CC256x TI Bluetooth Stack SPPLEDemo App

Return to CC256x MSP430 TI's Bluetooth stack Basic Demo APPS (http://processors.wiki.ti.com/index.php/CC256x_MSP430_TI_Bluetooth_Stack#Demos)

Return to CC256x Tiva TI's Bluetooth stack Basic Demo APPS (http://processors.wiki.ti.com/index.php/CC256x_Tiva_TI_Bluetooth_Stack#Demos)

Return to CC256x MSP432 TI's Bluetooth stack Basic Demo APPS (http://www.ti.com/lit/ug/swru453a/swru453a.pdf)

Return to CC256x STM32F4 TI's Bluetooth stack Basic Demo APPS (http://www.ti.com/lit/ug/swru428/swru428.pdf)

Contents

- ConfigureSPPLE
- LERead
- Loopback
- DisplayRawModeData
- AutomaticReadMode

Demo Overview

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

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 that it 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 on it 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.

This document talks about the SPPLE application in details.

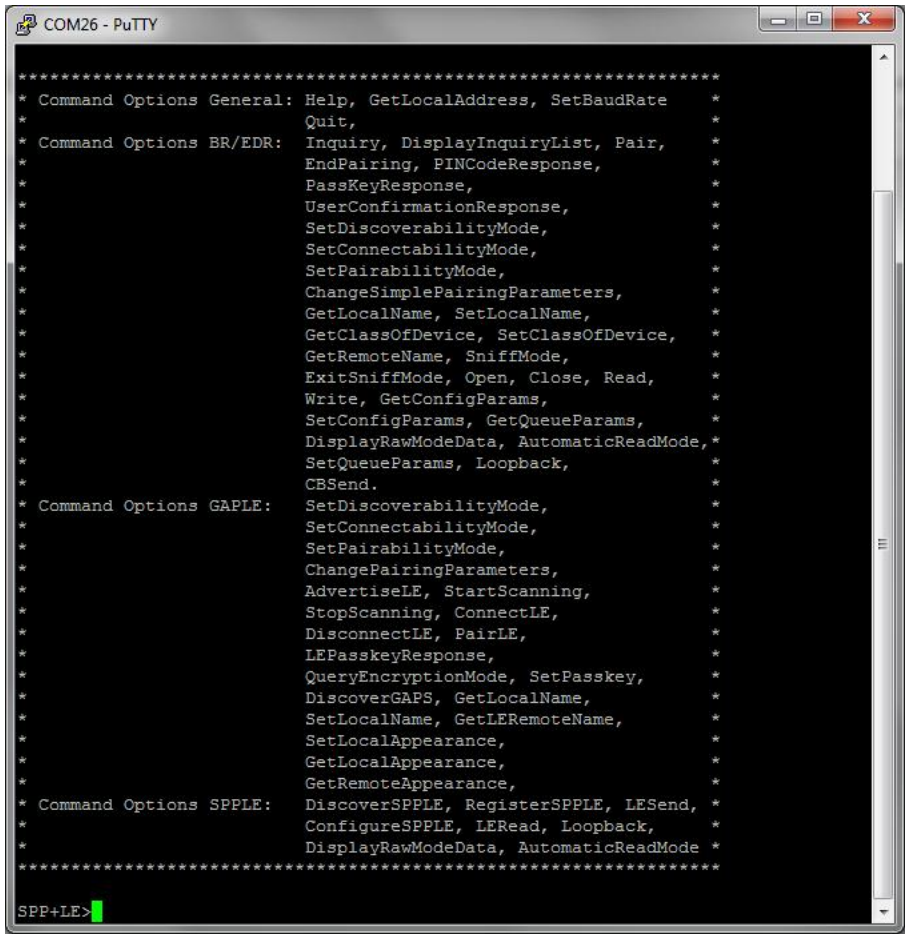
To read more about the BR/EDR version of SPP refer to this document SPP profile (http://processors.wiki.ti.com/index.php/CC256x_MSP430_TI's_Bluetooth_Stack_Basic_SPPDemo_APP).

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 over BLE.

It is recommended that the user visits the kit setup Getting Started Guide for MSP430 (http://processors.wiki.ti.com/index.php/CC256x_MSP430_TI's_Bluetooth_Stack_Basic_Demo_APPS), Getting Started Guide for TIVA (http://processors.wiki.ti.com/index.php/TIVA_TI's_Bluetooth_Stack_Basic_Demo_APPS), Getting Started Guide for MSP432 (http://www.ti.com/lit/ug/swru453a/swru453a.pdf) or Getting Started Guide for STM32F4 (http://www.ti.com/lit/ug/swru428/swru428.pdf) pages before trying the application described on this page.

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. It will show up 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. The serial parameters to use are 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 you should see the stack getting initialized on the terminal and the help screen will be displayed, which shows all of the commands. This device will become the server.



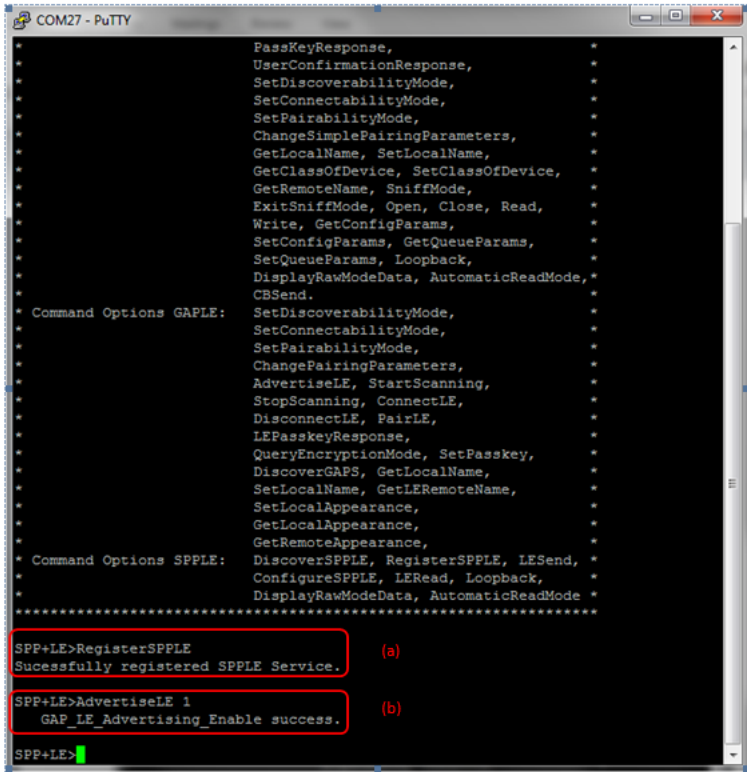
Now connect the second board via miniUSB or microUSB cable and follow the same steps performed before when running the Bluetooth code on the first board. The second device that is connected to the computer will be the client.

