CE226300 – BLE Environment Sensing Temperature with CYW20819

Objective

This code example demonstrates the implementation of a simple BLE Environmental Sensing Profile. It shows how to sense environmental conditions like temperature by using an on-chip ADC and sending values over a BLE 2M PHY link using the CYW20819 Bluetooth SoC. The instructions provided in this document are given for the CYW920819EVB-02 kit; however, they are also applicable for other supported kits that use the ModusToolbox™ IDE to build the code example.

Requirements

Tool: ModusToolbox™ IDE 1.1 with BT SDK 1.3 and later

Programming Language: C

Associated Parts: CYW20819, CYW20820, CYW20719

Related Hardware: CYW920819EVB-02, CYW920820EVB-02, CYW920719B2Q40EVB-01

Overview

This code example demonstrates the BLE Environmental Sensing Service (ESS) specification from the Bluetooth SIG on a supported device as listed in the requirements. The application demonstrates the ADC and BLE 5.0 (LE 2M PHY) capability of the Bluetooth SoC. “BLE_EnvironmentSensingTemperature” is the name of the code example that is available in ModusToolbox.

See the “Getting Started” section of ModusToolbox IDE User Guide to learn how to create an application from a code example. Some code examples are available within ModusToolbox while others need to be downloaded from GitHub. See the “Getting Started” section of the ModusToolbox IDE User Guide to learn how to download code examples from GitHub. This code example document also describes the hardware connections necessary for the desired behavior of the code example and how to verify the output.

Hardware Setup

This example uses the kit’s default configuration. Refer to the kit user guide, if required, to ensure that the kit is configured correctly.

The board should be connected to a PC using the USB micro-B cable. Also, make sure that the Thermistor Enable (J14) and Peripheral Enable (J18) jumpers are in the correct positions (shorted). See the “Jumper Setting” section in the kit user guide.

Software Setup

This code example consists of two parts: a BLE Peripheral device (GATT Server) and a BLE Central device (GATT Client). A smartphone or a PC will act as the BLE Central device whereas the programmed kit will act as the BLE Peripheral device.

For the Central device, download and install the CySmart app for iOS or Android. You can also use the CySmart Host Emulation Tool desktop application if you have access to the CY5677 CySmart BLE 4.2 USB Dongle. Alternatively, you can use any other mobile app that supports the BLE Environment Sensing Service.

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

iOS

Android

Operation

1. Connect the desired kit (for example, CYW920819EVB-02 Evaluation Kit) to your PC using the provided USB cable.

   The kit's USB interface should enumerate as two serial communication (COM) ports – WICED HCI UART and WICED Peripheral UART (PUART).

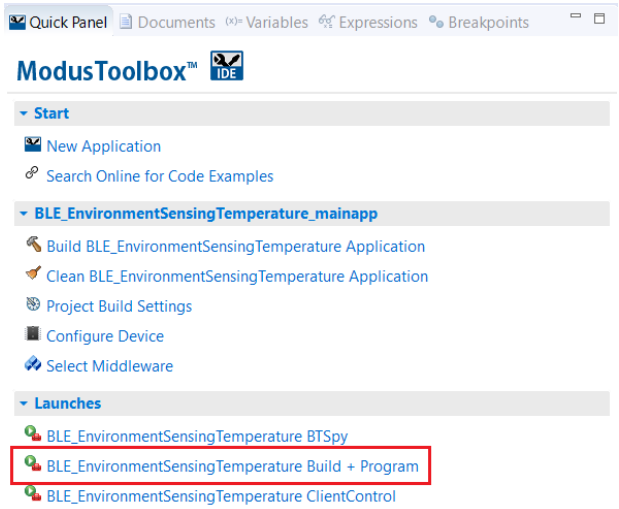
   - The WICED HCI UART port is used by the ModusToolbox programming interface for downloading the application code; no user configuration is required for this port.
   - The WICED Peripheral UART port is used in this code example for printing the BLE stack event messages and other application-level log messages. Use a serial terminal emulation tool such as PuTTY or Tera Term. See the respective kit user guide for more details on USB driver installation.

2. You can browse the collection of starter applications during application creation through **File** > **New** > **ModusToolbox IDE Application** and then choose the BLE_EnvironmentSensingTemperature application.

   NOTE: You can also import the code example from the downloaded or cloned GitHub repository into a new workspace. The CYW920819EVB-02, CYW920820EVB-02, and CYW920719B2Q40EVB-01 kits are pre-programmed with this Environment Sensing Profile code example out-of-the-box. For the most recent version of the code example, check the GitHub repository. You can find the Environmental Sensing Profile code example in this path: <kit_name>/apps/demo/env_sensing_temp. If you are not familiar with the import process, see KBA225201.

