CC256x TI Bluetooth Stack HRPDemo 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

Demo Overview

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

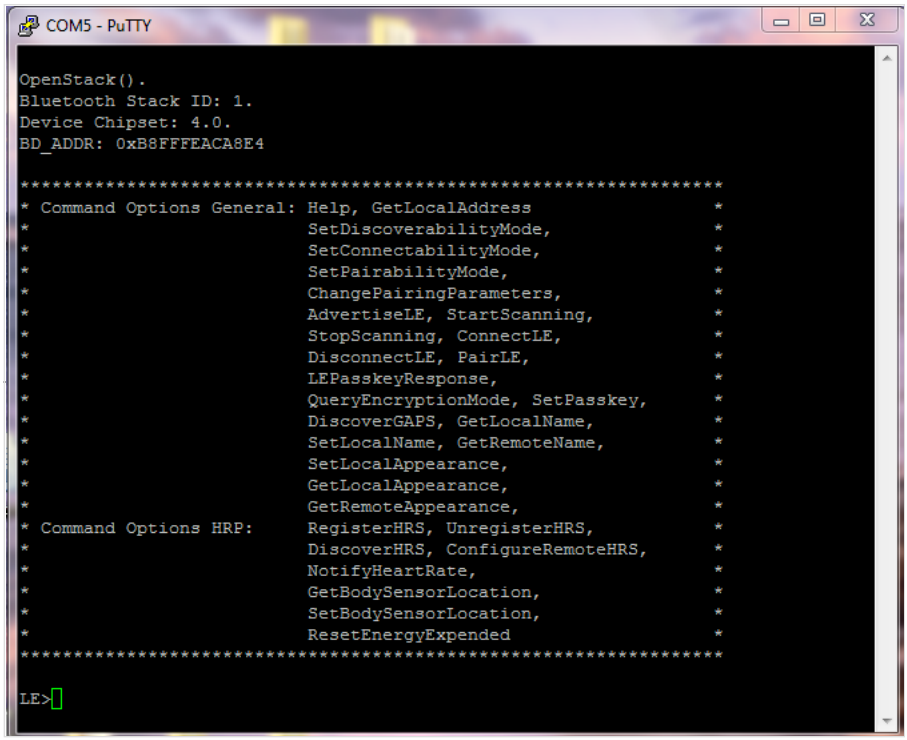
The Heart Rate profile (HRP) enables a Collector device to connect and interact with a Heart Rate sensor for use in healthcare applications. There are two roles defined in this profile. The first is the Sensor which measures the Heart Rate and the second is the collector which gets the Heart Rate and other settings from the sensor. Typically, the sensor would be present directly on the patient in a location such as the Heart or wrist measuring the temperature while the collector device is close by getting the Heart Rate from the sensor at regular intervals.

This application allows the user to use a console to use Bluetooth Low Energy (BLE) to establish connection between two BLE devices, notify the heart rate between the service and Client, get and change the location of the heart rate sensor.

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.



Now connect the second board via miniUSB or microUSB cable and follow the same steps performed before within the **Running the Bluetooth Code** section on the first board. The second device that is connected to the computer will be the **Client**.