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 a board when the stack is initialized.

Device 1 (Server) setup on the demo application

a) 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**.

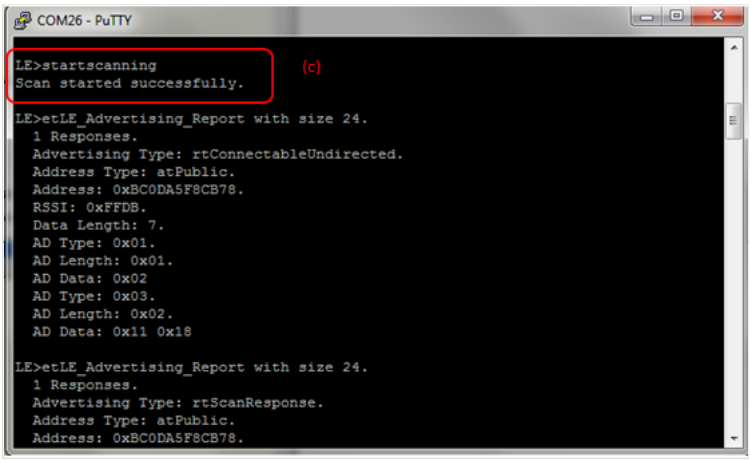


b) Next, the device acting as a server needs to advertise to other devices. This can be done by running **AdvertiseLE 1**.

Device 2 (Client) setup on the demo application

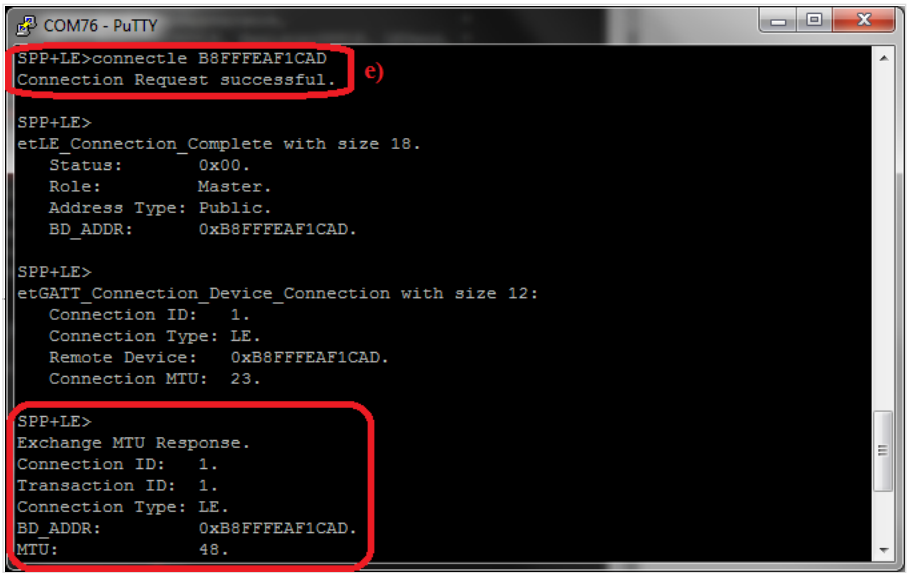
c) Place the device into client mode by typing **Client** on the console.

[Steps d and e are optional if you already know the Bluetooth address of the device that you want to connect to]
d) The client LE device can try to find which LE devices are in the vicinity using the command: **StartScanning**.
e) Once you have found the device, you can stop scanning by using the command: **StopScanning**.



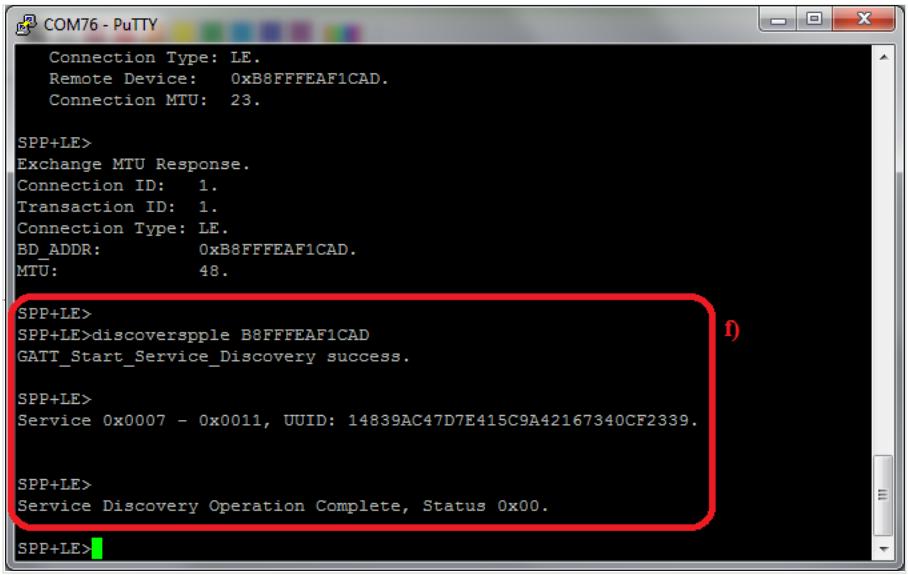
Initiating connection from device 2

f) Once the application on the client side knows the Bluetooth address of the device that is advertising, it can connect to that device using the command: **ConnectLE <Bluetooth Address>**

