TI Designs
UART to Bluetooth® *low energy (BLE) Bridge Design Guide*

TI Designs

TI Designs provide the foundation that you need, including methodology, testing, and design files, to quickly evaluate and customize the system. TI Designs help *you* accelerate your time to market.

Design Resources

TIDC-SPPBLE-SW-RD	Tool Folder Containing Design Files
CC2640	Product Folder
CC2650	Product Folder
BLE Stack	Product Folder

TI E2E™ Community

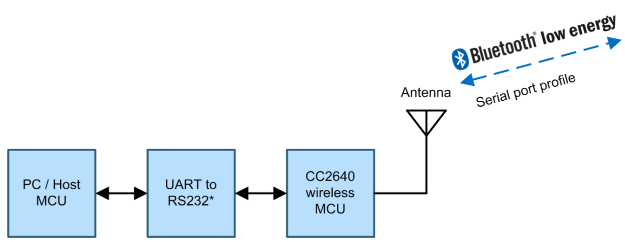
ASK Our E2E Experts
TI E2E BLE Forum

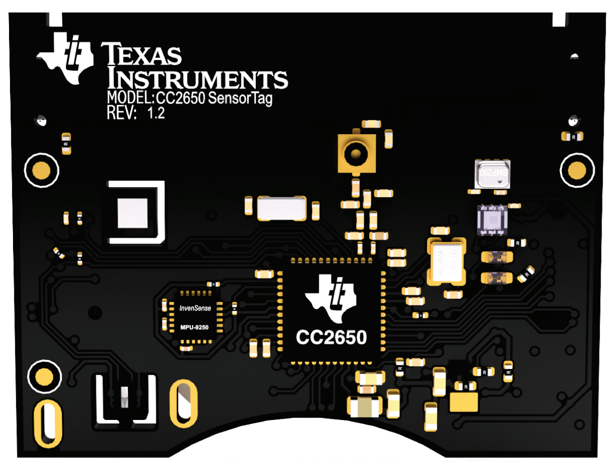
Design Features

- Enables Easy Integration Through Modular Code
- Runs On the SimpleLink™ *Bluetooth* low energy CC2640 wireless MCU
- Uses the TI Royalty Free BLE-Stack™
- Ports Easily to Other Boards Including the SimpleLink SensorTag 2.0
- Offers a Generic Design to Fit Various Applications

Featured Applications

- Cable Replacement
- Home Automation
- PC
- Wireless Sensors





An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

1 Overview

In *Bluetooth* classic (BR/EDR) systems, a Serial Port Profile (SPP) is an adopted profile defined by the *Bluetooth* Special Interest Group (SIG) used to emulate a serial port connection over a *Bluetooth* wireless connection. For *Bluetooth* low energy systems, an adopted profile for implementing SPP over BLE is undefined, thus emulation of a serial port must be implemented as a vendor-specific custom profile. The purpose of this document is to provide an overview of a custom BLE Serial Port Profile on the UART to BLE bridge reference design found on TI.com™ as part of TI Reference Designs™. This SPP over BLE implementation is a complete reference design software built on the existing BLE SDK V2.1. TI recommends reading the *SimpleLink Bluetooth low energy CC2640 wireless MCU Software Developer's Guide* (SWRU393) to learn about the BLE protocol stack and software before using these sample applications.

1.1 Introduction

BLE-Stack V2.1 contains reference project implementations that use both adopted and custom profiles. The APIs provided by the SDK, combined with the TI-RTOS real-time operating system, CC26XXWARE DriverLib, and peripheral drivers, let the application developer quickly implement a BLE custom application on a CC2640 wireless MCU. For a complete understanding of the capabilities of the TI BLE-Stack SDK, including an overview of the BLE protocol stack, TI recommends reading the *SimpleLink Bluetooth low energy CC2640 wireless MCU Software Developer's Guide* (SWRU393) before implementing a custom profile.

1.2 Terminology

The following table lists common abbreviations and terms and their meanings used in this guide.

Terms	Descriptions
SDK	Software Development Kit
BLE	*Bluetooth* low energy
SPP	Serial Port Profile
Client	Serial communication endpoint, connects to service
Server	Serial communication endpoint, provides service (such as the serial port service)
GATT	Generic Attribute, defines how data is transmitting as BLE packets over the air
IDE	Integrated Development Environment, integrated compiler, linker, and debugger such as IAR and CCS

2 Demonstration

This reference design consists of two IDE projects, the SPPBLEServer and the SPPBLEClient that run on their respective endpoints. These projects connect and exchange data wirelessly with each other. This capability creates a virtual serial link over the air. Each byte input can be sent and received by both the server and client. The SPPBLEServer is based on the SimpleBLEPeripheral project while the SPPBLEClient is based on the SimpleBLECentral project. The main difference between these projects is the addition of the Serial Data Interface (SDI) module, which is based on the NPI component in the HostTest project. TI designed the SPP BLE applications to use the UART transport layer but you could adapt this design to work with other serial protocols, such as SPI.