Demo Application

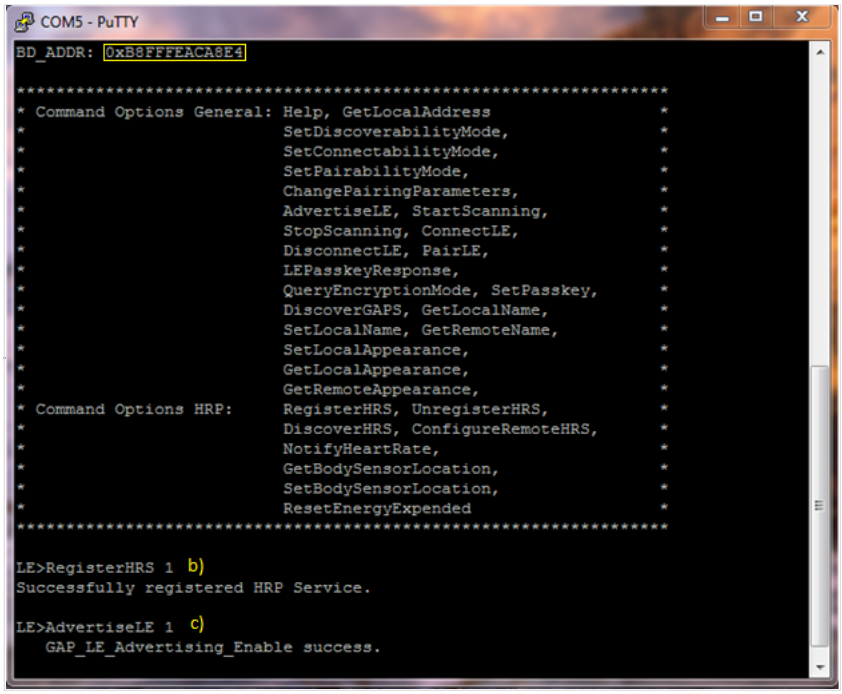
This section provides a description of how to use the demo application to connect two configured boards and communicate over Bluetooth. The Bluetooth Heart Rate Service (HRS) is a simple Client-Server connection process. We will setup one of the boards as a Server and the other board as a Client. We will then initiate a connection from the Client to the Server. Once connected, we can transmit data between the two devices over Bluetooth.

Server setup on the demo application

a) We will setup the first board as a **Server**. Note the Bluetooth address of the Server; we will later use this to initiate a connection from the Client.

b) Two commands are all that is needed to setup the Server. The first is RegisterHRS, so issue the **RegisterHRS 1** command.

c) Now use the AdvertiseLE command by issuing the **AdvertiseLE 1** command.

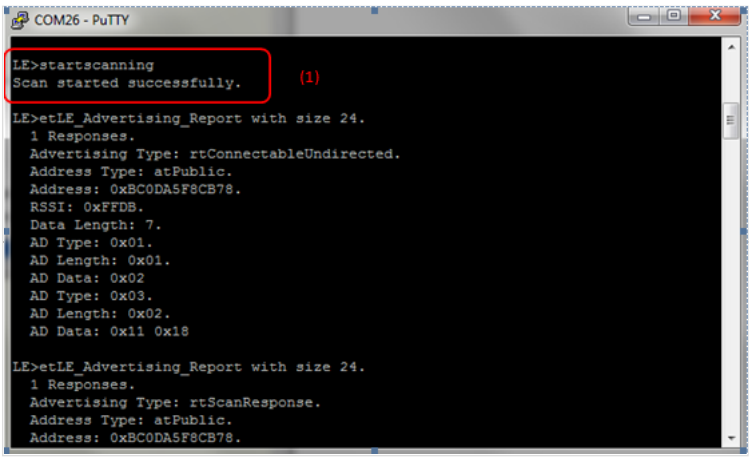


Initiating connection from the Client

Note: **Steps 1 and 2 are optional if you already know the Bluetooth address of the device that you want to connect to.**

1) The Client LE device can try to find which LE devices are in the vicinity issuing the command: **StartScanning**.

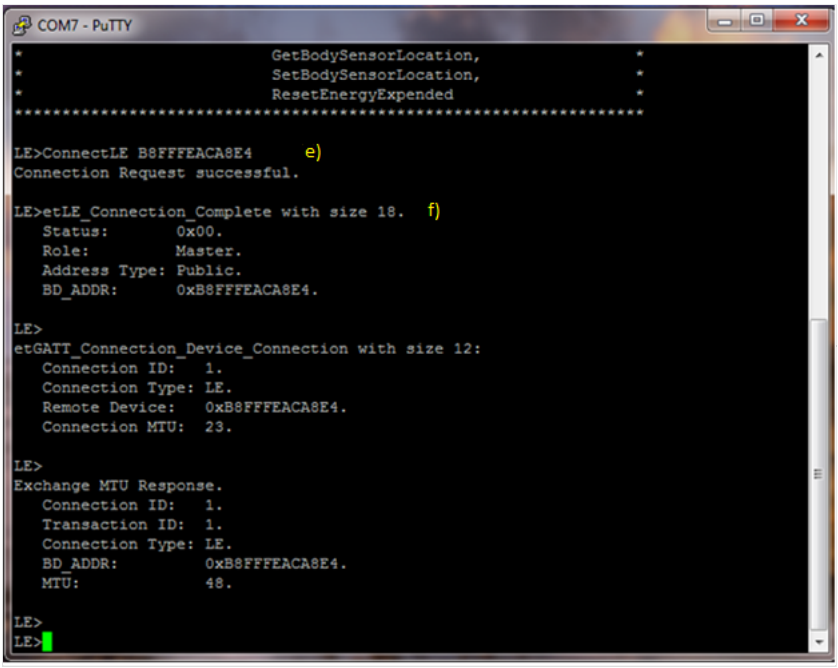
2) Once you have found the device, you can stop scanning by issuing the command: **StopScanning**



d) Retrieve the Bluetooth address of the first board that was configured as a **Server**.

e) Issue a **ConnectLE <BD_ADDR of Server>** command in the Client terminal.

f) When a **Client** successfully connects to a **Server**, both the Client and Server will output **LE_Connection_Complete** and information about the current connection.



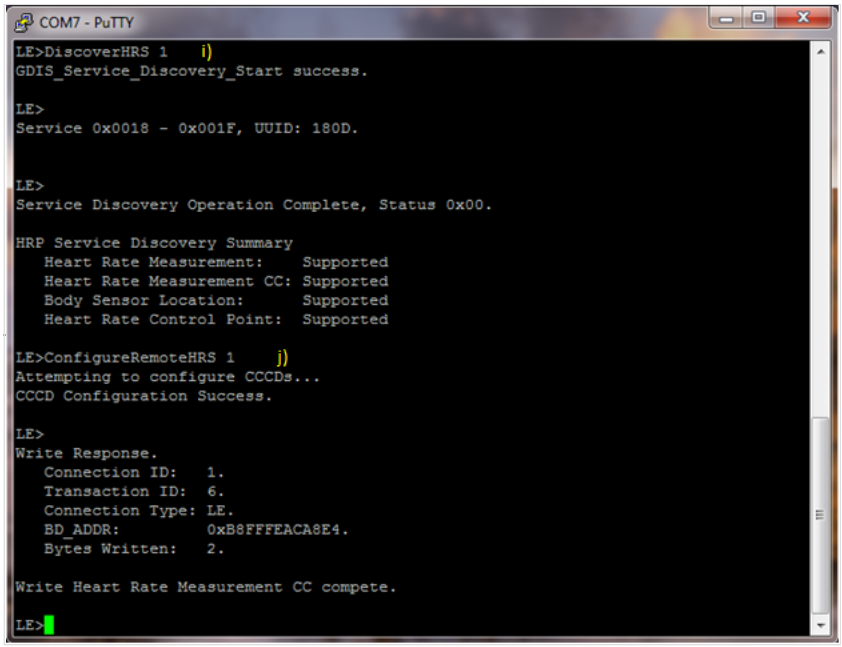
Sending Heart Rate Information between Client and Server

g) Now we have a connection established and both devices are ready to send data to each other.

h) Before heart rate information can be sent we must first initialize commands in the Client terminal.

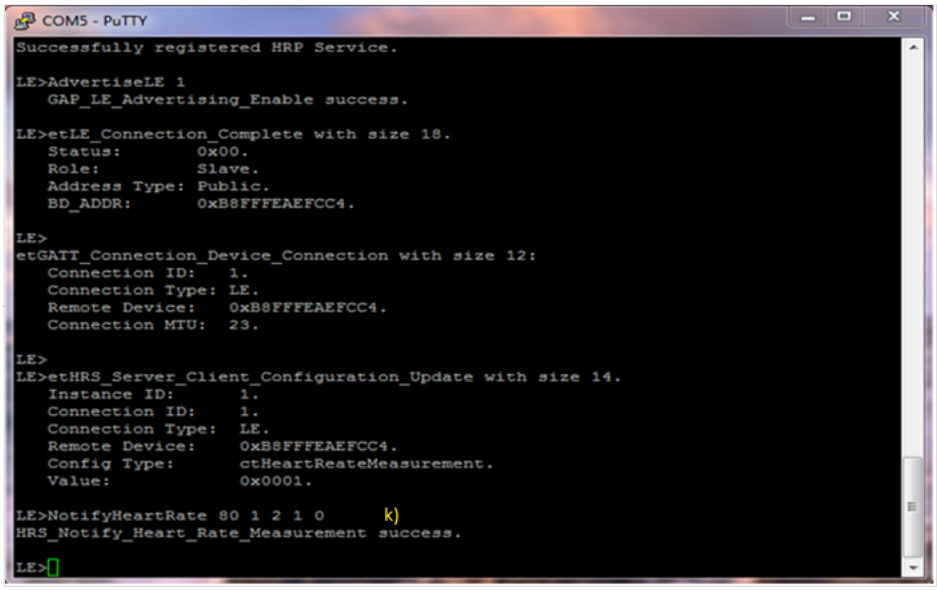
i) The two commands are **DiscoverHRS** and **ConfigureRemoteHRS**, so issue the command **DiscoverHRS 1**. A list of supported services should be displayed.

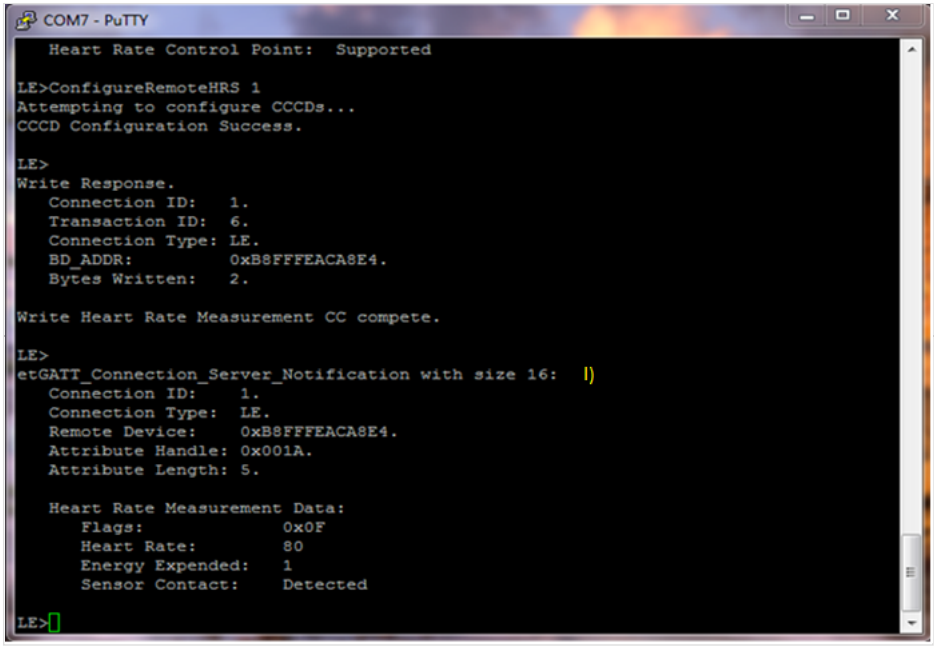
j) Now issue the **ConfigureRemoteHRS 1** command. If the configuration succeeds information about the CCCD configuration will be displayed in the Client and Server terminals.



k) To send information about the heart rate, use the **NotifyHeartRate** command in the Server terminal. The parameters for the NotifyHeartRate command are [HR (BPM)] [HR Format (0 = Byte, 1 = Word)] [Sensor Contact Status (0 = Not Supported, 1 = Supported/Not Detected, 2 = Supported/Detected] [Energy Expended (0 = Don't send, 1 = Send)] [RR Intervals (0 = None, X = Number of Intervals)]. Use the parameters **80 1 2 1 0** after the command.

l) Information about the heart rate should now be sent to the Client and be displayed in the **Client** terminal.





Setting a local appearance on the Server and checking that appearance from the Client.

m) In the **Server** terminal set the type of device with the command **SetLocalAppearance** and the parameter [index]. There are over 20 types of devices in the index, to look at them type first enter only SetLocalAppearance.

n) Now use the **DiscoverGAPS** command in the Client terminal.

o) Lastly enter **GetRemoteAppearance** in the Client terminal to see what type of device the Server had been set to.

Setting the location of the heart rate sensor on the Server and reading it from the Client

p) In the **Server** terminal set the location of the heart rate sensor with the command **SetBodySensorLocation** [type]. Type = 0 – Other, 1 – Chest, 2 – Wrist, 3 – Finger, 4 – Hand, 5 - Ear Lobe, 6 – Foot.

q) To see the location of the heart rate sensor type **GetBodySensorLocation** in the Client terminal.

Application Commands

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

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.

GAP 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.

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).

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
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR
(-56) BTPS_ERROR_GAP_NOT_INITIALIZED
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE

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 on successful execution and a negative value on all errors. 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
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-56) BTPS_ERROR_GAP_NOT_INITIALIZED
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-66) BTPS_ERROR_INSUFFICIENT_RESOURCES
(-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
(-105) BTPS_ERROR_SCAN_ACTIVE

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 StopScan() call
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-56) BTPS_ERROR_GAP_NOT_INITIALIZED
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE

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 on successful execution and a negative value on all errors. 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
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR
(-56) BTPS_ERROR_GAP_NOT_INITIALIZED
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-66) BTPS_ERROR_INSUFFICIENT_RESOURCES
(-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
(-111) BTPS_ERROR_CREATE_CONNECTION_OUTSTANDING
(-116) BTPS_ERROR_RANDOM_ADDRESS_IN_USE
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). This function will return zero if successful, or a negative return error code if there was an error condition. The GAP_LE_Add_Device_To_White_List function is provided to allow the local host the ability to add one (or more) devices to the white list maintained by the local device. This function will attempt to add as many devices as possible (from the specified list) and will return the number of devices added. 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). This function will return zero if successful, or a negative return error code if there was an error condition.

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
This command required one parameter which is the Bluetooth Address of the (currently connected) remote device that is to be disconnected.

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). 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 Pairing.

Possible Return Values
(0) Successfully Set Pairability Mode
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-6) INVALID_PARAMETERS_ERROR
(-56) BTPS_ERROR_GAP_NOT_INITIALIZED
(-66) BTPS_ERROR_INSUFFICIENT_RESOURCES
(-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
(-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
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR
(-56) BTPS_ERROR_GAP_NOT_INITIALIZED
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-66) BTPS_ERROR_INSUFFICIENT_RESOURCES
(-104) BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
(-107) BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
(-118) BTPS_ERROR_PAIRING_NOT_ACTIVE

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
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-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.

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.

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
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-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
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_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
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-59) BTPS_ERROR_ADDING_CALLBACK_INFORMATION

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).

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.

Heart Rate Profile Commands

RegisterHRS

Description
the RegisterHRS command is responsible for registering a HRP Service. This function will return 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.

Possible Return Values
(0) Successfully registered an HRP service
(-4) FUNCTION_ERROR
(-1000) HRS_ERROR_INVALID_PARAMETER
(-1002) HRS_ERROR_INSUFFICIENT_RESOURCES
(-1003) HRS_ERROR_SERVICE_ALREADY_REGISTERED

API Call
HRS_Initialize_Service(BluetoothStackID, HRS_HEART_RATE_CONTROL_POINT_RESET_ENERGY_EXPENDED_SUPPORTED, HRS_EventCallback, NULL, &HRSInstanceID);

API Prototype
int BTPSAPI HRS_Initialize_Service(unsigned int BluetoothStackID, unsigned long Supported_Commands, HRS_Event_Callback_t EventCallback, unsigned long CallbackParameter, unsigned int *ServiceID)

Description of API
This function is responsible for opening a HRS Server. The first parameter is the Bluetooth Stack ID on which to open the Server. The second parameter is the mask of supported Heart Rate Control point commands. The third parameter is the Callback function to call when an event occurs on this Server Port. The fourth parameter is a user-defined callback parameter that will be passed to the callback function with each event. The final parameter is a pointer to store the GATT Service ID of the registered HRS service. This can be used to include the service registered by this call. This function returns the positive, non-zero, Instance ID or a negative error code. Only 1 HRS Server may be open at a time, per Bluetooth Stack ID. The Supported_Commands parameter must be made up of bit masks of the form: HRS_HEART_RATE_CONTROL_POINT_XXX_SUPPORTED All Client Requests will be dispatch to the EventCallback function that is specified by the second parameter to this function.

UnRegisterHRS

Description
The UnRegisterHRS command is responsible for unregistering a HRP Service. This command will return 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.

Possible Return Values
(0) Successfully closed the HRS Server
(-4) FUNCTION_ERROR (HRP Service not registered)
(-1000) HRS_ERROR_INVALID_PARAMETER
(-1004) HRS_ERROR_INVALID_INSTANCE_ID

API Call
HRS_Cleanup_Service(BluetoothStackID, HRSInstanceID);

API Prototype
int BTPSAPI HRS_Cleanup_Service(unsigned int BluetoothStackID, unsigned int InstanceID)

Description of API
This function is responsible for closing a previously opened HRS Server. The first parameter is the Bluetooth Stack ID on which to close the Server. The second parameter is the InstanceID that was returned from a successful call to HRS_Initialize_Service(). This function returns a zero if successful or a negative return error code if an error occurs.

DiscoverHRS

Description
The DiscoverHRS command is responsible for performing a HRP Service Discovery Operation. This command will return a 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

Possible Return Values
(0) Service Discovery Start success
(-4) Function_Error

API Call
GDIS_Service_Discovery_Start(BluetoothStackID, ConnectionID, (sizeof(UUID)/sizeof(GATT_UUID_t)), UUID, GDIS_Event_Callback, sdHRS);

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 module can and should be modified for the customers use. This function is called to start a service discovery operation by the GDIS module.

ConfigureRemoteHRS

Description
The ConfigureRemoteHRS command is responsible for configuring a HRP Service on a remote device. This command will return zero on successful execution and a negative value on all errors.

Parameters
The ConfigureRemoteHRS requires only one parameter, Heart Rate Notify. 0 disables Heart Rate Notify and 1 enables Heart Rate Notify

Command Call Examples
ConfigureRemoteHRS 1 configures the service with Heart Rate Notify enabled.
ConfigureRemoteHRS 0 configures the service with Heart Rate Notify disabled.

Possible Return Values
(0) Successfully configured HRS on the remote device
(-4) FUNCTION_ERROR

NotifyHeartRate

Description
The NotifyHeartRate command is responsible for performing a Heart Rate Measurement notification to a connected remote device. This command will return zero on successful execution and a negative value on all errors.

Parameters
the NotifyHeartRate command requires 5 parameters. The first parameter is the BPM (beats per minute). The second is the Heart Rate format, 0 = Byte, 1 = Word. The third is the Sensor Contact Status, 0 = Not Supported, 1 = Supported/Not Detected, 2 = Supported/Detected. The fourth is Energy Expended, 0 = Don't Send, 1 = Send. The fifth and last is the number of RR Intervals, 0 = None, X = Number of Intervals.

Command Call Examples
NotifyHeartRate 80 1 2 1 0

Possible Return Values
(0) Successfully sent the Heart Rate Measurement
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR
(-1000) HRS_ERROR_INVALID_PARAMETER
(-1002) HRS_ERROR_INSUFFICIENT_RESOURCES
(-1004) HRS_ERROR_INVALID_INSTANCE_ID

API Call
HRS_Notify_Heart_Rate_Measurement(BluetoothStackID, HRSInstanceID, ConnectionID, HeartRatePtr)

API Prototype
Int BTPSAPI HRS_Notify_Heart_Rate_Measurement(unsigned int BluetoothStackID, unsigned int InstanceID, unsigned int ConnectionID, HRS_Heart_Rate_Measurement_Data_t *Heart_Rate_Measurement)

Description of API
This function is responsible for sending a Heart Rate Measurement to a specified remote device. The first parameter is the Bluetooth Stack ID of the Bluetooth Device. The second parameter is the InstanceID returned from a successful call to HRS_Initialize_Server(). The third parameter is the ConnectionID of the remote device to send the notification to. The final parameter is the Heart Rate Measurement data to notify. This function will return zero if the notification was sent successfully or a negative return error code if there was an error condition.

GetBodySensorLocation

Description
The GetBodySensorLocation command is responsible for reading the Body Sensor Location characteristic. It can be executed by a Server or a Client with an open connection to a remote Server. This command will return 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

Possible Return Values
(0) Successfully Set Body Sensor Location
(-4) FUNCTION_ERROR
(-103) BTPS_ERROR_FEATURE_NOT_AVAILABLE
(-1000) HRS_ERROR_INVALID_PARAMETER
(-1004) HRS_ERROR_INVALID_INSTANCE_ID

API Call
HRS_Query_Body_Sensor_Location(BluetoothStackID, HRSInstanceID, &location)

API Prototype
int BTPSAPI HRS_Query_Body_Sensor_Location(unsigned int BluetoothStackID, unsigned int InstanceID, Byte_t *Body_Sensor_Location)

Description of API
This function is responsible for querying the current location of the body sensor for the specified HRS instance. The first parameter is the Bluetooth Stack ID of the Bluetooth Device. The second parameter is the InstanceID returned from a successful call to HRS_Initialize_Server(). The final parameter is a pointer to return the current Body Sensor Location for the specified HRS instance. This function will return zero if the body sensor location was successfully set or a negative return error code if there was an error condition.

SetBodySensorLocation

Description
The SetBodySensorlocation command is responsible for writing the Body Sensor Location characteristic. It can be executed only by a Server. This command will return zero on successful execution and a negative value on all errors.

Parameters
The SetBodySensorLocation command requires one parameter which is the Location of the Heart Rate sensor. This parameter should be an integer between 1 and 6 with each number corresponding to a different location.
0 – Other, 1 – Chest, 2 – Wrist, 3 – Finger, 4 – Hand, 5 - Ear Lobe, 6 – Foot.

Command Call Examples
"SetBodySensorLocation 3" sets the sensor location to Finger.
"SetBodySensorLocation 5" sets the sensor location to Ear lobe.

Possible Return Values
(0) Successfully Set Body Sensor Location
(-4) FUNCTION_ERROR
(-103) BTPS_ERROR_FEATURE_NOT_AVAILABLE
(-1000) HRS_ERROR_INVALID_PARAMETER
(-1004) HRS_ERROR_INVALID_INSTANCE_ID

API Call
HRS_Set_Body_Sensor_Location(BluetoothStackID, HRSInstanceID, (Byte_t)TempParam->Params[0].intParam

API Prototype
Int BTPSAPI HRS_Set_Body_Sensor_Location(unsigned int BluetoothStackID, unsigned int InstanceID, Byte_t Body_Sensor_Location)

Description of API
This function is responsible for setting the location of the body sensor for the specified HRS instance. The Body_Sensor_Location parameter should be an enumerated value of the form HRS_BODY_SENSOR_LOCATION_XXX. This function will return zero if the body sensor location was successfully set or a negative return error code if there was an error condition.

ResetEnergyExpended

Description
The ResetEnergyExpended command is responsible for writing the Reset Energy Expended command to a remote Server Control Point. It can be executed only by a Client. This command will return a 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.

Possible Return Values
(0) successfully reset energy expended (-1000) HRS_ERROR_INVALID_PARAMETER

API Call
HRS_Format_Heart_Rate_Control_Command(ccResetEnergyExpended, HRS_HEART_RATE_CONTROL_POINT_VALUE_LENGTH, CommandBuffer)

API Prototype
int BTPSAPI HRS_Format_Heart_Rate_Control_Command(HRS_Heart_Rate_Control_Command_t Command, unsigned int BufferLength, Byte_t *Buffer)

Description of API

The Format_Heart_Rate_Control_Command function is responsible for formatting a Heart Rate Control Command into a user specified buffer. The first parameter is the command to format. The final two parameters contain the length of the buffer, and the buffer, to format the command into. This function returns a zero if successful or a negative return error code if an error occurs. The BufferLength and Buffer parameter must point to a buffer of at least HRS_HEART_RATE_CONTROL_POINT_VALUE_LENGTH in size. HRS_Heart_Rate_Control_Command_t is a ccResetEnergyExpended command which is placed in the buffer.

{{

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 HRPDemo 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 HRPDemo 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 HRPDemo 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 HRPDemo 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 HRPDemo 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 HRPDemo 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 HRPDemo 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 HRPDemo App** here.*

*For technical su please post you questions at http://e2e.ti.cor Please post on comments abo article **CC256x Bluetooth Sta HRPDemo Ap***

}}

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