User's Guide

CC256x TI Bluetooth Stack SPPLEDemo App

TEXAS INSTRUMENTS

ABSTRACT

This document talks about the SPPLE application in detail. The application allows the user to use a console to use Bluetooth Low Energy (BLE) to establish connection between two BLE devices, send Bluetooth commands and exchange data overBLE.

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

This application demonstrates a BR/EDR SPP based application as well as a custom application, SPPLE, over Bluetooth LE that is similar in functionality to the BR/EDR appliacation.The SPPLE Profile is similar to the SPP profile except the SPPLE uses LE transport compared to BR/EDR transport in the SPP profile. The SPP profile emulates serial cable connections. There are two roles defined in this profile. The first is the server that has the SPPLE service running and has open an server port. The client is a device that connects to the server. Both of these devices can then exchange data with each other.

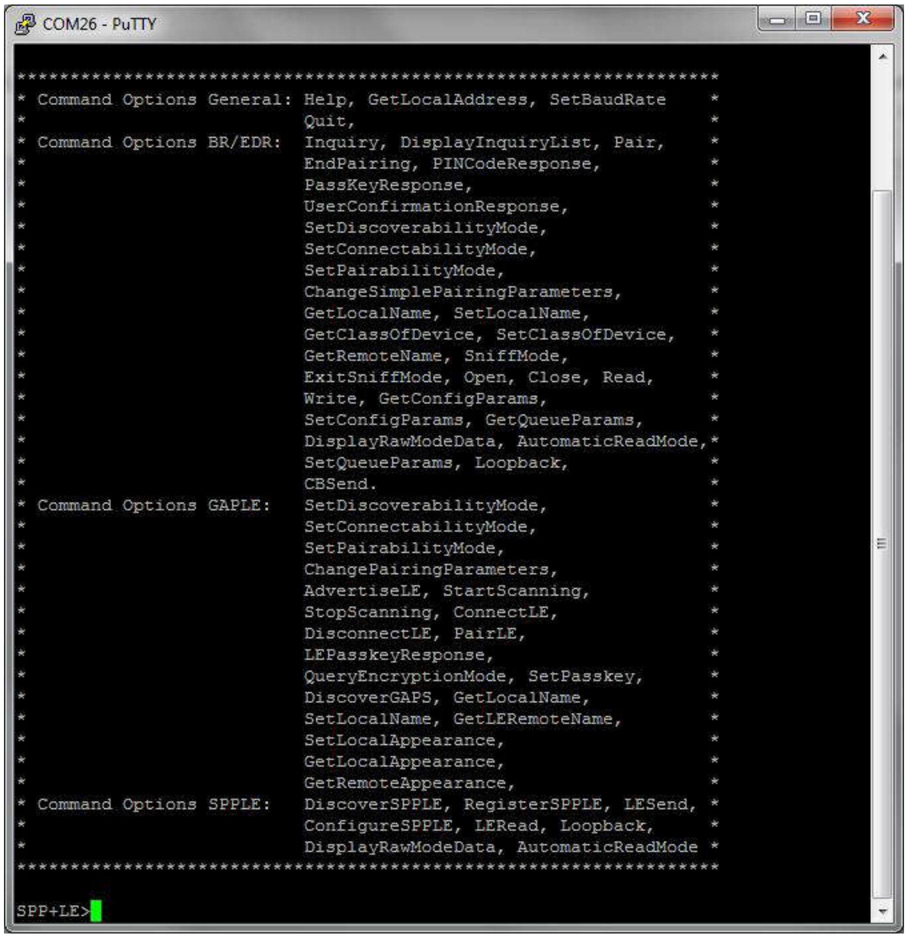
Visit the TI Dual-Mode Bluetooth® Stack on MSP432™ MCUs or Dual-Mode Bluetooth® Stack on STM32F4 MCUs pages before trying the application described on this page.

Note

The same instructions can be used to run this demo on the Tiva, MSP432 or STM32F4 Platforms.

2 Running the Bluetooth Code

Once the code is flashed, connect the board to a PC using a miniUSB or microUSB cable. Once connected, wait for the driver to install. The driver appears as MSP-EXP430F5438 USB -Serial Port(COM x), Tiva Virtual COM Port (COM x),XDS110 Class Application/User UART (COM x) for MSP432, under Ports (COM & LPT) in the Device manager. Attach a Terminal program like PuTTY to the serial port x for the board. Use the serial parameters 115200 Baud (9600 for MSP430), 8, n, 1. Once connected, reset the device using Reset S3 button (located next to the mini USB connector for the MSP430) and observe the stack getting initialized on the terminal and the help screen displays, which shows all of the commands. This device becomes the server. Connect the second board via miniUSB or microUSB cable and follow the same steps performed before when running the Bluetooth code on the first board. The second device that is connected to the computer is the client.



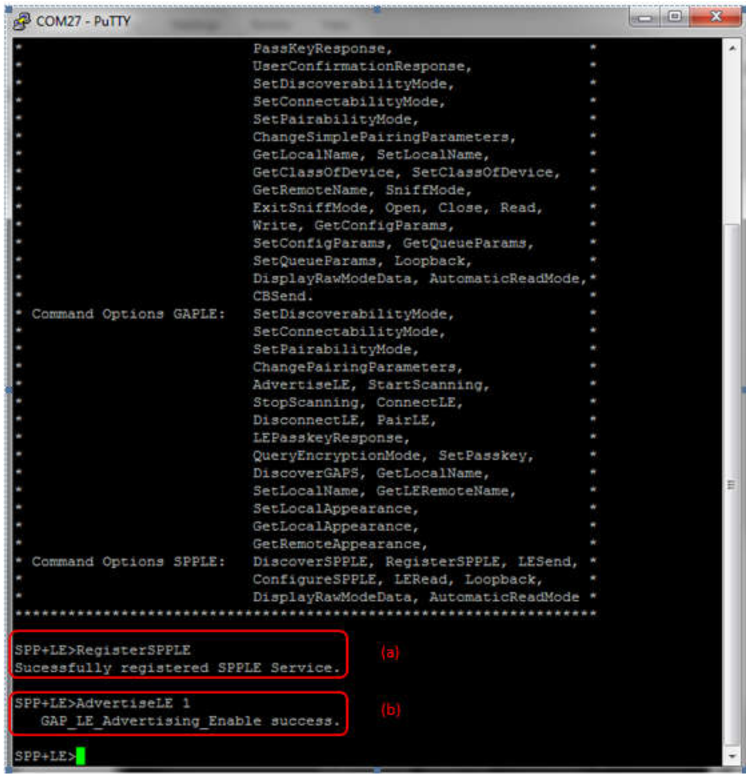
```
*******************************************************************
* Command Options General: Help, GetLocalAddress, SetBaudRate    *
*                          Quit,                                  *
* Command Options BR/EDR:  Inquiry, DisplayInquiryList, Pair,     *
*                          EndPairing, PINCodeResponse,           *
*                          PassKeyResponse,                       *
*                          UserConfirmationResponse,              *
*                          SetDiscoverabilityMode,                *
*                          SetConnectabilityMode,                 *
*                          SetPairabilityMode,                    *
*                          ChangeSimplePairingParameters,         *
*                          GetLocalName, SetLocalName,            *
*                          GetClassOfDevice, SetClassOfDevice,    *
*                          GetRemoteName, SniffMode,              *
*                          ExitSniffMode, Open, Close, Read,      *
*                          Write, GetConfigParams,                *
*                          SetConfigParams, GetQueueParams,       *
*                          DisplayRawModeData, AutomaticReadMode,*
*                          SetQueueParams, Loopback,              *
*                          CBSend.                                *
* Command Options GAPLE:   SetDiscoverabilityMode,                *
*                          SetConnectabilityMode,                 *
*                          SetPairabilityMode,                    *
*                          ChangePairingParameters,               *
*                          AdvertiseLE, StartScanning,            *
*                          StopScanning, ConnectLE,               *
*                          DisconnectLE, PairLE,                  *
*                          LEPasskeyResponse,                     *
*                          QueryEncryptionMode, SetPasskey,       *
*                          DiscoverGAPS, GetLocalName,            *
*                          SetLocalName, GetLERemoteName,         *
*                          SetLocalAppearance,                    *
*                          GetLocalAppearance,                    *
*                          GetRemoteAppearance,                   *
* Command Options SPPLE:   DiscoverSPPLE, RegisterSPPLE, LESend, *
*                          ConfigureSPPLE, LERead, Loopback,      *
*                          DisplayRawModeData, AutomaticReadMode *
*******************************************************************

SPP+LE>
```

3 Demo Application

Below is a description on how to use the demo application to connect two configured boards and communicate over bluetoothLE. The included application registers a custom service on aboard when the stack is initialized.

3.1 Device 1 (Server) Setup on the Demo Application

1. To start, place the device into server mode by typing: Server on the console. The SPP-LE Service can then be started by running RegisterSPPLE.
2. Next, the device acting as a server needs to advertise to other devices. This can be done by running AdvertiseLE 1.



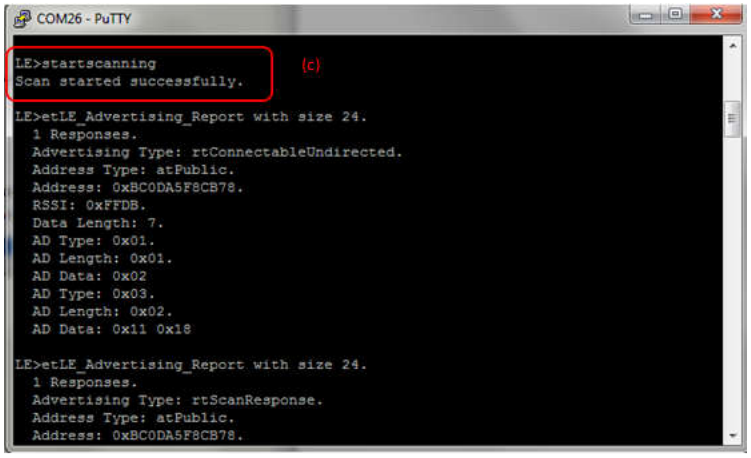
```
*                          PassKeyResponse,                      *
*                          UserConfirmationResponse,             *
*                          SetDiscoverabilityMode,               *
*                          SetConnectabilityMode,                *
*                          SetPairabilityMode,                   *
*                          ChangeSimplePairingParameters,        *
*                          GetLocalName, SetLocalName,           *
*                          GetClassOfDevice, SetClassOfDevice,   *
*                          GetRemoteName, SniffMode,             *
*                          ExitSniffMode, Open, Close, Read,     *
*                          Write, GetConfigParams,               *
*                          SetConfigParams, GetQueueParams,      *
*                          SetQueueParams, Loopback,             *
*                          DisplayRawModeData, AutomaticReadMode,*
*                          CBSend.                               *
* Command Options GAPLE:   SetDiscoverabilityMode,               *
*                          SetConnectabilityMode,                *
*                          SetPairabilityMode,                   *
*                          ChangePairingParameters,              *
*                          AdvertiseLE, StartScanning,           *
*                          StopScanning, ConnectLE,              *
*                          DisconnectLE, PairLE,                 *
*                          LEPasskeyResponse,                    *
*                          QueryEncryptionMode, SetPasskey,      *
*                          DiscoverGAPS, GetLocalName,           *
*                          SetLocalName, GetLERemoteName,        *
*                          SetLocalAppearance,                   *
*                          GetLocalAppearance,                   *
*                          GetRemoteAppearance,                  *
* Command Options SPPLE:   DiscoverSPPLE, RegisterSPPLE, LESend, *
*                          ConfigureSPPLE, LERead, Loopback,     *
*                          DisplayRawModeData, AutomaticReadMode *
******************************************************************

SPP+LE>RegisterSPPLE
Sucessfully registered SPPLE Service.

SPP+LE>AdvertiseLE 1
   GAP_LE_Advertising_Enable success.

SPP+LE>
```

3.2 Device 2 (Client) Setup on the Demo Application

1. Place the device into client mode by typing Client on the console.
2. The client LE device can try to find which LE devices are in the vicinity using the command: StartScanning.
3. Once you have found the device, you can stop scanning by using the command: StopScanning.