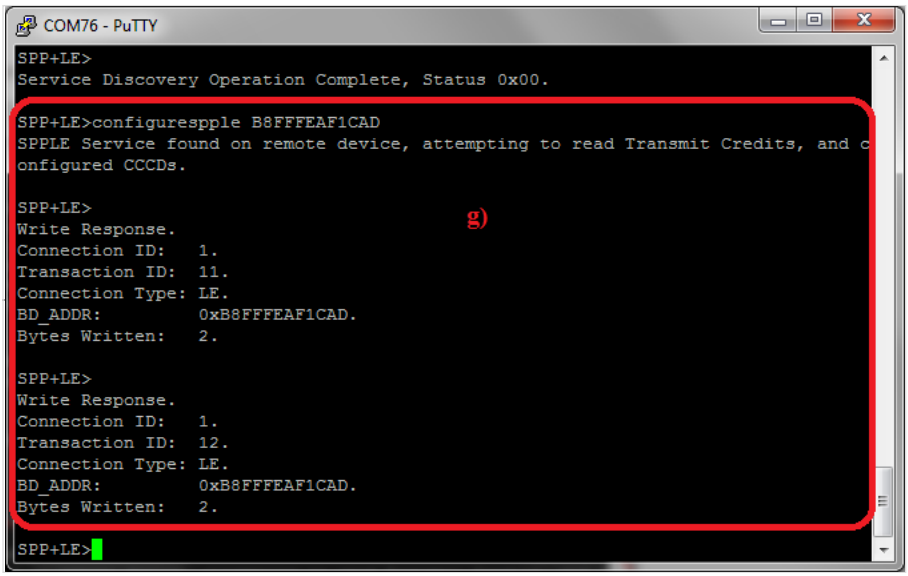


Identify supported services

g) After Initialization, the device needs to find out if SPP services are supported. To do this run **DiscoverSPPLE <Server BD-Address>** on the client.

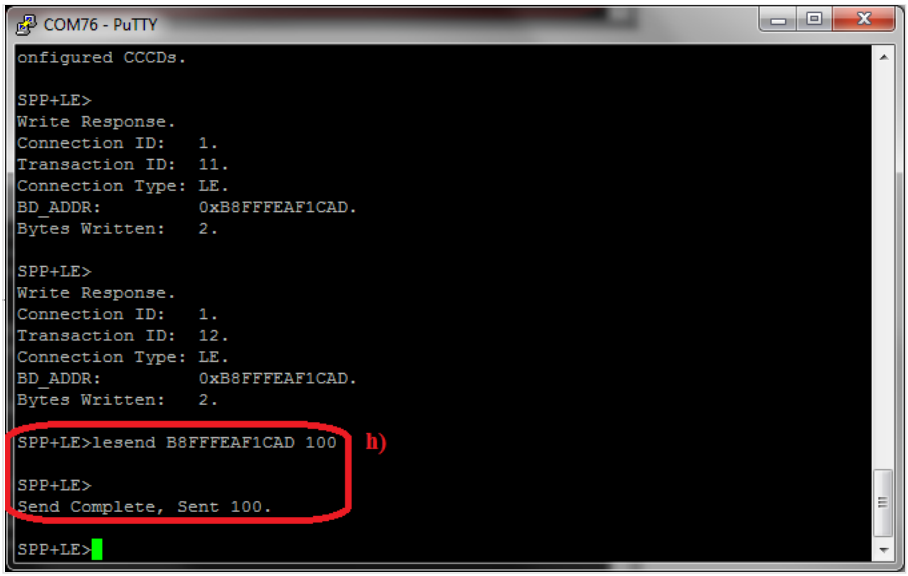


h) After finding out support for SPP-LE, we need to configure SPP-LE. This is done by running **ConfigureSPPLE <Server BD-Address>** on the client.



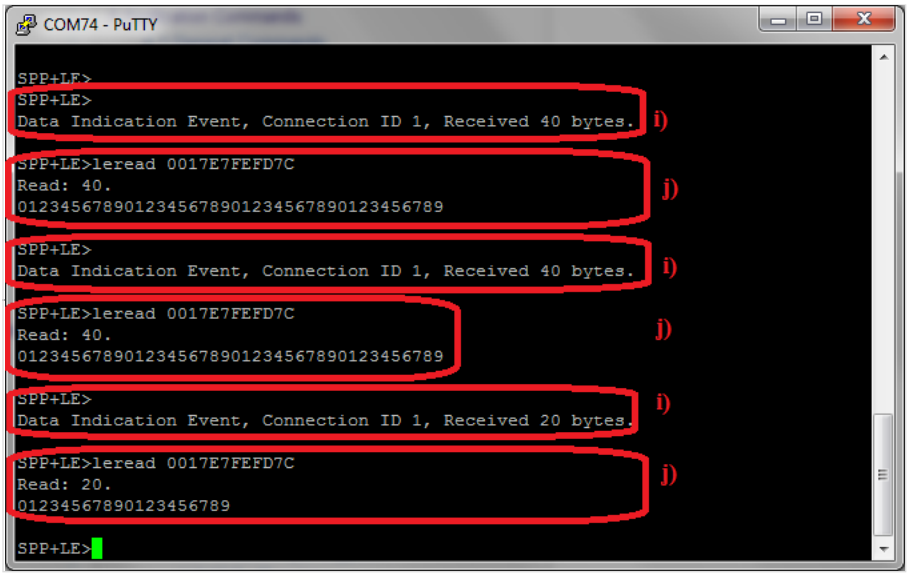
Data Transfer between Client and Server

i) After configuring we can send data between client and server. To send data we use **LESend <Remote Device BD-Address> <Number of bytes>**.



j) Once the other device receives the data it receives a Data Indication event.

k) The receiving device can then read the data that was sent using command: **LERead <Remote Device BD-Address>**



l) This will print out the data that was sent. This data was sent over BluetoothLE using a custom service of SPPLE in the sample application.

Multiple SPPLE Connections Guide

Two SPPLE Connections

a) 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.
b) 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 we connected to.
c) Similarly, Connect to the second device, discover and configure services on the second device.
d) To send data to the first remote device data we use **LeSend <BD-ADDR> <Number of Bytes to be sent>**
e) To send data to the second remote device data we use **LeSend <BD-ADDR> <Number of Bytes to be sent>**
f) To read data from the first remote device data we use **LeRead <BD-ADDR>**
g) To read data from the second remote device data we use **LeRead <BD-ADDR>**
h) When we turn on **Automaticreadmode**, **DisplayRawmodedata** or **Loopback** it turns it on for both connections.

- **One SPP and One SPPLE Connection**

a) In this version, we 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 device as SPP Client.
b) 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 we connected to.
c) Open an SPP server and let the second remote device connect to it.
d) To send data to the first remote device data we use **LeSend <BD-ADDR> <Number of Bytes to be sent>**
e) 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>**
f) To read data from the first remote device data we use **LeRead <BD-ADDR>**
g) To read data from the second remote device data we use **Read**.
h) When we turn on **Automaticreadmode**, **DisplayRawmodedata** or **Loopback** it turns it on for both connections.