3. Program the CYW20819 device on the kit. In the project explorer, select the **<App Name>** project. In the Quick Panel, scroll to the **Launches** section and click the **Build + Program** configuration as shown in Figure 1.

   Figure 1. Quick Links of ModusToolbox



   Note: If the download fails, it is possible that a previously loaded application is preventing programming. For example, the application might use a custom baud rate that the download process does not detect. 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 and release the **Reset** button (SW2), and finally release the **Recover** button (SW1).

4. Use the PUART serial interface to view the Bluetooth stack and application trace messages:

   a. Use a serial terminal emulation tool and connect to the **PUART** serial port. Configure the terminal emulation tool to access the serial port using settings listed in Table 1.

Table 1. WICED PUART Settings

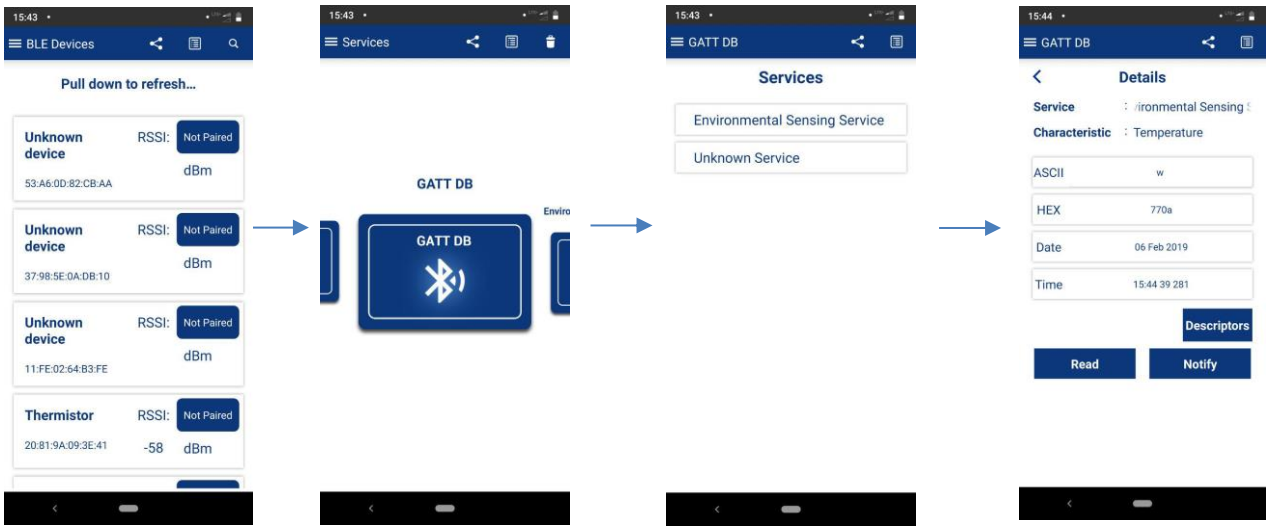
WICED PUART Serial Port Configuration	Value
Baud rate	115200 bps
Data	8 bits
Parity	None
Stop	1 bit
Flow control	None
Line Feed option	CR+LF or Auto

b. The Bluetooth stack and application trace messages will appear in the terminal window as shown in Figure 2. You may have to reset the device after opening the terminal emulator as you might have missed the initial trace messages after programming.

Figure 2. Log Messages on WICED PUART Serial Port

5. To test using the CySmart mobile app:
   a. Turn ON Bluetooth on your Android or iOS device.
   b. Launch the CySmart app. Figure 3 shows the steps for using the CySmart Android app. The screenshots provided in this document are from the Android app; these will vary slightly from the iOS version.

Figure 3. Testing with the CySmart App on Android



   c. Pull (swipe) down on the home screen of the CySmart app to start scanning for BLE Peripherals; your device appears in the CySmart app home screen with the Bluetooth device name as “Thermistor”. Select your device to establish a BLE connection. Once the connection is established, the red LED (LED2) turns ON.
   d. Select **GATT DB** from the carousel view. The CySmart app displays the available services.
6. Select **Environmental Sensing Service** from the list of available services. In the CySmart app for iOS devices, the service will be shown as an ‘unknown service’.
   a. At the bottom of the screen, you can find three buttons, namely,
      i. Descriptors (To view additional information about the Environmental Sensing Profile such as valid range, measurement interval, and measurement techniques)
      ii. Read (To read the last sampled temperature value once)
      iii. Notify (To receive notifications when the server sends an updated temperature value. The code example is configured to send a temperature value every 5 seconds)
   b. Tap the Notify button in the bottom of the screen.