Note

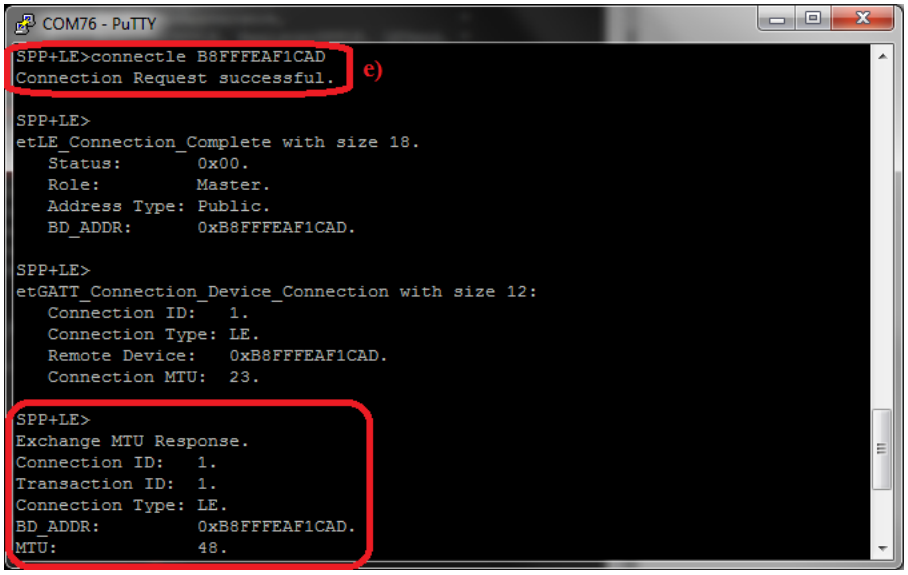
Steps 2 and 3 are optional if the Bluetooth address is already known.



```
LE>startscanning
Scan started successfully.

LE>etLE_Advertising_Report with size 24.
  1 Responses.
  Advertising Type: rtConnectableUndirected.
  Address Type: atPublic.
  Address: 0xBC0DA5F8CB78.
  RSSI: 0xFFDB.
  Data Length: 7.
  AD Type: 0x01.
  AD Length: 0x01.
  AD Data: 0x02
  AD Type: 0x03.
  AD Length: 0x02.
  AD Data: 0x11 0x18

LE>etLE_Advertising_Report with size 24.
  1 Responses.
  Advertising Type: rtScanResponse.
  Address Type: atPublic.
  Address: 0xBC0DA5F8CB78.
```

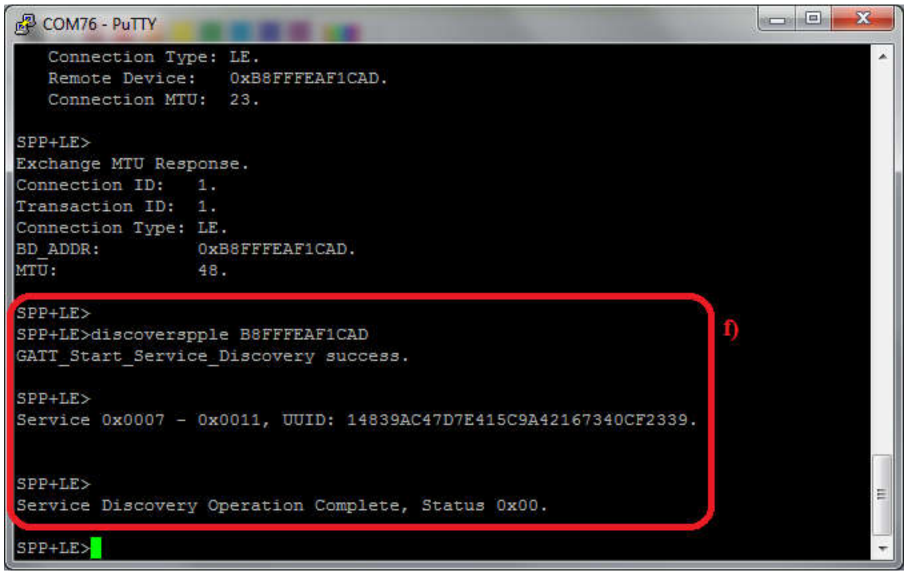
3.3 Initiating Connection from Device 2

1. Once the application on the client side knows the Bluetooth address of the device that is advertising, the application can connect to that device using the command: ConnectLE <BluetoothAddress>



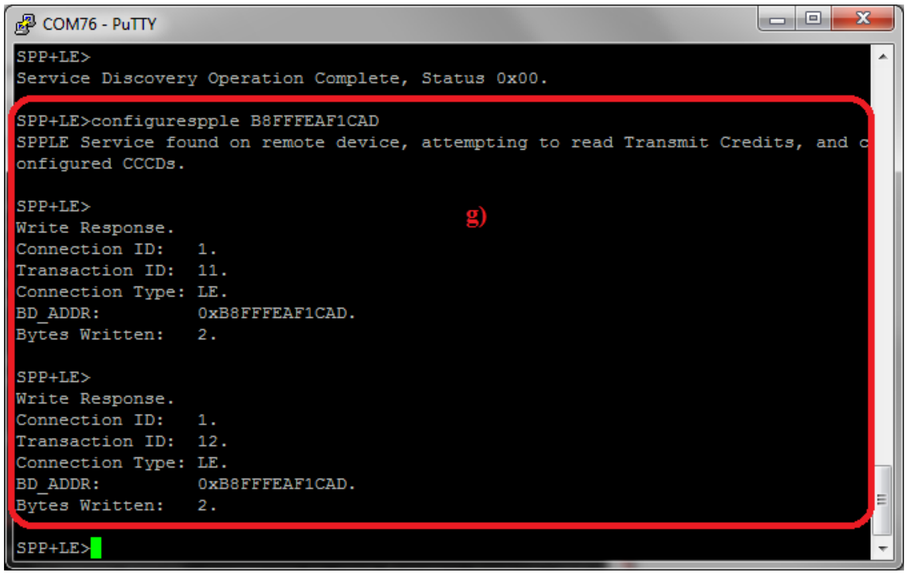
```
SPP+LE>connectle B8FFFEAF1CAD
Connection Request successful.

SPP+LE>
etLE_Connection_Complete with size 18.
   Status:        0x00.
   Role:          Master.
   Address Type:  Public.
   BD_ADDR:       0xB8FFFEAF1CAD.

SPP+LE>
etGATT_Connection_Device_Connection with size 12:
   Connection ID:    1.
   Connection Type:  LE.
   Remote Device:    0xB8FFFEAF1CAD.
   Connection MTU:   23.

SPP+LE>
Exchange MTU Response.
Connection ID:    1.
Transaction ID:   1.
Connection Type:  LE.
BD_ADDR:          0xB8FFFEAF1CAD.
MTU:              48.
```

3.4 Identify Supported Services



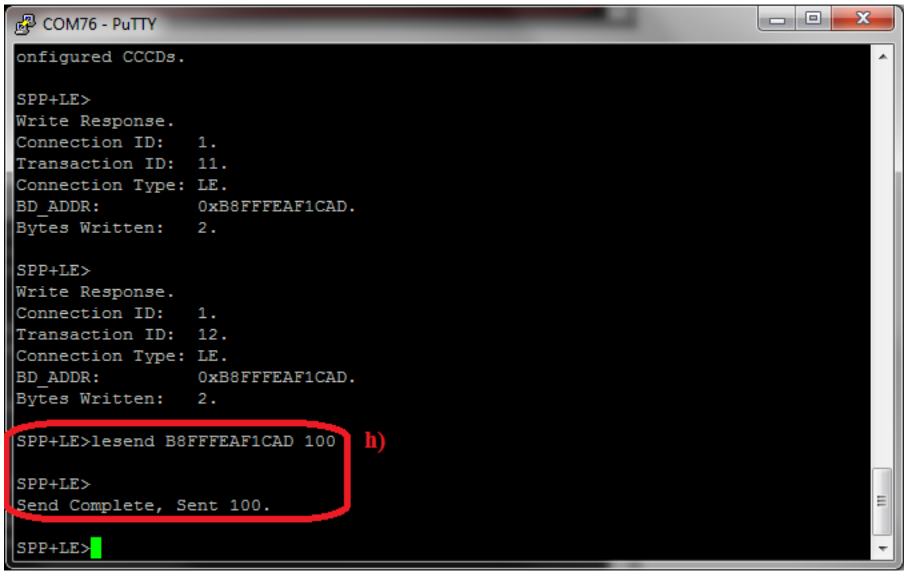
```
   Connection Type:  LE.
   Remote Device:    0xB8FFFEAF1CAD.
   Connection MTU:   23.

SPP+LE>
Exchange MTU Response.
Connection ID:    1.
Transaction ID:   1.
Connection Type:  LE.
BD_ADDR:          0xB8FFFEAF1CAD.
MTU:              48.

SPP+LE>
SPP+LE>discoverspple B8FFFEAF1CAD
GATT_Start_Service_Discovery success.

SPP+LE>
Service 0x0007 - 0x0011, UUID: 14839AC47D7E415C9A42167340CF2339.


SPP+LE>
Service Discovery Operation Complete, Status 0x00.

SPP+LE>
```

1. After Initialization, the device needs to find out if SPP services are supported. To do this run DiscoverSPPLE <Server BD-Address> on the client.
2. After finding out support for SPP-LE, we need to configure SPP-LE. This is done by running ConfigureSPPLE <Server BD-Address> on the client.



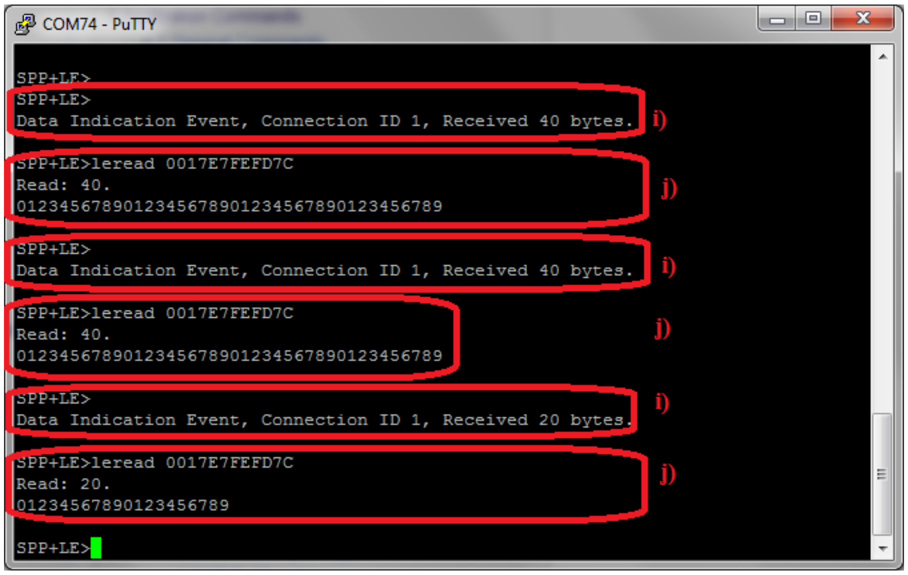
```
SPP+LE>
Service Discovery Operation Complete, Status 0x00.

SPP+LE>configurespple B8FFFEAF1CAD
SPPLE Service found on remote device, attempting to read Transmit Credits, and c
onfigured CCCDs.

SPP+LE>
Write Response.
Connection ID:   1.
Transaction ID:  11.
Connection Type: LE.
BD_ADDR:         0xB8FFFEAF1CAD.
Bytes Written:   2.

SPP+LE>
Write Response.
Connection ID:   1.
Transaction ID:  12.
Connection Type: LE.
BD_ADDR:         0xB8FFFEAF1CAD.
Bytes Written:   2.

SPP+LE>
```

3.5 Data Transfer Between Client and Server



```
onfigured CCCDs.

SPP+LE>
Write Response.
Connection ID:   1.
Transaction ID:  11.
Connection Type: LE.
BD_ADDR:         0xB8FFFEAF1CAD.
Bytes Written:   2.

SPP+LE>
Write Response.
Connection ID:   1.
Transaction ID:  12.
Connection Type: LE.
BD_ADDR:         0xB8FFFEAF1CAD.
Bytes Written:   2.

SPP+LE>lesend B8FFFEAF1CAD 100

SPP+LE>
Send Complete, Sent 100.

SPP+LE>
```

1. After configuring we can send data between client and server. To send data, use LESend <Remote Device BD-Address> <Number of bytes>.
2. Once the other device receives the data, the device also receives a Data Indication event.
3. The receiving device can then read the data that was sent using command: LERead <Remote Device BD-Address>
4. This prints out the data that was sent. This data was sent over BluetoothLE using a custom service of SPPLE in the sample application.



```
SPP+LE>
SPP+LE>
Data Indication Event, Connection ID 1, Received 40 bytes.
SPP+LE>leread 0017E7FEFD7C
Read: 40.
0123456789012345678901234567890123456789
SPP+LE>
Data Indication Event, Connection ID 1, Received 40 bytes.
SPP+LE>leread 0017E7FEFD7C
Read: 40.
0123456789012345678901234567890123456789
SPP+LE>
Data Indication Event, Connection ID 1, Received 20 bytes.
SPP+LE>leread 0017E7FEFD7C
Read: 20.
01234567890123456789

SPP+LE>
```

3.6 Multiple SPPLE Connections Guide

Two SPPLE Connections

1. In This version, we test two simultaneous SPPLE connections to the MSP430. The remote devices are used as a peripheral device while the MSP430 acts as the central device.
2. Connect to the first device, discover and configure services on the first device. When discovering services and configuring services we have to specify the remote BD_ADDR that weconnected to.
3. Similarly, Connect to the second device, discover and configure services on the second device.
4. To send data to the first remote device data we use LeSend <BD-ADDR> <Number of Bytes to be sent>.
5. To send data to the second remote device data we use LeSend <BD-ADDR> <Number of Bytes to be sent>.
6. To read data from the first remote device data we use LeRead <BD-ADDR>.
7. To read data from the second remote device data we use LeRead <BD-ADDR>.
8. When the Automaticreadmode, DisplayRawmodedata or Loopback turns on, both connections turn on.

One SPP and One SPPLE Connection

1. In this version, test an SPP connection and SPPLE Connection at the same time to the MSP430. One of the remote devices is used as a peripheral LE device while the remote deviceas SPP Client.
2. Connect to the first device, discover and configure services on the first device. When discovering services and configuring services we have to specify the remote BD_ADDR that weconnected to.
3. Open an SPP server and let the second remote device connect.
4. To send data to the first remote device data we use LeSend <BD-ADDR> <Number of Bytes to be sent>.
5. To send data to the second remote data we use CBSend <Number of Bytes to be sent> <Serial Port ID> . If we want to write a small amount of data we use the command Write <Serial Port ID>.
6. To read data from the first remote device data we use LeRead <BD-ADDR>.
7. To read data from the second remote device data we use Read.
8. Turn on Automaticreadmode, DisplayRawmodedata or Loopback (turns on for both connections.)