Demonstrating SPP LE on an iOS Device with the LightBlue App

LightBlue Overview

The LightBlue app is a free iOS app that allows you to test and demonstrate the GATT Profile using Bluetooth Low Energy (BLE). It allows you to create custom services and interact with servers with custom services. The app supports both the client and server roles of GATT. Here we will explain how to use the app with the SPPLEDemo application.

SPP LE Service Overview

SPP LE is not an official Bluetooth service. It is a custom service that is designed to demonstrate using Bluetooth Low Energy to send and receive data in a similar manner that Classic Bluetooth's SPP profile does. It uses a credit based protocol to send and receive data. In order 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 its maximum number of credits, it must wait for the remote device to provide it with more credits before it can continue sending. In this application 1 credit is equivalent to 1 byte (octet) of data.

Characteristics

SPP LE implements its 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 its 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 its 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. Next we'll demonstrate SPPLEDemo as the server and LightBlue as the client. If you haven't already done so, download the LightBlue app from the App Store and turn on Bluetooth on your iOS device.

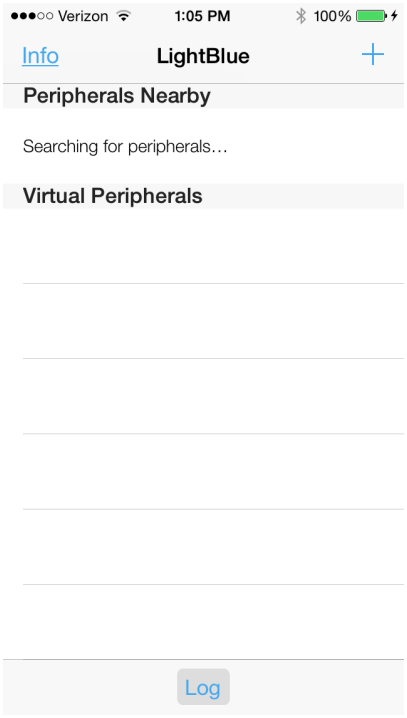
Note: For more information about characteristics, ATT Write Requests, and ATT Handle Value Notifications, please refer to the Attribute Protocol (ATT) and Generic Attribute Profile (GATT) specifications in the Bluetooth Core specification, which can be found on the Bluetooth SIG's website (https://www.bluetooth.org/en-us/specification/adopted-specifications).

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 (http://www.ti.com/tool/tibluetoothstack-sdk) was used running on a DK-TM4C123G.

LightBlue as the Client/SPPLEDemo as the Server

Connecting the Devices

First we need to establish a connection between the devices. To do this open the LightBlue app, you'll seen a screen similar to the following:

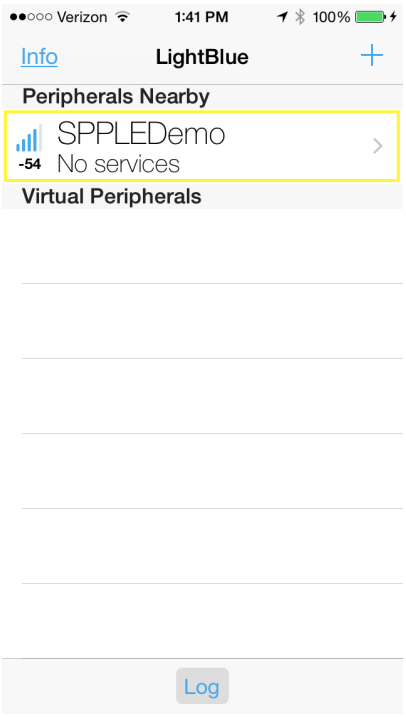


LightBlue Startup

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. You will see the following in the terminal:

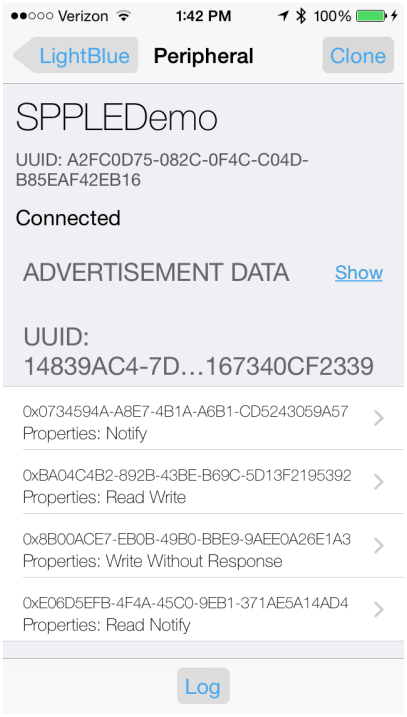
```
OpenStack().
Bluetooth Stack ID: 1.
Device Chipset: 4.1.
BD_ADDR: 0x0017e9d3581a

*****************************************************************
* 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 you will see the device shown in LightBlue:



SPPLEDemo Discovered by LightBlue

Next select the **SPPLEDemo** device in LightBlue, after doing so you should see the following screen:



LightBlue and SPPLEDemo Connected

In the SPPLEDemo terminal you will see 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.
```

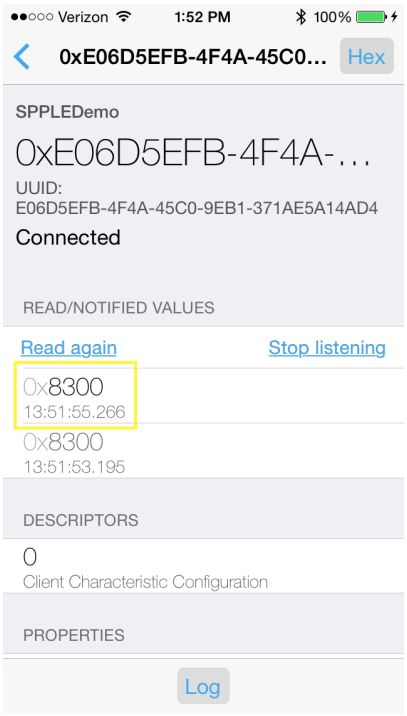
The devices are now connected.

Enabling Notifications

Next enable notifications on the Tx Characteristic and Rx Credits Characteristic in LightBlue by doing the following:

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

You will notice that after enabling notifications on the Rx Credits Characteristic (**0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4**) that SPPLEDemo sends its initial credits to LightBlue and you will see **0x8300** displayed twice in the app:



Initial Credits Received from SPPLEDemo

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.