      The CySmart app displays the temperature characteristic in two representations (ASCII and hexadecimal values), and the date and time of the last received temperature notification.

      The bytes in the 2-byte hexadecimal value need to be swapped to interpret the temperature. For example, if the hexadecimal representation in the CySmart app is 0xCE08, then swapping the bytes result in 0x08CE which is the decimal equivalent of 2254. The most significant byte (MSB) represents the integer part of the temperature in Celsius and the least significant byte (LSB) represent the fractional part of the temperature in degree celsius (i.e., 22.54 °C).

Upon disconnecting the Central device from the Peripheral device by user action in the CySmart App or by moving out of the range, the red LED (LED2) turns OFF. You will be notified in the PUART log about the remote device’s Bluetooth address that has recently disconnected.

Design and Implementation

This code example is written in a way that is generic across multiple devices and platforms. The code example application runs on the Arm® Cortex®-M4 core of the CYW20819, CYW20820, or CYW20719 Bluetooth SoC. The functionality of the code example will remain the same on all the supported kits; the application was developed using the ModusToolbox IDE.

This application demonstrates the ADC and BLE capability of the CYW20819 device. On the CYW920819EVB-02 kit, a negative temperature coefficient thermistor is used to measure the ambient temperature. The thermistor present on the board depends on the supported platform. For example, the CYW920819EVB-02 evaluation board uses the NCU15WF104F60RC temperature sensor from Murata. It is a 100-kΩ thermistor at 25°C with 1% accuracy connected to GPIO P8 on the CYW20819 device. The ADC is used to measure the voltage across the thermistor channel to determine the temperature. The temperature value is sent over BLE to a Central device, and to the PUART as debug trace messages. This project demonstrates the following features:

- Functionality of the ADC for temperature measurement
- BLE Environment Sensing Service (ESS) – GATT Read and Notify functionality
- Debug Trace messages
- Connection with one Central device
- BLE 5.0 Feature – LE 2M PHY if the Central device supports 2M PHY
- Connection status indication through LED
- Secure and non-secure Over-the-Air (OTA) firmware upgrade functionality

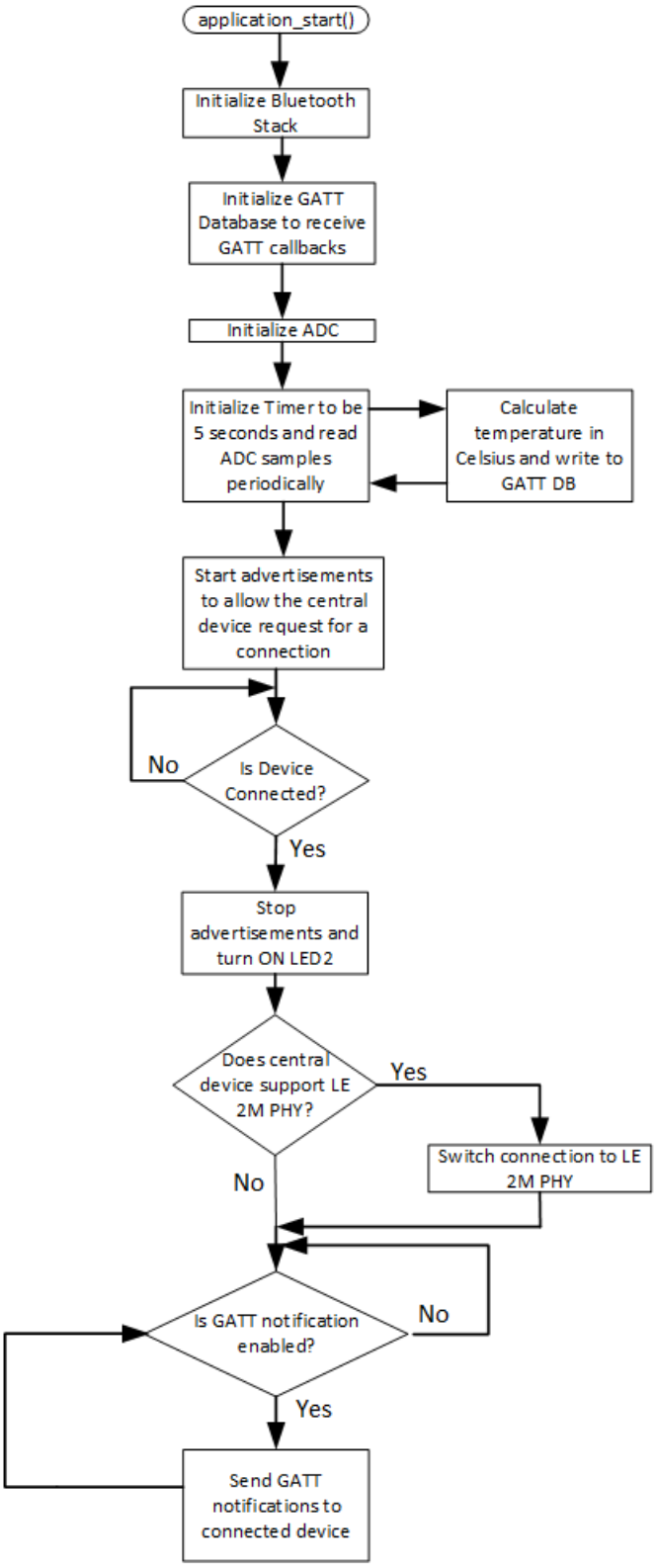
The project consists of the following files:

Table 2. BLE_EnvironmentSensing Project Files

File Name	Comments
thermistor_app.c	This file contains the `application_start()` function, which is the entry point of the user code execution and can be considered as the equivalent of the main() function in standard C. The application_start function initializes the Bluetooth stack by calling `wiced_bt_stack_init()`. This function also registers a Bluetooth management event callback function. After the Bluetooth stack is initialized, you have the control to execute your own application code based on different Bluetooth events received in the management callback. This management callback acts as a Finite State Machine (FSM) in executing the application code. For the Bluetooth Enabled Event (`BTM_ENABLED_EVT`), the `thermistor_app_init()` function is called. This function initializes the ADC, starts a timer to measure the temperature periodically by calling `seconds_timer_temperature_cb()`, registers a callback for GATT events, initializes the GATT database, and starts BLE advertisements. The thermistor_app.c file also holds the function `thermistor_set_advertisement_data()` which sets the advertisement data for the device to be discovered.
thermistor_gatt_handler.c *thermistor_gatt_handler.h*	These files consist of the code to perform GATT read handler, GATT write handler and GATT indication confirm handler. These files also hold the code to switch to LE 2M PHY if the Central device supports it.
thermistor_util_functions.c *thermistor_util_functions.h*	These files consist of the utility functions that will help make debugging and developing the application easier by providing more meaningful information. For example, this API provides meaningful strings for Bluetooth events, Bluetooth advertisement modes, GATT status messages, and GATT disconnection reasons.
wiced_app_cfg.c	These files contain the runtime Bluetooth stack configuration parameters like device name, advertisement/connection interval, and buffer pool configurations.
cycfg_bt.h *cycfg_gatt_db.c* *cycfg_gatt_db.h*	These files reside in the *GeneratedSource* folder under the application folder. They contain the GATT database information generated using the Bluetooth Configurator tool. There are a few manual edits made to these files to enable OTA firmware upgrade in the code example which is explained in the Over-the-Air (OTA) Firmware Upgrade section.
ecdsa256_pub.c	This file resides in the *secure* folder under the application folder. This file contains the public key for ecdsa256 encryption used in Secure OTA Firmware upgrade.
readme.txt	This file provides minimal information about the code example such as how to download the application and what to observe from the application, etc.

Flowchart

Figure 4. Flowchart of the BLE Environment Sensing Temperature Application



The flowchart as shown in Figure 4 gives an overview of the Environmental Sensing Profile application. The Environmental Sensing Profile in this application consists of one of the Bluetooth SIG-defined services, namely the Environment Sensing Service (ESS). This project implements only the temperature characteristic from the environmental sensing service. This characteristic supports notification and read operations, which allows the GATT Server to send data to the connected GATT Client device whenever new data is available and a read to be performed from the GATT Client device. The Bluetooth Configurator provided by ModusToolbox makes it easier to design and implement the GATT DB. The Bluetooth Configurator generates the *cycfg_gatt_db.c* and *cycfg_gatt_db.h* files. All Environmental Sensing Profile-related variables and functions are contained in these files. When the GATT DB is initialized after Bluetooth Stack initialization, all profile-related values will be ready to be advertised to the Central device. The procedure on how to use the Bluetooth Configurator is discussed later in this document.

The code example utilizes the on-board analog temperature sensor (thermistor) to send temperature values over BLE. The temperature sensor component on the board vary depending on the platform. On the CYW920819EVB-02, it is a Negative Temperature Coefficient (NTC) temperature sensor whose resistance increases when ambient temperature increases. The datasheet of the thermistor component provides details on the characteristics for varying resistance with respect to temperature. The thermistor library provides APIs independent of the thermistor interface on the board to the Cypress Bluetooth chip. This code example utilizes the thermistor library from BT SDK 1.3 and later.