4 Demonstrating SPP LE on an iOS Device with the LightBlue App

4.1 LightBlue Overview

The LightBlue app is a free iOS app that tests and demonstrates the GATT Profile using Bluetooth Low Energy (BLE).This app creates custom services and also interacts with servers that have custom services. The app supports both the client and server roles of GATT. The next section explains how to use the app with the SPPLEDemo application.

4.2 SPP LE Service Overview

SPP LE is not an official Bluetooth service. This is a custom service that is designed to demonstrate using Bluetooth Low Energy to send and receive data in a similar manner that ClassicBluetooth's SPP profile does. This uses a credit based protocol to send and receive data. For a device to send data to a remote device with the SPP LE protocol, the remote device must have provided the device with "credits". These credits specify how much data the device is allowed to send. When a device has sent the maximum number of credits, the device must wait for the remote device to provide more credits before sending. In this application 1 credit is equivalent to 1 byte (octet) of data.

4.2.1 Characteristics

SPP LE implements credit-based protocol using GATT characteristics. The SPP LE service has 4 characteristics:

Name	UUID	Purpose
Rx Characteristic	0x8B00ACE7-EB0B-49B0-BBE9-9AEE0A26E1A3	Client sends data to the server using this characteristic with an ATT Write Request.
Tx Credits Characteristic	0xBA04C4B2-892B-43BE-B69C-5D13F2195392	Client sends credits to the server using this characteristic with an ATT Write Request.
Tx Characteristic	0x0734594A-A8E7-4B1A-A6B1-CD5243059A57	Server sends data to the client using this characteristic with an ATT Handle Value Notification.
Rx Credits Characteristic	0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4	Server sends credits to the client using this characteristic with an ATT Handle Value Notification.

The client and server use these characteristics to send and receive data and credits. The following is a demonstration of the SPPLEDemo as the server and LightBlue as the client. Download the LightBlue app from the App Store and turn on Bluetooth on an iOS device.

Note

For more information about characteristics, ATT Write Requests, and ATT Handle Value Notifications, please refer to the Attribute Protocol (ATT) and Generic AttributeProfile (GATT) specifications in the Bluetooth Core specification, which can be found on the Bluetooth SIG's website

.

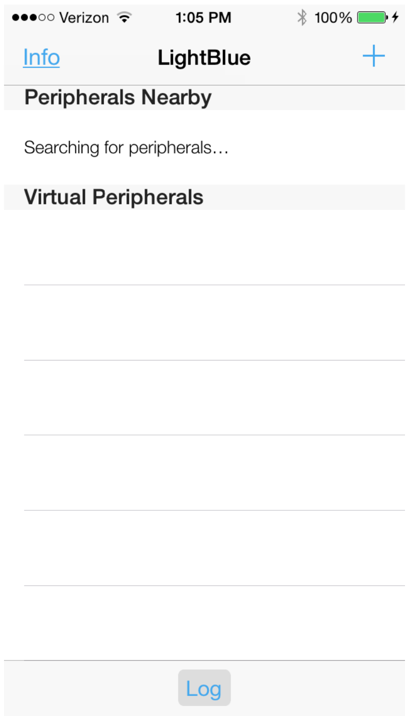
Note

The following instructions were confirmed in version 2.2.0 of LightBlue running on an iPhone 5 with iOS 8.1.3. These instructions can be used with the SPPLEDemo app from any TI Bluetooth SDK, but in this example the SPPLEDemo app from the Tiva v1.2 R2 SDK.

5 LightBlue as the Client/SPPLEDemo as the Server

5.1 Connecting the Devices

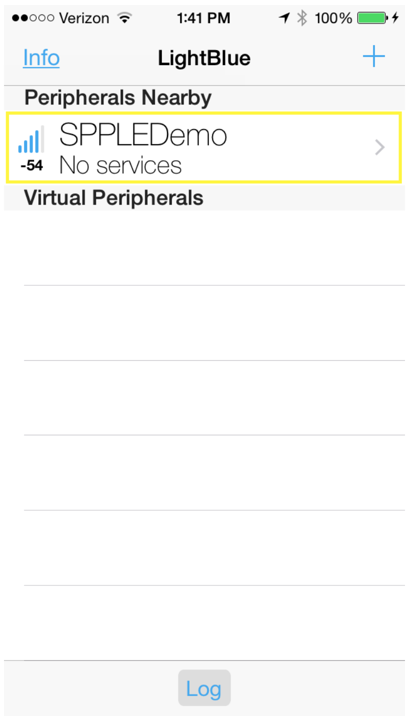
First establish a connection between the devices. To do this open the LightBlue app, observe a screen similar to the following:



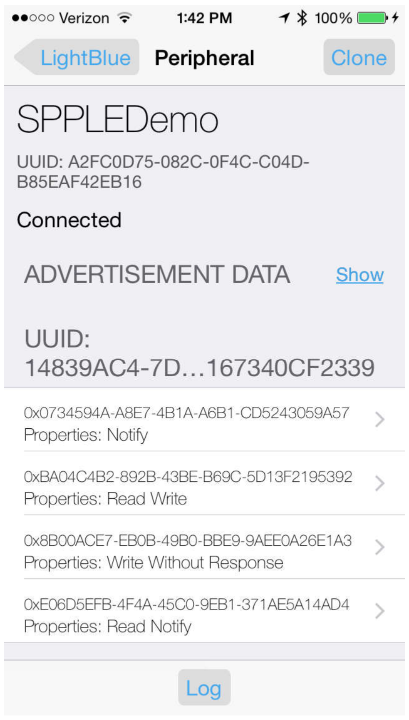
In the SPPLEDemo terminal start the app as a server, register the SPP LE Service, and begin advertising using the Server, RegisterSPPLE, and AdvertiseLE 1 commands. Observe the following in the terminal:

```
OpenStack().
Bluetooth Stack ID: 1.
Device Chipset: 4.1.
BD_ADDR: 0x0017e9d3581a

Command Options: Server, Client, Help

SPP+LE>Server

 Command Options General: Help, GetLocalAddress, SetBaudRate
 Quit,
 Command Options BR/EDR: Inquiry, DisplayInquiryList, Pair,
 EndPairing, PINCodeResponse,
 PassKeyResponse,
 UserConfirmationResponse,
 SetDiscoverabilityMode,
 SetConnectabilityMode,
 SetPairabilityMode,
 ChangeSimplePairingParameters,
 GetLocalName, SetLocalName,
 GetClassOfDevice, SetClassOfDevice,
 GetRemoteName, SniffMode,
 ExitSniffMode, Open, Close, Read,
 Write, GetConfigParams,
 SetConfigParams, GetQueueParams,
 SetQueueParams, Loopback,
 DisplayRawModeData, AutomaticReadMode,
 CBSend.
 Command Options GAPLE: SetDiscoverabilityMode,
 SetConnectabilityMode,
 SetPairabilityMode,
 ChangePairingParameters,
 AdvertiseLE, StartScanning,
 StopScanning, ConnectLE,
 DisconnectLE, PairLE,
 LEPasskeyResponse,
 QueryEncryptionMode, SetPasskey,
 DiscoverGAPS, GetLocalName,
 SetLocalName, GetLERemoteName,
 SetLocalAppearance,
 GetLocalAppearance,
 GetRemoteAppearance,
 Command Options SPPLE: DiscoverSPPLE, RegisterSPPLE, LESend,
 ConfigureSPPLE, LERead, Loopback,
 DisplayRawModeData, AutomaticReadMode

SPP+LE>RegisterSPPLE
Sucessfully registered SPPLE Service.
SPP+LE>AdvertiseLE 1
GAP_LE_Advertising_Enable success.
```

Now that SPPLEDemo is advertising, observe the device shown in LightBlue:



Next select the SPPLEDemo device in LightBlue, after doing so, observe the following screen:


●●○○○ Verizon
1:42 PM
100%
LightBlue
Peripheral
Clone
SPPLEDemo
UUID: A2FC0D75-082C-0F4C-C04D-B85EAF42EB16
Connected
ADVERTISEMENT DATA
Show
UUID: 14839AC4-7D…167340CF2339
0x0734594A-A8E7-4B1A-A6B1-CD5243059A57
Properties: Notify
0xBA04C4B2-892B-43BE-B69C-5D13F2195392
Properties: Read Write
0x8B00ACE7-EB0B-49B0-BBE9-9AEE0A26E1A3
Properties: Write Without Response
0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4
Properties: Read Notify
Log

In the SPPLEDemo terminal, observe the following:

```
etLE_Connection_Complete with size 16.
Status: 0x00.
Role: Slave.
Address Type: Random.
BD_ADDR: 0x5cfc3252180b.
SPP+LE>
etGATT_Connection_Device_Connection with size 16:
Connection ID: 2.
Connection Type: LE.
Remote Device: 0x5cfc3252180b.
Connection MTU: 23.
```

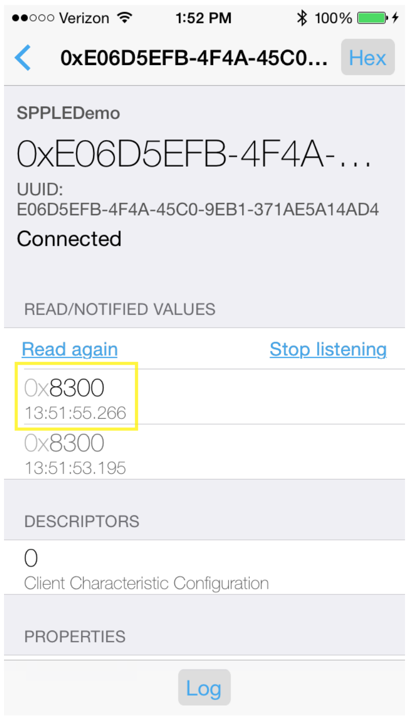
The devices are now connected.

5.2 Enabling Notifications

To enable notifications on the Tx Characteristic and Rx Credits Characteristic in LightBlue do the following:

1. Open the Tx Characteristic (0x0734594A-A8E7-4B1A-A6B1-CD5243059A57).
2. Choose listen for notifications.
3. Press the back button in the top left corner.
4. Open the Rx Credits Characteristic (0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4).
5. Choose listen for notifications.
6. Press the back button in the top left corner.

Observe that after enabling notifications on the Rx Credits Characteristic (0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4) that SPPLEDemo sends initial credits to LightBlue and observe the 0x8300 displayed twice in the app:



SPPLEDemo

0xE06D5EFB-4F4A-...

UUID:
E06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4

Connected

0x8300
13:51:55.266

0x8300
13:51:53.195

0
Client Characteristic Configuration

Note

The first instance of 0x8300 is seen because LightBlue read the characteristic automatically when the connection was first established.

Note

The data here is displayed in little-endian byte order, the actual number of credits is 0083 in hexadecimal, 131 in decimal.

5.3 Sending Data from LightBlue/Receiving Data in SPPLEDemo

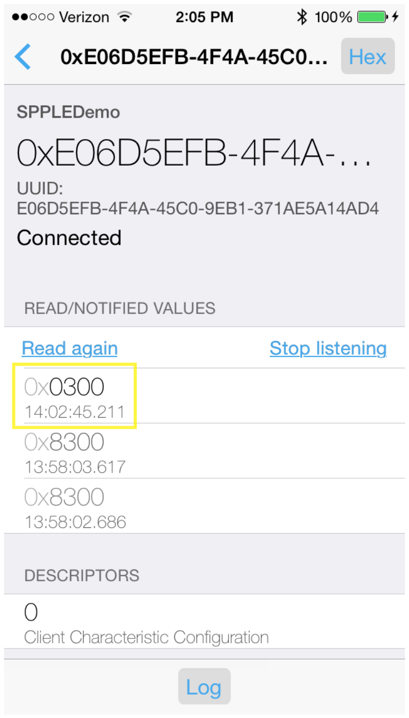
At this point the client (LightBlue) can send data to the server (SPPLEDemo). To send data from LightBlue to SPPLEDemo do the following:

1. Open the Rx Characteristic (0x8B00ACE7-EB0B-49B0-BBE9-9AEE0A26E1A3).
2. Choose write new value.
3. Type 414243(ABC in ASCII).
4. Choose Done.

In the SPPLEDemo terminal, observe a data indication event. To read the data run the LERead 5cfc3252180b command, obeserve the following in the terminal:

```
Data Indication Event, Connection ID 1, Received 3 bytes.
SPP+LE>LERead 5cfc3252180b
Read: 3.
ABC
```

Open the Rx Credits Characteristic (0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4) observe that SPPLEDemo has credited LightBlue with 3 more credits:


●●○○○ Verizon 2:05 PM 100%

0xE06D5EFB-4F4A-45C0... Hex

SPPLEDemo

0xE06D5EFB-4F4A-...