Sending Data from LightBlue/Receiving Data in SPPLEDemo

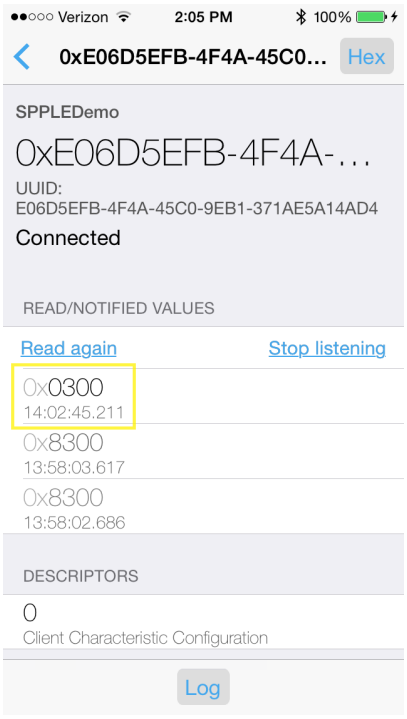
At this point the client (LightBlue) can send data to the server (SPPLEDemo). To send data from LightBlue to SPPLEDemo do the following:

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 you will see a data indication event. To read the data run the **LERead 5cfc3252180b** command, you will see the following in the terminal:

```
Data Indication Event, Connection ID 1, Received 3 bytes.

SPP+LE>LERead 5cfc3252180b
Read: 3.
ABC
```

Now if you open the Rx Credits Characteristic (**0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4**) you will see that SPPLEDemo has credited LightBlue with 3 more credits:



Additional Credits Received from SPPLEDemo

Sending Data from SPPLEDemo/Receiving Data in LightBlue

Now we will 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:

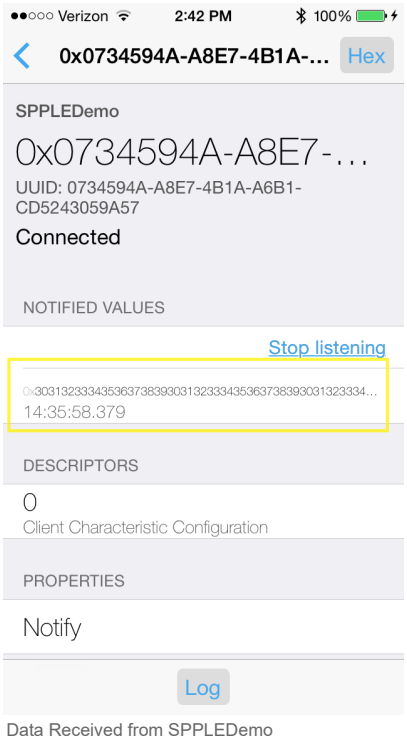
1. Open the Tx Credits Characteristic (**0xBA04C4B2-892B-43BE-B69C-5D13F2195392**).
2. Choose "Write new value".
3. Type **6400**. (100 credits = 0x0064 little-endian)
4. Choose **Done**.
5. Press the back button in the top left corner.

We have now given SPPLEDemo 100 credits. Now we can send data in SPPLEDemo using the **LESend 5cfc3252180b 100** command. You should see the following in the terminal:

```
SPP+LE>LESend 5cfc3252180b 100

Send Complete, Sent 100.
```

You can check that LightBlue received the data by:

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



Data Received from SPPLEDemo

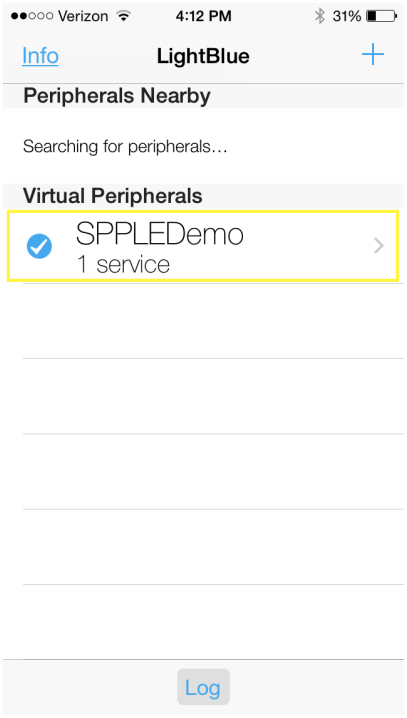
Now that LightBlue has received the data it needs to return the transmit credits to SPPLEDemo. This can be done by repeating the sequence above and re-writing **0x6400** to the Tx Credits Characteristic (**0xBA04C4B2-892B-43BE-B69C-5D13F2195392**).

LightBlue as the Server/SPPLEDemo as the Client

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

Connecting the Devices

The first step to connecting the devices is to add the SPP LE Service and its characteristics to LightBlue. It is possible to do this manually by creating a blank virtual peripheral in LightBlue and then adding the necessary service and characteristics, however, it's easier to simply clone SPPLEDemo when it is acting 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 will return to the devices list and you will now see SPPLEDemo listed as a Virtual Peripheral as seen below:



SPPLEDemo Shown as a Virtual Peripheral

Note: Make sure that the check box to the left of SPPLEDemo is checked, as seen in the image above. If it is not checked the iDevice is not advertising and SPPLEDemo will not be able to connectd to it.

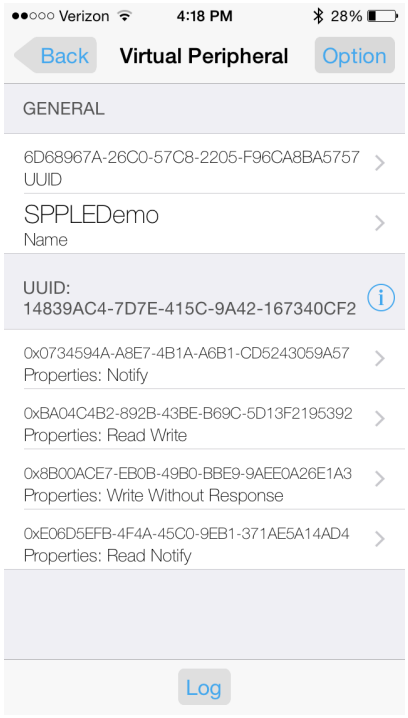
Now that we have cloned the SPP LE service we can now continue with connecting the devices. Next, restart SPPLEDemo and when prompted start the app as a client. Next scan for the iOS device using the **StartScanning** command. When the iOS device has been found stop the scan using the **StopScanning** command. Now we can connect to the iOS device using the **ConnectLE 5c75524c733a 1** command. After this you have about 10 seconds to run the **DiscoverSPPLE 5c75524c733a** command. After the SPP LE service discovery completes you have about 25 seconds to run the **ConfigureSPPLE 5c75524c733a**. If you wait longer than these times the iOS device will disconnect from SPPLEDemo. After the SPP LE characteristics are configured the 2 apps will stay connected, however, note that if the iOS device goes to sleep it will still close the connection. After running the commands just described you will see 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 was acting as the server we had to manually enable notifications with the LightBlue app, however, SPPLEDemo handles enabling notifications automatically when the **ConfigureSPPLE** command is run and this has already been taken care of.

