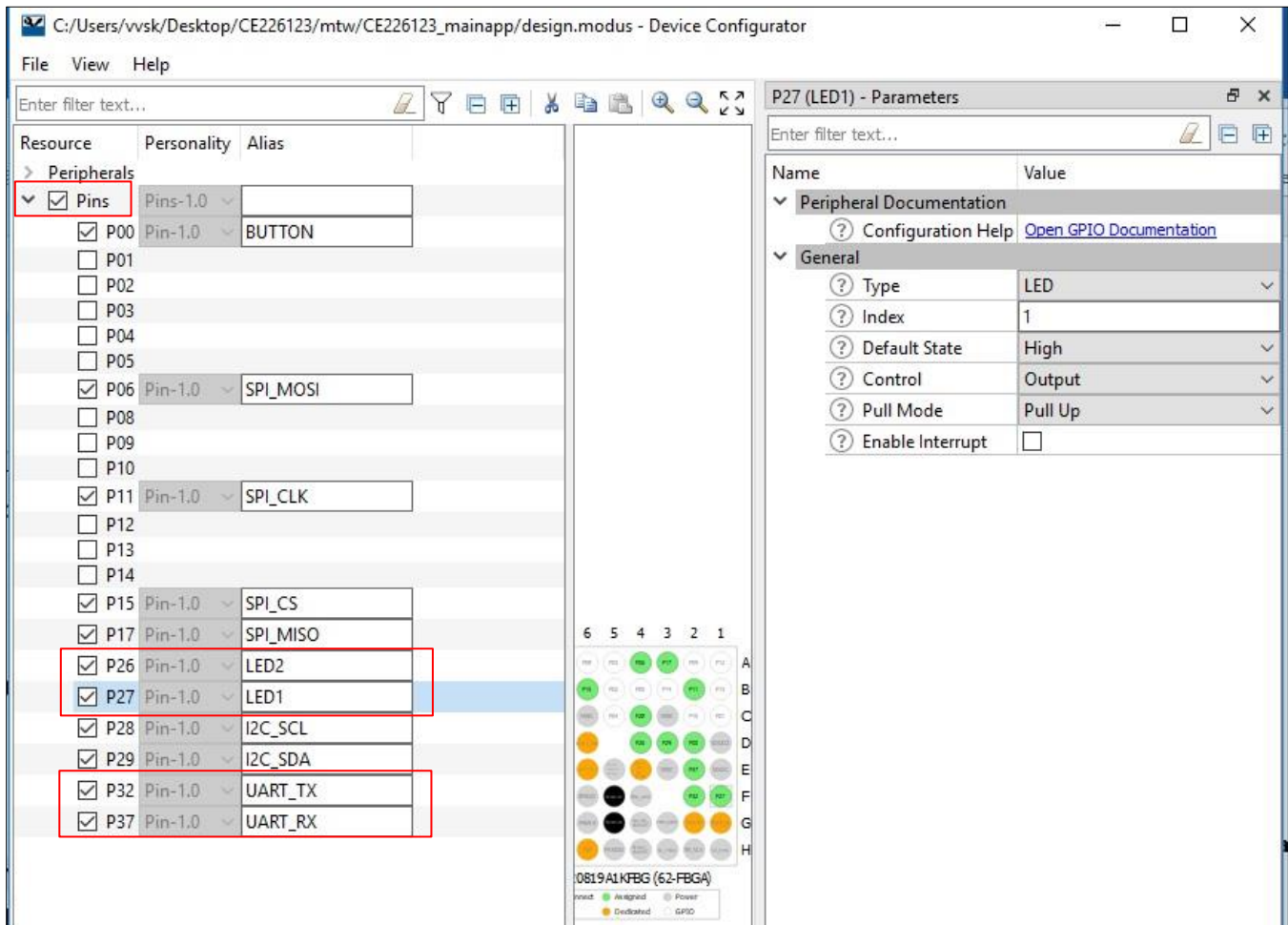
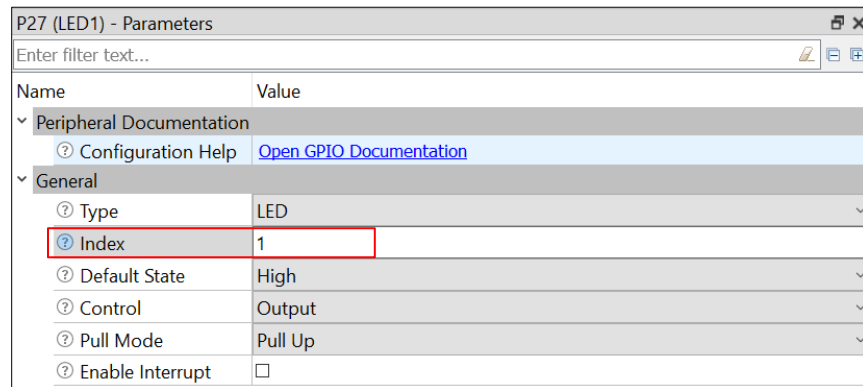


Table 1. Pin Mapping Details

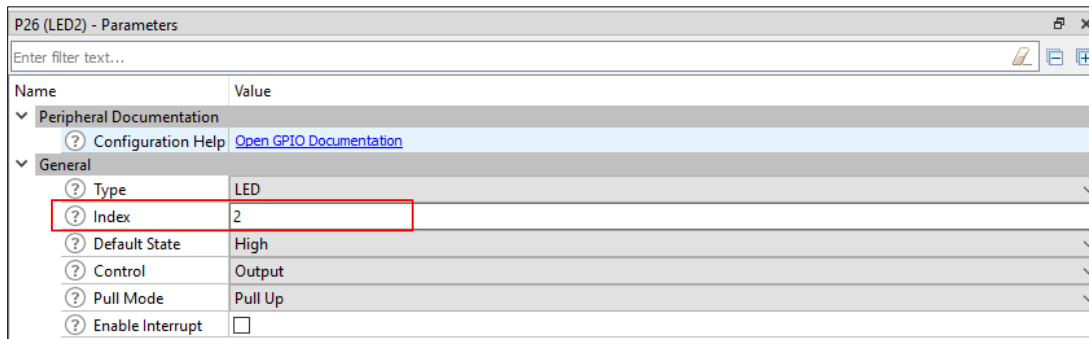
Pin	Alias	Purpose	Settings
P26	LED2	Mapped to the red LED (LED2) on the kit. Indicates IAS Alert Level.	See Figure 23 and Figure 24. Ensure that the Index setting for each LED is different as shown in the figures – the configurator does not do this checking. The Index value should also start from 1 (not 0). Refer <i>cycfg_pins.h</i> to see how the Index value is used by the configuration code.
P27	LED1	Mapped to the yellow LED (LED1) on the kit. Indicates the Advertising/Connected state of the BLE peripheral device.	
P32	UART_TX	Tx pin of UART peripheral	See Figure 25.
P37	UART_RX	Rx pin of UART peripheral	See Figure 26.