UUID:
E06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4

Connected

READ/NOTIFIED VALUES

Read again Stop listening

0x0300
14:02:45.211

0x8300
13:58:03.617

0x8300
13:58:02.686

DESCRIPTORS

0
Client Characteristic Configuration

Log

5.4 Sending Data from SPPLEDemo/Receiving Data in LightBlue

Send data from SPPLEDemo to LightBlue. First LightBlue needs to provide SPPLEDemo with transmit credits. To provide SPPLEDemo with transmit credits do the following in LightBlue:

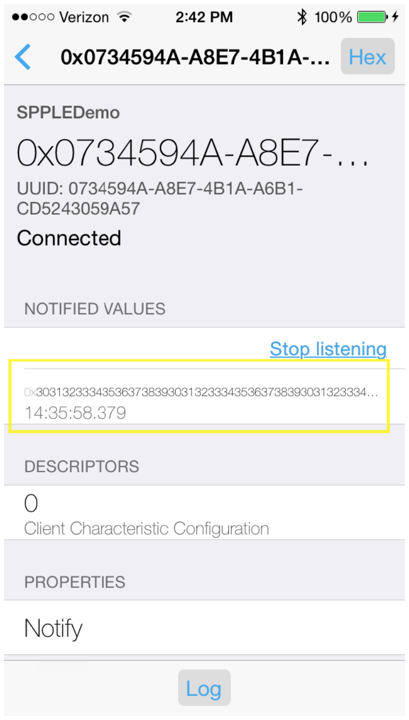
1. Open the Tx Credits Characteristic (0xBA04C4B2-892B-43BE-B69C-5D13F2195392).
2. Choose "Write new value".
3. Type 6400. (100 credits = 0x0064 little-endian).
4. Choose done.
5. Press the back button in the top left corner.

The SPPLEDemo has 100 credits and can send data in SPPLEDemo using the LESend 5cfc3252180b 100 command. Observe the following in the terminal:

```
SPP+LE>LESend 5cfc3252180b 100
Send Complete, Sent 100.
```

To check that LightBlue received the data:

1. Open the Tx Characteristic (0x0734594A-A8E7-4B1A-A6B1-CD5243059A57).
2. Observe a long 0x30313233... string of the received data in the list of NOTIFIED VALUES as seen below:



●●○○○ Verizon 2:42 PM 100%

0x0734594A-A8E7-4B1A-... Hex

SPPLEDemo

0x0734594A-A8E7-...

UUID: 0734594A-A8E7-4B1A-A6B1-CD5243059A57

Connected

NOTIFIED VALUES

Stop listening

0x3031323334353637383930313233343536373839303132333 4...
14:35:58.379

DESCRIPTORS

0
Client Characteristic Configuration

PROPERTIES

Notify

Log

LightBlue has received the data and needs to return the transmit credits to SPPLEDemo. This can be done by repeating the sequence above and re-writing 0x6400 to the TxCredits Characteristic (0xBA04C4B2-892B-43BE-B69C-5D13F2195392).

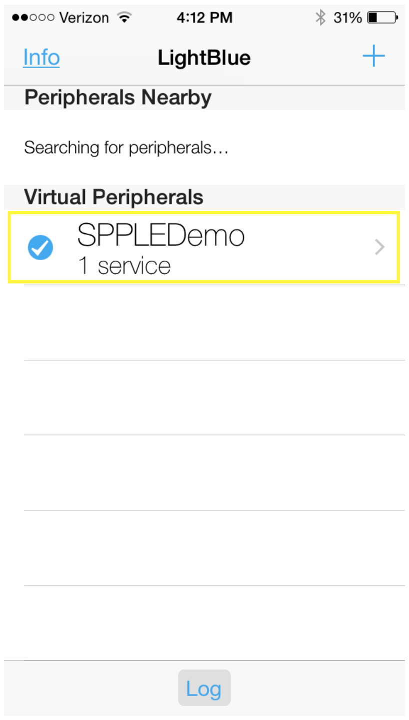
6 LightBlue as the Server/SPPLEDemo as the Client

Note

LightBlue in the server role does not support displaying the updated value of a characteristic even when written to. Therefore LightBlue is not able to send data from LightBlue toSPPLEDemo, SPPLEDemo is able to send data to LightBlue, but that data is not displayed in the app. This is a limitation of LightBlue.

6.1 Connecting the Devices

The first step to connecting the devices is to add the SPP LE Service and characteristics to LightBlue. To do this manually, create a blank virtual peripheral in LightBlue and then add the necessary service and characteristics. Another option is to clone SPPLEDemo while SPPLEDemo acts as the server. To clone SPPLEDemo first connect the 2 devices as described above. After the 2 devices are connected, choose the Clone option in the top right corner of the display. The app returns to the devices list and observe the SPPLEDemo listed as a Virtual Peripheral as seen below:



Note

Make sure that the check box to the left of SPPLEDemo is checked, as seen in the image. If not checked, the iDevice is not advertising and SPPLEDemo cannot be able to connect.

After cloning, the SPP LE service can now connect with the devices. Next, restart SPPLEDemo and when prompted start the app as a client. Next scan for the iOS device using the StartScanning command. When the iOS device has been found stop the scan using the StopScanning command. Now connect to the iOS device using the ConnectLE 5c75524c733a 1 command. After this, run the DiscoverSPPLE 5c75524c733a command within the 10 second timeframe. After the SPP LE service discovery completes, run the ConfigureSPPLE 5c75524c733a within the 25 second timeframe. The iOS device disconnects from SPPLEDemo if the commands are not run within these timeframes. After the SPP LE characteristics are configured the 2 apps stay connected, however, note that if the iOS device goes to sleep this closes the connection. After running the commands just described, observe output similar to the following in SPPLEDemo's terminal:

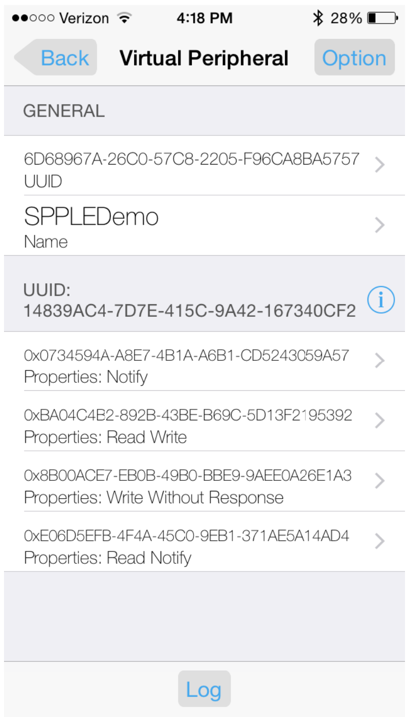
```
OpenStack().
Bluetooth Stack ID: 1.
Device Chipset: 4.1.
BD_ADDR: 0xd03972cdab68

 Command Options: Server, Client, Help

SPP+LE>Client

 Command Options General: Help, GetLocalAddress, SetBaudRate
 Quit,
 Command Options BR/EDR: Inquiry, DisplayInquiryList, Pair,
 EndPairing, PINCodeResponse,
 PassKeyResponse,
 UserConfirmationResponse,
 SetDiscoverabilityMode,
 SetConnectabilityMode,
 SetPairabilityMode,
 ChangeSimplePairingParameters,
 GetLocalName, SetLocalName,
 GetClassOfDevice, SetClassOfDevice,
 GetRemoteName, SniffMode,
 ExitSniffMode, Open, Close, Read,
 Write, GetConfigParams,
 SetConfigParams, GetQueueParams,
 DisplayRawModeData, AutomaticReadMode,
 SetQueueParams, Loopback,
 CBSend.
 Command Options GAPLE: SetDiscoverabilityMode,
 SetConnectabilityMode,
 SetPairabilityMode,
 ChangePairingParameters,
 AdvertiseLE, StartScanning,
 StopScanning, ConnectLE,
 DisconnectLE, PairLE,
 LEPasskeyResponse,
 QueryEncryptionMode, SetPasskey,
 DiscoverGAPS, GetLocalName,
 SetLocalName, GetLERemoteName,
 SetLocalAppearance,
 GetLocalAppearance,
 GetRemoteAppearance,
 Command Options SPPLE: DiscoverSPPLE, RegisterSPPLE, LESend,
 ConfigureSPPLE, LERead, Loopback,
 DisplayRawModeData, AutomaticReadMode

SPP+LE>StartScanning
Scan started successfully.
SPP+LE>
etLE_Advertising_Report with size 36.
1 Responses.
Advertising Type: rtConnectableUndirected.
Address Type: atRandom.
Address: 0x5c75524c733a.
RSSI: -71.
Data Length: 21.
AD Type: 0x01.
AD Length: 0x01.
AD Data: 0x1a
AD Type: 0x07.
AD Length: 0x10.
AD Data: 0x39 0x23 0xcf 0x40 0x73 0x16 0x42 0x9a 0x5c 0x41 0x7e 0x7d 0xc4 0x9a 0x83 0x14
SPP+LE>
etLE_Advertising_Report with size 36.
1 Responses.
Advertising Type: rtScanResponse.
Address Type: atRandom.
Address: 0x5c75524c733a.
RSSI: -71.
Data Length: 11.
AD Type: 0x09.
AD Length: 0x09.
AD Data: 0x53 0x50 0x50 0x4c 0x45 0x44 0x65 0x6d 0x6f
SPP+LE>StopScanning
Scan stopped successfully.
SPP+LE>ConnectLE 5c75524c733a 1
Connection Request successful.
SPP+LE>
```

```
etLE_Connection_Complete with size 16.
Status: 0x00.
Role: Master.
Address Type: Random.
BD_ADDR: 0x5c75524c733a.
SPP+LE>
etGATT_Connection_Device_Connection with size 16:
Connection ID: 1.
Connection Type: LE.
Remote Device: 0x5c75524c733a.
Connection MTU: 23.
SPP+LE>
Exchange MTU Response.
Connection ID: 1.
Transaction ID: 1.
Connection Type: LE.
BD_ADDR: 0x5c75524c733a.
MTU: 131.
SPP+LE>
SPP+LE>DiscoverSPPLE 5c75524c733a
GATT_Start_Service_Discovery success.
SPP+LE>
Service 0x000f - 0x001b, UUID: 14839ac47d7e415c9a42167340cf2339.
SPP+LE>
Service Discovery Operation Complete, Status 0x00.
SPP+LE>ConfigureSPPLE 5c75524c733a
SPPLE Service found on remote device, attempting to read Transmit Credits, and configured CCCDs.
SPP+LE>
Write Response.
Connection ID: 1.
Transaction ID: 15.
Connection Type: LE.
BD_ADDR: 0x5c75524c733a.
Bytes Written: 2.
SPP+LE>
Write Response.
Connection ID: 1.
Transaction ID: 16.
Connection Type: LE.
BD_ADDR: 0x5c75524c733a.
Bytes Written: 2.
```

Note

When SPPLEDemo acts as the server the user must manually enable notifications with the LightBlue app, however, SPPLEDemo handles enabling notifications automatically when the ConfigureSPPLE command is run.

Now that the 2 devices are connected and configured the devices can send and receive data between them. Now select the SPPLEDemo Virtual Peripheral in LightBlue to see the virtual peripheral's characteristics. Observe the following or similar on the iDevice's display:


●●○○○ Verizon 4:18 PM 28%

Back **Virtual Peripheral** Option

GENERAL

6D68967A-26C0-57C8-2205-F96CA8BA5757
UUID

SPPLEDemo
Name

UUID:
14839AC4-7D7E-415C-9A42-167340CF2

0x0734594A-A8E7-4B1A-A6B1-CD5243059A57
Properties: Notify

0xBA04C4B2-892B-43BE-B69C-5D13F2195392
Properties: Read Write

0x8B00ACE7-EB0B-49B0-BBE9-9AEE0A26E1A3
Properties: Write Without Response

0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4
Properties: Read Notify

Log

6.2 Sending Data from LightBlue/Receiving Data in SPPLEDemo

To confirm that SPPLEDemo has provided LightBlue with the transmit credits after the ConfigureSPPLE has finished running, open the open the SPPLEDemo Virtual Peripheral and choose the Credits Characteristic (0xBA04C4B2-892B-43BE-B69C-5D13F2195392). As mentioned above, LightBlue does not show updated values of characteristics when these are written and the user has no way to confirm that LightBlue received the data. Even though there is no confirmation that LightBlue has received transmit credits, data can still be sent from LightBlue to SPPLEDemo because LightBlue is primarily only a GATT Profile demonstration and doesn't have any knowledge of the SPP LE protocol used. LightBlue is unaware of the transmit credits present, and, for this reason, data can be sent from LightBlue to SPPLEDemo with or without transmit credits. To send data to SPPLEDemo, use the Tx Characteristic (0x0734594A-A8E7-4B1A-A6B1-CD5243059A57') and do the following in LightBlue:

1. Open the Tx Characteristic and choose the No value/hex option.
2. Type in 414243.
3. Choose Done.

In SPPLEDemo observe a data indication. To read the data use theLERead 5c75524c733a command. Observe the ABC displayed in the terminal, as seen below:

```
Data Indication Event, Connection ID 1, Received 3 bytes.
SPP+LE>LERead 5c75524c733a
Read: 3.
ABC
```

6.3 Sending Data from SPPLEDemo/Receiving Data in LightBlue

Note

As previously mentioned, LightBlue does not support showing the updated value of a characteristic when written. There is currently no way to confirm that LightBlue has received the data.

To send data from SPPLEDemo, LightBlue must first provide the credits. This can be done using the following in LightBlue:

- Open the Rx Credits Characteristic (0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4).
- Type in 6400. (100 credits = 0x0064 little-endian)
- Choose Done.

SPPLEDemo now has 100 transmit credits. Next, to send data in SPPLEDemo use the LESend 5c75524c733a 100 command. Observe the following in the terminal.

```
SPP+LE>LESend 5c75524c733a 100
Send Complete, Sent 100.
```

7 Application Commands

TI’s Bluetooth stack is implementation of the upper layers of the Bluetooth protocol stack. TI’s Bluetooth stack provides a robust and flexible software development tool that implements the Bluetooth Protocols and Profiles above the Host Controller Interface (HCI). TI’s Bluetooth stack's Application Programming Interface (API) provides access to the upper-layer protocols and profiles and can interface directly with the Bluetooth chips.