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



SPPLEDemo Shown as a Virtual Peripheral

.

Sending Data from LightBlue/Receiving Data in SPPLEDemo

At this point SPPLEDemo has provided LightBlue with transmit credits and did so when the ConfigureSPPLE command was ran. You should be able to confirm this by opening the SPPLEDemo Virtual Peripheral and choosing the Credits Characteristic (**0xBA04C4B2-892B-43BE-B69C-5D13F2195392**), however, as mentioned above LightBlue does not show updated values of characteristics when they are written to and we have no way to confirm that LightBlue received the data. Even though we can't confirm that LightBlue has received transmit credits, we can still send data from LightBlue to SPPLEDemo. This is true because LightBlue is primarily only a GATT Profile demonstration, it doesn't have any knowledge of the SPP LE protocol that we are using. It is unaware of the transmit credits it has or doesn't have, and, for this reason, we can send data 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 you will see a data indication. To read the data use the **LERead 5c75524c733a** command. You should see **ABC** displayed in the terminal, as seen below:

```
Data Indication Event, Connection ID 1, Received 3 bytes.

SPP+LE>LERead 5c75524c733a
Read: 3.
ABC
```

Sending Data from SPPLEDemo/Receiving Data in LightBlue

Note: As mentioned earlier LightBlue does not support showing the updated value of a characteristic when it is written to. We can send data to LightBlue, however, we have no way confirm the data was received.

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

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

SPPLEDemo now has 100 transmit credits. Next, to send data in SPPLEDemo use the **LESend 5c75524c733a 100** command. You will see the following in your terminal.

```
SPP+LE>LESend 5c75524c733a 100

Send Complete, Sent 100.
```

As mentioned earlier we have no way confirm that LightBlue actually received the data, this is a LightBlue limitation.

Demonstrating SPP LE on an iOS Device with the SPPLE Transfer App - LEGACY

Note: The SPPLE Transfer app is no longer available on the iOS app store as of Q1 2015, however, the LightBlue app is available and can be used as a replacement. Refer to the above section Demonstrating SPP LE on an iOS Device with the LightBlue App for a demonstration of using the LightBlue app with SPPLEDemo.

SPPLE is not a standard Bluetooth Profile. You will have to make sure the app can use the custom UUIDs that are needed to communicate and read and write to the app.

The MSP430 device can also connect to an iPhone running an SPPLE application. The application that we use on the iPhone is SPPLE Transfer (a.k.a. SPPLE Chat) which can be downloaded for free from the app store here (https://itunes.apple.com/us/app/spple-chat/id602017895?mt=8). There are some changes that need to be made to the **SPPLEDemo.c** file as well. See below:

- In the function **ConfigureSPPLE** make the following changes to the if statement. After the comment **/* Determine if a service discovery operation has been previously done */** and before the **else case**.

```
/* Determine if a service discovery operation has been      */
/* previously done.                                         */
   /*changed for using it with SPPLE chat application on iPhone*/
   if(TRUE)
   {
        Display(("SPPLE Service found on remote device, attempting to read Transmit Credits, and configured CCCDs.\r\n"));

        /* Enable Notifications on the proper characteristics.     */
        EnableDisableNotificationsIndications(LEContextInfo[LEConnectionIndex].ConnectionID, DeviceInfo->ClientInfo.Tx_Client_Configuration_Descriptor,
                    GATT_CLIENT_CONFIGURATION_CHARACTERISTIC_NOTIFY_ENABLE, GATT_ClientEventCallback_SPPLE);

        ret_val = 0;
}
else
```

```
   /* Determine if a service discovery operation has been      */
   /* previously done.                                         */
   if(SPPLE_CLIENT_INFORMATION_VALID(DeviceInfo->ClientInfo))

   {
      Display(("SPPLE Service found on remote device, attempting to read Transmit Credits, and

      /* Send the Initial Credits to the remote device.       */
      SPPLESendCredits(&(LEContextInfo[LEConnectionIndex]), DeviceInfo, LEContextInfo[LEConnec

      /* Enable Notifications on the proper characteristics.   */
      EnableDisableNotificationsIndications(LEContextInfo[LEConnectionIndex].ConnectionID, Dev
      EnableDisableNotificationsIndications(LEContextInfo[LEConnectionIndex].ConnectionID, Dev

      ret_val = 0;
   }
   else
```

```
   /* Determine if a service discovery operation has been      */
   /* previously done.                                         */
   /*changed for using it with SPPLE chat application on iPhone*/
   if(TRUE)
   {
      Display(("SPPLE Service found on remote device, attempting to read Transmit Credits, and confi

      /* Enable Notifications on the proper characteristics.      */
      EnableDisableNotificationsIndications(LEContextInfo[LEConnectionIndex].ConnectionID, DeviceInf

      ret_val = 0;
   }
   else
```

- In the function **SendDataCommand** add the following code after the **SendInfo.BytesSent = 0** and before the comment **/* Kick start the send process. */**.

```
LEContextInfo[LEConnectionIndex].SPPLEBufferInfo.TransmitCredits = 1000;
DeviceInfo->ServerInfo.Tx_Client_Configuration_Descriptor        = GATT_CLIENT_CONFIGURATION_CHARACTERISTIC_NOTIFY_ENABLE;
```

```
/* Get the count of the number of bytes to send.        */
LEContextInfo[LEConnectionIndex].SPPLEBufferInfo.SendInfo.BytesToSend  = (DWord_t)TempPa
LEContextInfo[LEConnectionIndex].SPPLEBufferInfo.SendInfo.BytesSent    = 0;
```

```
/* Get the count of the number of bytes to send.        */
LEContextInfo[LEConnectionIndex].SPPLEBufferInfo.SendInfo.BytesToSend  = (DWord_t)TempParam->Params[1].intParam;
LEContextInfo[LEConnectionIndex].SPPLEBufferInfo.SendInfo.BytesSent    = 0;
/*added for using it with SPPLE chat application on iPhone */
LEContextInfo[LEConnectionIndex].SPPLEBufferInfo.TransmitCredits = 1000;
DeviceInfo->ServerInfo.Tx_Client_Configuration_Descriptor       = GATT_CLIENT_CONFIGURATION_CHARACTERISTIC_NOTIFY_ENABLE;
```