Figure 23. LED1 Pin Settings



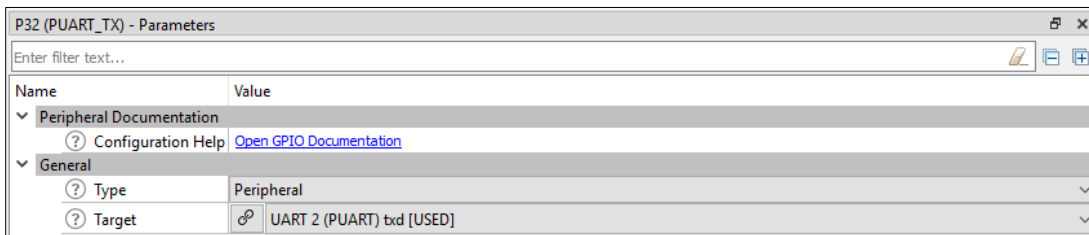
Name	Value
Peripheral Documentation	
Configuration Help	Open GPIO Documentation
General	
Type	LED
Index	1
Default State	High
Control	Output
Pull Mode	Pull Up
Enable Interrupt	<input type="checkbox"/>

Figure 24. LED2 Pin Settings



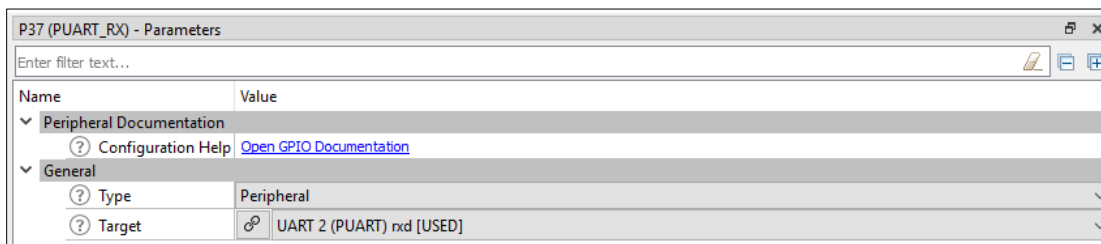
Name	Value
Peripheral Documentation	
Configuration Help	Open GPIO Documentation
General	
Type	LED
Index	2
Default State	High
Control	Output
Pull Mode	Pull Up
Enable Interrupt	<input type="checkbox"/>

Figure 25. PUART Tx Pin Settings



Name	Value
Peripheral Documentation	
Configuration Help	Open GPIO Documentation
General	
Type	Peripheral
Target	UART 2 (PUART) txd [USED]

Figure 26. PUART Rx Pin Settings



Name	Value
Peripheral Documentation	
Configuration Help	Open GPIO Documentation
General	
Type	Peripheral
Target	UART 2 (PUART) rxd [USED]

You have now completed the resource configuration for the application, the required configuration code has been generated in the files in the *GeneratedSource* folder, and the settings have also been stored in the *design.modus* file.

2. Configure Application Settings

Application Settings are miscellaneous project settings like specifying the Bluetooth device address, serial port number to download the application, enabling debug, etc. Application Settings can be viewed or modified by right-clicking the top-level project name in the project explorer and selecting **Change Application Settings** as shown in Figure 27.

Figure 27. Launch Application Settings dialog

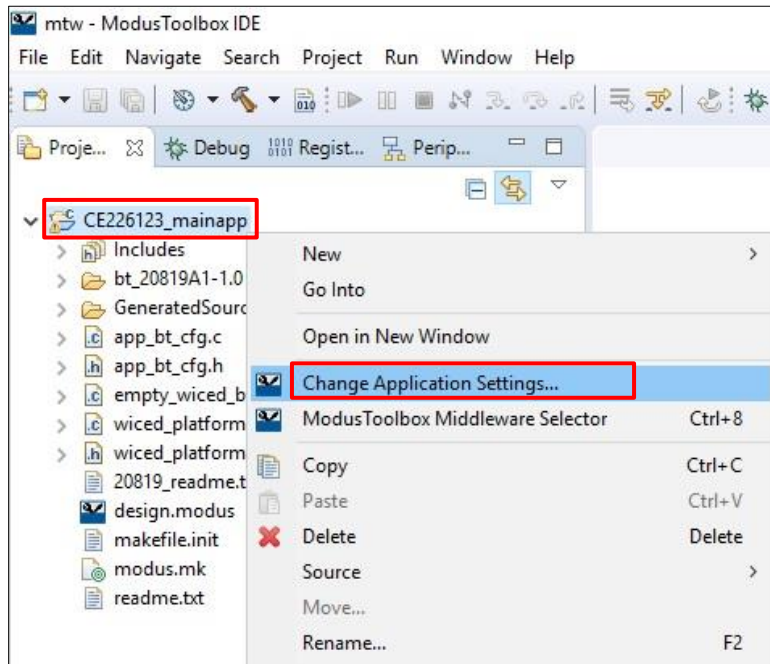
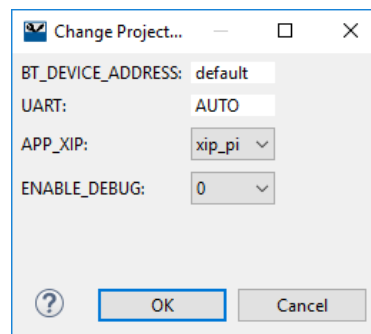


Figure 28 shows the Application Settings for the template application. You will not be making any changes to these settings for creating the Find Me application. One thing to note here is that the *BT_DEVICE_ADDRESS* parameter is set to the *AUTO* setting, which means the SDK will set a random Bluetooth device address for your device. To understand the meaning of the different parameters, refer to the “Application Settings” section in the *20819_readme.txt* file. Note that if you change any of the Application Settings, the IDE will also modify the *modus.mk* file automatically to reflect the updated settings.

Figure 28. Application Settings

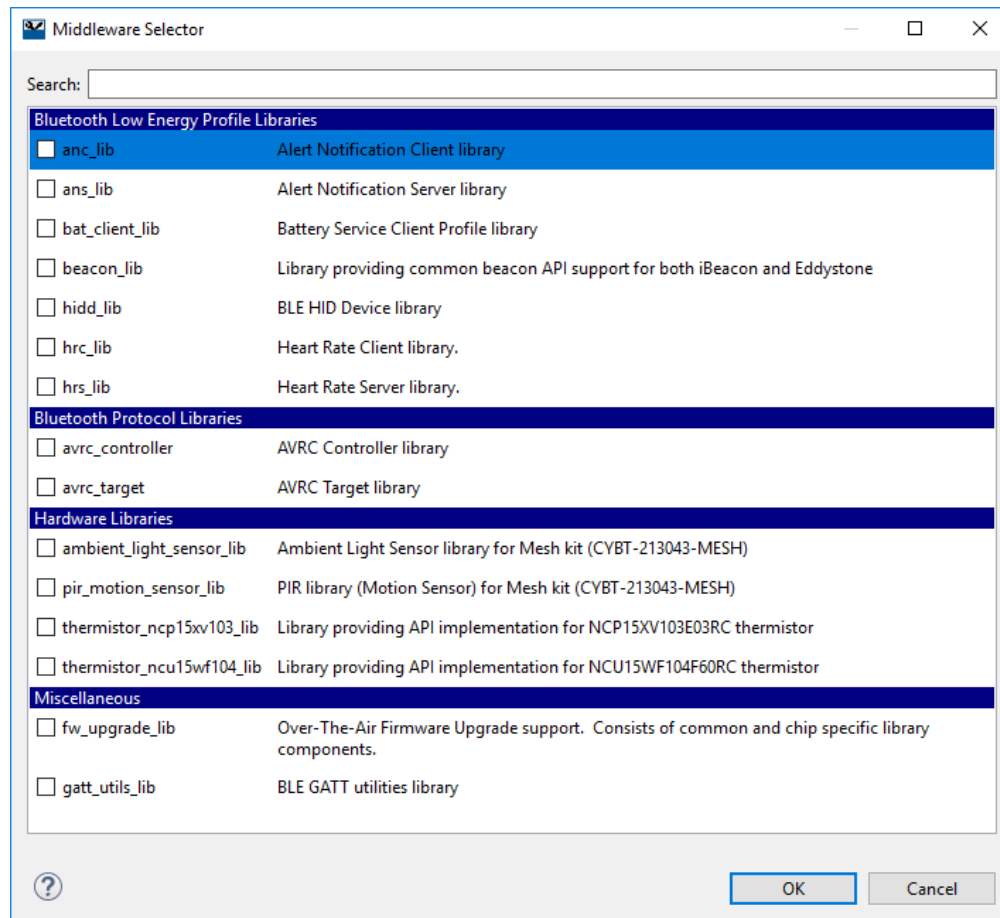


3. Middleware Selector

The Bluetooth SDK in ModusToolbox provides a 'Middleware Selector' dialog to select various middleware components for developing Bluetooth applications. To launch the Middleware Selector dialog, right-click the top-level project name in the project explorer, and select the **ModusToolbox Middleware Selector** option (refer to [Figure 27](#) for this option). Alternatively, you can click the **Select Middleware** link in the Quick panel. [Figure 29](#) shows the Middleware Selector dialog for the template application.

For the Find Me application, we will not be using any of these libraries, but you can refer to other code examples in the repository to understand how specific libraries are used. For example, the Beacon code example uses the *beacon_lib* and *fw_upgrade_lib* middleware components. Note that any time you make selections in the Middleware Selector, the *modus.mk* file gets updated automatically to include those middleware components in the relevant make variables. You do not have to manually edit the *modus.mk* file for middleware selection.

Figure 29. Middleware Selector



4.5 Part 3: Write the Application Code

At this point in the development process, you have created an application, configured the hardware resources, and generated the configuration code including the BLE GATT database. In this part, you examine the application code that implements the Find Me Target functionality.

Path	“Using CE directly” path (Evaluate existing Code Example (CE) directly)	“Working from Scratch” path (Use existing Code Example (CE) as reference only)
Actions	Ignore Step 1 - the CE already has all the necessary source files added. Read through the Firmware Description section to understand the firmware design.	Perform Step 1. Read through Firmware Description section to understand the firmware design.

The application code must do the major tasks as listed below.

- Perform system initialization including the Bluetooth stack.
- Implement Bluetooth stack event handler functions for different events like advertisement, connection, and attribute read/write requests.
- Implement user interface logic to update the LED state on the kit based on the events triggered.

We will be using a modular firmware development approach where each of the above tasks are contained in separate source/header files to aid in easy understanding of the code, and to enable code reuse.

1. Add files to your project (Required only for “Working from Scratch” flow)

- A. Locate the **BLE_FindMe** code example that you downloaded from the repository. See the [Prerequisites](#) section for details on the repository download process and code example location.
- B. Copy the following files from the **BLE_FindMe** code example top level folder to your *AppName_mainapp* folder inside the ModusToolbox workspace folder (*CE226123\mtw\CE226123_mainapp*). Replace any existing files using the native file explorer application of the OS. Alternately, you can drag/drop files into the *CE226123_mainapp* folder in the ModusToolbox IDE Project Explorer window.
 - *main.c*
 - *app_bt_event_handler.c*
 - *app_bt_event_handler.h*
 - *app_bt_cfg.c*
 - *app_bt_cfg.h*
 - *app_user_interface.c*
 - *app_user_interface.h*
 - *modus.mk*
- C. Delete the *empty_wiced_bt.c* file either from the ModusToolbox IDE project explorer or from your PC file explorer. You no longer need this file because the equivalent *application_start()* function is now part of the *main.c* file copied in the previous step.

4.6 Firmware Description

This section explains the application firmware of the Find Me application. The important source files relevant for the user application level code for this code example are listed in [Table 2](#).

Table 2. Important User Application Related Source Files

File Name	Comments
<i>cycfg_gatt_db.c</i> , <i>cycfg_gatt_db.h</i>	These files reside in the <i>GeneratedSource</i> folder under the application folder. They contain the GATT database information generated using the Bluetooth Configurator tool.
<i>app_bt_cfg.c</i> <i>app_bt_cfg.h</i>	These files contain the runtime Bluetooth stack configuration parameters like device name, and advertisement/connection settings.
<i>main.c</i>	Contains the <i>application_start()</i> function which is the entry point for execution of the user application code after device startup.
<i>app_bt_event_handler.c</i> <i>app_bt_event_handler.h</i>	These files contain the code for the Bluetooth stack event handler functions.
<i>app_user_interface.c</i> <i>app_user_interface.h</i>	These files contain the code for the application user interface (in this case, the LED) functionality.

4.6.1 BLE GATT Database

The *cycfg_gatt_db.c* and *cycfg_gatt_db.h* files contain the BLE GATT database definitions for the Find Me Target profile that was generated in the previous step using the Bluetooth Configurator tool. The GATT database is accessed by both the Bluetooth stack and the application code. The stack will directly access the attribute handles, UUIDs, and attribute permissions to process some of the Bluetooth events. The application code will access the GATT database to perform attribute read/write operations. The relevant database structures are listed below.

- `gatt_database[]`: This array contains the attribute handles, types and permissions. Note that this array does not contain the actual attribute values which are maintained as separate arrays as explained below.
- GATT Value Arrays**: The actual GATT database that contains the attribute values is declared as a series of `uint8_t` arrays under the section **GATT Initial Value Arrays** in *cycfg_gatt_db.c*. These arrays are also exposed as extern variables for application code access in the *cycfg_gatt_db.h* file. The FMP target application has these arrays defined by the names `app_gap_device_name[]`, `app_gap_appearance[]`, and `app_ias_alert_level[]`. `app_ias_alert_level[]` is the alert level characteristic corresponding to the IAS service that the Client will write to set the alert level. The actual write to this attribute is performed by the application code.
- `app_gatt_db_ext_attr_tbl[]`: This array of structures is a GATT lookup table that conveys the mapping of the attribute handles defined in `gatt_database[]` to the GATT value arrays. The application code uses this lookup table to perform the attribute read/write operations on the actual GATT arrays.

4.6.2 Bluetooth Stack Configuration Parameters

The *app_bt_cfg.c* and *app_bt_cfg.h* files contain the runtime Bluetooth stack configuration parameters like device name (`BT_LOCAL_NAME`), core stack configuration parameters (`wiced_bt_cfg_settings[]`), and the memory buffer pools used by the stack (`wiced_bt_cfg_buf_pools[]`). In the scope of this application note, we will not be covering these parameters. However, you can refer to the comments in the source files to learn about these parameters. Note that the device name defined in the `BT_LOCAL_NAME` variable is the one that will be used on the Peer device side to identify the device to establish a connection ("Find Me Target" in this case).

4.6.3 User Application Code Entry

The *main.c* file contains the *application_start()* function. This function is the entry point for execution of the user application code after device initialization is complete. In this code example, this function does two things:

- Selects the PUART serial port as the debug UART to view the trace messages and prints a startup message on the debug UART using the `WICED_BT_TRACE` function. The `WICED_BT_TRACE` function is used to send messages using `sprintf`-type formatting.

- Registers a Bluetooth stack management callback function by calling `wiced_bt_stack_init()`. The stack management callback function then typically controls the rest of the application based on Bluetooth events. Typically, only a minimal amount of application initialization is done in the `application_start()` function. Most of the application initialization is done in the stack callback function once the Bluetooth stack has been enabled. The stack callback function `app_bt_management_callback` is defined in `app_bt_event_handler.c`. A callback function is a function that is called by another function when a particular event happens.

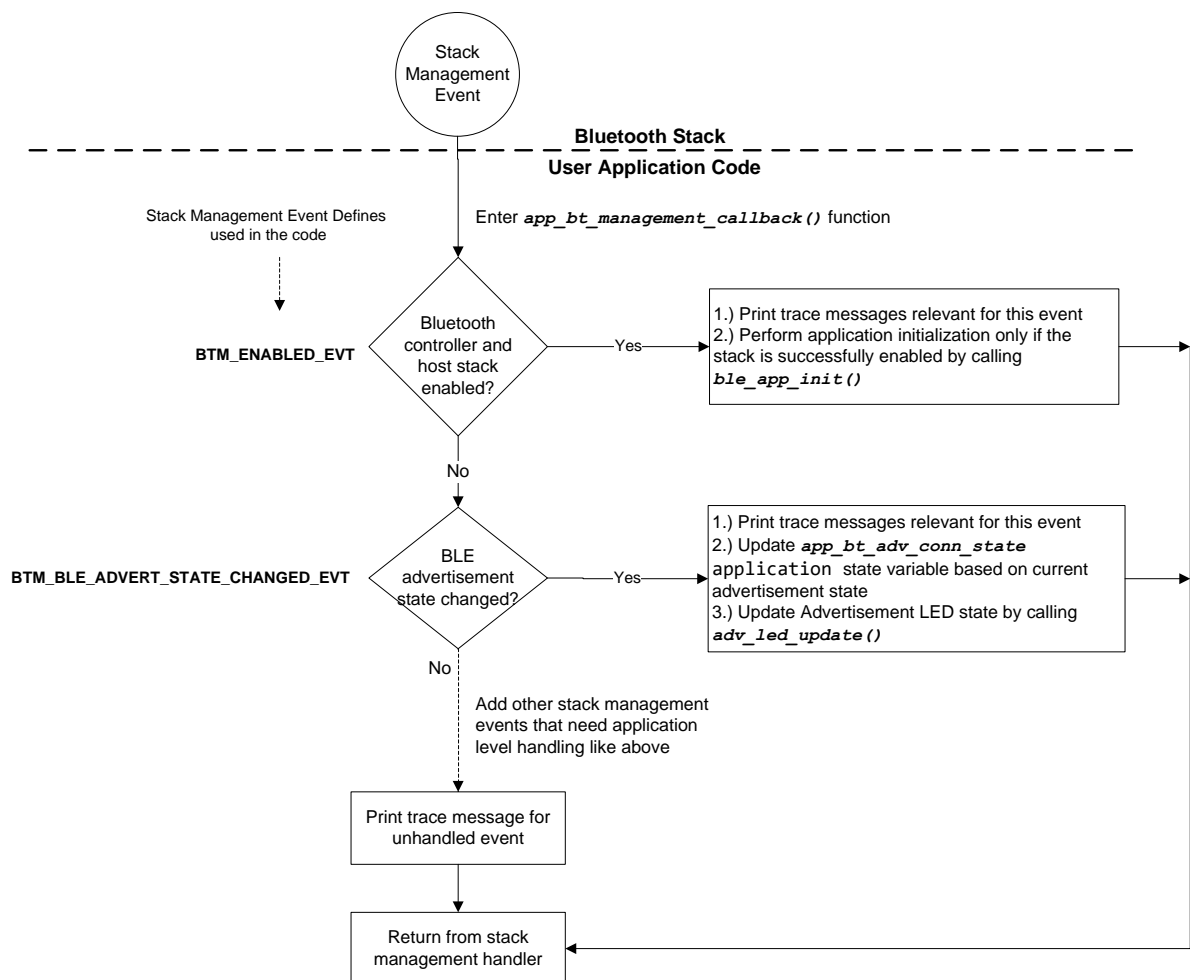
4.6.4 Bluetooth Stack Events

The `app_bt_event_handler.c` and `app_bt_event_handler.h` files contain the application code logic to handle the different types of events generated by the stack. At a high level, there are two categories of events that need to be handled: Bluetooth stack management events and GATT events.

4.6.4.1 Stack Management Events

The callback function `app_bt_management_callback` handles events like stack enabled, advertisement state change, security related events like pairing, and key exchange. This callback function is registered as a part of the `application_start()` function in `main.c`. Refer to the `wiced_bt_management_evt_t` definition in `wiced_bt_dev.h` for the list of management events. It is not required for the application code to handle all the management events. The events handled depend on the application requirements. Figure 30 shows the execution logic for the stack management event handler in this code example. As shown in Figure 30, only two management events (`BTM_ENABLED_EVT` and `BTM_BLE_ADVERT_STATE_CHANGED_EVT`) are handled in the stack management callback function.

Figure 30. Bluetooth Stack Management Event Handler Function Flow



At this point, it is pertinent to discuss the `BTM_ENABLED_EVT`. This is an essential management event that should be handled in all CYW20819 based applications, and it signifies that the Bluetooth stack has been enabled. All the application code initialization is done only after the Bluetooth stack has been enabled successfully by calling the `ble_app_init()` function as [Figure 30](#) shows.

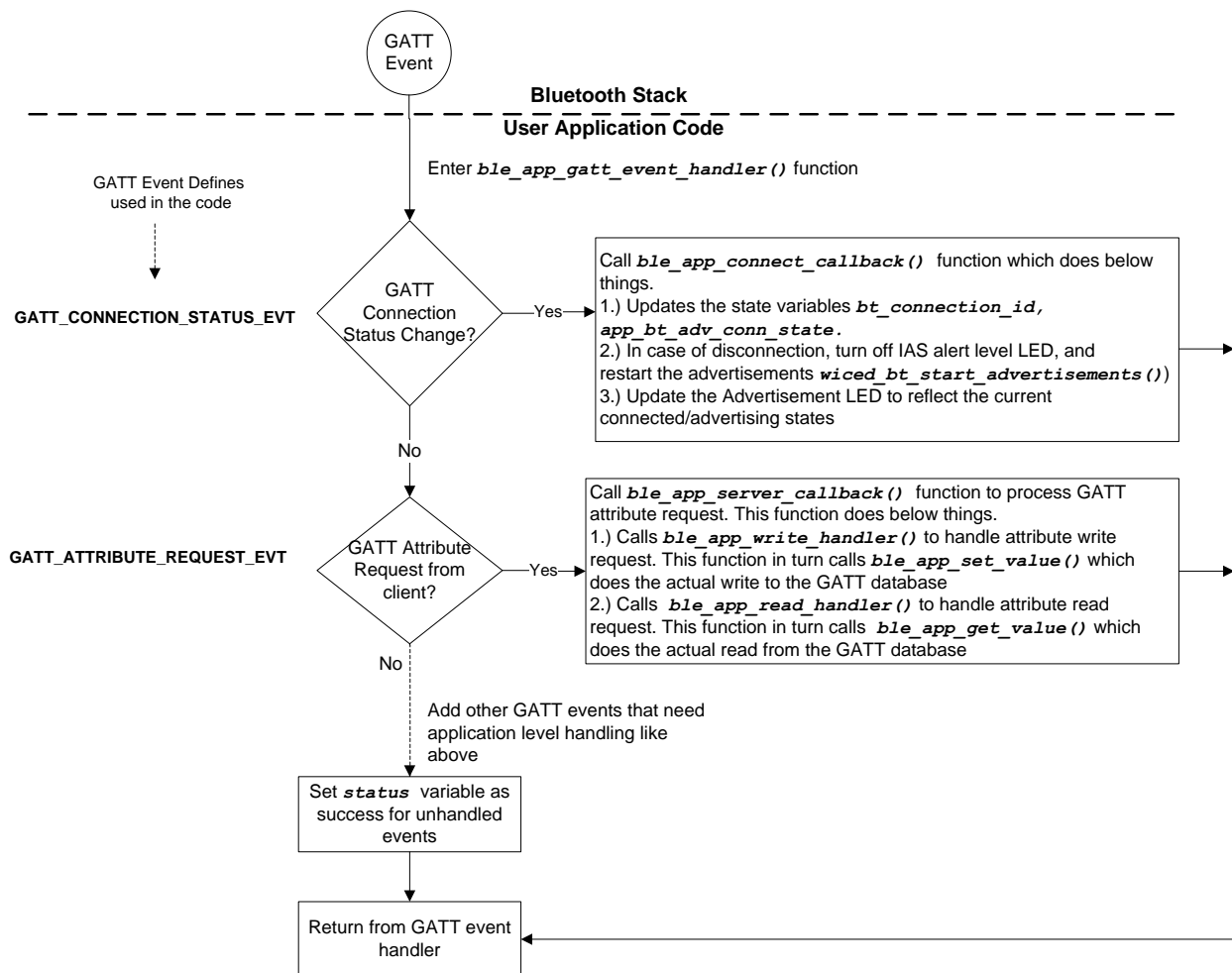
The `ble_app_init()` function, defined in `app_bt_event_handler.c`, performs the initialization tasks listed below. For any CYW20819-based application that you create, you should add the required initialization code in this function.

- Initializes the user interface (LED) logic by calling `app_user_interface_init()` defined in `app_user_interface.c`.
- Disables pairing by calling `wiced_bt_set_pairable_mode()`. For this application, the pairing feature is not used.
- Configure the advertisement packet data by calling `ble_app_set_advertisement_data()` defined in `app_bt_event_handler.c`. Look at this function definition in the code example to understand how to configure the elements of an advertisement packet.
- Register the callback function to handle GATT events (`ble_app_gatt_event_handler()`) by calling `wiced_bt_gatt_register()`.
- Initialize the GATT database (`gatt_database`) defined in `cycfg_gatt_db.c` by calling `wiced_bt_gatt_db_init()`.
- As the final step of the initialization process, the device starts advertising by calling the function `wiced_bt_start_advertisements()`.

4.6.4.2 GATT Events

The `ble_app_gatt_event_handler()` function handles GATT events like connection and attribute request events. This function is registered with a call to `wiced_bt_gatt_register()` from the `ble_app_init()` function in `app_bt_event_handler.c`. Refer to the `wiced_bt_gatt_evt_t` definition in `wiced_bt_gatt.h` for the list of GATT events. It is not required for the application code to handle all the GATT events. The events handled depend on the application requirements. Figure 31 shows the execution logic for the GATT event handler in this code example. As shown in Figure 31, only two GATT events (`GATT_CONNECTION_STATUS_EVT` and `GATT_ATTRIBUTE_REQUEST_EVT`) are handled in the function.

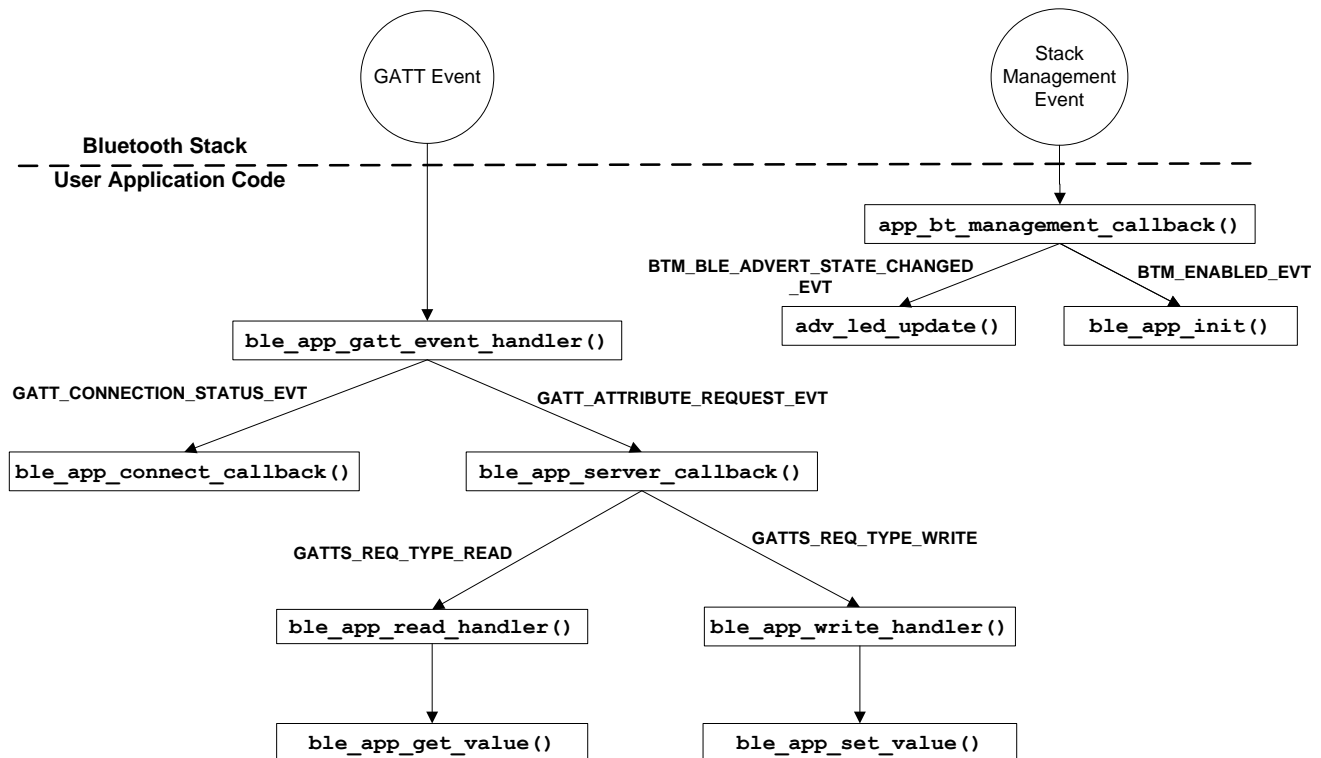
Figure 31. GATT Event Handler



At this point, it is pertinent to discuss about the `GATT_ATTRIBUTE_REQUEST_EVT`. This event is used to process the GATT Attribute read/write operations. Figure 32 gives information on the functions that are called in the case of read or write operations. In this code example, when the Find Me Locator updates IAS alert level characteristic on the CYW20819 device, `GATT_ATTRIBUTE_REQUEST_EVT` is triggered, which in turn calls the series of functions related to the attribute write request. At the end of the write operation, the `app_ias_alert_level[]` function in the GATT database in `cycfg_gatt_db.c` gets updated with the alert level set by the Find Me Locator and the LED is set appropriate to the alert level.

Figure 32 shows the function call chart summarizing the sequence of function calls for different stack events for this application. All these functions (except `adv_led_update()`) are defined in `app_bt_event_handler.c`. Refer to the source code to understand the implementation details of these functions.

Figure 32. Bluetooth Stack Events Function Call Chart



4.6.5 User Interface Logic

The *app_user_interface.c* and *app_user_interface.h* files contain the application code to handle the user interface logic. The design uses two LEDs for user interface whose details are given below.

- LED1 (Yellow LED) on the kit is used to indicate the advertising/connected state of the BLE peripheral device. LED1 is in the OFF state when the device is not advertising, blinking state while advertising, and always ON state when connected to the peer device. Refer to the *adv_led_update()* function for implementation details. A global state variable *app_bt_adv_conn_state* is used to update the LED state. The *adv_led_update()* function is called from two places in the application code:
 - The *app_bt_management_callback()* function updates LED1 when the advertisement state changes (stack management event *BTM_BLE_ADVERT_STATE_CHANGED_EVT*)
 - The *ble_app_connect_callback()* function updates LED1 when the connection state changes (GATT event *GATT_CONNECTION_STATUS_EVT*)
- LED2 (Red LED) on the kit is used to indicate the IAS alert characteristic level when the device is connected to a peer device. When connected to a peer device, LED2 is in the OFF state for low alert, blinking state for mid alert, and ON state for high alert. When the device is not connected to any peer device, LED2 is in the OFF state. Refer to the *ias_led_update()* function for implementation details. The *adv_led_update()* function is called from two places in the application code:
 - The *ble_app_set_value()* function updates LED2 when an attribute write request to the IAS alert level characteristic is done from the Client side.
 - The *ble_app_connect_callback()* function drives LED2 to the OFF state when a disconnection occurs (GATT event *GATT_CONNECTION_STATUS_EVT*)

4.7 Part 4: Build, Program, and Test Your Design

This section shows how to build the application and program the CYW20819 device on the CYW920819EVB-02 kit. It also explains how to test the Find Me Profile BLE design using the CySmart mobile app, and using the PUART serial interface to view the Bluetooth stack and application trace messages.

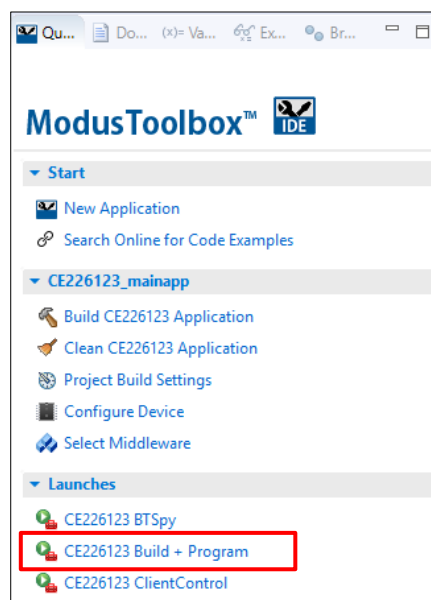
At this point, it is assumed that you have followed the previous steps in this application note to develop the Find Me Profile application.

Path	“Using CE directly” path (Evaluate existing Code Example (CE) directly)	“Working from Scratch” path (Use existing Code Example (CE) as reference only)
Actions	Perform all the steps in this section	Perform all the steps in this section

1. Connect the kit to your PC using the provided USB cable.
2. The USB Serial interface on the kit provides access to the two UART interfaces of the CYW20819 device – WICED HCI UART, and WICED Peripheral UART (PUART). The HCI UART interface is used only for downloading the application code in this code example; the PUART interface is for printing the Bluetooth stack and application trace messages. Use your favorite serial terminal application and connect to the PUART serial port. Configure the terminal application to access the serial port using the following settings.

Baud rate: 115200 bps; Data: 8 bits; Parity: None; Stop: 1 bit; Flow control – None; New line for receive data: Line Feed (LF) or Auto setting
3. **Build and Program the Application:** In the project explorer, select the **<App Name>_mainapp** project. In the Quick Panel, scroll to the **Launches** section, and click the **<App Name> Build + Program** configuration as shown in [Figure 33](#).

Figure 33. Programming the CYW20819 Device from ModusToolbox



Note 1: If the download fails, it is possible that a previously loaded application is preventing programming. For example, the application may use a custom baud rate that the download process does not detect, or the device may be in a low-power mode. In that case, it may be necessary to put the board in recovery mode, and then try the programming operation again from the IDE. To enter recovery mode, first, press and hold the **Recover** button (SW1), then press the **Reset** button (SW2), release the **Reset** button (SW2), and then release the **Recover** button (SW1).

Note 2: If you encounter errors in the application Build process, read the error messages in the IDE console window, and revisit the relevant previous steps in this document to check if you have missed or wrongly done any of those steps.

4. To test using the CySmart mobile app, follow the steps below (see equivalent CySmart app screenshots in [Figure 34](#) for iOS, [Figure 35](#) for Android):
 - a. Turn ON Bluetooth on your Android or iOS device.
 - b. Launch the CySmart app.
 - c. Press the reset switch on the CYW920819EVB-02 kit to start sending advertisements. The yellow LED (LED1) starts blinking to indicate that advertising has started. Advertising will stop after 90 seconds if a connection has not been established.
 - d. Swipe down on the CySmart app home screen to start scanning for BLE Peripherals; your device appears in the CySmart app home screen. Select your device to establish a BLE connection. Once the connection is established, the yellow LED (LED1) changes from blinking state to always ON state.
 - e. Select the 'Find Me' Profile from the carousel view. (Swipe left or right to rotate the carousel).
 - f. Select an Alert Level value on the Find Me Profile screen. Observe that the state of the red LED (LED2) on the device changes based on the alert level.

Figure 34. Testing with the CySmart App on iOS

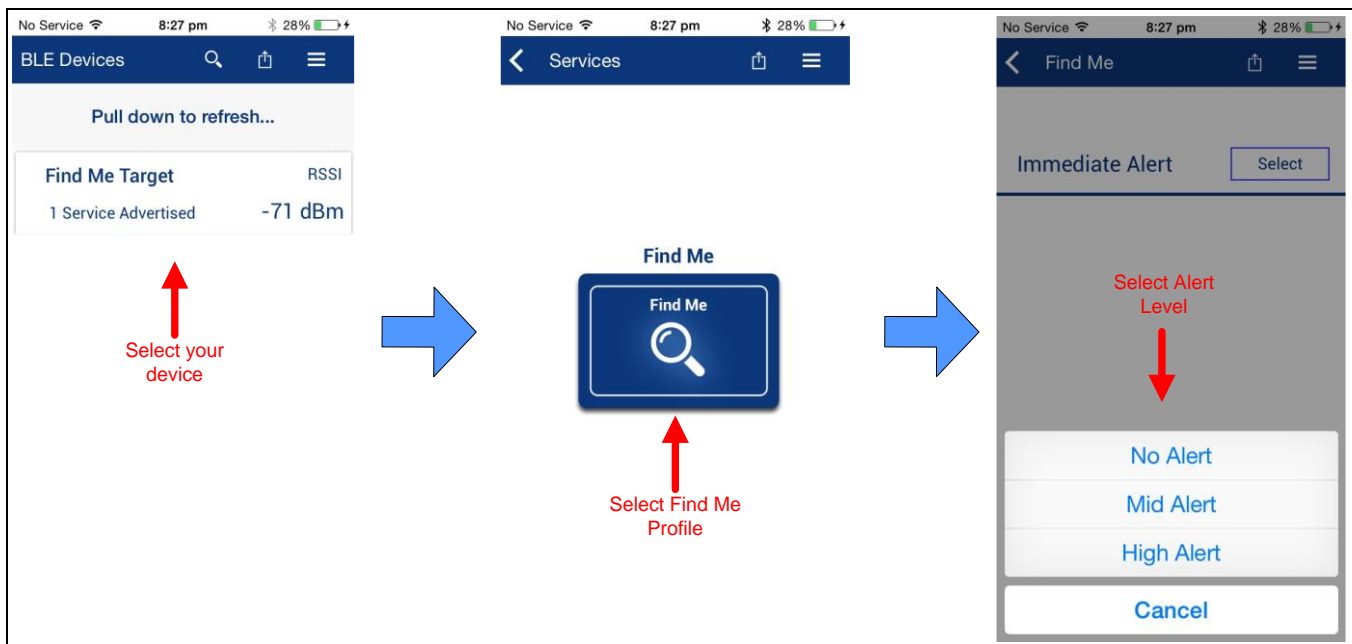
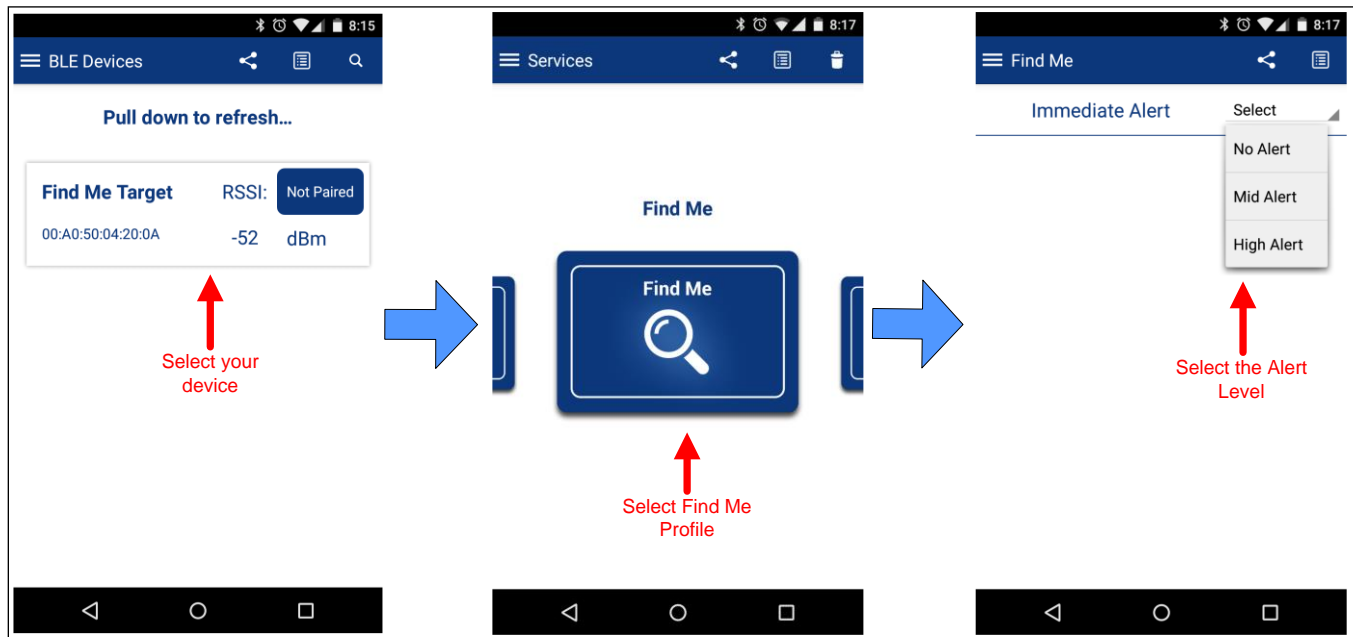
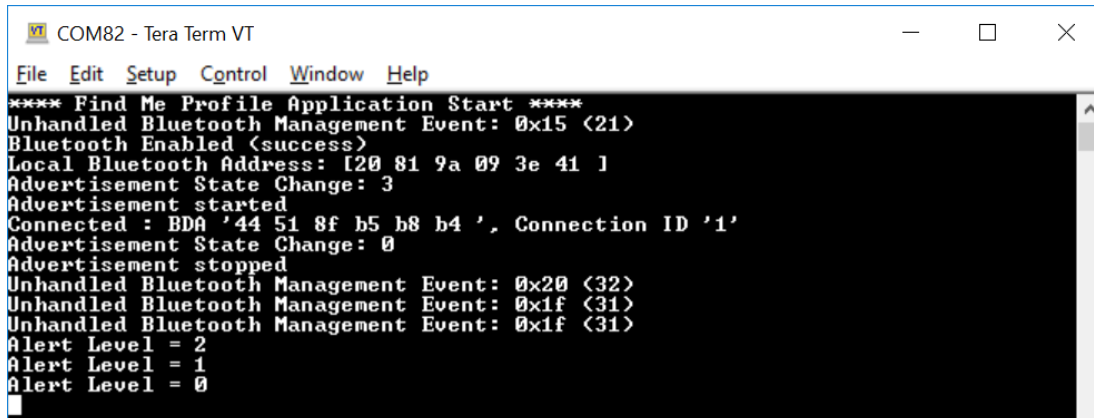


Figure 35. Testing with the CySmart App on Android



5. Use the UART serial port to view the Bluetooth stack and application trace messages in the terminal window as shown in Figure 36.

Figure 36. Log Messages on WICED UART Serial Port



Now you have successfully developed a simple BLE application for the CYW20819 device using ModusToolbox IDE. For further learning about CYW20819 device including technical documents, additional code examples, refer to the [Technical Resources](#) section.

5 Summary

This application note explored the CYW20819 BT MCU device architecture, the associated development tools, and the steps to create a simple BLE application for CYW20819 using ModusToolbox. CYW20819 is a BT 5.0-compliant, standalone baseband processor with an integrated 2.4-GHz transceiver with support for both BLE and classic BT. The device is intended for use in audio (source), sensors (medical, home, security), HID and remote-control functionality as well as a host of other IoT applications.

Cypress provides a wealth of code examples, application notes, and other technical documents to help you quickly develop CYW20819 based Bluetooth applications that meets your end application requirements. Refer the [Technical Resources](#) section to continue learning more about the CYW20819 device and develop Bluetooth applications.

6 Technical Resources

Application Notes	
AN225684 – Getting Started with CYW20819	This document.
Code Examples	
Visit the CYW20819 code examples repository in Cypress GitHub portal for a comprehensive collection of code examples using ModusToolbox IDE	
Device Documentation	
CYW20819 Device Datasheet	
Development Kits	
CYW920819EVB-02 Evaluation Kit	
Tool Documentation	
ModusToolbox IDE	The Cypress IDE for IoT designers

Document History

Document Title: AN225684 – Getting Started with CYW20819

Document Number: 002-25684

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6490998	VVSK	02/21/2019	New application note

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at [Cypress Locations](#).

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#) | [PSoC 6 MCU](#)

Cypress Developer Community

[Community](#) | [Code Examples](#) | [Projects](#) | [Videos](#) | [Blogs](#)
| [Training](#) | [Components](#)

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, "Security Breach"). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. "High-Risk Device" means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. "Critical Component" means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress's published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.