The basic bluetooth application included with MSP-EXP430F5438, Tiva DK-TM4C129X, MSP432and STM32F4 is a Serial Port Profile Application.

Also view the pages TI Dual-Mode Bluetooth® Stack on MSP432™ MCU and Dual-Mode Bluetooth® Stack on STM32F4 MCUs.

This page describes the various commands that a user of the application can use. Each command is a wrapper over a TI’s Bluetooth stack API which gets invoked with the parameters selected by the user. This is a subset of the APIs available to the user. TI’s Bluetooth stack API documentation describes all of the API's in detail.

8 General Commands

8.1 Help (DisplayHelp)

Description

The Help command is responsible for displaying the current Command Options for either Serial Port Client or Serial Port Server. The input parameter to this command is completely ignored, and only needs to be passed in because all Commands that can be entered at the prompt pass in the parsed information. This command displays the current command options that are available and always returns zero.

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of the command.

Possible Return Values

This command always returns 0.

8.2 Get Local Address

Description

The GetLocalAddress command is responsible for querying the Bluetooth Device Address of the local Bluetooth Device. This function returns zero on a successful execution and a negative value on all errors. A Bluetooth Stack ID must exist before attempting to call this command.

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of the Query.

Possible Return Values

- (0) Successfully Query Local Address
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR

API Call

GAP_Query_Local_BD_ADDR(BluetoothStackID, &BD_ADDR)

API Prototype

*int BTPSAPI GAP_Query_Local_BD_ADDR(unsigned int BluetoothStackID, BD_ADDR_t *BD_ADDR)*

Description of API

This function is responsible for querying (and reporting) the device address of the local Bluetooth device. The second parameter is a pointer to a buffer that is to receive the device address of the local Bluetooth device. If this function is successful, the buffer that the BD_ADDR parameter points to is filled with the device address read from the local Bluetooth device. If this function returns a negative value, then the device address of the local Bluetooth device was NOT able to be queried (error condition).

8.3 Set Baud Rate

Description

The SetBaudRate command is responsible for changing the current baud rate used to talk to the radio. This function ONLY configures the baud rate for a TI Bluetooth chipset. This command requires that a valid Bluetooth Stack ID exists.

Parameters

This command requires one parameter. The value is an integer representing a value used for the baud rate. The options are 0 (for Baud Rate of 115200), 1 (for Baud Rate 230400), 2 (for Baud Rate 460800), 3 (for Baud Rate 921600), 4 (for Baud Rate 1843200), or 5 (for Baud Rate 3686400). The maximum baud rate default is 921600 so options 4 and 5 are disabled.

Command Call Examples

- "SetBaudRate 0" Attempts to set the baud rate to 115200.
- "SetBaudRate 1" Attempts to set the baud rate to 230400.
- "SetBaudRate 2" Attempts to set the baud rate to 460800.
- "SetBaudRate 3" Attempts to set the baud rate to 921600.

Possible Return Values

- (0) Successfully Set Baud Rate
- (-4) FUNCTION_ERROR
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

API Call

HCI_Reconfigure_Driver(BluetoothStackID, FALSE, &(Data.DriverReconfigureData))

API Prototype

*int BTPSAPI HCI_Reconfigure_Driver(unsigned int BluetoothStackID,Boolean_t ResetStateMachines,HCI_Driver_Reconfigure_Data_t *DriverReconfigureData)*

Description of API

This function issues the appropriate call to an HCI driver to request the HCI Driver to reconfigure itself with the corresponding configuration information.

8.4 Quit

Use this command to return to the intial command screen.

9 BR/EDR Commands

For BR/EDR Commands refer to the document SPP Profile (http://processors.wiki.ti.com/index.php/CC256x_MSP430_TI's_Bluetooth_Stack_Basic_SPPDemo_APP) sections GenericAccess Profile Commands and Serial Port Profile Commands.

10 GAPLE Commands

The Generic Access Profile defines standard procedures related to the discovery and connection of Bluetooth devices. This defines modes of operation that are generic to all devices and allows for procedures which use those modes to decide how a device can interact with other Bluetooth devices. Discoverability, Connectability, Pairability, Bondable Modes, and Security Modes can all be changed using Generic Access Profile procedures. All of these modes affect the interaction two devices can have with one another. GAP also defines the procedures for how to bond two Bluetooth devices.

10.1 Set Discoverability Mode

Description

The SetDiscoverabilityMode command is responsible for setting the Discoverability Mode of the local device. This command returns zero on successful execution and a negative value on all errors. The Discoverability Mode in LE is only applicable when advertising. If a device is not advertising, then the device is not discoverable. The value set by this command is used as a parameter in the command AdvertiseLE.

Parameters

This command requires only one parameter which is an integer value that represents a Discoverability Mode. This value must be specified as 0 (for Non-Discoverable Mode), 1 (forLimited Discoverable Mode), or 2 (for General Discoverable Mode).

Command Call Examples

- "SetDiscoverabilityMode 0" Attempts to change the Discoverability Mode of the local device to Non-Discoverable.
- "SetDiscoverabilityMode 1" Attempts to change the DiscoverabilityMode of the local device to Limited Discoverable.
- "SetDiscoverabilityMode 2" Attempts to change the Discoverability Mode of the local device to General Discoverable.

Possible Return Values

- (0) Successfully Set Discoverability Mode Parameter
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR

10.2 Set Connectability Mode

Description

The SetConnectabilityMode command is responsible for setting the Connectability Mode of the local device. This command returns zero on successful execution and a negative value on all errors. The Connectability Mode in LE is only applicable when advertising. If a device is not advertising, then the device is not connectable. The value set by this command is used as a parameter in the command AdvertiseLE.

Parameters

This command requires only one parameter which is an integer value that represents a Connectability Mode. This value must be specified as 0 (for Non-Connectable) or 1 (for Connectable).

Command Call Examples

- "SetConnectabilityMode 0" Attempts to set the local device’s Connectability Mode to Non-Connectable.
- "SetConnectabilityMode 1" Attempts to set the local device’s ConnectabilityMode to Connectable.

Possible Return Values

- (0) Successfully Set Connectability Mode Paramet

- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR

10.3 Set Pairability Mode

Description

The SetPairabilityMode command is responsible for setting the pairability mode of the local device. This command returns zero on successful execution and a negative value on all errors.

Parameters

This command requires only one parameter which is an integer value that represents a pairability mode. This value must be specified as 0 (for Non-Pairable), 1 (for Pairable) or 2 (for Pairable with Secure Simple Pairing).

Command Call Examples

- "SetPairabilityMode 0" Attempts to set the Local Device's Pairability Mode to Non-Pairable.
- "SetPairabilityMode 1" Attempts to set the Local Device's Pairability Mode to Pairable.

Possible Return Values

- (0) Successfully Set Pairability Mode
- (-4) FUNCTION_ERROR
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR

API Call

GAP_LE_Set_Pairability_Mode(BluetoothStackID, PairabilityMode)

API Prototype

int BTPSAPI GAP_LE_Set_Pairability_Mode(unsigned int BluetoothStackID, GAP_LE_Pairability_Mode_t PairableMode)

Description of API

This function is provided to allow the local host the ability to change the pairability mode used by the local host. This function returns zero if successful or a negative return error code if there was an error condition.

10.4 Change Pairing Parameters

Description

The ChangePairingParameters command is responsible for changing the LE Pairing Parameters that are exchanged during the Pairing procedure. This command returns zero on successful execution and a negative value on all errors.

Parameters

This command requires five parameters which are the I/O Capability, the Bonding Type, the MITM Requirement, the SC Enable, and the P256 debug mode:

1. The first parameter must be specified as 0 (for Display Only), 1 (for Display Yes/No), 2 (for Keyboard Only), 3 (for No Input/Output) or 4 (for Keyboard/Display).
2. The second parameter must be specified as 0 (for No Bonding) or 1 (for Bonding), when at least one of the devices is set to No Bonding, the LTK won't be stored.
3. The third parameter must be specified as 0 (for No MITM) or 1 (for MITM required).
4. The fourth parameter must be specified as 0 (for SC disabled) or 1 (for SC enabled), legacy pairing procedure takes place when using SC disable.

5. The fifth parameter must be specified as 0 (for Debug Mode disabled) or 1 (for P256 debug mode enabled), but only when using SC pairing. P256 debug mode is relevant when set, the values of the P256 private and public keys are pre-defined according to the Bluetooth specification instead of random.

Command Call Examples

- "ChangeSimplePairingParameters 3 0 0 0 0" Attempts to set the I/O Capability to No Input/Output, Bonding Type set to No Bonding, turns off MITM Protection, disable secure connections and disable debug mode.
- "ChangeSimplePairingParameters 2 0 1 1 0 " Attempts to set the I/O Capability to Keyboard Only, Bonding Type set to No Bonding, activates MITM Protection, enabling secure connections and disable debug mode.
- "ChangeSimplePairingParameters 1 1 1 1 1" Attempts to set the I/O Capability to Display Yes/No, bonding type set to Bonding, activates MITM Protection, enabling secure connections and enabling debug mode.

Possible Return Values

- (0) Successfully Set Pairability Mode
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR

10.5 Advertise LE

Description

The AdvertiseLE command is responsible for enabling LE advertisements. This command returns zero on successful execution and a negative value on all errors.

Parameters

The only parameter necessary decides whether advertising reports are sent or are disabled. To disable, use 0 as the first parameter, to enable, use 1 instead.

Command Call Examples

- “AdvertiseLE 1” Attempts to enable Low Energy Advertising on the local Bluetooth device.
- “AdvertiseLE 0” Attempts to disable Low Energy Advertising on the local Bluetooth device.

Possible Return Values

- (0) Successfully Set Pairability Mode
- (-4) FUNCTION_ERROR
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR

API Calls

Depending on the First Parameter Value:

- *GAP_LE_Advertising_Disable(BluetoothStackID)*
- *GAP_LE_Set_Advertising_Data(BluetoothStackID, (Advertisement_Data_Buffer.AdvertisingData.Advertising_Data[0] + 1), &(Advertisement_Data_Buffer.AdvertisingData))*
- *GAP_LE_Set_Scan_Response_Data(BluetoothStackID, (Advertisement_Data_Buffer.ScanResponseData.Scan_Response_Data[0] + 1), &(Advertisement_Data_Buffer.ScanResponseData))*
- *GAP_LE_Advertising_Enable(BluetoothStackID, TRUE, &AdvertisingParameters, &ConnectabilityParameters, GAP_LE_Event_Callback, 0)*

API Prototypes

- *int BTPSAPI GAP_LE_Advertising_Disable(unsigned int BluetoothStackID)*
- *int BTPSAPI GAP_LE_Set_Advertising_Data(unsigned int BluetoothStackID, unsigned int Length, Advertising_Data_t *Advertising_Data)*
- *int BTPSAPI GAP_LE_Set_Scan_Response_Data(unsigned int BluetoothStackID, unsigned int Length, Scan_Response_Data_t *Scan_Response_Data)*
- *int BTPSAPI GAP_LE_Set_Advertising_Data(unsigned int BluetoothStackID, unsigned int Length, Advertising_Data_t *Advertising_Data)*
- *int BTPSAPI GAP_LE_Set_Advertising_Data(unsigned int BluetoothStackID, unsigned int Length, Advertising_Data_t *Advertising_Data)*

Description of API

- The GAP_LE_Advertising_Disable function is provided to allow the local host the ability to cancel (stop) an on-going advertising procedure. This function returns zero if successful or a negative return error code if there was an error condition.
- The GAP_LE_Set_Advertising_Data is provided to allow the local host the ability to set the advertising data that is used during the advertising procedure (started via the GAP_LE_Advertising_Enable function). This function returns zero if successful or a negative return error code if there was an error condition.
- The GAP_LE_Set_Scan_Response_Data function is provided to allow the local host the ability to set the advertising data that is used during the advertising procedure (started via the GAP_LE_Advertising_Enable function). This function returns zero if successful or a negative return error code if there was an error condition.
- TheGAP_LE_Set_Advertising_Data function is provided to allow the local host the ability to set the advertising data that is used during the advertising procedure (started via theGAP_LE_Advertising_Enable function). This function returns zero if successful or a negative return error code if there was an error condition.

10.6 Start Scanning

Description

The StartScanning command is responsible for starting an LE scan procedure. This command returns zero if successful and a negative value if an error occurred. This command calls the StartScan (unsigned in BluetoothStackID) function which performs the scan.

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of the scan.

Possible Return Values

- (0) Successfully started the LE Scan Procedure
- (-1) Bluetooth Stack ID is Invalid during the StartScan() call
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-66) BTPS_ERROR_INSUFFICIENT_RESOURCES
- (-105) BTPS_ERROR_SCAN_ACTIVE
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR

API Call

GAP_LE_Perform_Scan(BluetoothStackID, stActive, 10, 10, latPublic, fpNoFilter, TRUE, GAP_LE_Event_Callback, 0)

API Prototype

int BTPSAPI GAP_LE_Perform_Scan(unsigned int BluetoothStackID, GAP_LE_Scan_Type_t ScanType, unsigned int ScanInterval, unsigned int ScanWindow,GAP_LE_Address_Type_t LocalAddressType, GAP_LE_Filter_Policy_t FilterPolicy, Boolean_t FilterDuplicates, GAP_LE_Event_Callback_t GAP_LE_Event_Callback, unsignedlong CallbackParameter)

Description of API

The GAP_LE_Perform_Scan function is provided to allow the local host the ability to begin an LE scanning procedure. This procedure is similar in concept to the inquiry procedure in Bluetooth BR/EDR as this can be used to discover devices that have been instructed to advertise. This function returns zero if successful or a negative return error code if there is an error condition.

10.7 Stop Scanning

Description

The StopScanning command is responsible for stopping an LE scan procedure. This command returns zero if successful and a negative value if an error occurred. This command calls the StopScan (unsigned in BluetoothStackID) function which performs the scan.

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of the command.

Possible Return Values

- (0) Successfully stopped the LE Scan Procedure
- (-1) Bluetooth Stack ID is Invalid during the StartScan() call
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR

API Call

GAP_LE_Cancel_Scan(BluetoothStackID)

API Prototype

int BTPSAPI GAP_LE_Cancel_Scan(unsigned int BluetoothStackID)

Description of API

The GAP_LE_Cancel_Scan function is provided to allow the local host the ability to cancel (stop) an on-going scan procedure. This function returns zero if successful or a negative return error code if there is an error condition.

10.8 Connect LE

Description

The ConnectLE command is responsible for connecting to an LE device. This command returns zero if successful and a negative value if an error occurred. This command calls the ConnectLEDevice (unsigned in BluetoothStackID, BD_ADDR_t BD_ADDR, Boolean_t UseWhiteList) function using ConnectLEDevice (BluetoothStackID, BD_ADDR, FALSE).

Parameters

The only parameter required is the Bluetooth Address of the remote device. This can easily be found using the StartScanning command if the advertising device is in proximity during the scan.

Command Call Examples

- “ConnectLE 001bdc05b617” Attempts to send a connection request to the Bluetooth device with the BD_ADDR of 001bdc05b617.
- “ConnectLE 000275e126FF” Attempts to send a connection request to the Bluetooth device with the BD_ADDR of 000275e126FF.

Possible Return Values

- (0) Successfully Set Pairability Mode
- (-4) FUNCTION_ERROR
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-116) BTPS_ERROR_RANDOM_ADDRESS_IN_USE
- (-111) BTPS_ERROR_CREATE_CONNECTION_OUTSTANDING
- (-66) BTPS_ERROR_INSUFFICIENT_RESOURCES
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR
- GAP_LE_ERROR_WHITE_LIST_IN_USE

API Calls

- *GAP_LE_Create_Connection(BluetoothStackID, 100, 100, Result?fpNoFilter:fpWhiteList, latPublic, Result?&BD_ADDR:NULL, latPublic, &ConnectionParameters,GAP_LE_Event_Callback, 0)*
- *GAP_LE_Remove_Device_From_White_List(BluetoothStackID, 1, &WhiteListEntry, &WhiteListChanged)*
- *GAP_LE_Add_Device_To_White_List(BluetoothStackID, 1, &WhiteListEntry, &WhiteListChanged)*

Note

These two APIs can generally be ignored unless the WhiteList is enabled in the call to ConnectLEDevice.

API Prototypes

- *int BTPSAPI GAP_LE_Create_Connection(unsigned int BluetoothStackID, unsigned int ScanInterval, unsigned int ScanWindow, GAP_LE_Filter_Policy_t InitatorFilterPolicy,GAP_LE_Address_Type_t RemoteAddressType, BD_ADDR_t *RemoteDevice, GAP_LE_Address_Type_t LocalAddressType, GAP_LE_Connection_Parameters_t*ConnectionParameters, GAP_LE_Event_Callback_t GAP_LE_Event_Callback, unsigned long CallbackParameter)*
- *int BTPSAPI GAP_LE_Remove_Device_From_White_List(unsigned int BluetoothStackID, unsigned int DeviceCount, GAP_LE_White_List_Entry_t *WhiteListEntries, unsigned int*RemovedDeviceCount)*
- *int BTPSAPI GAP_LE_Add_Device_To_White_List(unsigned int BluetoothStackID, unsigned int DeviceCount, GAP_LE_White_List_Entry_t *WhiteListEntries, unsigned int*AddedDeviceCount)*

Description of API

The GAP_LE_Create_Connection function is provided to allow the local host the ability to create a connection to a remote device using the Bluetooth LE radio. The connection process is asynchronous in nature and the caller is notified via the GAP LE event callback function (specified in this function) when the connection completes. This function returns zero if successful, or a negative return error code if there is an error condition.

The GAP_LE_Remove_Device_From_White_List function is provided to allow the local host the ability to remove one (or more) devices from the white list maintained by the local device. This function attempts to delete as many devices as possible (from the specified list) and returns the number of devices deleted. The GAP_LE_Read_White_List_Size function can be used to determine how many devices the local device supports in the white list (simultaneously).

10.9 Disconnect LE

Description

The DisconnectLE command is responsible for disconnecting from an LE device. This command returns zero on successful execution and a negative value on all errors. This command requires that a valid Bluetooth Stack ID exists before running.

Parameters

The only parameter required is the Bluetooth Address of the remote device that is connected.

Command Call Examples

- “DisconnectLE 001bdc05b617” Attempts to send a disconnection request to the Bluetooth Device with the BD_ADDR of 001bdc05b617.
- “DisconnectLE 000275e126FF” Attempts to send a disconnection request to the Bluetooth Device with the BD_ADDR of 000275e126FF.

Possible Return Values

- (0) Successfully disconnected remote device
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR

API Call

GAP_LE_Disconnect(BluetoothStackID, BD_ADDR)

API Prototype

int BTPSAPI GAP_LE_Disconnect(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR)

API Description

The GAP_LE_Disconnect function provides the ability to disconnect from a remote device. This function returns zero if successful or a negative return error code if there is an error condition.

10.10 Pair LE

Description

The PairLE command is provided to allow a mechanism of Pairing (or requesting security if a slave) to the connected device. This command calls the SendPairingRequest (BD_ADDR_tBD_ADDR, Boolean_t ConnectionMaster) function using SendPairingRequest (ConnectionBD_ADDR, LocalDeviceIsMaster).

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of the command.

Possible Return Values

- (0) Successfully Set Pairability Mode
- (-4) FUNCTION_ERROR

- (-6) INVALID_PARAMETERS_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
- (-66) BTPS_ERROR_INSUFFICIENT_RESOURCES
- (-107) BTPS_ERROR_INVALID_DEVICE_ROLE_MODE

API Calls

- *GAP_LE_Pair_Remote_Device(BluetoothStackID, BD_ADDR, &Capabilities, GAP_LE_Event_Callback, 0)*
- *GAP_LE_Request_Security(BluetoothStackID, BD_ADDR, Capabilities.Bonding_Type, Capabilities.MITM, GAP_LE_Event_Callback, 0)*

API Prototypes

- *int BTPSAPI GAP_LE_Pair_Remote_Device(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_LE_Pairing_Capabilities_t *Capabilities, GAP_LE_Event_Callback_tGAP_LE_Event_Callback, unsigned long CallbackParameter)*
- *int BTPSAPI GAP_LE_Request_Security(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_LE_Bonding_Type_t Bonding_Type, Boolean_t MITM,GAP_LE_Event_Callback_t GAP_LE_Event_Callback, unsigned long CallbackParameter)*

Description of API

The GAP_LE_Pair_Remote_Device function is provided to allow a means to pair with a remote, connected, device. This function accepts the device address of the currently connected device to pair with, followed by the pairing capabilities of the local device. This function also accepts as input the GAP LE event callback information to use during the pairing process. This function returns zero if successful or a negative error code if there is an error. This function can only be issued by the master of the connection (the initiator of the connection). The reason is that a slave can only request a security procedure, it cannot initiate a security procedure. The GAP_LE_Request_Security function is provided to allow a means for a slave device to request that the master (of the connection) perform a pairing operation or re-establishing prior security. This function can only be called by a slave device. The reason for this is that the slave can only request for security to be initiated, it cannot initiate the security process itself. This function returns zero if successful or a negative error code if there is an error.

10.11 LE Pass Key Response

Description

The LEPassKeyResponse command is responsible for issuing a GAP Authentication Response with a Pass Key value specified via the input parameter. This command returns zero on successful execution and a negative value on all errors.

Parameters

The PassKeyResponse command requires one parameter which is the Pass Key used for authenticating the connection. This is a string value which can be up to 6 digits long (with a valuebetween 0 and 999999).

Command Call Examples

- "PassKeyResponse 1234" Attempts to set the Pass Key to "1234."
- "PassKeyResponse 999999" Attempts to set the Pass Key to "999999."

This value represents the longest Pass Key value of 6 digits.

Possible Return Values

- (0) Successful Pass Key Response
- (-4) FUNCTION_ERROR

- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-118) BTPS_ERROR_PAIRING_NOT_ACTIVE
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
- (-66) BTPS_ERROR_INSUFFICIENT_RESOURCES
- (-107) BTPS_ERROR_INVALID_DEVICE_ROLE_MODE

API Call

GAP_LE_Authentication_Response(BluetoothStackID, CurrentRemoteBD_ADDR, &GAP_LE_Authentication_Response_Information)

API Prototype

*int BTPSAPI GAP_LE_Authentication_Response(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_LE_Authentication_Response_Information_t*GAP_LE_Authentication_Information)*

Description of API

This function is provided to allow a mechanism for the local device to respond to GAP LE authentication events. This function is used to specify the authentication information for the specified Bluetooth device. This function accepts as input, the Bluetooth protocol stack ID of the Bluetooth device that has requested the authentication action, and the authentication response information (specified by the caller).

10.12 LE Query Encryption

Description

The LEQueryEncryption command is responsible for quering the Encryption Mode for an LE Connection. This command returns zero on successful execution and a negative value on all errors.

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of the query.

Possible Return Values

- (0) Successfully Queried Encryption Mode
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE

API Call

GAP_LE_Query_Encryption_Mode(BluetoothStackID, ConnectionBD_ADDR, &GAP_Encryption_Mode)

API Prototype

*int BTPSAPI GAP_LE_Query_Encryption_Mode(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Encryption_Mode_t *GAP_Encryption_Mode)*

Description of API

This function is provided to allow a means to query the current encryption mode for the LE connection that is specified.

10.13 Set Passkey

Description

The SetPasskey command is responsible for querying the encryption mode for an LE Connection. This command returns zero on successful execution and a negative value on all errors.

Note

SetPasskey Command works only when pairing.

Parameters

The SetPasskey command requires one parameter which is the Pass Key used for authenticating the connection. This is a string value which can be up to 6 digits long (with a value between 0 and 999999).

Command Call Examples

- "SetPasskey 0" Attempts to remove the Passkey.
- "SetPasskey 1 987654" Attempts to set the Passkey to 987654.
- "SetPasskey 1" Attempts to set the Passkey to the default Fixed Passkey value.

Possible Return Values

- (0) Successful Pass Key Response
- (-4) FUNCTION_ERROR
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE

API Calls

One of these is chosen depending on the first parameter:

- *GAP_LE_Set_Fixed_Passkey(BluetoothStackID, &Passkey)*
- *GAP_LE_Set_Fixed_Passkey(BluetoothStackID, NULL)*

API Prototype

*int BTPSAPI GAP_LE_Set_Fixed_Passkey(unsigned int BluetoothStackID, DWord_t *Fixed_Display_Passkey)*

Description of API

This function is provided to allow a means for a fixed passkey to be used whenever the local Bluetooth device is chosen to display a passkey during a pairing operation. This fixed passkey is only used when the local Bluetooth device is chosen to display the passkey, based on the remote I/O Capabilities and the local I/O capabilities.

10.14 Discover GAPS

Description

The DiscoverGAPS command is provided to allow an easy mechanism to start a service discovery procedure to discover the Generic Access Profile Service on the connected remote device.

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of the service discovery.

Possible Return Values

- (0) Successfully discovered the Generic Access Profile Service.
- (-4) Function Error (on failure).

API Call

GDIS_Service_Discovery_Start(BluetoothStackID, ConnectionID, (sizeof(UUID)/sizeof(GATT_UUID_t)), UUID, GDIS_Event_Callback, sdGAPS)

API Prototypes

*int BTPSAPI GDIS_Service_Discovery_Start(unsigned int BluetoothStackID, unsigned int ConnectionID, unsigned int NumberOfUUID, GATT_UUID_t *UUIDList,GDIS_Event_Callback_t ServiceDiscoveryCallback, unsigned long ServiceDiscoveryCallbackParameter)*

Description of API

The GDIS_Service_Discover_Start is in an application module called GDIS that is provided to allow an easy way to perform GATT service discovery. This module can be modified for customer use. This function is called to start a service discovery operation by the GDIS module.

10.15 Get Local Name

Description

The GetLocalName command is responsible for querying the name of the local Bluetooth Device. This command returns zero on a successful execution and a negative value on all errors. A Bluetooth Stack ID must exist before attempting to call this command.

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of thequery.

Possible Return Values

- (0) Successfully Queried Local Device Name
- (-8) INVALID_STACK_ID_ERROR
- (-4) FUNCTION_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR
- (-65) BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

API Call

*GAP_Query_Local_Device_Name(BluetoothStackID, 257, (char *)LocalName)*

API Prototype

*int BTPSAPI GAP_Query_Local_Device_Name(unsigned int BluetoothStackID, unsigned int NameBufferLength, char *NameBuffer)*

Description of API

This function is responsible for querying (and reporting) the user friendly name of the local Bluetooth device. The final parameters to this function specify the buffer and the buffer length of the buffer that is to receive the local device name. The NameBufferLength parameter is at least MAX_NAME_LENGTH+1 to hold the maximum allowable device name plus a single character to hold the NULL terminator. If this function is successful, this function returns zero, and the buffer that NameBuffer points to is filled with a NULL terminated ASCII representation of the local device name. If this function returns a negative value, then the local device name was NOT able to be queried (error condition).