2.1 Project Overview

The structure of the project is similar to the SimpleBLEPeripheral project in terms of the layout of the project directory. The application directory contains the application source code and header files. The project contains two tested configurations listed below. TI has tested and used both IAR and CCS IDEs in this guide. Refer to Section 5 for the versions in this guide. The demonstration uses the CC2650DK, which contains two SmartRF06 boards and two CC2650 evaluation modules (EM). You can replace the EM boards with other development boards such as the SensorTag 2.0. For convenience, the SPPBLEServer project contains a separate configuration set up to run on the SensorTag 2.0. The reference design includes layout and schematic files for an RS232 DevPack, which can connect to the UART interface on the SensorTag 2.0.

FlashROM— Application and stack image using the SmartRF06+CC2650EM hardware platform

FlashROM_SensorTag— Application and stack image using the TI SensorTag hardware platform

2.1.1 Using the Interface

The following button inputs exist for the application on the client side (SPPBLEClient project):

- Button Up: If not connected, start or stop device discovery. If connected to a device, alternate sample read and write requests.
- Button Left: Scroll through the device discovery results.
- Button Select: Connect to or disconnect from the device.
- Button Right: If connected, send a parameter update request.
- Button Down: Does nothing, left for user application

Use the up, left, and select buttons for the client side to discover, select, and connect, respectively, to the server device (SPPBLEServer project).

2.1.2 Hardware and Software Requirements

- 1× CC2650 Development Kit (includes 2× SmartRF06 boards and 2× CC2650EM-7ID)
- 2× USB micro cables
- A terminal application (such as TeraTerm or RealTerm)
- The TI BLE Stack V2.1
- A software patch from TI Designs (.zip)

2.1.3 Operating

1. Unzip this file and replace the files in the respective directories in the SDK.
2. Verify SPPBLEServer and SPPBLEClient sample application have been added with existing applications here: C:\Texas_Instruments\simplelink\ble_cc26xx_2_01_00_44423\Projects\ble
3. Build and download one CC2650EM board with the SPPBLEClient project. (Alternatively, flash the CC2650EM with the precompiled hex (CC2640_SmartRF-7ID_SPPBLEClient_no.hex) using Flash Programmer 2.
4. Build and download the other CC2650EM board with the SPPBLEServer project.
5. Connect both boards to a terminal application through USB cables to a PC.
6. Power cycle the boards.
7. Verify the debug output in the terminal application.

The following table shows the default UART settings used by the applications.

UART Parameters	Default Value
Baud Rate	115200
Data Length	8
Parity	None
Stop Bits	1
Flow Control	None

The server side should start advertising.

1. Press up to initiate a device discovery on the client side.
2. Press left until the correct server device is found.
3. Press select to connect to the server device.
4. Verify that the terminal output is connected.

After the client device scans and connects, the client device discovers the serial port service and configures it to enable notifications of the serial port service data characteristic. The client device may also discover the device information service for the manufacturing and serial numbers.

On the client side, debug output in terminal indicates notifications have been enabled. Entering data in one terminal will transfer to the other terminal wirelessly.

When the link terminates, the server will advertise.

The client device may ask the server for read-only device information. Section 2.2.7 presents details on the supported items for this project. These details include the model number, the serial number, the UART configuration, and UART status (including the number of bytes sent and received).

2.2 Software Description

The application is implemented in spp_ble_server.c. The application is simple and the main processing is implemented within the application task function. The application gets the UART data from the SDI layer and sends it over the air in notification packets. The application does not directly receive wireless data; the data goes to the profile layer and gets sent to the UART by the SDI layer. You can implement queues to transfer the data to the application layer for further processing.

2.2.1 Initializating

The initialization of the application is similar to other projects in terms of API function calls. The SPPBLEServer task function calls the SPPBLEServer_init function before running the main task. This function configures parameters in the peripheral profile, the GAP, and the GAP bond manager. The function sets up the serial port service with standard GATT and GAP services in the attribute server and lets you change the parameters of UART. You can set up the registration for receiving UART messages from the SDI layer. Also during this phase, the SPPBLEServer_init function calls the GAPRole_StartDevice function to set up the GAP functions then calls the GAPBondMgr_Register to register with the bond manager.