The CYW20719 has a Hardware Floating Point (FPU) whereas CYW20819/CYW20820 do not have a Hardware Floating Point Unit (FPU). The source files of the thermistor library hold a mapping table between the thermistor's resistance values and temperatures. The lookup function calculates the temperatures at the previous resistance and the next resistance to calculate the slope. Using this slope value, an accurate temperature is calculated. The temperature value is sent once every 5 seconds to the Central device when connected and GATT notifications are enabled by the Central. To send the temperature values once every 5 seconds, an application timer is used which can be configured as MILLI_SECONDS_PERIODIC_TIMER or SECONDS_PERIODIC_TIMER.

When the Peripheral device is connected, LED2 will be ON; when it is disconnected, LED2 will be OFF. To turn the LED ON and OFF, generic GPIO functions are used to drive the output pin HIGH or LOW. The LEDs present in the supported kits are active LOW LEDs, which means that the LED turns ON when the GPIO is driven LOW.

The *thermistor_gatt_handler.c* and *thermistor_gatt_handler.h* files handle the functionality of GATT callbacks from the Central device. On receiving a connection request, the Bluetooth stack gives a GATT event to the application of `wiced_bt_gatt_evt_t` type. For example, the LED toggle functionality is implemented in the GATT connection callback. The connection event also sets the PHY Rx and Tx connection to LE 2M PHY if the Central device supports it as specified by the Bluetooth 5.0 standard. On a disconnection event, the code resets the Client Characteristic Configuration Descriptor (CCCD) value so that on a reconnect event, notifications will be disabled.

In *wiced_app_cfg.c,* all the runtime Bluetooth stack configuration parameters are defined; these will be initialized during Bluetooth stack initialization. Some of the configurations include device name, connection interval, advertisement interval, advertisement channels to use, number of client connections, and Maximum Transmission Unit (MTU). You also have the flexibility to configure the buffer pool size, which helps in optimizing the memory and transmission rate depending on the application use case.

The project also contains *thermistor_util_functions.c* and *thermistor_util_functions.h* which provide APIs to see meaningful messages in debug logs in the PUART. Most of the status messages in Bluetooth stack are represented in enumerated values. These functions allow you to view the respective strings instead of the enumerated values.

Over-the-Air (OTA) Firmware Upgrade

This code example demonstrates OTA firmware upgrade using a middleware library called *fw_upgrade_lib* and a custom BLE service called *FWUpgradeService*. The service and its characteristic have fixed UUIDs, properties, and permissions. These have been pre-configured in the code example with the correct values and settings using the Bluetooth Configurator as shown in the Resources and Settings section.

The library functions expect specific handles for the OTA service and characteristics. If you modify/save the Bluetooth Configurator settings, it may be necessary to manually edit the *cycfg_gatt_db.h* file inside the GeneratedSource folder to use the following handle definitions after saving from the Bluetooth Configurator.

```
#include "wiced_bt_ota_firmware_upgrade.h"

/* Service FWUpgradeService */
#define HDLS_FWUPGRADESERVICE
HANDLE_OTA_FW_UPGRADE_SERVICE
/* Characteristic FWUpgradeContolPoint */
```

```
#define HDLC_FWUPGRADESERVICE_FWUPGRADECONTOLPOINT
HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_CONTROL_POINT
#define HDLC_FWUPGRADESERVICE_FWUPGRADECONTOLPOINT_VALUE
HANDLE_OTA_FW_UPGRADE_CONTROL_POINT
/* Descriptor Client Characteristic Configuration */
#define HDLD_FWUPGRADESERVICE_FWUPGRADECONTOLPOINT_CLIENT_CHAR_CONFIG
HANDLE_OTA_FW_UPGRADE_CLIENT_CONFIGURATION_DESCRIPTOR
/* Characteristic FWUpgradeData */
#define HDLC_FWUPGRADESERVICE_FWUPGRADEDATA
HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_DATA
#define HDLC_FWUPGRADESERVICE_FWUPGRADEDATA_VALUE
HANDLE_OTA_FW_UPGRADE_DATA
```

Initialization of the OTA library is done during application initialization in *thermistor_app.c*. All other OTA functionality is contained in the GATT callback handler in *thermistor_gatt_handler.c*.

The file used for OTA upgrade is created during a build and is called *mainapp_download.ota.bin*. The application's build folder depends on the build configuration used; by default, it is set to Debug. The build must have the setting *OTA_FW_UPGRADE=1* for the *.bin* file to be created. This is setting is enabled by default in the code example.

Cypress provides executables and source code for OTA firmware upgrade applications for Windows 10 and Android in the SDK installation folder at:

```
<install_folder>/ModusToolbox_1.1/libraries/bt_sdk-1.1/components/BT-
SDK/common/peer_apps/ota_firmware_upgrade.
```

See the *readme.txt* file in that folder for additional information.

The code example is configured for a non-secure OTA service by default. Secure OTA uses elliptic curve cryptography to implement a secure method to upgrade the firmware of Bluetooth devices. Secure OTA can be enabled by defining the build setting *OTA_SECURE_FIRMWARE_UPGRADE*. The Secure OTA settings can be changed by right-clicking on the <**App Name**> project in the Workspace Explorer and selecting **Change Application Settings…**

This setting changes the UUID for the OTA service in the *cycfg_gatt_db.c* file as shown below. Note that if you modify/save the Bluetooth Configurator settings, it may be necessary to manually edit the *cycfg_gatt_db.c* file in the *GeneratedSource* folder.

```
#ifdef OTA_SECURE_FIRMWARE_UPGRADE
    PRIMARY_SERVICE_UUID128(HDLS_FWUPGRADESERVICE, UUID_OTA_SEC_FW_UPGRADE_SERVICE),
#else
    /* Primary Service: FWUpgradeService */
    PRIMARY_SERVICE_UUID128 (HDLS_FWUPGRADESERVICE, __UUID_SERVICE_FWUPGRADESERVICE),
#endif
```

Secure OTA uses public/private key encryption. The code example has a file located at *secure/ecdsa256_pub.c* containing the public key. To use secure OTA, you should generate your own public/private key pair using the tools available in the following folder or your preferred tool set.:

<intall_folder>/ModusToolbox_1.1/tools/wiced-tools-1.0/BT/ecdsa256/bin

The executable *ecdsa_genkey.exe* will generate the key pair and the *ecdsa256_pub.c* file, which should be replaced in the *secure* folder in the code example.

The executable *ecdsa_sign.exe* is used to sign the *.bin* file after a build so that it can be loaded using the OTA upgrade service.

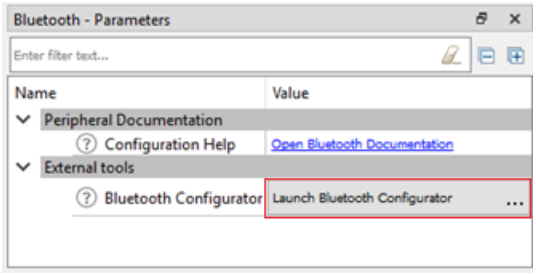
For more details on the Secure OTA firmware upgrade, see the *WICED Secure Over-the-Air Firmware Upgrade* document in ModusToolbox Help.

Resources and Settings

This section explains the ModusToolbox resources and their configuration as used in this code example. Note that all the configuration explained in this section has already been done in the code example. 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 source files. These files are stored in the application's *GeneratedSource* folder.

- **Device Configurator:** The Device Configurator is used to enable and 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. Any time you make a change in the Device Configurator, click **File** > **Save** to save the updated configuration. This code example uses the default platform configuration for supported platforms.
- **Bluetooth Configurator**: The Bluetooth Configurator is used for creating and modifying the BLE GATT database. 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** shortcut under the **Bluetooth-Parameters** window, as shown in Figure 5.

Figure 5. Launch Bluetooth Configurator



The Bluetooth Configurator settings for this code example are shown in Figure 6. Note that the Environmental Sensing Service corresponding to the Environmental Sensing Profile has been added as shown in Figure 7. Figure 7 shows how the Environment Sensing profile was added to create the GATT database. Figure 8 shows the OTA FWUpgradeService attributes that is used in this code example using Bluetooth Configurator. Any time you make a change in the Bluetooth Configurator, ensure that you save the changes.

Figure 6. Bluetooth Configurator

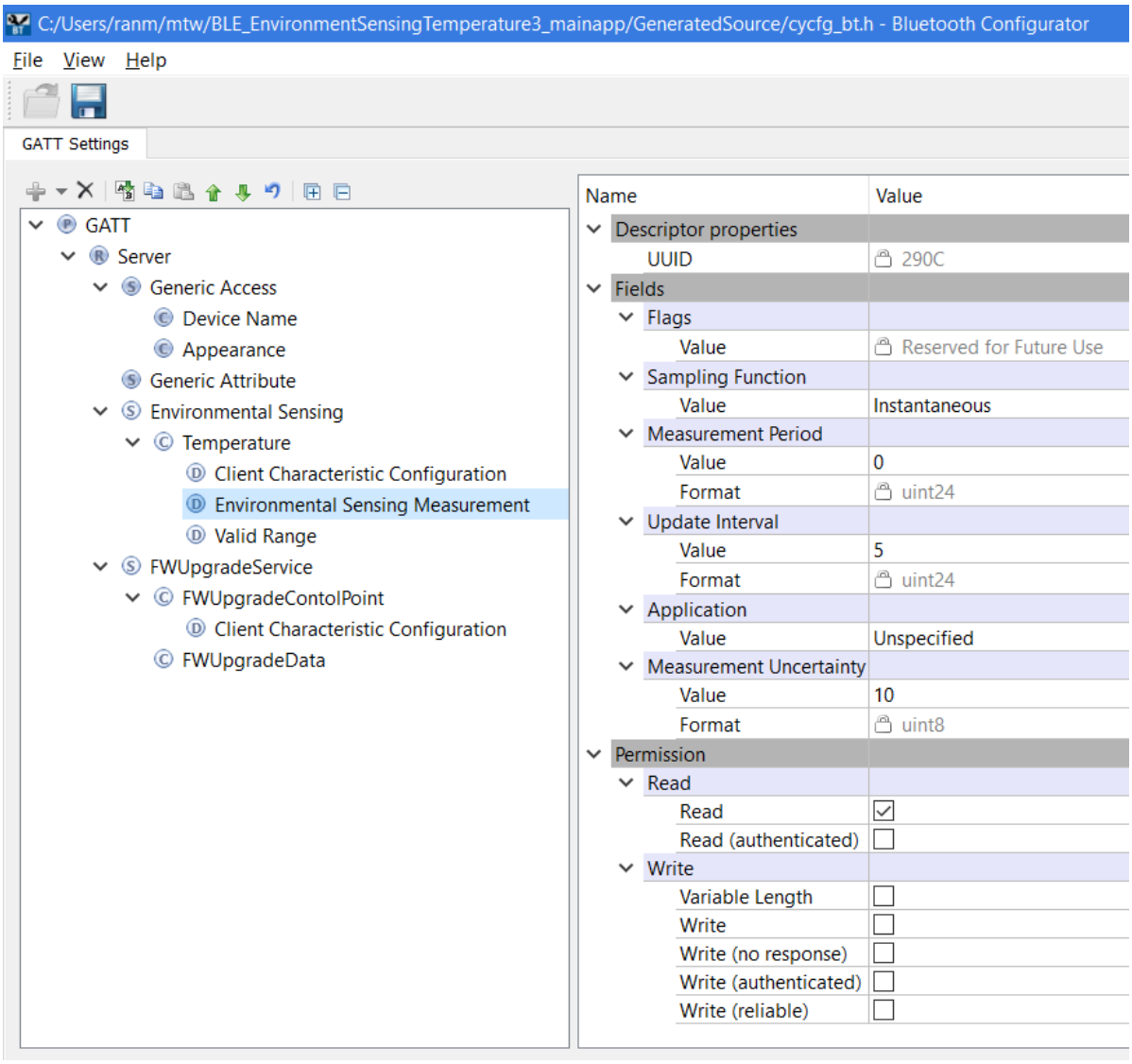


Figure 7. Environment Sensing Profile Bluetooth Configurator

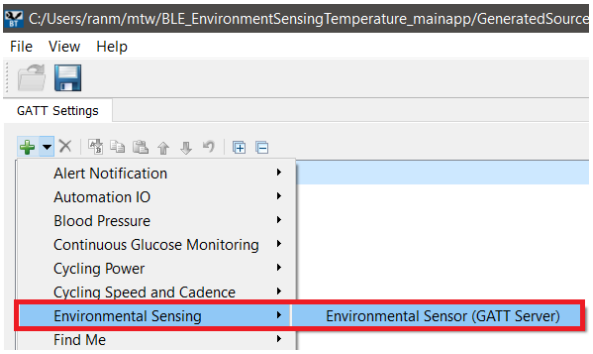
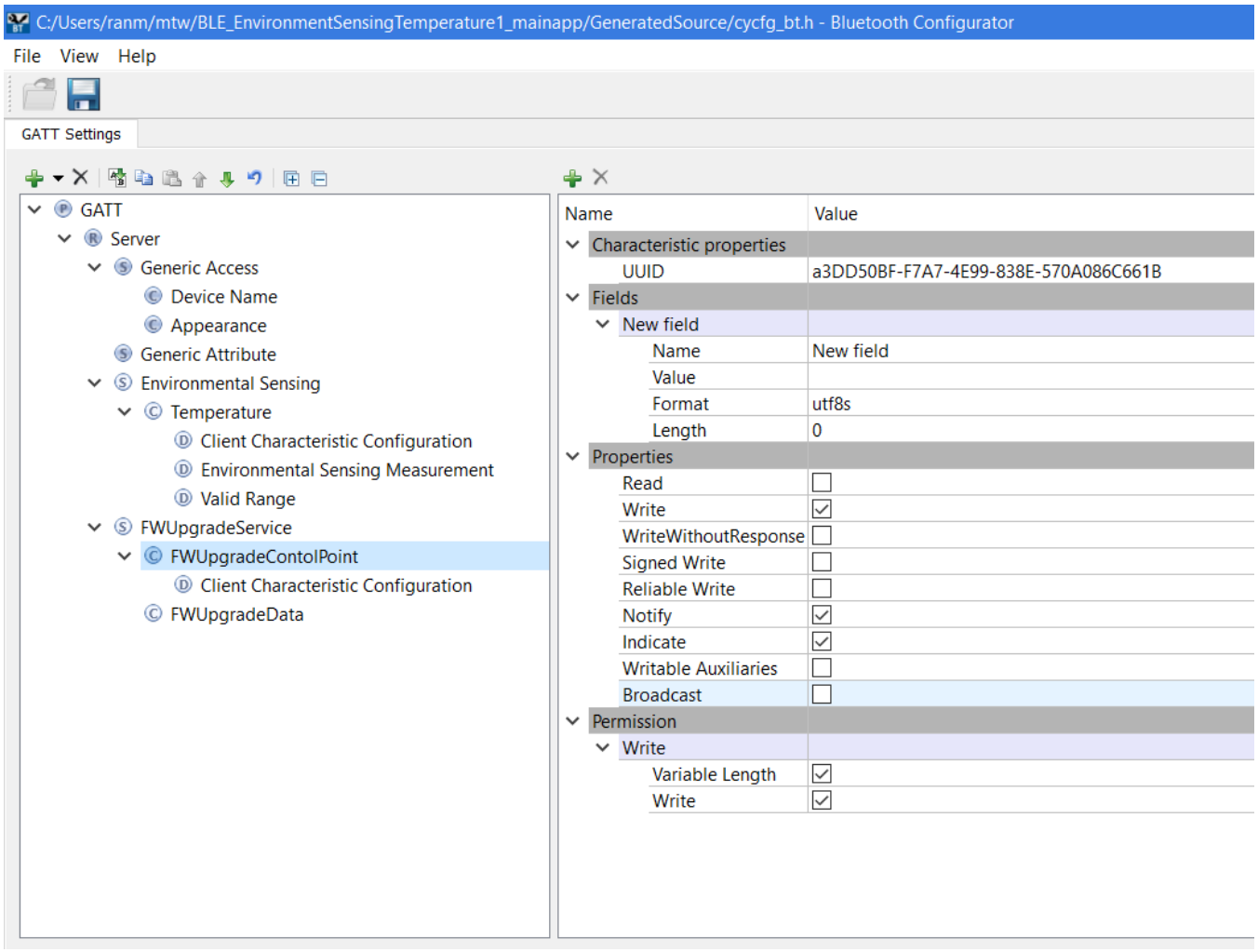


Figure 81. OTA FWUpgradeService Bluetooth Configurator



Related Documents

Application Notes		
AN225684 – Getting Started with CYW208xx		Describes CYW20819 and CYW20820 Bluetooth SoC and how to build your first BLE application using the device in ModusToolbox IDE.
Code Examples		
Visit the Cypress GitHub repository for a comprehensive collection of code examples using ModusToolbox IDE		
Device Documentation		
CYW20719 Device Datasheet	CYW20819 Device Datasheet	CYW20820 Device Datasheet
CYW20819 Feature and Peripheral Guide		This document introduces the CYW20819 ultra-low power, dual-mode BT v5.0 wireless MCU with an Arm® Cortex®- M4 CPU by exploring the CYW20819 device architecture along with the development tools to write applications using ModusToolbox software.
CYW20719 Product Guide		This document introduces the CYW20719 ultra-low power dual-mode BT v5.0 wireless MCU with an Arm® Cortex®-M4 CPU by exploring both CYW20719 device architecture along with the development tools to write applications using WICED Studio™.
CY5677 CySmart BLE 4.2 USB Dongle		CY5677 CySmart BLE 4.2 USB Dongle is a BLE-USB bridge featuring a Bluetooth 4.2-compliant PSoC® 4 BLE, enabling you to test and debug Bluetooth® Low Energy (BLE) systems.
Development Kits		
CYW920819EVB-02	CYW920820EVB-02	CYW920719B2Q40EVB-01
Tool Documentation		
ModusToolbox IDE		The Cypress IDE for IoT designers

Document History

Document Title: CE226300 – BLE Environment Sensing Temperature with CYW20819

Document Number: 002-26300

Revision	ECN	Submission Date	Description of Change
**	6490178	02/20/2019	New code example
*A	6616465	07/08/2019	The code example has been modified to support new platforms such as CYW920820EVB-02 and CYW920719B2Q40EVB-01. Over-the-Air (OTA) Firmware Upgrade section has been newly added to the document.

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

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