10.16 Set Local Name

Description

The SetLocalName command is responsible for setting the name of the local Bluetooth device to a specified name. This command returns zero on a successful execution and a negative value on all errors. A Bluetooth Stack ID must exist before attempting to call this command.

Parameters

One parameter is necessary for this command. The specified device name must be the only parameter (which means there are no spaces in the name. If spaces are in the name, only the first section of the name is set.)

Command Call Examples

- "SetLocalName New_Bluetooth_Device_Name" Attempts to set the local device name to "New_Bluetooth_Device_Name."
- "SetLocalName New Bluetooth Device Name" Attempts toset the local device name to "New Bluetooth Device Name" but only sets the first parameter, which makes the local device name "New."
- "SetLocalName MSP430" Attempts to setthe local device name to "MSP430."

Possible Return Values

- (0) Successfully Set Local Device Name
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-8) INVALID_STACK_ID_ERROR
- (-4) FUNCTION_ERROR
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR

API Call

GAP_Set_Local_Device_Name(BluetoothStackID, TempParam->Params[0].strParam)

API Prototype

*int BTPSAPI GAP_Set_Local_Device_Name(unsigned int BluetoothStackID, char *Name)*

Description of API

This function is provided to allow the changing of the device name of the local Bluetooth device. The name parameter must be a pointer to a NULL terminated ASCII string of at most MAX_NAME_LENGTH (not counting the trailing NULL terminator). This function returns zero if the local device name was successfully changed, or a negative return error code if there is an error condition.

10.17 Get Remote Name

Description

The GetRemoteName command is responsible for querying the Bluetooth Device Name of a Remote Device. This command returns zero on a successful execution and a negative value on all errors. The command requires that a valid Bluetooth Stack ID exists before running and is called after using the Inquiry command. The DisplayInquiryList command is useful in this situation to find which Remote Device goes with which Inquiry Index.

Parameters

The GetRemoteName command requires one parameter which is the Inquiry Index of the Remote Bluetooth Device. This value can be found after an Inquiry or displayed when the command DisplayInquiryList is used. Command Call Examples "GetRemoteName 5" attempts to query the Device Name for the Remote Device that is at the fifth Inquiry Index. "GetRemoteName 8" attempts to query the Device Name for the Remote Device that is at the eighth Inquiry Index.

Possible Return Values

- (0) Successfully Queried Remote Name
- (-6) INVALID_PARAMETERS_ERROR
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-59) BTPS_ERROR_ADDING_CALLBACK_INFORMATION
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR

API Call

GAP_Query_Remote_Device_Name(BluetoothStackID, InquiryResultList[(TempParam->Params[0].intParam – 1)], GAP_Event_Callback, (unsigned long)0)

API Prototype

int BTPSAPI GAP_Query_Remote_Device_Name(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Event_Callback_t GAP_Event_Callback, unsigned longCallbackParameter)

Description of API

This function is provided to allow a mechanism to query the user-friendly Bluetooth device name of the specified remote Bluetooth device. This function accepts as input the Bluetooth device address of the remote Bluetooth device to query the name of and the GAP event callback information that is to be used when the remote device name process has completed. This function returns zero if successful, or a negative return error code if the remote name request was unable to be submitted. If this function returns successful, then the caller is notified via the specified callback when the remote name information has been determined (or if there is an error). This function cannot be used to determine the user-friendly name of the local Bluetooth device. The GAP_Query_Local_Name function is used to query the user-friendly name of the local Bluetooth device. Because this function is asynchronous in nature (specifying a remote device address), this function notifies the caller of the result via the specified callback. The caller is free to cancel the remote name request at any time by issuing the GAP_Cancel_Query_Remote_Name function and specifying the Bluetooth device address of the Bluetooth device that was specified in the original call to this function. When the callback is cancelled, the operation is still attempted and then the callback is cancelled (i.e. the GAP module can still perform the remote name request, but no callback is ever issued).

10.18 LE User Confirmation Response

Description

The LEUserConfirmationResponse command is responsible for issuing a GAP LE Authentication Response with a user confirmation value specified via the input parameter. This function returns zero on successful execution and a negative value on all errors.

Parameters

This command requires one parameter which indicates if confirmation is accepted or not. 0 = decline, 1 = accept.

Command Call Examples

- "LEUserConfirmationResponse 0" Attempts to respond with a decline value.
- "LEUserConfirmationResponse 1" Attempts to respond with a accept value.

Possible Return Values

- (0) Success.
- (-4) FUNCTION_ERROR.
- (-6) INVALID_PARAMETERS_ERROR.
- (-1) BTPS_ERROR_INVALID_PARAMETER.
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID.
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED.
- (-57) BTPS_ERROR_DEVICE_HCI_ERROR.
- (-66) BTPS_ERROR_INSUFFICIENT_RESOURCES.
- (-98) BTPS_ERROR_DEVICE_NOT_CONNECTED.
- (-103) BTPS_ERROR_FEATURE_NOT_AVAILABLE.
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE.
- (-107) BTPS_ERROR_INVALID_DEVICE_ROLE_MODE.
- (-118) BTPS_ERROR_PAIRING_NOT_ACTIVE.
- (-119) BTPS_ERROR_INVALID_STATE.
- (-120) BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE.
- (-122) BTPS_ERROR_NUMERIC_COMPARISON_FAILED.

API Call

GAP_LE_Authentication_Response(BluetoothStackID, CurrentLERemoteBD_ADDR, &GAP_LE_Authentication_Response_Information)

API Prototype

*int BTPSAPI GAP_LE_Authentication_Response(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_LE_Authentication_Response_Information_t*GAP_LE_Authentication_Information)*

Description

The following function is provided to allow a mechanism for the local device to respond to GAP LE authentication events. This function is used to set the authentication information for the specified Bluetooth device. This function accepts as input, the Bluetooth protocol stack ID followed by the remote Bluetooth device address that is currently executing apairing/authentication process, followed by the authentication response information. This function returns zero if successful, or a negative return error code if there is an error.

10.19 Enable SC Only

Description

The EnableSCOnly command enables LE Secure Connections (SC) only mode. In case this mode is enabled, pairing request from peers that support legacy pairing only is rejected. Please note that in case this mode is enabled, the SC flag in the LE_Parameters must be set to TRUE. This function returns zero on successful execution and a negative value on all errors.

Parameters

This command requires one parameter which indicates if Secure connections only mode is set or not. 0 = SC Only mode is off, 1 = SC Only mode is on.

Command Call Examples

- “EnableSCOnly 0” Disable secure connections only mode.
- “EnableSCOnly 1” Enable secure connections only mode.

Possible Return Values

- (0) Success.
- (-4) FUNCTION_ERROR.
- (-6) INVALID_PARAMETERS_ERROR.
- (-8) INVALID_STACK_ID_ERROR.
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID.
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED.
- (-103) BTPS_ERROR_FEATURE_NOT_AVAILABLE.
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE.
- (-120) BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE.

API Call

GAP_LE_SC_Only_Mode(BluetoothStackID, EnableSCOnly)

API Prototype

int BTPSAPI GAP_LE_SC_Only_Mode (unsigned int BluetoothStackID, Boolean_t EnableSCOnly)

Description of API

The following function is provided to allow a configuration of LE Secure Connecions only mode. The upper layer uses this function before the beginning of LE SC pairing, in case the function asks to reject a device that supports only a legacy pairing. This mode can be used when it is more important for a device to have high security than for the device to maintain backwards compatibility with devices that do not support SC. This function accepts as parameters the Bluetooth stack ID of the Bluetooth device, and a boolean EnableSCOnly that enable or disable the SC only mode. This function is used once, before the first pairing process. This function returns zero if successful or a negative error code.

10.20 Regenerate P256 Local Keys

Description

The following function allows the user to generate new P256 private and local keys. This function shall NOT be used in the middle of a pairing process. This is relevant for LE SecureConenctions pairing only. This function returns zero on successful execution and a negative value on all errors.

Parameters

No parameters are necessary.

Command Call Examples

“RegenerateP256LocalKeys” Attempts to generate new P256 private and local keys.

Possible Return Values

- (0) Success.
- (-4) FUNCTION_ERROR.
- (-8) INVALID_STACK_ID_ERROR.
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID.
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED.
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE.
- (-117) BTPS_ERROR_PAIRING_ACTIVE.
- (-120) BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE.

API Call

GAP_LE_SC_Regenerate_P256_Local_Keys (BluetoothStackID)

API Prototype

int BTPSAPI GAP_LE_SC_Regenerate_P256_Local_Keys (unsigned int BluetoothStackID)

Description of API

The following function is provided to allow a regeneration of the P-256 private and local puclic keys. This function is relevant only in case of LE SC pairing. This function accepts as parameters the Bluetooth stack ID of the Bluetooth device. This functions shall NOT be used while performing pairing. This function returns zero if successful or a negative error code.

10.21 SC Generate OOB Local Params

Description

To perform LE SC pairing in OOB method, generate local random and confirmation values before the pairing process starts. The following function allows the user to generate OOB local parameters. This function shall NOT be used in the middle of a pairing process. This is relevant for LE SC pairing only. This function returns zero on successful execution and a negative value on all errors.

Parameters

No parameters are necessary.

Command Call Examples

“SCGenerateOOBLocalParams” Attempts to generate local random and confirmation values before the pairing process starts.

Possible Return Values

- (0) Success.
- (-4) FUNCTION_ERROR.
- (-8) INVALID_STACK_ID_ERROR.
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID.
- (-56) BTPS_ERROR_GAP_NOT_INITIALIZED.
- (-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE.
- (-117) BTPS_ERROR_PAIRING_ACTIVE.
- (-120) BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE.

API Call

GAP_LE_SC_OOB_Generate_Parameters(BluetoothStackID, &OOBLocalRandom, &OOBLocalConfirmation)

API Prototype

*int BTPSAPI GAP_LE_SC_OOB_Generate_Parameters(unsigned int BluetoothStackID, SM_Random_Value_t *OOB_Local_Rand_Result, SM_Confirm_Value_t*OOB_Local_Confirm_Result)*

Description of API

The following function is provided to allow the use of LE Secure Connections (SC) pairing in Out Of Band (OOB) association method. The upper layer uses this function to generate the the local OOB random value, and OOB confirmation value (ra/rb and Ca/Cb) as defined in the Bluetooth specification. This function accepts as parameters the Bluetooth stack ID of the Bluetooth device, and pointers to buffers that recieve the generated local OOB random, and OOB confirmation values. This function returns zero if successful or a negative error code.

10.22 Set Local Appearance

Description

The SetLocalAppearence command is provided to set the local device appearance that is exposed by the GAP Service (GAPS).

Parameters

The one parameter the SetLocalAppearence command requires is the Local Device Appearance.

Possible Return Values

- (0) Success.
- (-4) Function error (on failure).

API Call

GAPS_Set_Device_Appearance(BluetoothStackID, GAPSInstanceID, Appearance)

API Prototype

int BTPSAPI GAPS_Set_Device_Appearance (unsigned int BluetoothStackID, unsigned int InstanceID, Word_t DeviceAppearance)

Description of API

This function allows a mechanism of setting the local device appearance that is exposed as part of the GAP Service API (GAPS).

10.23 Get Local Appearance

Description

The GetLocalAppearence command is provided to read the local device appearance that is exposed by the GAP Service (GAPS).

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome.

Possible Return Values

- (0) Success.
- (-4) Function error (on failure).

API Call

GAPS_Query_Device_Appearance(BluetoothStackID, GAPSInstanceID, &Appearance)

API Prototype

*int BTPSAPI GAPS_Query_Device_Appearance(unsigned int BluetoothStackID, unsigned int InstanceID, Word_t *DeviceAppearance)*

Description of API

This function allows a mechanism of reading the local device appearance that is exposed as part of the GAP Service API (GAPS).

11 SPPLE Commands

11.1 Discover SPPLE

Description

The following function is responsible for performing a SPPLE Service Discovery Operation. This function returns zero on successful execution and a negative value on errors.

Parameters

The only parameter required is the Bluetooth Address of the remote device that is connected.

Command Call Examples

- “DiscoverSPPLE 001bdc05b617” Attempts to discover services of the Bluetooth Device with the BD_ADDR of 001bdc05b617.
- “DiscoverSPPLE 000275e126FF” Attempts to discover services of the Bluetooth Device with the BD_ADDR of 000275e126FF.

Possible Return Values

- (0) Successfully started a SPP LE Service Discovery.
- (-4) Function Error (on failure).

API Call

GDIS_Service_Discovery_Start(BluetoothStackID, ConnectionID, (sizeof(UUID)/sizeof(GATT_UUID_t)), UUID, GDIS_Event_Callback, 0)

API Prototype

*int BTPSAPI GDIS_Service_Discovery_Start(unsigned int BluetoothStackID, unsigned int ConnectionID, unsigned int NumberOfUUID, GATT_UUID_t *UUIDList,GDIS_Event_Callback_t ServiceDiscoveryCallback, unsigned long ServiceDiscoveryCallbackParameter)*

Description of API

The GDIS_Service_Discover_Start is in an application module called GDIS that is provided to allow an easy way to perform GATT service discovery. This function is called to start a service discovery operation by the GDIS module.

11.2 Register SPPLE

Description

The following function is responsible for registering a SPPLE Service. This function returns zero on successful execution and a negative value on errors.

Parameters

Including parameters is not necessary when using this command. A parameter has no effect on the outcome of registering a SPPLE Service.

Possible Return Values

- (0) Successfully registered a SPPLE Service.
- (-4) Function Error (on failure).

API Call

*GATT_Register_Service(BluetoothStackID, SPPLE_SERVICE_FLAGS, SPPLE_SERVICE_ATTRIBUTE_COUNT, (GATT_Service_Attribute_Entry_t *)SPPLE_Service,&ServiceHandleGroup, GATT_ServerEventCallback, 0)*

API Prototype

*int BTPSAPI GATT_Register_Service(unsigned int BluetoothStackID, Byte_t ServiceFlags, unsigned int NumberOfServiceAttributeEntries, GATT_Service_Attribute_Entry_t*ServiceTable, GATT_Attribute_Handle_Group_t *ServiceHandleGroupResult, GATT_Server_Event_Callback_t ServerEventCallback, unsigned long CallbackParameter)*

Description of API

The following function is provided to allow a means to add a GATT Service to the local GATT Database. The first parameter is Bluetooth stack ID of the Bluetooth Device. The second parameter is a bit mask field that specifies the type of service being registered, which must be non-zero (i.e. at least one bit must be set). The third parameter is the number of entries in the service attribute array that is pointed to by the fourth parameter. The fourth parameter is an array that contains the attributes for the service being registered. The next parameter is a pointer to a buffer that stores the attribute handle range of the registered service. The final two parameters specify the GATT server callback and callback parameter that can be used whenever a client request to the GATT server cannot be satisified internally by the local GATT module. This function returns a positive non-zero service ID if successful, or a negative return error code if there is an error. If this function returns successfully then the ServiceHandleGroupResult buffer contains the service's attribute handle range.

11.3 LE Send

Description

The following function is responsible for sending a number of characters to a remote device to which a connection exists. The function receives a parameter that indicates the number of bytes to be transferred. This function returns zero on successful execution and a negative value on errors. Depending on what the device role for SPPLE is, server or client, the APIfunction that is called is either a GATT_Handle_Value_Notification or a GATT_Write_Without_Response_Request; which notifies the receiving credit characteristic or sends a write with out response packet to the transmission credit characteristic respectively.

Parameters

LESend requires two parameters. The first is the remote Bluetooth address of the device you are sending to. The second is the number of bytes to send. This value has to be greater than 10.

Command Call Examples

- "LeSend 0017E7FEFD7C 100" Attempts to send 100 bytes of data to 0017E7FEFD7C.
- "LeSend B8FFFEAF1CAD 25" Attempts to send 25 bytes of data to B8FFFEAF1CAD.

Possible Return Values

- (0) Successfully Sent Data
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-67) BTPS_ERROR_RFCOMM_NOT_INITIALIZED
- (-85) BTPS_ERROR_SPP_NOT_INITIALIZED

API Call

GATT_Handle_Value_Notification(BluetoothStackID, SPPLEServiceID, ConnectionID, SPPLE_TX_CHARACTERISTIC_ATTRIBUTE_OFFSET, (Word_t)DataCount, SPPLEBuffer)

Or

GATT_Write_Without_Response_Request(BluetoothStackID, ConnectionID, DeviceInfo->ClientInfo.Rx_Characteristic, (Word_t)DataCount, SPPLEBuffer)

API Prototype

*int BTPSAPI GATT_Handle_Value_Notification(unsigned int BluetoothStackID, unsigned int ServiceID, unsigned int ConnectionID, Word_t AttributeOffset, Word_tAttributeValueLength, Byte_t *AttributeValue)*

Or

*int BTPSAPI GATT_Write_Without_Response_Request(unsigned int BluetoothStackID, unsigned int ConnectionID, Word_t AttributeHandle, Word_t AttributeLength, void*AttributeValue)*

Description of API

The first of these API functions allows a means of sending a Handle/Value notification to a remote GATT client. The first parameter to this function is the Bluetooth stack ID of the localBluetooth stack. The second parameter is the service ID of the service that is sending the Handle/Value notification. The third parameter specifies the connection ID of the connection to send the Handle/Value notification to. The fourth parameter specifies the offset in the service table (registered via the call to the GATT_Register_Service() function) of the attribute that is being notified. The fifth parameter is the length (in bytes) of the attribute value that is being notified. The sixth parameter is a pointer to the actual attribute value to notify. This function returns a non-negative value that represents the actual length of the attribute value that is notified, or a negative return error code if there is an error.

The second of these API functions is provided to allow a means of performing a write without response request to remote device for a specified attribute. The first parameter to this function is the Bluetooth stack ID of the local Bluetooth stack, followed by the connection ID of the connected remote device, followed by the handle of the attribute to write, followed by the length of the value data to write (in bytes), followed by the actual value to write. This function returns the number of bytes written on success or a negative error code.

11.4 Configure SPPLE

Description

The following function is responsible to configure a SPPLE Service on a remote device. This function returns zero on successful execution and a negative value on errors. The following function enables notifications of the proper characteristics based on a specified handle; depending what the device role for SPPLE is, server or client, the API function that is called is either a GATT_Handle_Value_Notification or a GATT_Write_Without_Response_Request; which notifies the receiving credit characteristic or sends a write with out response packet to the transmission credit characteristic respectively.

Parameters

The only parameter required is the Bluetooth Address of the remote device that is connected.

Command Call Examples

- “ConfigureSPPLE 001bdc05b617” Attempts to configure services of the Bluetooth Device with the BD_ADDR of 001bdc05b617.
- “ConfigureSPPLE 000275e126FF” Attempts to configure services of the Bluetooth Device with the BD_ADDR of 000275e126FF.

Possible Return Values

- (0) Successfully configured a SPPLE Service.
- (-4) Function Error (on failure).

API Call

GATT_Write_Request(BluetoothStackID, ConnectionID, ClientConfigurationHandle, sizeof(Buffer), &Buffer, ClientEventCallback, 0)

And

GATT_Handle_Value_Notification(BluetoothStackID, SPPLEServiceID, ConnectionID, SPPLE_RX_CREDITS_CHARACTERISTIC_ATTRIBUTE_OFFSET, WORD_SIZE, (Byte_t)&Credits)*

Or

GATT_Write_Without_Response_Request(BluetoothStackID, ConnectionID, DeviceInfo->ClientInfo.Tx_Credit_Characteristic, WORD_SIZE, &Credits)

API Prototype

*int BTPSAPI GATT_Write_Request(unsigned int BluetoothStackID, unsigned int ConnectionID, Word_t AttributeHandle, Word_t AttributeLength, void *AttributeValue,GATT_Client_Event_Callback_t ClientEventCallback, unsigned long CallbackParameter)*

And

*int BTPSAPI GATT_Handle_Value_Notification(unsigned int BluetoothStackID, unsigned int ServiceID, unsigned int ConnectionID, Word_t AttributeOffset, Word_tAttributeValueLength, Byte_t *AttributeValue)*

Or

*int BTPSAPI GATT_Write_Without_Response_Request(unsigned int BluetoothStackID, unsigned int ConnectionID, Word_t AttributeHandle, Word_t AttributeLength, void*AttributeValue)*

Description of API

The first of these API functions is provided to allow a means of performing a write request to a remote device for a specified attribute. The first parameter to this function is the Bluetoothstack ID of the local Bluetooth stack, followed by the connection ID of the connected remote device, followed by the handle of the attribute to write the value of, followed by the length of the value (in bytes), followed by the the actual value data to write. The final two parameters specify the GATT client event callback function and callback parameter (respectively) that can be called when a response is received from the remote device. This function returns the positive, non-zero, Transaction ID of the request or a negative error code.

The second of these API functions allows a means of sending a Handle/Value notification to a remote GATT client. The first parameter to this function is the Bluetooth stack ID of the local Bluetooth stack. The second parameter is the service ID of the service that is sending the Handle/Value notification. The third parameter specifies the connection ID of the connection to send the Handle/Value notification to. The fourth parameter specifies the offset in the service table (registered via the call to the GATT_Register_Service() function) of the attribute that is being notified. The fifth parameter is the length (in bytes) of the attribute value that is being notified. The sixth parameter is a pointer to the actual attribute value to notify. This function returns a non-negative value that represents the actual length of the attribute value that was notified, or a negative return error code if there is an error.

The third of these API functions is provided to allow a means of performing a write without response request to remote device for a specified attribute. The first parameter to this function is the Bluetooth stack ID of the local Bluetooth stack, followed by the connection ID of the connected remote device, followed by the handle of the attribute to write, followed by the length of the value data to write (in bytes), followed by the actual value to write. This function returns the number of bytes written successfully or a negative error code.

11.5 LE Read

Description

The following function is responsible for reading data sent by a remote device to which a connection exists. This function returns zero on successful execution and a negative value on errors. Depending what the device role for SPPLE is, server or client, the API function that is called is either a GATT_Handle_Value_Notification or aGATT_Write_Without_Response_Request; which notifies the receiving credit characteristic or sends a write with out response packet to the transmission credit characteristic respectively.

Parameters

The only parameter required is the Bluetooth Address of the remote device that is connected.

Command Call Examples

- "LeRead 001bdc05b617" Attempts to read data of the Bluetooth Device with the BD_ADDR of 001bdc05b617.
- "LeRead 000275e126FF" Attempts to read data of the Bluetooth Device with the BD_ADDR of 000275e126FF.

Possible Return Values

- (0) Successfully Read Data
- (-6) INVALID_PARAMETERS_ERROR
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-67) BTPS_ERROR_RFCOMM_NOT_INITIALIZED
- (-85) BTPS_ERROR_SPP_NOT_INITIALIZED
- (-86) BTPS_ERROR_SPP_PORT_NOT_OPENED

API Call

GATT_Handle_Value_Notification(BluetoothStackID, SPPLEServiceID, ConnectionID, SPPLE_RX_CREDITS_CHARACTERISTIC_ATTRIBUTE_OFFSET, WORD_SIZE, (Byte_t)&Credits)*

Or

GATT_Write_Without_Response_Request(BluetoothStackID, ConnectionID, DeviceInfo->ClientInfo.Tx_Credit_Characteristic, WORD_SIZE, &Credits)

API Prototype

*int BTPSAPI GATT_Handle_Value_Notification(unsigned int BluetoothStackID, unsigned int ServiceID, unsigned int ConnectionID, Word_t AttributeOffset, Word_tAttributeValueLength, Byte_t *AttributeValue)*

Or

*int BTPSAPI GATT_Write_Without_Response_Request(unsigned int BluetoothStackID, unsigned int ConnectionID, Word_t AttributeHandle, Word_t AttributeLength, void*AttributeValue)*

Description of API

The first of these API functions allows a means of sending a Handle/Value notification to a remote GATT client. The first parameter to this function is the Bluetooth stack ID of the localBluetooth stack. The second parameter is the service ID of the service that is sending the Handle/Value notification. The third parameter specifies the connection ID of the connection to send the Handle/Value notification to. The fourth parameter specifies the offset in the service table (registered via the call to the GATT_Register_Service() function) of the attribute that is being notified. The fifth parameter is the length (in bytes) of the attribute value that is being notified. The sixth parameter is a pointer to the actual attribute value to notify. This function returns a non-negative value that represents the actual length of the attribute value that was notified, or a negative return error code if there was an error.

The second of these API functions is provided to allow a means of performing a write without response request to remote device for a specified attribute. The first parameter to this function is the Bluetooth stack ID of the local Bluetooth stack, followed by the connection ID of the connected remote device, followed by the handle of the attribute to write, followed by the length of the value data to write (in bytes), followed by the actual value to write. This function returns the number of bytes written on success or a negative error code.

11.6 Loopback

Description

The Loopback command is responsible for setting the application state to support loopback mode. This command returns zero on successful execution and a negative value on errors.

Parameters

This command requires one parameter which indicates if loopback can be supported. 0 = loopback not active, 1 = loopback active.

Command Call Examples

- "Loopback 0" sets loopback support to inactive.
- "loopback 1" sets loopback support to active.

Possible Return Values

- (0) Successfully set loopback support.
- (-6) INVALID_PARAMETERS_ERROR.

11.7 Display Raw Mode Data

Description

The following function is responsible for setting the application state to support displaying Raw Data. This function returns zero on successful execution and a negative value on errors.

Parameters

This command accepts one parameter which indicates if displaying raw data mode can be supported. 0 = Display Raw Data Mode inactive, 1 = Display Raw Data active.

Command Call Examples

- "DisplayRawModeData 0" sets Display Raw Mode support inactive.
- "DisplayRawModeData 1" sets Display Raw Mode support active.

Possible Return Values

- (0) Successfully sets Display Raw Data Mode support.
- (-6) INVALID_PARAMETERS_ERROR.

11.8 Automatic Read Mode

Description

The AutomaticReadMode command is responsible for setting the application state to support Automatically reading all data that is received through SPP. This function returns zeroon successful execution and a negative value on errors.

Parameters

This command accepts one parameter which indicates if automatic read mode can be supported. 0 = Automatic Read Mode inactive, 1 = Automatic Read Mode active.

Command Call Examples

- "AutomaticReadMode 0" sets Automatic Read Mode support to inactive.
- "AutomaticReadMode 1" sets Automatic Read Mode support to active.

Possible Return Values

- (0) Successfully set Automatic Read Mode support.
- (-6) INVALID_PARAMETERS_ERROR

12 References

- Texas Instruments, *TI Dual-Mode Bluetooth® Stack on MSP432™ MCUs*, User's Guide.
- Texas Instruments, *Dual-Mode Bluetooth® Stack on STM32F4 MCUs*, User's Guide.

13 Revision History

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

DATE	REVISION	NOTES
August 2023	*	Initial Release