- Load the SPP LE profile on to the MSP430F5438A device by rebuilding the project and flashing it from the project.
- Set up a Terminal Program for the Serial Port that the device is connected to. The serial parameters to use are 115200 Baud, 8, no, 1 and no flow control. Once connected, reset the

```
    device using Reset S3 button and you should see the stack getting initialized on the terminal.
```

- On the iPhone open the SPPLE chat application. Choose peripheral mode and turn on advertising.
- On the MSP430 device, **StartScanning** to find out devices in the area that are connectable. The Bluetooth address of the Iphone should show up something like this:

```
etLE_Advertising_Report with size 36.
  1 Responses.
  Advertising Type: rtConnectableUndirected.
  Address Type: atRandom.
  Address: 0x79F20C012372.
  RSSI: 0xFFFFFFCB.
  Data Length: 29.
  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
  AD Type: 0x09.
  AD Length: 0x06.
  AD Data: 0x69 0x50 0x68 0x6F 0x6E 0x65

LE>etLE_Advertising_Report with size 36.
  1 Responses.
```

```
Advertising Type: rtScanResponse.
Address Type: atRandom.
Address: 0x79F20C012372.
RSSI: 0xFFFFFFCB.
Data Length: 0.
```

- The address type will be random. Note down the address of the device specified.
- Connect to the remote device using the **Connectle <bd-addr> 1** command where the bd-addr is the previously noted address.
- Discover services using **Discoverspple** and configure services using **Configurespple**.
- Now the two devices are connected. Data from the iphone can be send by typing text on the text box and hitting send.
- Data from the MSP device can be sent using the **Senddata** command. It is read and displayed automatically in the output window of the app.

SPP Demo

To use classic SPP on this demo, please follow same instructions here: CC256x MSP430 TI's Bluetooth stack Basic SPPDemo APP (http://processors.wiki.ti.com/index.php/CC256x_MSP430_TI's_Bluetooth_Stack_Basic_SPPDemo_APP)

Application Commands

TI's Bluetooth stack is implementation of the upper layers of the Bluetooth protocol stack. TI's Bluetooth stackprovides 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 (http://www.ti.com/tool/msp-exp430f5438), Tiva DK-TM4C129X (http://www.ti.com/tool/dk-tm4c129x), MSP432 (http://www.ti.com/tool/msp-exp432p401r) and STM32F4 (http://www.st.com/en/evaluation-tools/stm3240g-eval.html) is a Serial Port Profile Application.

An overview of the application and other applications can be read at the Getting Started Guide (http://processors.wiki.ti.com/index.php/CC256x_MSP430_TI's_Bluetooth_Stack_Basic_Demo_APPS) for MSP430, Getting Started Guide (http://processors.wiki.ti.com/index.php/TIVA_TI's_Bluetooth_Stack_Basic_Demo_APPS) for Tiva M4, Getting Started Guide (http://www.ti.com/lit/ug/swru453a/swru453a.pdf) for MSP432 and Getting Started Guide (http://www.ti.com/lit/ug/swru428/swru428.pdf) for STM32F4.

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 (**TI_Bluetooth_Stack_Version-Number\Documentation** or for STM32F4, **TI_Bluetooth_Stack_Version-Number\RTOS_VERSION\Documentation**) describes all of the API's in detail.

General Commands

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
It is not necessary to include parameters when using this command. A parameter will have no effect on the outcome of the command.

Possible Return Values
This command always returns 0

GetLocalAddress

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
It is not necessary to include parameters when using this command. A parameter will have 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 will be 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).

SetBaudRate

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 disable.

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.

Quit

Using this command will take you back to the intial command screen.

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 Generic Access Profile Commands and Serial Port Profile Commands.

GAPLE Commands

The Generic Access Profile defines standard procedures related to the discovery and connection of Bluetooth devices. It defines modes of operation that are generic to all devices and allows for procedures which use those modes to decide how a device can be interacted with by 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 may have with one another. GAP also defines the procedures for how bond two Bluetooth devices.

SetDiscoverabilityMode

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 it is not discoverable. The value set by this command will be 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 (for Limited 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 Discoverability Mode 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

SetConnectabilityMode

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 it is not connectable. The value set by this command will be 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 Connectability Mode to Connectable.

Possible Return Values

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

SetPairabilityMode

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 will return zero if successful or a negative return error code if there was an error condition.

ChangePairingParameters

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.
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).
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.
The third parameter must be specified as 0 (for No MITM) or 1 (for MITM required).
The fourth parameter must be specified as 0 (for SC disabled) or 1 (for SC enabled), when using SC disable, legacy pairing procedure will take place.
The fifth parameter must be specified as 0 (for Debug Mode disabled) or 1 (for P256 debug mode enabled), Only when using SC pairing, P256 debug mode is relevant and when it is set, the values of the P256 private and public keys will be 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

AdvertiseLE

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 will return 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 will return 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 will return zero if successful or a negative return error code if there was an error condition. The GAP_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 the GAP_LE_Advertising_Enable function). This function will return zero if successful or a negative return error code if there was an error condition.

StartScanning

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 int BluetoothStackID) function which performs the scan.

Parameters

It is not necessary to include parameters when using this command. A parameter will have 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, unsigned long 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 in that it can be used to discover devices that have been instructed to advertise. This function will return zero if successful, or a negative return error code if there was an error condition.

StopScanning

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 int BluetoothStackID) function which performs the scan.

Parameters

It is not necessary to include parameters when using this command. A parameter will have no effect on the outcome of disabling Scanning.

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 will return zero if successful or a negative return error code if there was an error condition.

ConnectLE

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 int 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);
(these two APIs can generally be ignored unless the WhiteList is enabled in the call to ConnectLEDevice)
GAP_LE_Remove_Device_From_White_List(BluetoothStackID, 1, &WhiteListEntry, &WhiteListChanged);
GAP_LE_Add_Device_To_White_List(BluetoothStackID, 1, &WhiteListEntry, &WhiteListChanged);

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 will be notified via the GAP LE event callback function (specified in this function) when the connection completes. This function will return zero if successful, or a negative return error code if there was 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 will attempt to delete as many devices as possible (from the specified list) and will return 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).