2.2.2 Event Processing

The application has a main event processing function in SPPBLEServer_taskFxn.

This function handles events as follows:

Queues:

- appUARTMsgQueue: Services UART data messages received from the SDI.
- appMsgQueue: Processes application messages from the lower-level BLE stack.

Events:

- SBP_PERIODIC_EVT: Acts as a placeholder for periodic processing for application.
- SBP_UART_CONFIG_EVT: Changes UART parameters such as baud rate.

SPPBLEServer_processAppMsg handles application messages as follows:

- SBP_STATE_CHANGE_EVT: Processes a pending GAP Role state change event.
- SBP_CHAR_CHANGE_EVT: Processes a pending SPP characteristic value change event. When the client device writes to the config characteristic, it sets an event to initiate the change of UART hardware parameters.

2.2.3 Callbacks

The application callback functions are as follows:

- SPPBLEServer_stateChangeCB: Callback from GAP Role that indicates a role state change.
- SPPBLEServer _charValueChangeCB: Callback from SPP that indicates a characteristic value change.

2.2.3.1 Queues

Other tasks can add to the application queues to initiate further message processing as follows:

- SPPBLEServer_enqueueMsg: Creates a message and puts the message in RTOS appMsgQueue queue.
- SPPBLEServer_enqueueUARTMsg: Creates a message and puts the message in RTOS appUARTMsgQueue queue.

2.2.4 Sending Data Wirelessly

The client will send WriteNoRsp packets to the server. The server side sends data through notifications. When the SDI receives UART data, the SDI places it in the appUARTMsgQueue queue. The queue is processed in the SPPBLEServer task as part of the application thread.

For example, in spp_ble_server.c:

```
static void SPPBLEServer_taskFxn(UArg a0, UArg a1)
{
…
if (!Queue_empty(appUARTMsgQueue))
{
…
// Send the notification
  retVal = SerialPortService_SetParameter(SERIALPORTSERVICE_CHAR_DATA, pMsg->length,
pMsg->data);

…
```

NOTE: To send notifications, you must enable them. The application sends serial data in notification packets wirelessly when the queue is not empty.

2.2.5 Receiving Data Wirelessly

The server will receive this data in the SerialPortService_WriteAttrCB function. The server notifies the application side.

For example, in SerialPortService.c:

```
      //Write the value
      if ( status == SUCCESS )
      {
        uint8 *pCurValue = (uint8 *)pAttr->pValue;

        //Copy/Store data to the GATT table entry
        memset(pCurValue, 0, SERIALPORTSERVICE_DATA_LEN);
        memcpy(pCurValue, pValue, len);

        //Send Data to UART
        SDITask_sendToUART(pCurValue, len);

        if (len > 0)
        {
         SerialPortService_AddStatusRXBytes( len );
        }

        notifyApp = SERIALPORTSERVICE_CHAR_DATA;
      }

      break;
```

When the server receives data, it also sends data to the UART terminal.

2.2.6 Optimizing Data Throughput

Effective throughput depends on several factors, such as the connection interval and capability of the peer device. By decreasing the connection interval parameter in the application or limiting application processing, you can maximize 1-way throughput. Throughput depends on the limitations of the peer device that might have stricter guidelines on connection intervals and number of packets per connection event than TI's CC2640 wireless MCU.

To support higher throughput, increase the number of buffers allocated for TX buffers in the controller by adding the following into the preprocessor symbols. This value increases the maximum number of packets that you can queue. Refer to *SimpleLink Bluetooth low energy CC2640 wireless MCU Software Developer's Guide* (SWRU393) for more details:

```
MAX_NUM_PDU = 15
```

Increase the heap size in the application to accommodate the increased heap memory usage.

```
HEAPMGR_SIZE = 4096
```

2.2.7 GATT Table

Figure 1 is the GATT table with the handles and UUIDs of characteristics in the SPPBLEServer project. The table was generated using BTOOL.

ConHnd	Handle	Uuid	Uuid Description	Value	Properties
0x0000	0x0001	0x2800	GATT Primary Service Declaration	00:18	
0x0000	0x0002	0x2803	GATT Characteristic Declaration	02:03:00:00:2A	
0x0000	0x0003	0x2A00	Device Name	SPP BLE Server	Rd 0x02
0x0000	0x0004	0x2803	GATT Characteristic Declaration	02:05:00:01:2A	
0x0000	0x0005	0x2A01	Appearance	00:00	Rd 0x02
0x0000	0x0006	0x2803	GATT Characteristic Declaration	02:07:00:04:2A	
0x0000	0x0007	0x2A04	Peripheral Preferred Connection Parameters	50:00:A0:00:00:00:E8:03	Rd 0x02
0x0000	0x0008	0x2800	GATT Primary Service Declaration	01:18	
0x0000	0x0009	0x2803	GATT Characteristic Declaration	20:0A:00:05:2A	
0x0000	0x000A	0x2A05	Service Changed		Ind 0x20
0x0000	0x000B	0x2902	Client Characteristic Configuration	00:00	
0x0000	0x000C	0x2800	GATT Primary Service Declaration	0A:18	
0x0000	0x000D	0x2803	GATT Characteristic Declaration	02:0E:00:23:2A	
0x0000	0x000E	0x2A23	System ID	00:01:02:00:00:03:04:05	Rd 0x02
0x0000	0x000F	0x2803	GATT Characteristic Declaration	02:10:00:24:2A	
0x0000	0x0010	0x2A24	Model Number String	Model Number	Rd 0x02
0x0000	0x0011	0x2803	GATT Characteristic Declaration	02:12:00:25:2A	
0x0000	0x0012	0x2A25	Serial Number String	Serial Number	Rd 0x02
0x0000	0x0013	0x2803	GATT Characteristic Declaration	02:14:00:26:2A	
0x0000	0x0014	0x2A26	Firmware Revision String	Firmware Revision	Rd 0x02
0x0000	0x0015	0x2803	GATT Characteristic Declaration	02:16:00:27:2A	
0x0000	0x0016	0x2A27	Hardware Revision String	Hardware Revision	Rd 0x02
0x0000	0x0017	0x2803	GATT Characteristic Declaration	02:18:00:28:2A	
0x0000	0x0018	0x2A28	Software Revision String	Software Revision	Rd 0x02
0x0000	0x0019	0x2803	GATT Characteristic Declaration	02:1A:00:29:2A	
0x0000	0x001A	0x2A29	Manufacturer Name String	Manufacturer Name	Rd 0x02
0x0000	0x001B	0x2803	GATT Characteristic Declaration	02:1C:00:2A:2A	
0x0000	0x001C	0x2A2A	IEEE 11073-20601 Regulatory Certification Data List	FE:00:65:78:70:65:72:69:6D:65:6E:74:61:6C	Rd 0x02
0x0000	0x001D	0x2803	GATT Characteristic Declaration	02:1E:00:50:2A	
0x0000	0x001E	0x2A50	PnP ID	01:0D:00:00:00:10:01	Rd 0x02
0x0000	0x001F	0x2800	GATT Primary Service Declaration	00:00:00:00:00:00:00:B0:00:40:51:04:E0:C0:00:F0	
0x0000	0x0020	0x2803	GATT Characteristic Declaration	14:21:00:00:00:00:00:00:00:00:B0:00:40:51:04:E1:C0:00:F0	
0x0000	0x0021	0xF000C0E104514000B000000000000000	Unknown		Wwr Nfy 0x14
0x0000	0x0022	0x2902	Client Characteristic Configuration	00:00	
0x0000	0x0023	0x2901	Characteristic User Description	Data Characteristic	
0x0000	0x0024	0x2803	GATT Characteristic Declaration	02:25:00:00:00:00:00:00:00:00:B0:00:40:51:04:E2:C0:00:F0	
0x0000	0x0025	0xF000C0E204514000B000000000000000	Unknown	00:00:00:00:00:00:00	Rd 0x02
0x0000	0x0026	0x2901	Characteristic User Description	Status Characteristic	
0x0000	0x0027	0x2803	GATT Characteristic Declaration	0A:28:00:00:00:00:00:00:00:00:B0:00:40:51:04:E3:C0:00:F0	
0x0000	0x0028	0xF000C0E304514000B000000000000000	Unknown	2D:00:02	Rd Wr 0x0A
0x0000	0x0029	0x2901	Characteristic User Description	Config Characteristic	

Figure 1. GATT Table

3 Serial Port Service Specification

TI designed the serial port service to be versatile and easy to use. This protocol can be adapted to most UART communication protocols (including RS-232).

The serial port service has the following UUID: F000C0E0-0451-4000-B000-00000000-0000.

The service has three characteristics:

Characteristic	UUID
Data	F000C0E1-0451-4000-B000-00000000-0000
Status	F000C0E2-0451-4000-B000-00000000-0000
Config	F000C0E3-0451-4000-B000-00000000-0000

Because the service has dedicated UUIDs, it can operate with different client devices that implement the same service with same UUIDs. For example, a smart phone application can communicate with any other devices that implement this serial port service.

See the TI Serial Port Service specification document for more information on the serial port service.

4 Porting To Other Projects

This section describes how to port the SPPBLEServer to other projects.

4.1 Porting SPPBLEServer to Other Projects

1. Copy the SDI folder to C:\Texas_Instruments\simplelink\ble_cc26xx_2_01_00_44423\Components.
2. Copy the SerialPortService folder to C:\Texas_Instruments\simplelink\ble_cc26xx_2_01_00_44423\Projects\ble\Profiles.
3. Define SDI_USE_UART in preprocessor symbols.
4. In preprocessor settings, add the following include directories:
   - $PROJ_DIR$/../../../../../../../Components/sdi.
   - $PROJ_DIR$/../../../../../../../Projects/ble/Profiles/SerialPortService/CC26xx.
   - $PROJ_DIR$/../../../../../../../Projects/ble/Profiles/SerialPortService.
5. Create the SDI task in main.c.
6. Include header file to avoid compile errors.
7. Initialize semaphore.
8. Add other changes in SPPBLEServer.c (check the difference between the two files [SPPBLEServer.c and SimpleBLEPeripheral.c]), including adding the service and the queues for UART message processing.

4.2 Running SPPBLEServer on SensorTag

The SPPBLEServer application has a project configuration for the SensorTag. You can select the configuration in the workspace pane by selecting FlashROM_SensorTag configuration.

To emulate this configuration, do the following:

1. Remove or exclude the existing board file.
2. Add Board.c from C:\Texas_Instruments\simplelink\ble_cc26xx_2_01_00_44423\Projects\ble\SensorTag\CC26xx\Source\Application\Board_patch.
3. Replace $PROJ_DIR$/../../../../../../../Projects/ble/SensorTag/CC26xx/Source/Application/Board_patch/CC26XXST_0120 in preprocessor settings.
4. In preprocessor symbols, remove the LCD driver: xTI_DRIVERS_LCD_INCLUDED.
5. Connect to the port enumerated as XDS110 Class Application/User UART when connected to the Debugger DevPack mounted on the SensorTag.

5 Test Results

The demonstration of the application indicates that TI has tested the SPP BLE projects successfully. The following table shows functional tests performed using various configurations.

Test Case	Device Under Test	Compiler	Peer Device	Status
Successfully send/receive file	SPP BLE Server	IAR 7.40.2	SPP BLE Client	Passed
Successfully send/receive file	SPP BLE Server	CCS 6.1.0	SPP BLE Client	Passed
Successfully send/receive bytes	SPP BLE Server	IAR 7.40.2	BLE Scanner on Android 5.0.1	Passed
Send/Receive bytes	SPP BLE Server	CCS 6.1.0	LightBlue on iOS 7.0.4	Passed

Troubleshooting

Several problems might be encountered while porting or using the SPP BLE application. The following table lists a few common problems and their solutions.

Symptom	Solution
Project does not download/debug.	Debugger information may be lost during the porting process. Go to project settings and configure the Debugger section to match other BLE projects like the SimpleBLEPeripheral project.
Unable to open Application UART	Restart the computer.
Error connecting to target	Ensure correct debugger is selected in project settings.
No output on UART terminal	Verify the correct board file is used in the project

For further information and addendums to this guide, see the wiki.

6 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDC-SPPBLE-SW-RD.

7 References

Included with the TI *Bluetooth* Low Energy V2.1 SDK Release (path and file references in this document assume that the BLE development kit software has been installed to the default path C:\Texas_Instruments\simplelink\ble_cc26xx_2_01_00_44423\Projects\ble\Profiles\).

1. *SimpleLink Bluetooth low energy CC2640 wireless MCU Software Developer's Guide* (SWRU393)
2. TI BLE Stack V2.1 www.ti.com/ble-stack
3. *CC2540 and CC2541 Mini Development Kit User's Guide* (SWRU270)

8 About the Author

Zahid Haq is an applications engineer for the *Bluetooth* Smart Applications team based at TI San Diego. Zahid has more than 3 years' experience in supporting and developing with TI *Bluetooth* devices. Zahid earned his Bachelor of Science in Electrical Engineering (BSEE) from University of Texas at Austin.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

Revision History

Changes from May 18, 2015 to September 1, 2015 **Page**

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS