Bluetooth® Low Energy
Sensor Tag Hands On

1. Introduction

Thank you for purchasing a Texas Instruments (TI) *Bluetooth®* low energy (BLE) Sensor Tag Development Kit. The purpose of this document is to give an overview of the hardware and software included in the kit and to provide an introduction into BLE.

The information in this guide will get you up and running with the kit. For more detailed information on BLE technology and the TI BLE protocol stack, please consult the Texas Instruments *Bluetooth®* Low Energy Software Developer's Guide.

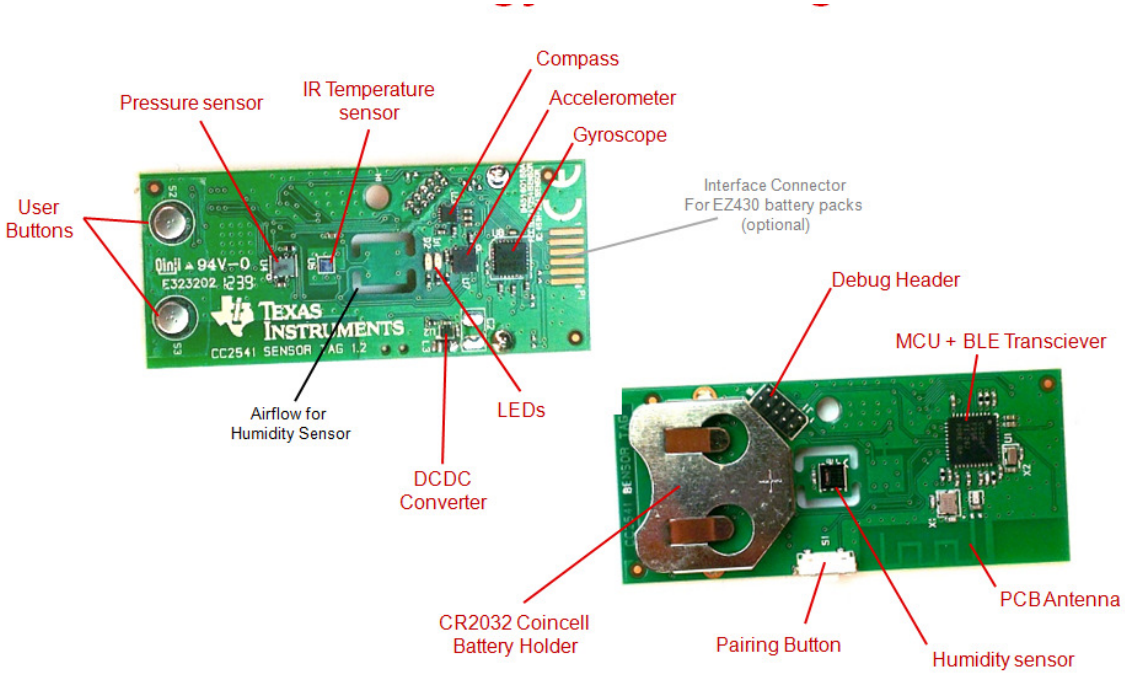
1.1 Kit Contents Overview

The kits contain the following hardware components including cables:

	CC2541 Sensor Tag	CC2540 Dongle	Plastic Case for Sensor Tag
Sensor Tag Kit	•	•	•

The **CC2541 Sensor Tag** is designed to act as a Peripheral Device (BLE Slave). Plastic casing for the sensor tag is also included. The sensor tag operates on a single CR2032 coin cell battery and includes a two-colored LED and the following sensors: temperature, humidity, pressure, accelerometer, gyroscope, and magnetometer.

The sensor tag uses I2C to interface to the different sensors. It is a FCC, IC, and ETSI certified solution. An overview of the sensor tag is shown below:



The **CC2540 USB Dongle** can be used to emulate any type of Bluetooth low energy behavior but is usually used as a Central Device (BLE Master). It connects to a Windows PC's USB Port, and is pre-loaded with the necessary software to receive commands from the PC tool BTool. That is, it acts as a network processor by default.

Caution! The kits include a non-rechargeable lithium battery. Always make sure the battery is removed from the CC2540/41 Sensor tag when it is connected to an external power source (Do not apply voltage > 3.6V). Dispose the battery properly and keep out of the reach of children. If swallowed, contact a physician immediately.

Caution! The kits contain ESD sensitive components. Handle with care to prevent permanent damage.

1.2 System Requirements

To use the TI BLE software, a PC running Microsoft Windows (XP or later) is required, as well as Microsoft .NET Framework 3.5 Service Pack 1 (SP1) or greater.

In order to check whether your system has the appropriate .NET Framework, open up the Windows Control Panel, and select “Add or Remove Programs”. Amongst the list of currently installed programs, you should see “Microsoft .NET Framework 3.5 SP1”, as shown in Figure 1:

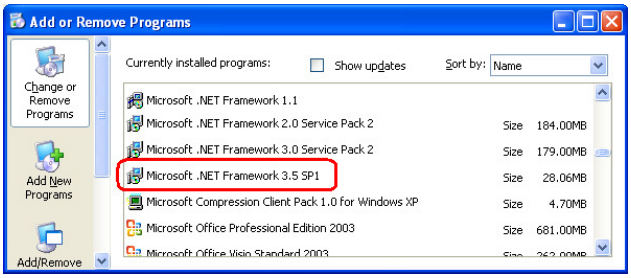


Figure 1 System Requirements, .NET Framework 3.5 SP1

If you do not see it in the list, you can download the framework from Microsoft.

From a hardware standpoint, the Windows PC must contain one free USB port. An additional free USB port is required in order to use the CC Debugger and the USB Dongle simultaneously.

IAR Embedded Workbench for 8051 development environment is required in order to make changes to the sensor tag software. More information on IAR can be found in the Texas Instruments *Bluetooth*® Low Energy Software Developer’s Guide **Error! Reference source not found.**.

2. Getting Started

This section describes how to set up the software and get started with the Development Kit. It is assumed that the Sensor tag comes pre-programmed out of the box. If not, please see Chapter 4 for details on how to program the sensor tag with the latest firmware. In addition, this section assumes that the latest version of the TI BLE software (v1.3.1 as of the release of this document) has been installed. The latest BLE software can be downloaded at www.ti.com/ble-stack.

2.1 Associate Driver with USB Dongle

After the software installation is complete, the USB Dongle driver must be associated with the device in order to use the demo application. To associate the USB Dongle driver, first you must connect the USB Dongle to the PC's USB port, or to a USB hub that connects to the PC.

The first time that the dongle is connected to the PC, a message will most probably pop-up, indicating that Windows does not recognize the device.

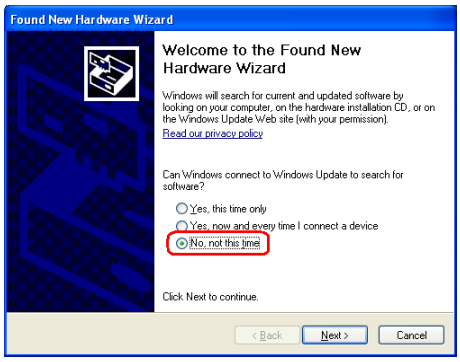


Figure 2 PC, Found New Hardware

When prompted whether to use Windows Update search for software, select "No, not this time" and press the "Next" button. On the next screen, select the option "Install from a list or specific location (Advanced)", and press the "Next" button:

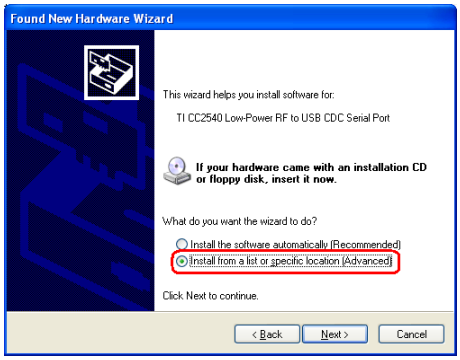


Figure 3 PC, Install Driver

On the next screen, click the checkbox labeled "Include this location in the search:", and click the "Browse" button. Select the following directory (assuming the default installation path was used):

C:\Texas Instruments\BLE-CC254x-1.3.1\Accessories\Drivers

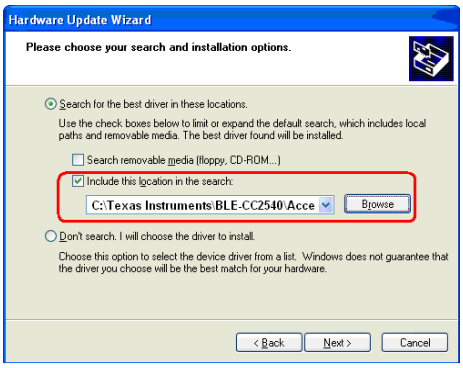


Figure 4 PC, Select Driver

Click the "Next" button. This should install the driver. It will take a few seconds for the file to load. If the installation was successful, you should see the screen to the below. Click the "Finish" button to complete the installation.

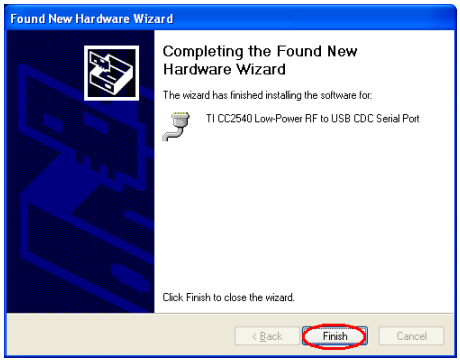


Figure 5 PC, CDC Driver Installation Complete

2.2 Determining the COM Port

Once the driver is installed, you need to determine which COM port Windows has assigned to the USB Dongle. After you have completed the USB Dongle driver association in section 2.1, right-click on the "Computer" icon on your Start and select "Properties", as shown in Figure 5.

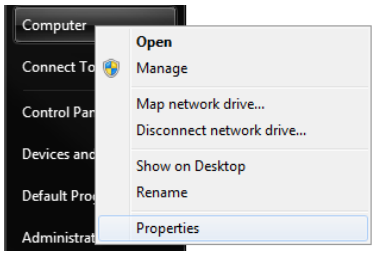


Figure 6 Win7 PC, Finding Computer Properties

The "System Properties" window should open up. Click "Device Manager as shown in Figure 7.

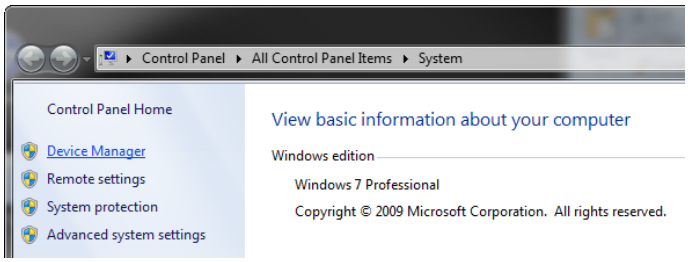


Figure 7 Win7 PC, Finding Device Manager

A list of all hardware devices should appear. Under the section "Ports (COM & LPT)", the device "TI CC2540 Low-Power RF to USB CDC Serial Port" should appear. Next to the name should be the port number (for example, the CC2540USB Dongle uses COM8 in Figure 8).

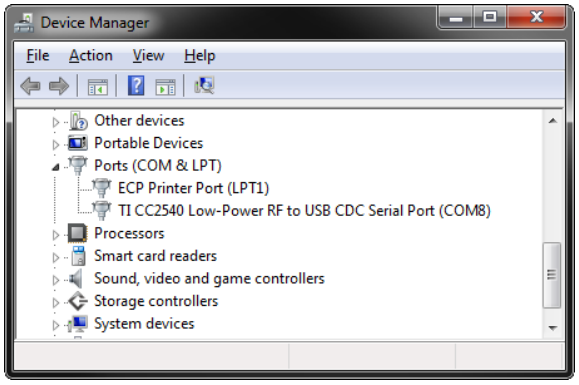


Figure 8 Win7 PC, Connected Ports List

Take note of this port number, as it will be needed in order to use BTool. You may close the device manager at this point.

3. Using BTool

BTool is a PC Application that allows a user to form a connection between two BLE devices. BTool works by communicating with the CC2540 USB Dongle, acting as a network processor, by means of HCI vendor specific commands. The USB Dongle software (when running the HostTestRelease project) and driver create a virtual serial port over the USB interface. BTool, running on the PC, communicates with the USB Dongle through this virtual serial port.

More information on the network processor configuration and the HostTestRelease project can be found in the Texas Instruments *Bluetooth*® Low Energy Software Developer's Guide. More information on the HCI interface, as well as details on the HCI vendor specific commands that are used by the CC2540/41, can be found in the TI BLE Vendor Specific HCI Reference Guide. These documents can be found in the Documents folder of the stack install directory.

For this section, a PC running windows 7 has been used, but the procedures are essentially the same for other windows version, such as XP.

3.1 Starting the Application

To start the application, go into your programs by choosing Start > Programs > Texas Instruments > BLE-CC254x-1.3.1 > BTool. On Start-up you should be able to set the Serial Port Settings. Set the "Port" value to the COM port earlier noted in Section 3.2. For the other settings, use the default values as shown in Figure 9. Press "OK" to connect to the CC2540 USB Dongle.

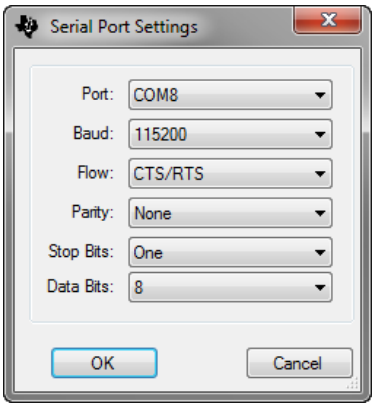


Figure 9 BTool, Serial Port settings

When connected you should see the screen presented in Figure 10. The screen indicates that you now have a serial port connection to the CC2540 USB Dongle. The screen is divided up into a few sections: the left sidebar contains information on the CC2540 USB Dongle status. The left side of the sub-window contains a log of all messages sent from the PC to the CC2540 USB Dongle and received by the PC from the CC2540 USB Dongle. The right side of the sub-window contains a GUI for control of the CC2540 USB Dongle. The bottom pane is the attribute explorer which we will discuss later on.

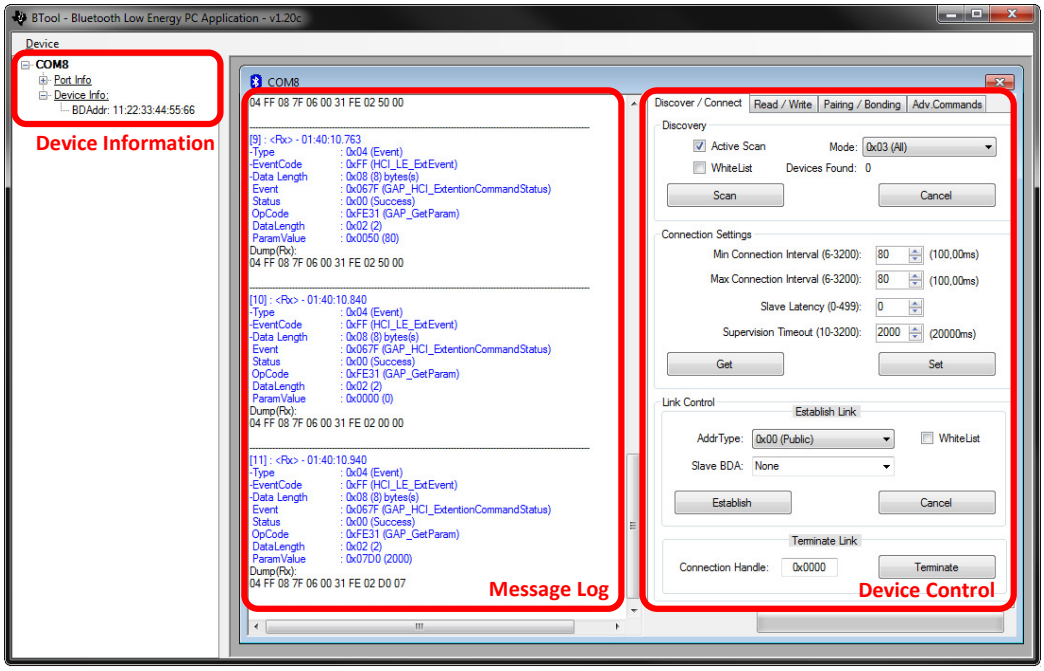


Figure 10 BTool, Overview

3.2 Creating a BLE Connection between USB Dongle and Sensor tag

At this point the USB Dongle (central) is ready to discover other BLE devices that are advertising. The sensor tag should be preloaded with the sensor tag application. The full project and application source code files for the sensor tag are included in the BLE software development kit.

At this time you will want to insert the battery (or remove and re-insert the battery to reset the device) into the sensor tag (peripheral). You should also assemble the plastic and rubber portions of the kit to minimize ESD on the board.

In order to ensure that you are connecting to the correct device, **you need to know your sensor tag's address**. To save time for this tutorial, we have included your address on the bottom of the lid of your development kit. Alternatively, you can refer to section 5.3.2 for instructions to read the sensor tag's primary address.

3.2.1 Making the Sensor tag Discoverable

When the sensor tag powers up, it will not immediately go into a discoverable state. To enable advertising and make the sensor tag discoverable, press the "pairing button" on the side of the sensor tag once. This will turn advertisements on; making the device discoverable for 30 seconds (this value is defined in the *Specification of the Bluetooth System*). After that time, the device will return to standby mode. To make the device discoverable again, simply press the button once again. During discoverable mode, the LED will flash green.

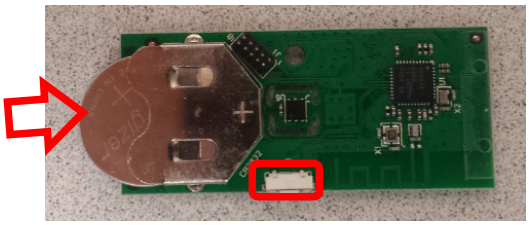


Figure 11 Press Side Button to Turn On Advertisements

3.2.2 Scanning for Devices

In BTool, Press the “Scan” button under the “Discover / Connect” tab, as shown in Figure 12.

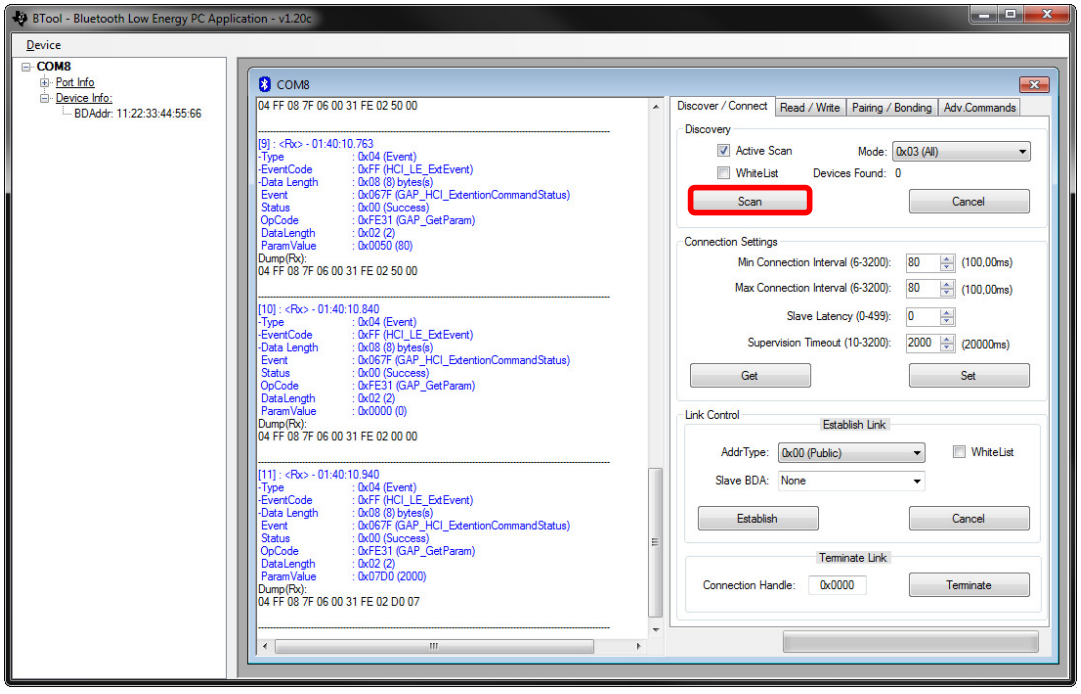


Figure 12 BTool, Scan for Devices

The USB Dongle will begin search for other BLE devices. As devices are found, the log on the left side of the screen will display the devices discovered. After 10 seconds, the device discovery process will complete, and the USB Dongle will stop scanning. A summary of all the scanned devices will be displayed in the log window. In the example in Figure 13, one peripheral device was discovered while scanning. If you do not want to wait through the full 10 seconds of scanning, the “Cancel” button can be pressed alternatively, which will stop the device discovery process. The address of any scanned devices will appear in the “Slave BDA” section of the “Link Control” section in the bottom right corner of the sub-window.

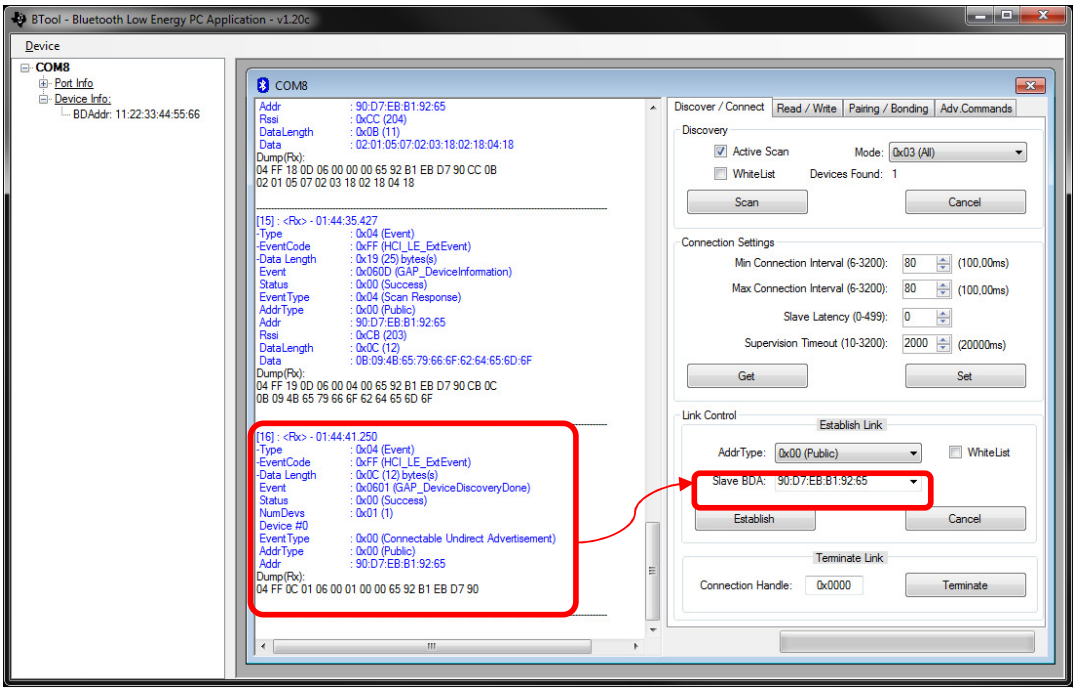


Figure 13 BTool, Slave Address

3.2.3 Selecting Connection Parameters

Before establishing a connection, you can set up the desired connection parameters. The default values of 100ms connection interval, 0 slave latency, and 20s supervision timeout should serve as a good starting point; however for different applications you may want to experiment with these values.

Once the desired values have been set, be sure to click the “Set” button; otherwise the settings will not be saved. Note that the connection parameters must be set before a connection is established; changing the values and clicking the “Set” button while a connection is active will not change the settings of an active connection. The connection must be terminated and re-established to use the new parameters. (The *Bluetooth* specification does support connection parameter updates while a connection is active; however this must be done using either an L2CAP connection parameter update request, or using a direct HCI command. More information can be found in the *Specification of the Bluetooth System*)

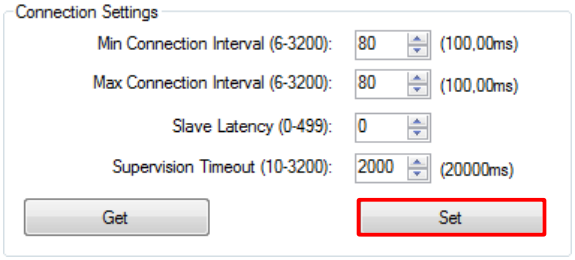


Figure 14 BTool, Connection Settings

3.2.4 Establishing a Connection

To establish a connection with the sensor tag, select the address of the device to connect with and click the “Establish” button as shown in Figure 15.

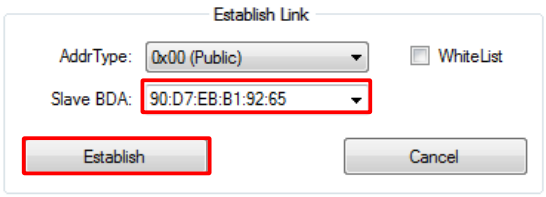


Figure 15 BTool, Establish Connection

If the sensor tag is still in discoverable mode, a connection should be established (if more than 30 seconds have passed since the device was previously made discoverable, press the right button on the sensor tag once again). Once a connection is established, the message window will return a "GAP_EstablishLink" event message with a "Status" value of "0x00 (Success)" as shown in Figure 16.

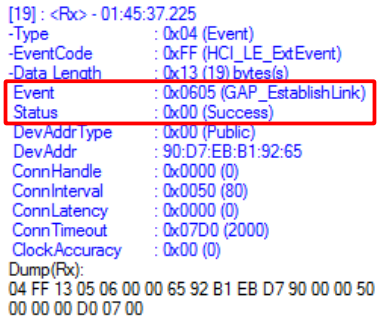


Figure 16 BTool Log, Link Established

In BTool, you can see your connected peripheral device in the Device Information field, as shown in Figure 17.

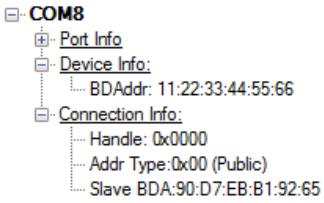


Figure 17 BTool, Device Information

3.3 Using the Sensor Tag's GATT Profiles

We will now begin investigating the sensor tag's GATT profiles. Besides the standard GAP, GATT, and device information services, the sensor tag contains the following GATT services: temperature, accelerometer, humidity, magnetometer, barometer, gyroscope, simple keys, and test. You will find the sensor tag complete attribute below and it can be used as a reference. Services are shown in yellow, characteristics are shown in blue, and characteristic values / descriptors are shown in grey.

Services are constructed of characteristics, each of which have, at minimum, a declaration and a value, and may have a client configuration and/or a user description. The actual payload data is stored with the characteristic values. All application data that is being sent or received in Bluetooth low energy must be contained within characteristic values. This section details a step-by-step process that demonstrates several processes for reading, writing, discovering, and notifying GATT characteristic values using BTool.

In a *Bluetooth* low energy system, upon connection, the Central Device (GATT Client) performs a service discovery on the Peripheral device (GATT server) to build up an attribute table. This attribute table will provide handles (internal addresses of the characteristics) which can be used by the Client to access the data located in the Server. The service discovery is typically an automated process that can be started with a single command. In BTool however, the automated service discovery is not implemented (although it's still possible to perform it manually). To simplify the evaluation of the sensor tag, the attribute table will be known and is shown below so it is possible to use handles directly to read out data.

Sensor Tag Application: Complete Attribute Table

TI Base UUID: F000XXXX-0451-4000-B000-000000000000. 128-but UUIDs are typed 'bold'

handle (hex)	handle (dec)	Type (hex)	Type (#DEFINE)	Hex / Text Value (default)	GATT Server Permissions	Notes
0x1	1	0x2800	GATT_PRIMARY_SERVICE_UUID	0x1800 (GAP_SERVICE_UUID)	GATT_PERMIT_READ	Start of GAP Service (Mandatory)
0x2	2	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 03 00 (handle: 0x0003) 00 2A (UUID: 0x2A00)	GATT_PERMIT_READ	Device Name characteristic declaration
0x3	3	0x2A00	GAP_DEVICE_NAME_UUID	"Sensor Tag"	GATT_PERMIT_READ	Device Name characteristic value
0x4	4	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 05 00 (handle: 0x0005) 01 2A (UUID: 0x2A01)	GATT_PERMIT_READ	Appearance characteristic declaration
0x5	5	0x2A01	GAP_APPEARANCE_UUID	0x0000	GATT_PERMIT_READ	Appearance characteristic value
0x6	6	0x2803	GATT_CHARACTER_UUID	0A (properties: read/w rite) 07 00 (handle: 0x0007) 02 2A (UUID: 0x2A02)	GATT_PERMIT_READ	Peripheral Privacy Flag characteristic declaration
0x7	7	0x2A02	GAP_PERI_PRIVACY_FLAG_UUID	0x00 (GAP_PRIVACY_DISABLED)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Peripheral Privacy Flag characteristic value
0x8	8	0x2803	GATT_CHARACTER_UUID	0A (properties: read/w rite) 09 00 (handle: 0x0009) 03 2A (UUID: 0x2A03)	GATT_PERMIT_READ	Reconnection address characteristic declaration
0x9	9	0x2A03	GAP_RECONNECT_ADDR_UUID	00:00:00:00:00:00	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Reconnection address characteristic value
0xA	10	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 0B 00 (handle: 0x000B) 04 2A (UUID: 0x2A04)	GATT_PERMIT_READ	Peripheral Preferred Connection Parameters characteristic declaration
0xB	11	0x2A04	GAP_PERI_CONN_PARAM_UUID	50 00 (100ms preferred min connection interval) A0 00 (200ms preferred max connection interval) 00 00 (0 preferred slave latency) E8 03 (10000ms preferred supervision timeout)	GATT_PERMIT_READ	Peripheral Preferred Connection Parameters characteristic declaration
0xC	12	0x2800	GATT_PRIMARY_SERVICE_UUID	0x1801 (GATT_SERVICE_UUID)	GATT_PERMIT_READ	Start of GATT Service (mandatory)
0xD	13	0x2803	GATT_CHARACTER_UUID	20 (properties: indicate only) 0E 00 (handle: 0x000E) 05 2A (UUID: 0x2A05)	GATT_PERMIT_READ	Service Changed characteristic declaration
0xE	14	0x2A05	GATT_SERVICE_CHANGED_UUID	(null value)	(none)	Service Changed characteristic value
0xF	15	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01:00" to enable notifications, "00:00" to disable
0x10	16	0x2800	GATT_PRIMARY_SERVICE_UUID	0x180A (DEVINFO_SERV_UUID)	GATT_PERMIT_READ	Start of Device Information Service
0x11	17	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 11 00 (handle 0x0011) 23 2A (UUID 0x2A23)	GATT_PERMIT_READ	System ID characteristic declaration
0x12	18	0x2A23	DEVINFO_SYSTEM_ID_UUID	xx xx xx 00 00 xx xx xx (xx's are IEEE address)	GATT_PERMIT_READ	System ID
0x13	19	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 13 00 (handle 0x0013) 24 2A (UUID 0x2A24)	GATT_PERMIT_READ	Model Number String characteristic declaration
0x14	20	0x2A24	DEVINFO_MODEL_NUMBER_UUID	"Model Number"	GATT_PERMIT_READ	Model Number String
0x15	21	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 15 00 (handle 0x0015) 25 2A (UUID 0x2A25)	GATT_PERMIT_READ	Serial Number String characteristic declaration
0x16	22	0x2A25	DEVINFO_SERIAL_NUMBER_UUID	"Serial Number"	GATT_PERMIT_READ	Serial Number String
0x17	23	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 17 00 (handle 0x0017) 26 2A (UUID 0x2A26)	GATT_PERMIT_READ	Firmw are Revision String characteristic declaration
0x18	24	0x2A26	DEVINFO_FIRMWARE_REV_UUID	"Firmw are Revision"	GATT_PERMIT_READ	Firmw are Revision String
0x19	25	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 19 00 (handle 0x0019) 27 2A (UUID 0x2A27)	GATT_PERMIT_READ	Hardw are Revision String characteristic declaration
0x1A	26	0x2A27	DEVINFO_HARDWARE_REV_UUID	"Hardw are Revision"	GATT_PERMIT_READ	Hardw are Revision String
0x1B	27	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 1B 00 (handle 0x001B) 28 2A (UUID 0x2A28)	GATT_PERMIT_READ	Softw are Revision String characteristic declaration
0x1C	28	0x2A28	DEVINFO_SOFTWARE_REV_UUID	"Softw are Revision"	GATT_PERMIT_READ	Softw are Revision String
0x1D	29	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 1D 00 (handle 0x001D) 29 2A (UUID 0x2A29)	GATT_PERMIT_READ	Manufacturer Name String characteristic declaration
0x1E	30	0x2A29	DEVINFO_MANUFACTURER_NAME_UUID	"Manufacturer Name"	GATT_PERMIT_READ	Manufacturer Name String
0x1F	31	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 1F 00 (handle 0x001F) 2A 2A (UUID 0x2A2A)	GATT_PERMIT_READ	IEEE 11073-20601 Regulatory Certification Data List characteristic declaration
0x20	32	0x2A2A	DEVINFO_11073_CERT_DATA_UUID	FE 00 65 78 70 65 72 69 6D 65 6E 74 61 6C	GATT_PERMIT_READ	IEEE 11073-20601 Regulatory Certification Data List
0x21	33	0x2803	GATT_CHARACTER_UUID	02 (read permissions) 22 00 (handle 0x0022) 50 2A (UUID 0x2A50)	GATT_PERMIT_READ	PnP ID characteristic declaration
0x22	34	0x2A2A	PNPID_DATA_UUID	FE 00 65 78 70 65 72 69 6D 65 6E 74 61 6C	GATT_PERMIT_READ	PnP ID

0x23	35	0x2800	GATT_PRIMARY_SERVICE_UUID	**0xAA00** (IRTEMPERATURE_SERV_UUID)	GATT_PERMIT_READ	Start of Sensor Profile Temperature Service
0x24	36	0x2803	GATT_CHARACTER_UUID	12 (properties: read/notify) 25 00 (handle: 0x0025) 01 AA (UUID: **0xAA01**)	GATT_PERMIT_READ	
0x25	37	**0xAA01**	IRTEMPERATURE_DATA_UUID	00:00:00:00 (4 bytes)	GATT_PERMIT_READ	ObjectLSB:ObjectMSB:AmbientLSB:AmbientMSB
0x26	38	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01:00" to enable notifications, "00:00" to disable
0x27	39	0x2901	GATT_CHAR_USER_DESC_UUID	"IR Temp. Data" (14 bytes)	GATT_PERMIT_READ	
0x28	40	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 29 00 (handle: 0x0029) 02AA (UUID: **0xAA02**)	GATT_PERMIT_READ	
0x29	41	**0xAA02**	IRTEMPERATURE_CONF_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01" to start Sensor and Measurements, "00" to put to sleep
0x2A	42	0x2901	GATT_CHAR_USER_DESC_UUID	"IR Temp. Conf." (15 bytes)	GATT_PERMIT_READ	
0x2B	43	0x2800	GATT_PRIMARY_SERVICE_UUID	**0xAA10** (ACCELEROMETER_SERV_UUID)	GATT_PERMIT_READ	Start of Sensor Profile Accelerometer Service
0x2C	44	0x2803	GATT_CHARACTER_UUID	12 (properties: read/notify) 2D 00 (handle: 0x002D) 11 AA (UUID: **0xAA11**)	GATT_PERMIT_READ	
0x2D	45	**0xAA11**	ACCELEROMETER_DATA_UUID	00:00:00 (3 bytes)	GATT_PERMIT_READ	X : Y : Z Coordinates
0x2E	46	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01:00" to enable notifications, "00:00" to disable
0x2F	47	0x2901	GATT_CHAR_USER_DESC_UUID	"Accel. Data" (14 bytes)	GATT_PERMIT_READ	
0x30	48	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 31 00 (handle: 0x0031) 12 AA (UUID: **0xAA12**)	GATT_PERMIT_READ	
0x31	49	**0xAA12**	ACCELEROMETER_CONF_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01" to start Sensor and Measurements, "00" to put to sleep
0x32	50	0x2901	GATT_CHAR_USER_DESC_UUID	"Accel. Conf." (15 bytes)	GATT_PERMIT_READ	
0x33	51	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 34 00 (handle: 0x0034) 13 AA (UUID: **0xAA13**)	GATT_PERMIT_READ	
0x34	52	**0xAA13**	ACCELEROMETER_PERI_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Period = [Input*10] ms, default 1000 ms, lower limit 100 ms
0x35	53	0x2901	GATT_CHAR_USER_DESC_UUID	"Acc. Period" (12 bytes)	GATT_PERMIT_READ	
0x36	54	0x2800	GATT_PRIMARY_SERVICE_UUID	**0xAA20** (HUMIDITY_SERV_UUID)	GATT_PERMIT_READ	Start of Sensor Profile Humidity Service
0x37	55	0x2803	GATT_CHARACTER_UUID	12 (properties: read/notify) 38 00 (handle: 0x0038) 21 AA (UUID: **0xAA21**)	GATT_PERMIT_READ	
0x38	56	**0xAA21**	HUMIDITY_DATA_UUID	00:00:00:00 (4 bytes)	GATT_PERMIT_READ	TempLSB:TempMSB:HumidityLSB:HumidityMSB
0x39	57	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01:00" to enable notifications
0x3A	58	0x2901	GATT_CHAR_USER_DESC_UUID	"Humid. Data" (14 bytes)	GATT_PERMIT_READ	
0x3B	59	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 3C 00 (handle: 0x003C) 22 AA (UUID: **0xAA22**)	GATT_PERMIT_READ	
0x3C	60	**0xAA22**	HUMIDITY_CONF_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01" to start Sensor and Measurements, "00" to put to sleep
0x3D	61	0x2901	GATT_CHAR_USER_DESC_UUID	"Humid. Conf." (15 bytes)	GATT_PERMIT_READ	
0x3E	62	0x2800	GATT_PRIMARY_SERVICE_UUID	**0xAA30** (MAGNETOMETER_SERV_UUID)	GATT_PERMIT_READ	Start of Sensor Profile Magnetometer Service
0x3F	63	0x2803	GATT_CHARACTER_UUID	12 (properties: read/notify) 40 00 (handle: 0x0040) 31 AA (UUID: **0xAA31**)	GATT_PERMIT_READ	
0x40	64	**0xAA31**	MAGNETOMETER_DATA_UUID	00:00:00:00:00:00 (6 bytes)	GATT_PERMIT_READ	XLSB:XMSB:YLSB:YMSB: ZLSB:ZMSB Coordinates
0x41	65	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01:00" to enable notifications, "00:00" to disable
0x42	66	0x2901	GATT_CHAR_USER_DESC_UUID	"Mag. Data" (10 bytes)	GATT_PERMIT_READ	
0x43	67	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 44 00 (handle: 0x0044) 32 AA (UUID: 0xAA32)	GATT_PERMIT_READ	
0x44	68	**0xAA32**	MAGNETOMETER_CONF_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01" to start Sensor and Measurements, "00" to put to sleep
0x45	69	0x2901	GATT_CHAR_USER_DESC_UUID	"Mag. Conf." (11 bytes)	GATT_PERMIT_READ	
0x46	70	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 47 00 (handle: 0x0047) 33 AA (UUID: **0xAA33**)	GATT_PERMIT_READ	
0x47	71	**0xAA33**	MAGNETOMETER_PERI_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Period = [Input*10]ms, default 2000ms, lower limit 100 ms
0x48	72	0x2901	GATT_CHAR_USER_DESC_UUID	"Mag. Period" (12 bytes)	GATT_PERMIT_READ	

0x49	73	0x2800	GATT_PRIMARY_SERVICE_UUID	**0xAA40** (BAROMETER_SERV_UUID)	GATT_PERMIT_READ	Start of Sensor Profile Barometer Service
0x4A	74	0x2803	GATT_CHARACTER_UUID	12 (properties: read/notify) 4B 00 (handle: 0x004B) 41 AA (UUID: **0xAA41**)	GATT_PERMIT_READ	
0x4B	75	**0xAA41**	BAROMETER_DATA_UUID	00:00:00:00 (4 bytes)	GATT_PERMIT_READ	TempLSB:TempMSB:PressureLSB:PressureMSB
0x4C	76	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	
0x4D	77	0x2901	GATT_CHAR_USER_DESC_UUID	"Barometer Data" (15 bytes)	GATT_PERMIT_READ	
0x4E	78	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 53 00 (handle: 0x0053) 42 AA (UUID: **0xAA42**)	GATT_PERMIT_READ	
0x4F	79	**0xAA42**	BAROMETER_CONF_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write "01" to start Sensor and Measurements, "00" to put to sleep, "02" to read calibration values from sensor
0x50	80	0x2901	GATT_CHAR_USER_DESC_UUID	"Barometer Conf." (16 bytes)	GATT_PERMIT_READ	
0x51	81	0x2803	GATT_CHARACTER_UUID	02 (properties: read only) 4F 00 (handle: 0x004F) 43 AA (UUID: **0xAA43**)	GATT_PERMIT_READ	
0x52	82	**0xAA43**	BAROMETER_CALI_UUID	00:00:...:00:00 (16 bytes)	GATT_PERMIT_READ	When write 02 to Barometer Conf. has been issued, the calibration values is found here
0x53	83	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	
0x54	84	0x2901	GATT_CHAR_USER_DESC_UUID	"Barometer Cali." (16 bytes)	GATT_PERMIT_READ	
0x55	85	0x2800	GATT_PRIMARY_SERVICE_UUID	**0xAA50** (GYROSCOPE_SERV_UUID)	GATT_PERMIT_READ	Start of Sensor Profile Gyroscope Service
0x56	86	0x2803	GATT_CHARACTER_UUID	12 (properties: read/notify) 57 00 (handle: 0x0057) 51 AA (UUID: **0xAA51**)	GATT_PERMIT_READ	
0x57	87	**0xAA51**	GYROSCOPE_DATA_UUID	00:00:00:00:00:00 (6 bytes)	GATT_PERMIT_READ	XLSB:XMSB:YLSB:YMSB: ZLSB:ZMSB
0x58	88	0x2902	GATT_CLIENT_CHAR_CFG_UUID	00:00 (2 bytes)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	
0x59	89	0x2901	GATT_CHAR_USER_DESC_UUID	"Gyro. Data" (11 bytes)	GATT_PERMIT_READ	
0x5A	90	0x2803	GATT_CHARACTER_UUID	0A (properties: read/write) 5B 00 (handle: 0x005B) 52 AA (UUID: **0xAA52**)	GATT_PERMIT_READ	
0x5B	91	**0xAA52**	GYROSCOPE_CONF_UUID	1 (1 byte)	GATT_PERMIT_READ | GATT_PERMIT_WRITE	Write 0 to turn off gyroscope, 1 to enable X axis only, 2 to enable Y axis only, 3 = X and Y, 4 = Z only, 5 = X and Z, 6 = Y and Z, 7 = X, Y and Z
0x5C	92	0x2901	GATT_CHAR_USER_DESC_UUID	"Gyro. Conf." (13 bytes)	GATT_PERMIT_READ	
0x5D	93	0x2800	GATT_SERVICE_UUID	0xFFE0 (SK_KEYPRESSED_UUID)	GATT_PERMIT_READ	Start of Simple Keys Service
0x5E	94	0x2803	GATT_CHARACTER_UUID	10 (notify permission) 34 00 (handle 0x0034) E1 FF (UUID 0xFFE1)	GATT_PERMIT_READ	Keys state characteristic declaration
0x5F	95	0xFFE1	SK_KEYPRESSED_UUID	0	(none)	Keys state characteristic value (bit mask of left / right key presses). Side key as bit 2 in test mode only.
0x60	96	0x2902	GATT_CLIENT_CHAR_CFG_UUID	0x0000	GATT_PERMIT_READ | GATT_PERMIT_WRITE	
0x61	97	0x2901	GATT_CHAR_USER_DESC_UUID	"Key Press State"	GATT_PERMIT_READ	Keys state characteristic user description
0x62	98	0x2800	GATT_SERVICE_UUID	**0xAA60** (TEST_SERVICE_UUID)	GATT_PERMIT_READ	Start of TestService
0x63	99	0x2803	GATT_CHARACTER_UUID	02 (read permission) 64 00 (handle 0x0064) 61 AA (UUID: **0xAA61**)	GATT_PERMIT_READ	Test Data characteristic declaration
0x64	100	**0xAA61**	TEST_DATA_UUID	1 byte	GATT_PERMIT_READ	Test Data: 1 bit set of each test passed
0x65	101	0x2901	GATT_CHAR_USER_DESC_UUID	"Test Data" (10 bytes)	GATT_PERMIT_READ	
0x66	102	0x2803	GATT_CHARACTER_UUID	0A (read/write permission) 68 00 (handle 0x0068) 62 AA (UUID: **0xAA62**)	GATT_PERMIT_READ	Test Config characteristic declaration
0x67	103	**0xAA62**	TEST_CONFIG_UUID	1 byte	GATT_PERMIT_READ	Test Config: bit 7 - enable test mode, bit 1 - set LED2, bit 0 - set LED 1
0x68	104	0x2901	GATT_CHAR_USER_DESC_UUID	"Test Config" (12 bytes)	GATT_PERMIT_READ	

3.3.1 Reading a Characteristic Value by UUID

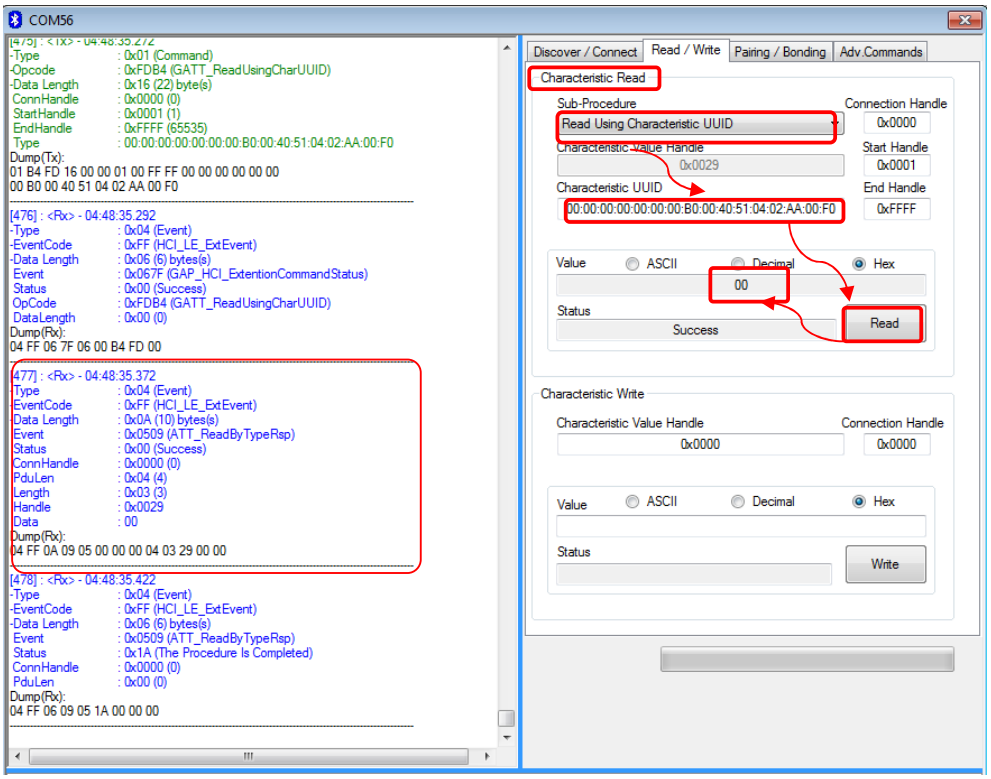
A characteristic value is essentially where the data payload is stored, which could be, for example, temperature data or battery level. It is the stored data in a server that a client wants to access. A characteristic is a discrete value that has, at minimum, the following three properties associated with it:

1. A handle (address)
2. A type (UUID)
3. A set of permissions

Let's consider the IR Temperature service: handles 0x23 to 0x2A as seen above. This service has two characteristics: IR temperature data and IR temperature config. We must first enable the IR sensor by writing to the IR temperature config characteristic. We can then read the temperature by reading from the IR temperature data characteristic. First, let's read the IR temperature config characteristic to ensure that isn't already enabled (it won't be). The simplest way to read its value is to use the "Read Characteristic by UUID" sub-procedure. To do this, you will first need to click the "Read / Write" tab in BTool. Select the option "Read Using Characteristic UUID" under the "Sub-Procedure" option in the "Characteristic Read" section at the top of the screen. Enter the UUID we are looking for. The UUID from the table above is 0xAA02. However, this is a 128-bit UUID so we must add the TI Base UUID. The effective UUID we are looking for is F000**AA02**-0451-4000-B000-000000000000. Also, we must enter this LSB to MSB in BTool with each byte separated by a colon. So enter 00:00:00:00:00:00:00:00:B0:00:40:51:04:**02:AA**:00:F0 in the "Characteristic UUID" box, and click the "Read" button as shown below.

An attribute protocol *Read by Type Request* packet gets sent over the air from the central device to the peripheral device, and an attribute protocol *Read by Type Response* packet gets sent back from the peripheral device to the central device. The value "00" is displayed in the "Value" box, and "Success" is displayed in the "Status" box. The "00" indicates that the temperature sensor is not enabled. In addition, the message window will display information on the *Read by Type Response* packet that was received by the central device. The message includes not only the characteristic's data value, but also the handle of the characteristic value (0x0029 in this case).

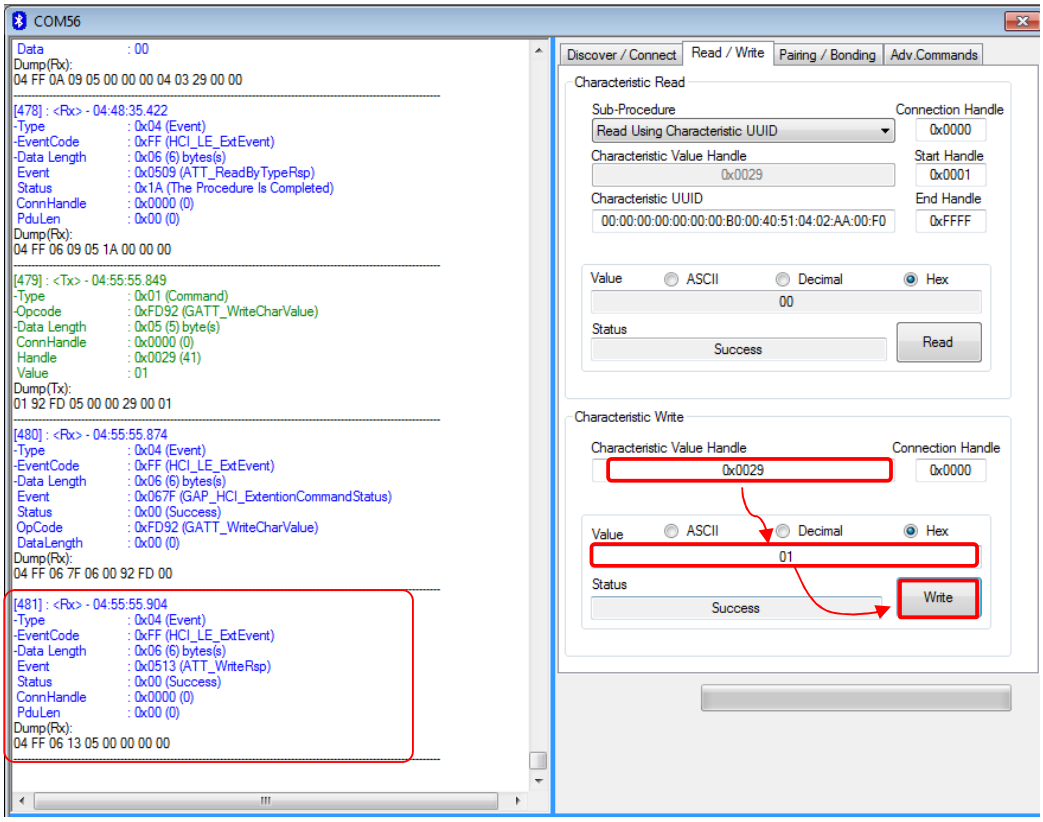
****Note that, as you read attributes from the peripheral, the attribute table in the bottom pane begins to fill up. You can actually fill this entire table up initially by choosing ATT_FindInfoReq in the Adv. Commands tab. You can then read and write to many characteristics by clicking on their respective column in the table. However, it is recommended to go through these manual steps first to gain understanding.



3.3.2 Writing a Characteristic Value

In the previous section, the handle of the config characteristic in the IR Temperature service was found to be 0x0029. Knowing this, and based on the fact that the characteristic has both read and write permissions, it is possible for us to write a new value. Enter “0x0029” into the “Characteristic Value Handle” box in the “Characteristic Write” section, and enter “01” in the “Value” section (the format can be set to either “Decimal” or “Hex”) to enable the temperature sensor. Click the “Write” button as shown below.

An attribute protocol *Write Request* packet gets sent over the air from the central device to the peripheral device, and an attribute protocol *Write Response* packet gets sent back from the peripheral device to the central device. The status box will display “Success”, indicating that the write was successful.

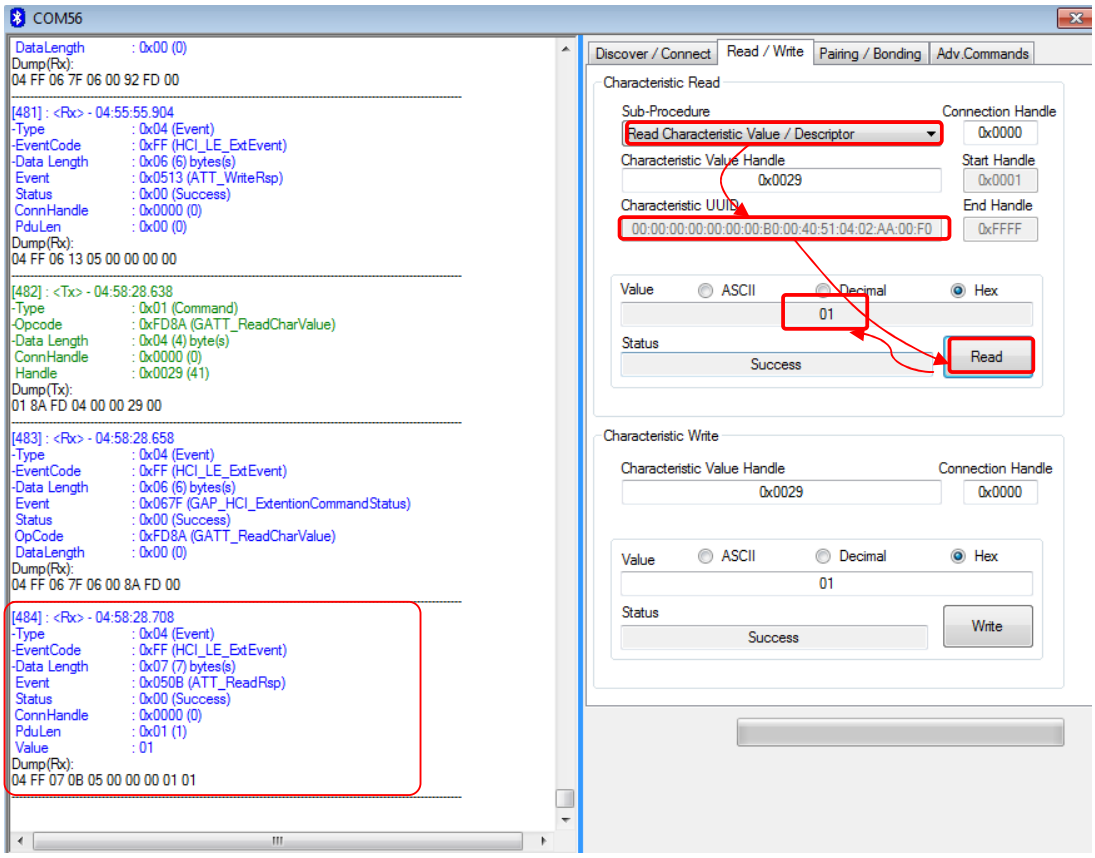


Write a Characteristic Value

3.3.3 Reading a Characteristic Value by Handle

This time, instead of reading the value by its UUID, the value will be read by its handle. Select the option “Read Characteristic Value / Descriptor” under the “Sub-Procedure” option in the “Characteristic Read” section. Enter “0x0029” in the “Characteristic Value Handle” box, and click the “Read” button as shown below.

An attribute protocol *Read Request* packet gets sent over the air from the central device to the peripheral device, and an attribute protocol *Read Response* packet gets sent back from the peripheral device to the central device. The new value is displayed in the “Value” box, and “Success” is displayed in the “Status” box. This value should match the value that was written in the previous step.



Read a Characteristic Value by Handle

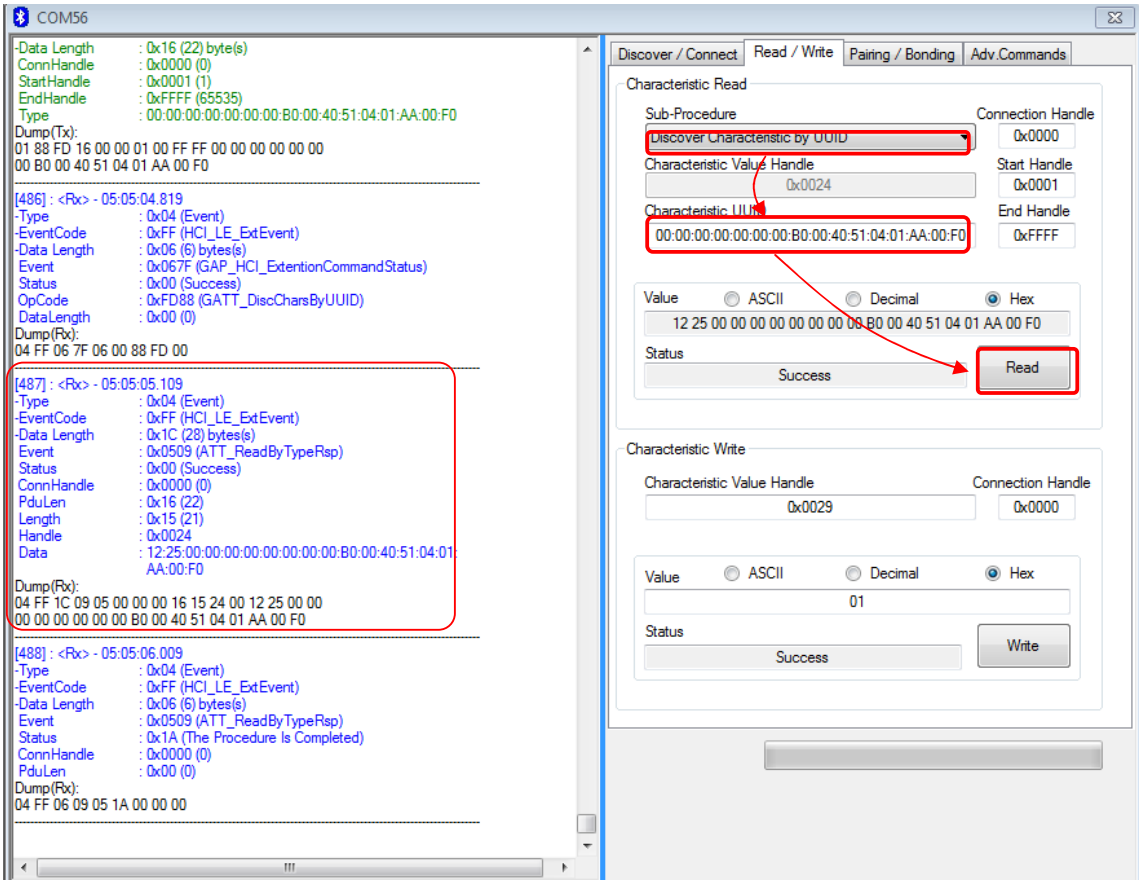
3.3.4 Discovering a Characteristic by UUID

Consider a situation in which you want to know if the GATT server has a characteristic with a given UUID and, if so, where it is located in the attribute table. In this case, we need to discover a characteristic by its UUID. By doing this, we will not only get the handle of the UUID, but we will also get the properties of the characteristic. The UUID of the IR temperature data characteristic is, not including the TI 128-bit mask, 0xAA01. Select the option "Discover Characteristic by UUID" under the "Sub-Procedure" option in the "Characteristic Read" section at the top of the screen. Adding the mask and reversing the bytes, enter 00:00:00:00:00:00:00:B0:00:40:51:04:**01:AA**:00:F0 in the "Characteristic UUID" box, and click the "Read" button as shown below.

A series of attribute protocol *Read by Type Request* packets get sent over the air from the central device to the peripheral device, and for each request an attribute protocol *Read by Type Response* packet gets sent back from the peripheral device to the central device. Essentially, the central device is reading every attribute on the peripheral device with a UUID of 0x2803 (this is the UUID for a characteristic declaration as defined in *Specification of the Bluetooth System*), and checking the "Characteristic Value UUID" portion of each declaration to see if it matches the UUID we are looking for. The procedure is complete once every characteristic declaration has been read.

The procedure will find one instance of the characteristic with type 0xFFF2, and display "12 25 00 00 00 00 00 00 00 B0 00 40 51 04 01 AA 00 F0" (the value of the declaration) in the "Value" box, with "Success" displayed in the "Status" box. As per the *Bluetooth* specification, the first byte "12" tells us that the properties of the characteristic are read and notify. The second and third bytes "25 00" tell us that the handle of the characteristic value is 0x0025. The remaining bytes tell the UUID of the characteristic.

Now that the temperature sensor is enabled, using the steps above, read the value of IR temperature data characteristic (handle 0x0025). This data would need to be converted to a human-readable temperature value. Regardless, you should be able to increase / decrease the least significant bytes by cooling / warming the sensor.



Discover a Characteristic by UUID

3.3.5 Enabling Notifications

In BLE, it is possible for a GATT server device to "push" characteristic value data out to a client device, without being prompted with a read request. This process is called a "characteristic value notification". Notifications are useful in that they allow a device in a BLE connection to send out as much or as little data as required at any point in time. In addition, since no request from the client is required, the overhead is reduced and the data is transmitted more efficiently. The Sensor Tag software contains an example in which notifications can be demonstrated.

KeyPressed (handle 0x005E), the only characteristic in the SimpleKeyService, has notify-only properties. When any of the keys are pressed / released, the KeyPressed characteristic value gets set by the application and the SimpleKeys profile will check to see if notifications are enabled. If they are enabled, the profile will send a notification of the value to the client device.

Before notifications can be enabled, the handle of the fourth characteristic must be found. This can be done by using the "Discover Characteristic by UUID" process (see section 3.3.4), with the UUID value set to **E1:FF**. Note that this is only a 16-bit UUID. This will be changed to a 128-bit UUID in a future release as this is the preferred method to avoid UUID overlaps. The procedure will find one instance of the characteristic with type 0xFFE1, and display "10 5F 00 E1 FF" (the value of the declaration) in the "Value" box, with "Success" displayed in the "Status" box. As per the *Bluetooth* specification, the first byte "10" tells us that the properties of the characteristic are notify-only. The second and third bytes "5F 00" tell us that the handle of the characteristic value is 0x005F. The fourth and fifth bytes tell the UUID of the characteristic, 0xFFE1.

In order to enable notifications, the client device must write a value of 0x0001 to the client characteristic configuration descriptor for the particular characteristic. The handle for the client characteristic configuration descriptor immediately follows the characteristic value's handle. Therefore, a value of 0x0001 must be written to handle 0x0060. Enter "0x0060" into the "Characteristic Value Handle" box in the "Characteristic Write" section, and enter "01:00" in the "Value" section (note that the LSB is entered

first, and the MSB is entered last and all client characteristic configurations are two-byte values). Click the "Write Value" button. The status box will display "Success", indicating that the write was successful.

Every time a key is pressed or released, an attribute protocol *Handle Value Notification* packet gets sent from the peripheral device to the central device. With each notification, the value of the characteristic at handle is displayed in the log window.

Note that it is also possible to send indications in the same manner if the characteristic has indication property. In this case, the only difference is that the GATT client must respond to the indication to verify that it received the data.

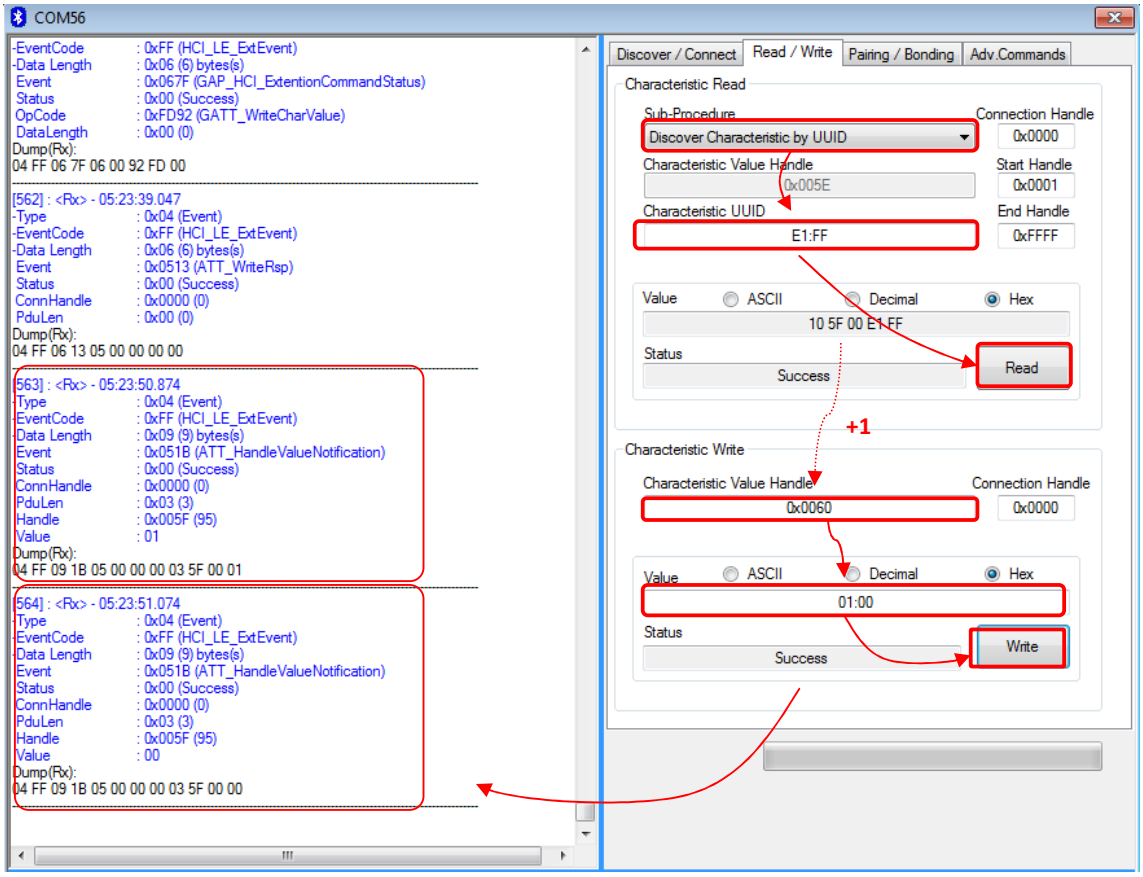


Figure 18 BTool, Enable Notifications

It is important to note that the simple GATT profile included with the BLE development kit does not conform to any standard profile specification available from the *Bluetooth* SIG. The profile, including the GATT characteristic definition, the UUID values, and the functional behavior, was developed by Texas Instruments for use with the CC2540DK or CC2542EMK development kit, and is intended as a demonstration of the capabilities of the *Bluetooth* low energy protocol.

3.4 Using BLE Security

BTool also includes the ability to make use of security features in BLE, including encryption, authentication, and bonding.

3.4.1 Encrypting the Connection

To encrypt the link, the pairing process must be initiated. Click on the "Pairing / Bonding" tab in BTool. In the "Initiate Pairing" section at the top of the screen, check the boxes labeled "Bonding Enabled" and "Authentication (MITM) Enabled", and click the button "Send Pairing Request", as shown in Figure 19. This will send the request to the peripheral device.

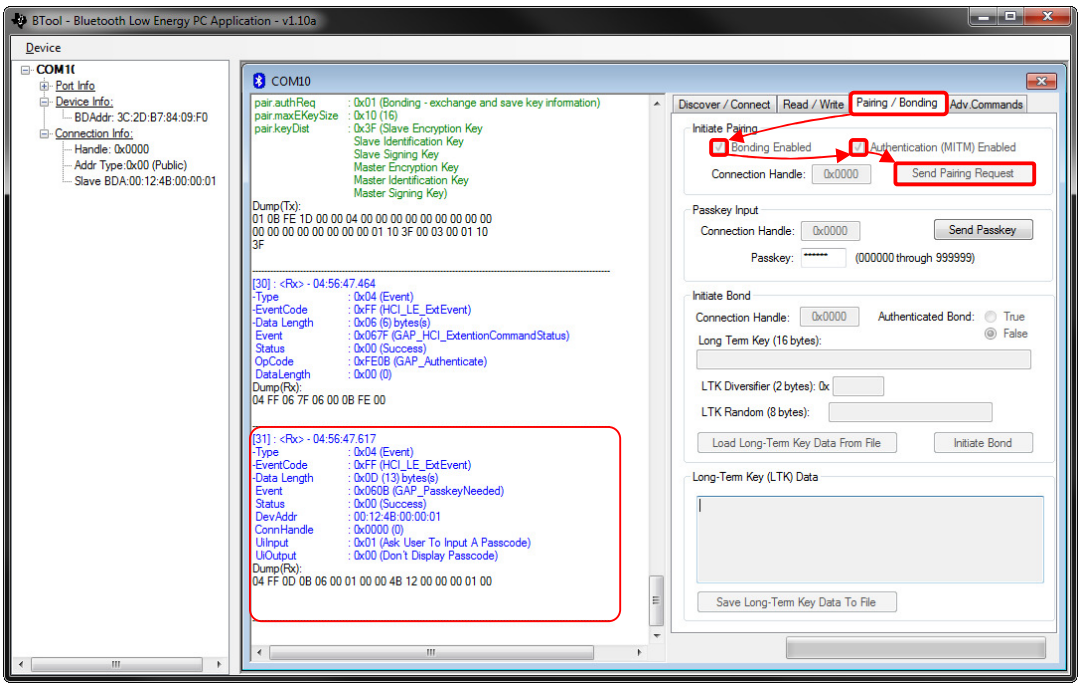


Figure 19 BTool, Send Pairing Request

The peripheral will send a pairing response in return, which will require a six-digit passcode to be entered by the user in order to complete the process. Typically, this passcode is intended to be used by a peripheral device containing a display. By displaying the passkey on the peripheral device and requiring the user to enter it in on the central device's user interface, the link is authenticated, in that it has been verified that the connection has not been hijacked using a man-in-the-middle (MITM) attack.

In the case of the SimpleBLEPeripheral software, a fixed passcode "000000" is used (this value can be modified in the source code). In the box labeled "Passkey" in the "Passkey Input" section, enter the value "000000" and click the "Send Passkey" button, as shown in Figure 20. Note that if you do not send the passkey within 30 seconds after receiving the pairing response message, the pairing process will fail, and you will need to re-send the pairing request.

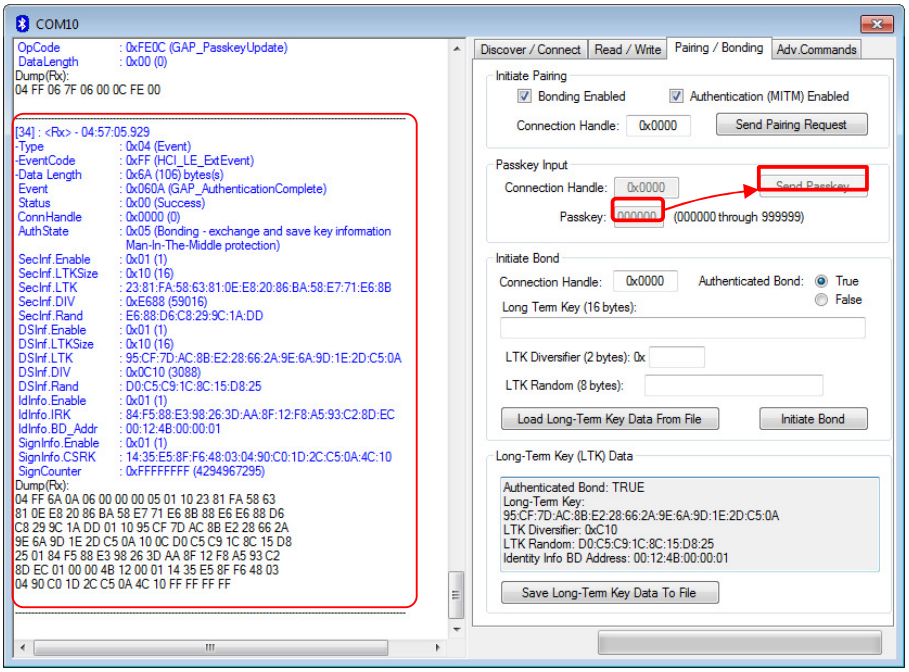


Figure 20 BTool, Send Passkey

When pairing is successfully completed, you will see a "GAP_AuthenticationComplete" event in the log window, with a "Success" status. The BLE connection is now encrypted.

3.4.2 Using Bonding and Long-Term Keys

Bonding is a feature in BLE that allows a device, after initial pairing with a peer, to remember specific information about that peer device. In particular, the long-term key data that is generated during the initial pairing process can be stored locally. If the connection is then terminated and the two devices later reconnect, this data can be used to quickly re-initiate encryption without needing to go through the full pairing process and/or use a passkey. In addition, if a client device had enabled notifications of any characteristics on the server device while the two devices were bonded, the server device will remember the setting and the client will not have to re-enable them.

After pairing has been completed with bonding enabled, the "Long-Term Key (LTK) Data" will be populated with some of the data from the "GAP_AuthenticationComplete" event that was generated during the encryption process. This data is required for re-initiating encryption upon reconnect. Click the "Save Long-Term Key Data to File" button to save this information to file, as shown in Figure 21. The data is saved as in a "comma separated value" (CSV) format as simple text, and can be store anywhere on disk. Be sure to note the location that the file is stored.

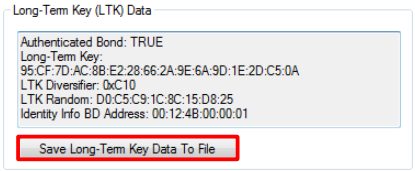


Figure 21 BTool, Save Long-Term Key Data to File

Within the peripheral device, a similar process is going on, in that the SimpleBLEPeripheral software contains a bond manager that is storing the long-term key data that it had generated during encryption. Since the SimpleBLEPeripheral does not have a file system, it is simply storing the data in the nonvolatile memory of the CC2540/41. More information on the bond manager can be found in Texas Instruments *Bluetooth*® Low Energy Software Developer's Guide **Error! Reference source not found.**.

With a bond now active, you can enable notifications of a characteristic value and have that setting remembered for later. Note that if notifications were enabled before going through the pairing process, then the setting will not be stored. Therefore, you will need to re-write the value "01:00" to a client characteristic configuration descriptor. For example, write "01:00" to handle 0x002F to enable the

periodic notifications, as was done in section 3.3.5. You should now be receiving a notification once every five seconds. Because the devices are paired with bonding enabled, the bond manager in the SimpleBLEPeripheral software will store the client characteristic configuration descriptor data in nonvolatile memory.

To verify that bonding worked, you will need to disconnect and re-connect. Click on the “Discover / Connect” tab and click the “Terminate” button at the bottom of the screen to disconnect from the peripheral device, as shown in Figure 22. The message window will show a “GAP_TerminateLink” event with “Success” status. In addition, the connection information in the upper-left corner of the screen will disappear.

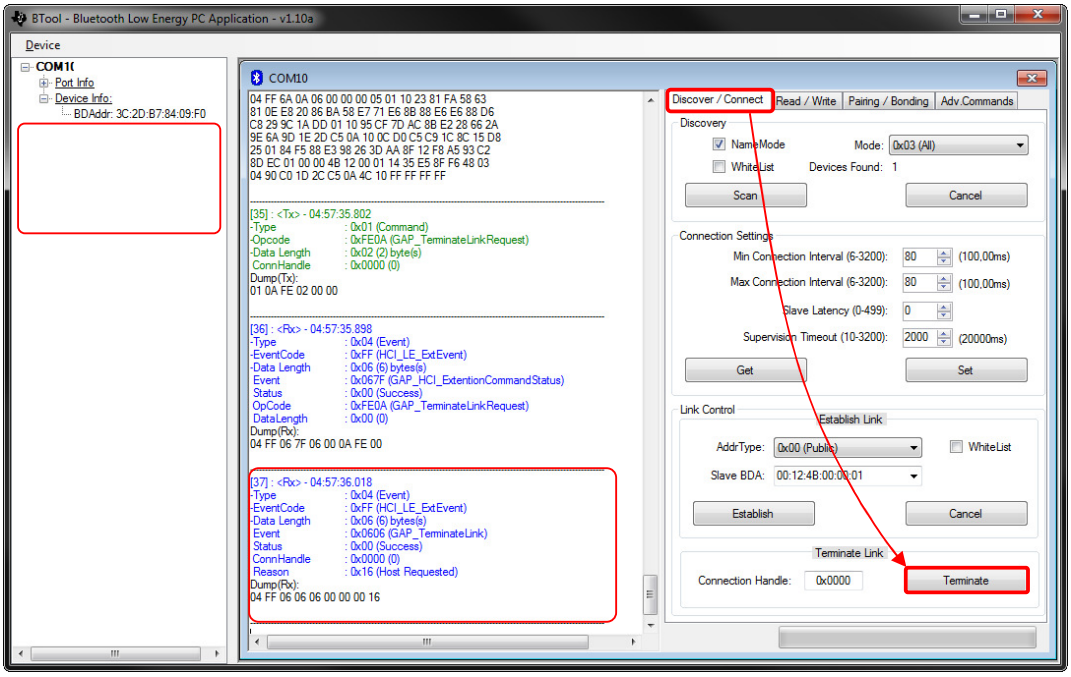


Figure 22 BTool, Terminate Link

At a later time, re-connect with the peripheral device following the procedure in section 3.2.4. Once connected, you will notice that the periodic notifications are no longer enabled. This is because the Simple GATT profile will always reset the value of the client characteristic configuration descriptor back to “00:00” if a connection is terminated or if the device resets.

To re-initiate encryption and re-enable the periodic notifications, return to the “Pairing / Bonding” tab. In the “Initiate Bond” section, click the “Load Long-Term Key Data From File” button, and select the file in which the data was previously stored. The data fields will get automatically populated from the data in the file. Click the “Initiate Bond” button to re-enable encryption, as shown in Figure 23.

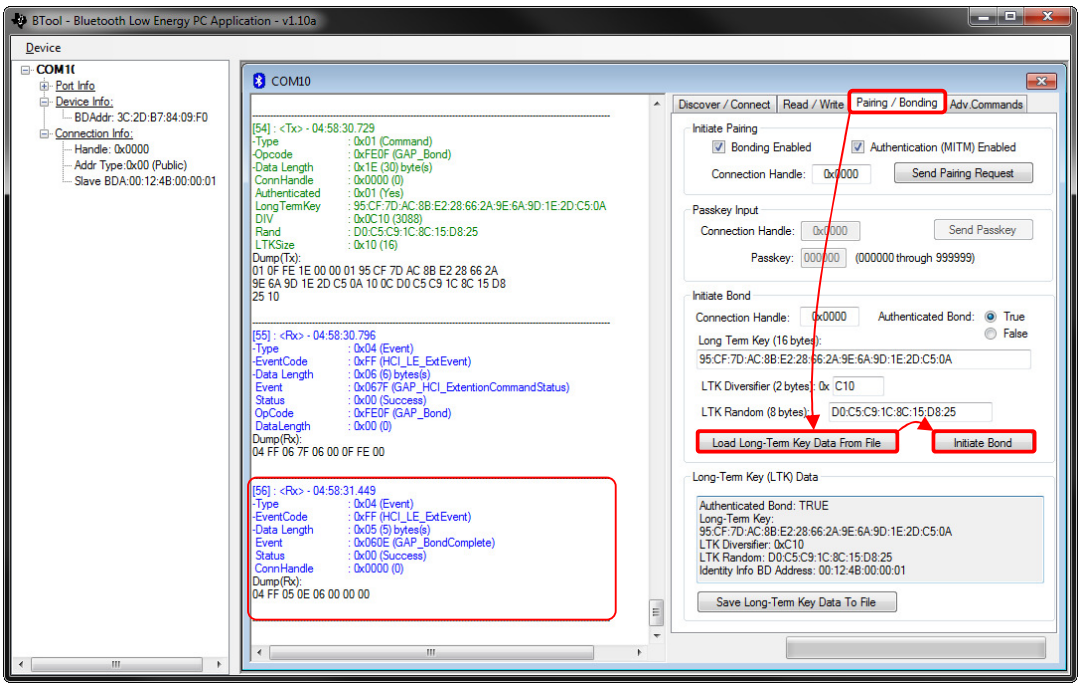


Figure 23 BTool, Re-initiate Encryption

A "GAP_BondComplete" event with "Success" status will be displayed in the message window. This indicates that the link has been re-encrypted, which can be verified by reading the fifth characteristic value in the SimpleGATTProfile at handle 0x0032. You will also now be receiving periodic notifications of the fourth characteristic value, as the client characteristic configuration descriptor value of the characteristic has been restored. Any changes to the client characteristic configuration descriptor value (i.e. turning off notifications) will be saved to nonvolatile memory and remembered for next time that encryption is initiated using the long-term key.

3.5 Additional Sample Applications

In addition to the Sensor tagDemo application, the BLE software development kit includes project and source code files for several additional applications and profiles, including:

- Blood Pressure Sensor- with simulated measurements
- Emulated Keyboard- press the two buttons on the sensor tag to simulate keyboard presses
- Heart Rate Sensor- with simulated measurements
- Health Thermometer- with simulated measurements
- Glucose Sensor – with simulated measurements
- SimpleBLEPeripheral - with proprietary profile which implements all various types of permissions

More information on these projects can be found in the Texas Instruments BLE Sample Applications Guide **Error! Reference source not found.**.

4. Programming / Debugging the CC2540 or CC2541

The CC Debugger included with the CC254*X*DK-MINI kit allows for debugging using IAR Embedded Workbench for 8051, as well as for reading and writing hex files to the CC2540/41 flash memory using the SmartRF Flash Programmer software. SmartRF Flash Programmer also has the capability to change the IEEE address of the CC2540/41 device. The BLE software development kit includes hex files for both the USB Dongle as well as the sensor tag. This section details the hardware setup when using the CC Debugger, as well as information on using SmartRF Flash Programmer. Information on using IAR Embedded Workbench for debugging can be found in the Texas Instruments *Bluetooth*® Low Energy Software Developer's Guide **Error! Reference source not found.**.

4.1 Hardware Setup for Sensor tag

If the sensor tag is viewed with the LED on top and the coin cell battery holder at the bottom, then the set of pins closer to the top are the ones that should be used for connecting to the debugger. Pin 1 is the pin on the lower right side as shown in Figure 24.

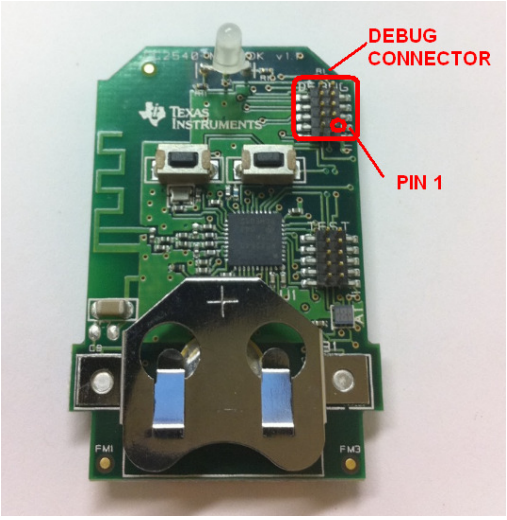


Figure 24 CC2540 Sensor tag, Debug Connector

Connect the CC Debugger to the sensor tag as shown below. Be sure that the ribbon cable is oriented properly, with the red stripe connected to pin 1 as shown in Figure 25.

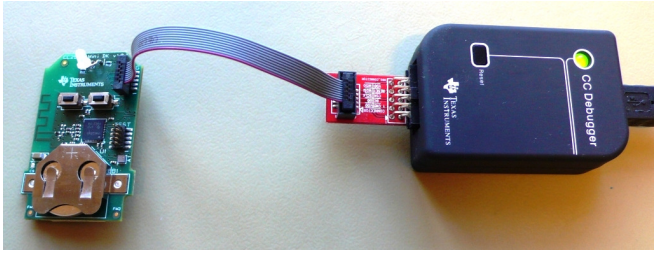


Figure 25 CC2540 Sensor tag Connected to CC Debugger

Insert a coin cell battery in the sensor tag to supply power to the target. **NB!** Note the orientation of the battery (+ up, - down). Next, connect the CC Debugger to the PC's USB port and then to the sensor tag. Note that the CC debugger will by default not supply any power, but it will sense the voltage on the target (in this case the sensor tag) for proper level shifting of the debug signals. The status indicator LED on the CC Debugger should turn on. If the LED is red, that means no CC2540/41 device was detected. If it is green, then a CC2540/41 device has been detected. If the sensor tag is connected and the LED is red, try pressing the reset button on the CC Debugger. This resets the debugger and re-checks for a CC2540/41 device. If the LED still does not turn green, re-check that all cables are securely connected. Also verify that the CC Debugger has the latest firmware (see section 4.3).

Figure 26 CC Debugger Interface

Once the CC Debugger is set up with the status indicator LED showing green, you are ready to either read or write a hex file from the board, or to start debugging a project using IAR Embedded Workbench.

Power Savings Tip: Do not leave the CC Debugger connected to the sensor tag for and extended period of time with the battery in the sensor tag. This will cause a higher, constant current draw from the battery, and will significantly reduce the battery life.

If you intend to perform a lot of debugging and expect to leave the debugger connected to the sensor tag for a long time, it is possible to supply power directly from the CC Debugger. In this case, the first thing you need to do is to remove the battery. This is important in order to avoid any charging current to the battery.

On the CC2540Sensor tag, locate the pads for resistor R1, which are located immediately next to the debug header. Using a soldering iron, solder a small piece of wire across the two pads, shorting them together as shown in Figure 27.

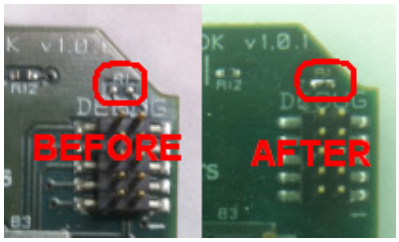


Figure 27 CC2540, Power Device Using CC Debugger

On the CC2541 Sensor tag, short-circuit the two pins on the P1 connector, next to the LED, with the small jumper included in the kit.

WARNING! This kit includes a non-rechargeable lithium battery. To minimize risk of personal injury and/or property damage due to potential of explosion/rupture of battery due to charging the coin cell, always make sure battery is completely removed from the CC2541 Sensor tag before trying to power it from the CC Debugger. As with any lithium battery, proper disposal should always be done and keep out of the reach of children at all times.

4.2 Hardware Setup for USB Dongle

The setup process for flashing the USB Dongle is very similar to the process when flashing the sensor tag. First, plug the USB Dongle into a PC USB port (or a USB hub), as shown in Figure 28.

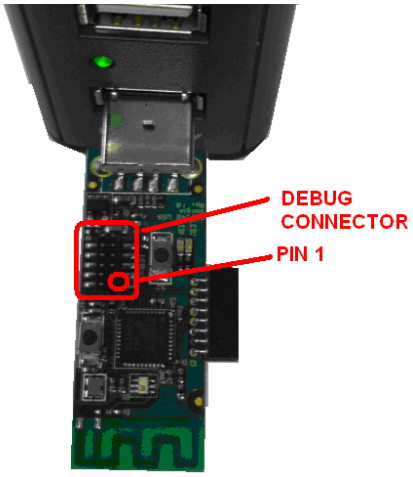


Figure 28 CC2540 USB Dongle

Connect the CC Debugger to the USB Dongle as shown below. Be sure that the ribbon cable is oriented properly, with the red stripe connected to pin 1 as shown in Figure 29.

Figure 29 CC2540 USB Dongle Connected to CC Debugger

Connect the CC Debugger to the PC USB port. The status indicator LED on the CC Debugger should turn on. If the LED is red, that means no CC2540 device was detected. If it is green, then a CC2540 device has been detected. If the USB Dongle is connected and the LED is red, try pressing the reset button on the CC Debugger. This resets the debugger and re-checks for a CC2540 device. If the LED still does not turn green, re-check that all cables are securely connected.

Figure 30 CC Debugger Interface

Once the CC Debugger status LED is showing green, as shown in Figure 30, you are ready to use IAR to debug or to read or write a hex file from/to the USB Dongle.

4.3 Using SmartRF Flash Programmer Software

Note: the instructions in the section apply to the latest version of SmartRF Flash Programmer (version 1.12.6), which is available at the following URL: http://www.ti.com/tool/flash-programmer

To start the application go into your programs by choosing Start > All Programs > Texas Instruments > SmartRF Flash Programmer > SmartRF Flash Programmer. The program start-up screen is shown in Figure 31.

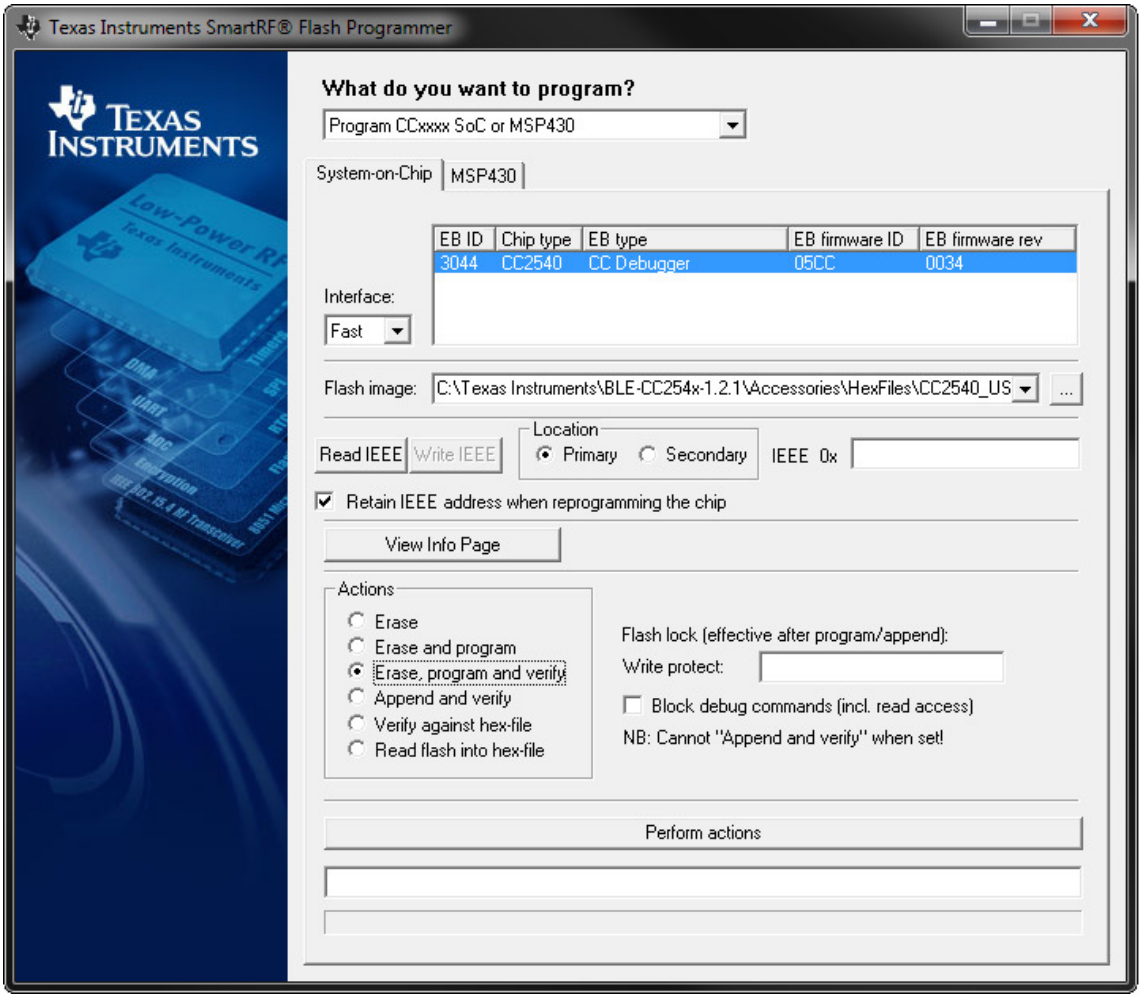


Figure 31 Flash Programmer

Note. If you get prompted to update the EB Firmware (CC Debugger), follow the presented instructions to update the CC Debugger.

4.3.1 Reading or Writing a Hex File to the CC2540/41

To read or write a hex file to the CC2540/41, select the "System-on-Chip" tab (default). The connected CC2540/41 should be detected and show up in the list of devices. Under "Flash image" select the desired hex file that you would like to write to the device. If you are reading from the CC2540/41, under "Flash image" enter the desired path and filename for the hex file. To write to the CC2540/41, under "Actions" select "Erase, program and verify". To read from the CC2540/41, under "Actions" select "Read flash into hex-file". To begin the read or write, click the button "Perform actions".

If the action completes successfully, you should see the progress bar at the bottom of the window fill up, and either one of the following two messages, depending on whether a write or a read was performed: "CC254*X* - ID*XXXX*: Erase, program and verify OK" or "CC254*X* - ID*XXXX*: Flash read OK".

4.3.2 Reading or Writing the CC2540/41 Device Address

Every CC2540/41 device comes pre-programmed with a unique 48-bit IEEE address. This is referred to as the device's "primary address", and cannot be changed. It is also possible to set a "secondary address" on a device, which will override the primary address upon power-up. Flash Programmer can be used to read the primary address, as well as to read or write the secondary address.

To read the primary address of a device connected to the CC Debugger, select "Primary" under the "Location" option, and click the "Read IEEE" button. The primary device address should appear in the box on the right as shown in Figure 32.

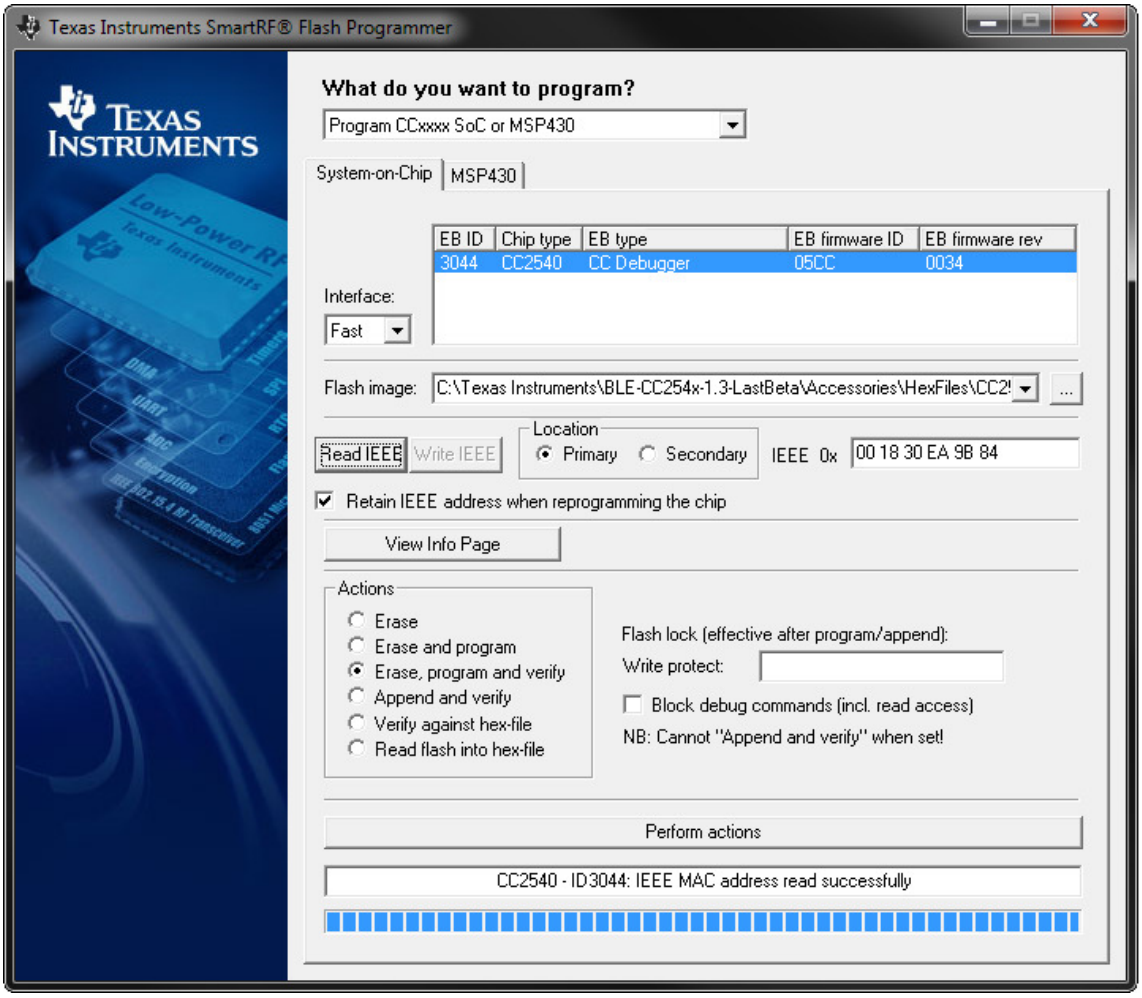


Figure 32 Flash Programmer, Read Primary address

To read the secondary address, select "Secondary" under the "Location" option, and click the "Read IEEE" button. The secondary device address should appear in the box on the right.

To set a new secondary address, select "Secondary" under the "Location" option, and enter the desired address in the box on the right. Click the "Write IEEE" button to perform the write. If the secondary device is set to "FF FF FF FF FF FF", the device will use the primary address. If the secondary device is set to anything else, the secondary address will be used.

5. SmartRF™ Packet Sniffer

The SmartRF™ Packet Sniffer is a PC software application used to display and store RF packets captured with a listening RF hardware node. Various RF protocols are supported, included *Bluetooth* low energy. The Packet Sniffer filters and decodes packets and displays them in a convenient way, with options for filtering and storage to a binary file format.

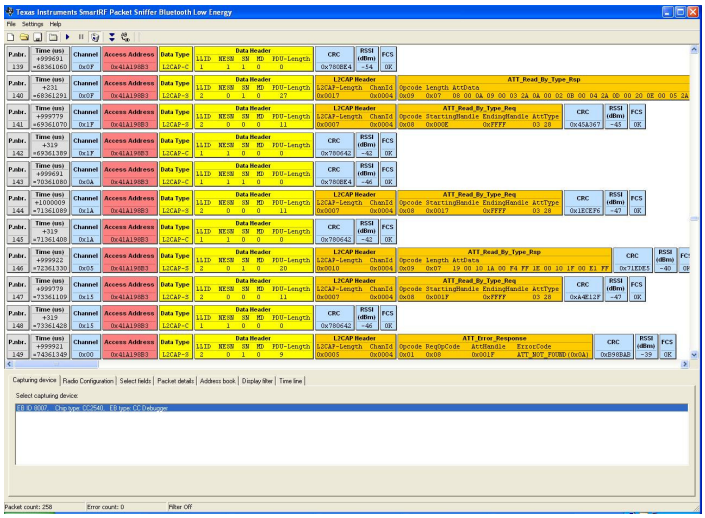


Figure 33 SmartRF Packet Sniffer

The CC2540 USB Dongle included with the CC2540/41 Mini Development Kit can be used as the listening hardware node, and can be useful when debugging *Bluetooth* low energy software applications. The SmartRF™ Packet Sniffer software can be downloaded at http://www.ti.com/tool/packet-sniffer.