DisconnectLE

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 will return zero if successful, or a negative return error code if there was an error condition.

PairLE

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_t BD_ADDR, Boolean_t ConnectionMaster) function using SendPairingRequest(ConnectionBD_ADDR, LocalDeviceIsMaster).

Parameters

It is not necessary to include parameters when using this command. A parameter will have no effect on the outcome of Pairing.

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_t GAP_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 was 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 was an error.

LEPassKeyResponse

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 value between 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).

LEQueryEncryption

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
It is not necessary to include parameters when using this command. A parameter will have 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.

SetPasskey

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 you are 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
(Depending on the First Parameter one of these will be chosen)
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.

DiscoverGAPS

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
It is not necessary to include parameters when using this command. A parameter will have 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 and should be modified for the customers use. This function is called to start a service discovery operation by the GDIS module.

GetLocalName

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
It is not necessary to include parameters when using this command. A parameter will have no effect on the outcome of the Query.

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 buffer length of the buffer that is to receive the local device name. The NameBufferLength parameter should be 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 will be 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).

SetLocalName

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 should not be spaces in the name or only the first section of the name will be 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 to set the Local Device Name to "New Bluetooth Device Name" but only sets the first parameter, which would make the Local Device Name "New." "SetLocalName MSP430" Attempts to set the 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 will return zero if the local device name was successfully changed, or a negative return error code if there was an error condition.

GetRemoteName

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 it should be called after using the Inquiry command. The DisplayInquiryList command would be 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
(-4) FUNCTION_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 long CallbackParameter);

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 success, then the caller will be notified via the specified callback when the remote name information has been determined (or there was an error). This function cannot be used to determine the user-friendly name of the local Bluetooth device. The GAP_Query_Local_Name function should be 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 will notify 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. It should be noted that when the callback is cancelled, the operation is attempted to be cancelled and the callback is cancelled (i.e. the GAP module still might perform the remote name request, but no callback is ever issued).

LEUserConfirmationResponse

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 Response with a decline value.
“LEUserConfirmationResponse 1” Attempts to Response 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 of API

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 a pairing/authentication process, followed by the authentication response information.This function returns zero if successful, or a negative return error code if there was an error.

EnableSCOnly

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 will be 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 will use this function before the beginning of LE SC pairing, in case it asks to reject a device that supports only legacy pairing. This mode should be used when it is more important for a device to have high security than it is for it 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 should be used ones, before the first pairing process. This function returns zero if successful or a negative error code.

RegenerateP256LocalKeys

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. It is relevant for LE Secure Conenctions 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.

SCGenerateOOBLocalParams

Description
In order to be able to perform LE SC pairing in OOB method we need to 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. It 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 will use 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 will recieve the generated local OOB random, and OOB confirmation values. This function returns zero if successful or a negative error code.

SetLocalAppearence

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

Parameters
The SetLocalAppearence command requires one parameter which is the Local Device Appearance you wish to be set.

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).

GetLocalAppearence

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

Parameters
It is not necessary to include parameters when using this command. A parameter will have 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).

GetRemoteAppearence

Description
The GetRemoteAppearence command is provided to read the device appearance from the connected remote device that is exposed as part of the GAP Service. The GAP Service on the remote device must have already been discovered using the DiscoverGAPS command.

Parameters
It is not necessary to include parameters when using this command. A parameter will have no effect on the outcome

Possible Return Values
(0) Success.
(-4) Function error (on failure).

API Call
GATT_Read_Value_Request(BluetoothStackID, ConnectionID, DeviceInfo->GAPSClientInfo.DeviceAppearanceHandle, GATT_ClientEventCallback_GAPS, (unsigned long)DeviceInfo->GAPSClientInfo.DeviceAppearanceHandle)

API Prototype
int BTPSAPI GATT_Read_Value_Request(unsigned int BluetoothStackID, unsigned int ConnectionID, Word_t AttributeHandle, GATT_Client_Event_Callback_t ClientEventCallback, unsigned long CallbackParameter)

Description of API
This function allows a mechanism of reading an attribute from a connected device.

SPPLE Commands

DiscoverSPPLE

Description
The following function is responsible for performing a SPPLE Service Discovery Operation. This function will return 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.

RegisterSPPLE

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

Parameters
It is not necessary to include parameters when using this command. A parameter will have 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 will store the attribute handle range of the registered service. The final two parameters specify the GATT server callback and callback parameter that will be used whenever a client request to the GATT server cannot be satisified internally by the local GATT module. This function will return a positive non-zero service ID if successful, or a negative return error code if there was an error. If this function returns success then the ServiceHandleGroupResult buffer will contain the service's attribute handle range.

LESend

/ 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 byte to be transferred. This function will return 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 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_t AttributeValueLength, 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 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 will return 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 will return the number of bytes written on success or a negative error code.

ConfigureSPPLE

Description

The following function is responsible to configure a SPPLE Service on a remote device. This function will return 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_t AttributeValueLength, 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 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 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 will be called when a response is received from the remote device. This function will return 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 will return 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 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 will return the number of bytes written on success or a negative error code.

LERead

Description
The following function is responsible for reading data sent by a remote device to which a connection exists. This function will return 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 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
“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_t AttributeValueLength, 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 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 will return 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 will return the number of bytes written on success or a negative error code.

Loopback

Description

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

Parameters

This command requires one parameter which indicates if loopback should 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

DisplayRawModeData

Description

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

Parameters

This command accepts one parameter which indicates if displaying raw data mode should 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

AutomaticReadMode

Description

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

Parameters

This command accepts one parameter which indicates if automatic read mode should 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

{{

1. switchcategory:MultiCore=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **CC256x TI Bluetooth Stack SPPLEDemo App** here.

Keystone=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **CC256x TI Bluetooth Stack SPPLEDemo App** here.

C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **CC256x TI Bluetooth Stack SPPLEDemo App** here.*

DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **CC256x TI Bluetooth Stack SPPLEDemo App** here.*

MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **CC256x TI Bluetooth Stack SPPLEDemo App** here.*

OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **CC256x TI Bluetooth Stack SPPLEDemo App** here.*

OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **CC256x TI Bluetooth Stack SPPLEDemo App** here.*

MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **CC256x TI Bluetooth Stack SPPLEDemo App** here.*

*For technic please post questions a http://e2e.ti. Please pos comments article **CC2 Bluetooth SPPLEDen here.*

}}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity