CC256x TI Bluetooth Stack HFPDemo 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)

Contents

Demo Overview

The Hands free profile allows the user to demonstrate the use of Hands-free profile on an embedded device. The hands free profile is used to connect a headset or speakerphone with a mobile device to provide remote control and voice connections. The Hands-free profile supports two roles, Hands free and Audio Gateway. This document demonstrates how to use the hands free role of the profile.

Note: ***An external codec MUST be connected to the CC256x I2S/PCM interface to play and record audio.***

Note: **The same instructions can be used to run this demo on the MSP430 Platform.**

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)** 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.

Demo Application

- This section provides a description of how to use the demo application to connect two configured board and communicate over Bluetooth. Bluetooth HFP is a simple Client-Server connection process with one side operating in the Audio-Gateway role and the other operating in the Handsfree role. We will setup one of the boards as a Handsfree server and use an android phone as Client. Once connected, we can transmit data between the two devices over Bluetooth.

Server setup on the demo application

a) Perform the steps mentioned earlier in **Running the Bluetooth Code** section to initialize the application.

```
*****************************************************************
* Command Options: Inquiry, DisplayInquiryList, Pair,           *
*                  EndPairing, PINCodeResponse, PassKeyResponse, *
*                  UserConfirmationResponse,                    *
*                  SetDiscoverabilityMode, SetConnectabilityMode,*
*                  SetPairabilityMode,                          *
*                  ChangeSimplePairingParameters,               *
*                  GetLocalAddress, GetLocalName, SetLocalName, *
*                  GetClassOfDevice, SetClassOfDevice,          *
*                  GetRemoteName, OpenHFServer, CloseHFServer   *
*                  ManageAudio, AnswerCall, HangUpCall, Close,  *
*                  Help                                         *
*****************************************************************
```

b) Give a name for the MSP430 using the **SetLocalName**. In our example we give it a name of **hfpserver**.

```
HFRE16>SetLocalName hfpserver
Local Device Name set to: hfpserver.
```

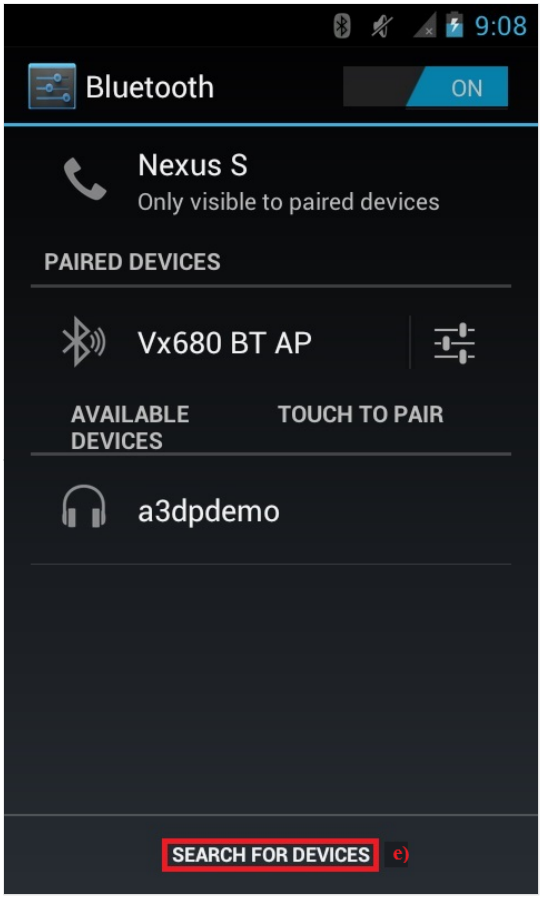
c) Open a HFPServer using the command, **OpenHFServer <port number>**. Here we use **OpenHFServer 1** to open a port.

```
HFRE16>OpenHFServer 1
HFRE_Open_HandsFree_Server_Port: Function Successful.
HFRE_Register_HandsFree_SDP_Record: Function Successful.
```

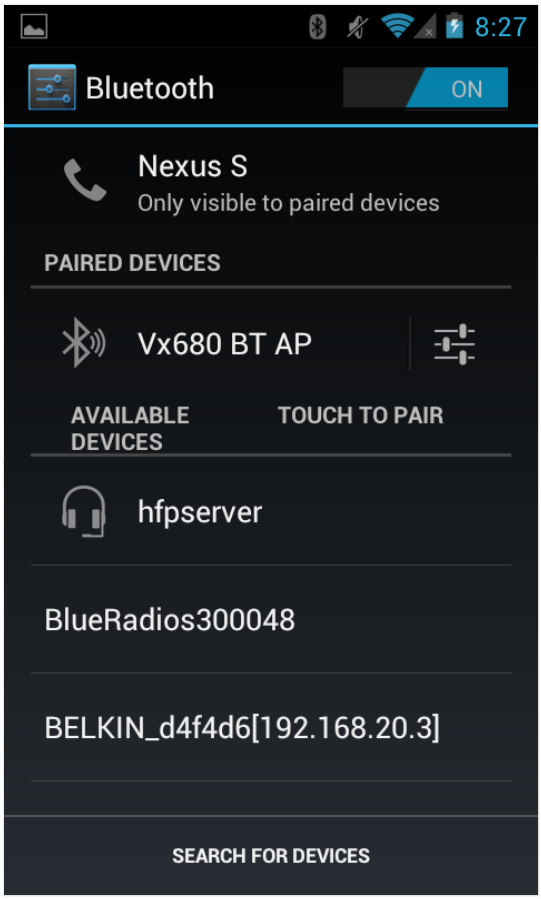
Client setup and device discovery

d) Open the bluetooth settings menu on the android phone **Settings->Bluetooth** . We should get a menu like this.

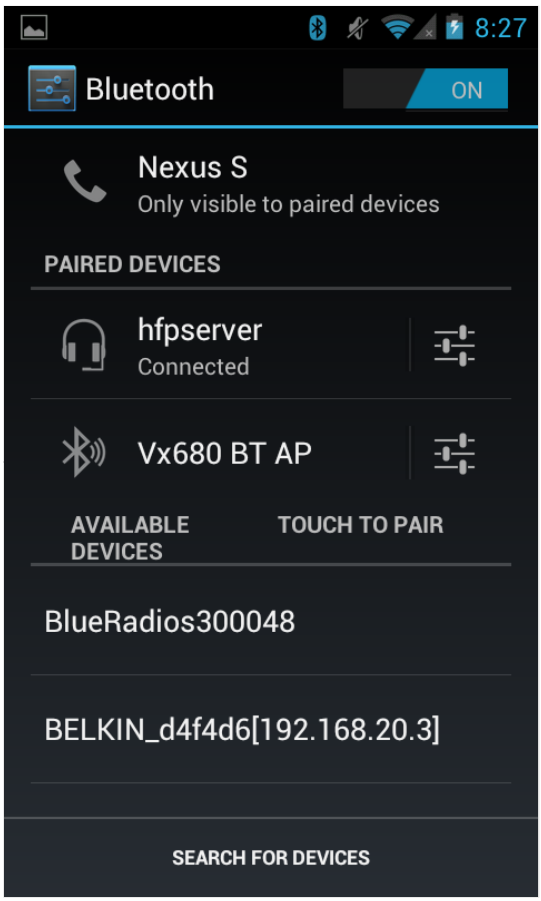
e) Hit on **search for devices**. The phone should begin looking for other bluetooth devices.



f) A Tiva **HFPDemo** Demo should appear like shown below in the picture. Click on the device to begin pairing.



g) After the devices are paired, the device should show connected on the phone side and on the Tiva.



```
HFRE16>
HFRE Open Port Indication, ID: 0x0001, Board: 0x00027232591C.

HFRE16>
HFRE Control Indicator Status Confirmation, ID: 0x0001, Description: SERVICE, Va
lue: FALSE.

HFRE16>
HFRE Control Indicator Status Confirmation, ID: 0x0001, Description: CALL, Value
: FALSE.

HFRE16>
HFRE Control Indicator Status Confirmation, ID: 0x0001, Description: CALL_SETUP,
 Value: 0.

HFRE16>
HFRE Control Indicator Status Confirmation, ID: 0x0001, Description: CALLSETUP,
Value: 0.

HFRE16>
HFRE Control Indicator Status Confirmation, ID: 0x0001, Description: CALLHELD, V
alue: FALSE.

HFRE16>
HFRE Control Indicator Status Confirmation, ID: 0x0001, Description: SIGNAL, Val
ue: 0.

HFRE16>
HFRE Control Indicator Status Confirmation, ID: 0x0001, Description: ROAM, Value
: FALSE.

HFRE16>
HFRE Open Service Level Connection Indication, ID: 0x0001
                    RemoteSupportedFeaturesValid: TRUE
                              RemoteSupportedFeatures: 0x00000511X
                  RemoteCallHoldMultipartySupport: 0x00000000X
HFRE_Enable Call Line Identification
```

HandsFree Options

h) To Manage the audio connection to the headset. We use **ManageAudio 0** to release the SCO connection and **ManageAudio 1** to Setup the SCO connection.

```
HFRE16>ManageAudio 1
HFRE_Setup_Audio_Connection: Function Successful.

HFRE16>
HFRE Audio Connection Indication, ID: 0x0001, Status: 0x0000.
```

```
HFRE16>ManageAudio 0
HFRE_Release_Audio_Connection: Function Successful.
```

i) To Answer an Incoming Call use **AnswerCall** and to hang up a call use the **HangUpCall** command.

```
HFRE16>AnswerCall
HFRE_Answer_Incoming_Call: Function Successful.

HFRE16>
HFRE Command Result, ID: 0x0001, Type 0 Code 0.
```

```
HFRE16>HangUpCall
HFRE_Hang_Up_Call: Function Successful.

HFRE16>
HFRE Command Result, ID: 0x0001, Type 0 Code 0.
```

j) To Close the HFserver use the **CloseHFServer** command.

```
HFRE16>CloseHFServer
Server Closed.
```

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.

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**) describes all of the API's in detail.

Generic Access Profile 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 DisplayHelp command will display the Command Options menu. Depending on the UI_MODE of the device (Server or Client), different commands will be used in certain situations. The Open and Close commands change their use depending on the mode the device is in.

Parameters

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

Possible Return Values

The return value is always 0

Inquiry

Description

The Inquiry command is responsible for performing a General Inquiry for discovering Bluetooth Devices. The command requires that a valid Bluetooth Stack ID exists before running. This command returns zero on a successful call or a negative value if an error occurred during execution. The inquiry will last 10 seconds unless 20 devices (MAX_INQUIRY_RESULTS) are found before that time limit.

Parameters

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

Possible Return Values

(0) Successful Inquiry Procedure
(-1) BTPS_ERROR_INVALID_PARAMETER
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-58) BTPS_ERROR_INVALID_MODE

API Call

GAP_Perform_Inquiry(BluetoothStackID, itGeneralInquiry, 0, 0, 10, MAX_INQUIRY_RESULTS, GAP_Event_Callback, (unsigned long) NULL);

API Prototype

int BTPSAPI GAP_Perform_Inquiry(unsigned int BluetoothStackID, GAP_Inquiry_Type_t GAP_Inquiry_Type, unsigned int MinimumPeriodLength, unsigned int MaximumPeriodLength, unsigned int InquiryLength, unsigned int MaximumResponses, GAP_Event_Callback_t GAP_Event_Callback, unsigned long CallbackParameter);

Description of API

This function is provided to allow a mechanism for starting an Inquiry Scan Procedure. The first parameter to this function is the Bluetooth Protocol Stack of the Bluetooth Device that is to perform the Inquiry. The second parameter is the type of Inquiry to perform. The third and fourth parameters are the Minimum and Maximum Period Lengths which are specified in seconds (only valid in case a Periodic Inquiry is to be performed). The fifth parameter is the Length of Time to perform the Inquiry, specified in seconds. The sixth parameter is the Number of Responses to wait for. The final two parameters represent the Callback Function (and parameter) that is to be called when the specified Inquiry has completed. This function returns zero is successful, or a negative return error code if an Inquiry was unable to be performed. Only ONE Inquiry can be performed at any given time. Calling this function with an outstanding Inquiry is in progress will fail. The caller can call the GAP_Cancel_Inquiry() function to cancel a currently executing Inquiry procedure. The Minimum and Maximum Inquiry Parameters are optional and, if specified, represent the Minimum and Maximum Periodic Inquiry Periods. The called should set BOTH of these values to zero if a simple Inquiry Procedure is to be used (Non-Periodic). If these two parameters are specified, then these two parameters must satisfy the following formula:
MaximumPeriodLength > MinimumPeriodLength > InquiryLength

Pair

Description

The Pair command is responsible for initiating bonding with a remote Bluetooth Device. The function returns zero on a successful execution and a negative value on all errors. A Bluetooth Stack ID must exist before attempting to pair and the device must not already be connected to any device (including the one it tries to pair with). It is also important to note that the use of the Inquiry command before calling Pair is necessary to connect to a remote device. Both General and Dedicated bonding are supported.

Parameters

The Pair command requires one or two parameters with specific values in order to work successfully. The first parameter is the Inquiry Index of the remote Bluetooth Device. This parameter is always necessary. This can be found after an Inquiry or displayed when the command DisplayInquiryList is used. If the desired remote device does not appear in the list, it cannot be paired with. The second parameter is the bonding type used for the pairing procedure. It is an optional parameter which is only required if General Bonding is desired for the connection. This must be specified as either 0 (for Dedicated Bonding) or 1 (for General Bonding). If only one parameter is given, the Bonding Type will be Dedicated Bonding.

Command Call Examples

"Pair 5 0" Attempts to pair with the remote device at the fifth Inquiry Index using Dedicated Bonding.
"Pair 5" Is the exact same as the above example. If no parameters, the Bonding Type will be Dedicated.
"Pair 8 1" Attempts to pair with the remote device at the eighth Inquiry Index using General Bonding.

Possible Return Values

(0) Successful Pairing
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-1) BTPS_ERROR_INVALID_PARAMETER
(-59) BTPS_ERROR_ADDING_CALLBACK_INFORMATION
(-8) BTPS_ERROR_DEVICE_HCI_ERROR

API Call

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

API Prototype

int BTPSAPI GAP_Initiate_Bonding(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Bonding_Type_t GAP_Bonding_Type, GAP_Event_Callback_t GAP_Event_Callback, unsigned long CallbackParameter);

Description of API

This function is provided to allow a means to Initiate a Bonding Procedure. This function can perform both General and Dedicated Bonding based upon the type of Bonding requested. This function accepts as input, the Bluetooth Protocol Stack ID of the Local Bluetooth device that is perform the Bonding, the Remote Bluetooth address of the Device to Bond with, the type of bonding to perform, and the GAP Event Callback Information that will be used to handle Authentication Events that will follow if this function is successful. If this function is successful, then all further information will be returned through the Registered GAP Event Callback. It should be noted that if this function returns success that it does NOT mean that the Remote Device has successfully Bonded with the Local Device, ONLY that the Remote Device Bonding Process has been started. This function will only succeed if a Physical Connection to the specified Remote Bluetooth device does NOT already exist. This function will connect to the Bluetooth device and begin the Bonding Process. If General Bonding is specified, then the Link is maintained, and will NOT be terminated until the GAP_End_Bonding function has been called. This will allow any higher level initialization that is needed on the same physical link. If Dedicated Bonding is performed, then the Link is terminated automatically when the Authentication Process has completed.Due to the asynchronous nature of this process, the GAP Event Callback that is specified will inform the caller of any Events and/or Data that is part of the Authentication Process. The GAP_Cancel_Bonding function can be called at any time to end the Bonding Process and terminate the link (regardless of which Bonding method is being performed).When using General Bonding, if an L2CAP Connection is established over the Bluetooth Link that was initiated with this function, the Bluetooth Protocol Stack MAY or MAY NOT terminate the Physical Link when (and if) an L2CAP Disconnect Request (or Response) is issued. If this occurs, then calling the GAP_End_Bonding function will have no effect (the GAP_End_Bonding function will return an error code in this case).

EndPairing

Description

The EndPairing command is responsible for ending a previously initiated bonding session with a remote device. The function returns zero on a successful execution and a negative value on all errors. A Bluetooth Stack ID must exist before attempting to end pairing and the device must already be connected to a remote device. It is also important to note that the use of the Pair and Inquiry commands before calling EndPairing are necessary to disconnect from a remote device.

Parameters

The EndPairing 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. It should be the same value as the first parameter used in the Pair command, unless a new Inquiry has been called after pairing. If this is the case, find the Bluetooth Address of the device used in the Pair command.

Command Call Examples

"EndPairing 5" Attempts to end pairing with the remote device at the fifth Inquiry Index.
"EndPairing 8" Attempts to end pairing with the remote device at the eighth Inquiry Index.

Possible Return Values

(0) Successful End Pairing
(-2)BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-1)BTPS_ERROR_INVALID_PARAMETER
(-58)BTPS_ERROR_INVALID_MODE
(-4) FUNCTION_ERROR
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR

API Call

GAP_End_Bonding(BluetoothStackID, InquiryResultList[(TempParam->Params[0].intParam – 1)]);

API Prototype

int BTPSAPI GAP_Initiate_Bonding(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Bonding_Type_t GAP_Bonding_Type, GAP_Event_Callback_t GAP_Event_Callback, unsigned long CallbackParameter);

Description of API

This function is provided to allow a means to terminate a connection that was established via a call to the GAP_Initiate_Bonding function (that specified general bonding as the bonding type to perform). This function has NO effect if the bonding procedure was initiated using dedicated bonding (or the device is already disconnected). This function accepts the Bluetooth device address of the remote Bluetooth device that was specified to be bonded with (general bonding). This function terminates the ACL connection that was established and it guarantees that NO GAP Event Callbacks will be issued to the GAP Event Callback that was specified in the original GAP_Initiate_Bonding function call (if this function returns success).

PINCodeResponse

Description

The PINCodeResponse command is responsible for issuing a GAP Authentication Response with a PIN Code value specified via the input parameter. 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 function. The device must also be in the middle of an on-going Pairing operation that was started by the local device or a remote device.

Parameters

The PINCodeResponse command requires one parameter which is the PIN Code used for authenticating the connection. This is a string value which can be up to 16 digits long. The initiator of the Pairing will see a message displayed during the Pairing Procedure to call this command. A responder will receive a message to call this command after the initiator has put in the PIN Code.

- **Command Call Examples**

"PINCodeResponse 1234" Attempts to set the PIN Code to "1234."

"PINCodeResponse 5921302312564542 Attempts to set the PIN Code to "5921302312564542." This value represents the longest PIN Code value of 16 digits.

Possible Return Values

(0) Successful PIN Code 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

(-57) BTPS_ERROR_DEVICE_HCI_ERROR

API Call

GAP_Authentication_Response(BluetoothStackID, CurrentRemoteBD_ADDR, &GAP_Authentication_Information);

API Prototype

*int BTPSAPI GAP_Authentication_Response(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Authentication_Information_t *GAP_Authentication_Information);*

Description of API

This function is provided to allow a mechanism for the local device to respond to GAP 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).

PassKeyResponse

Description

The PassKeyResponse command is responsible for issuing a GAP Authentication Response with a Pass Key value via the input parameter. 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 function. The device must also be in the middle of an on-going Pairing operation that was started by the local device or a remote device.

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

(-57) BTPS_ERROR_DEVICE_HCI_ERROR

API Call

GAP_Authentication_Response(BluetoothStackID, CurrentRemoteBD_ADDR, &GAP_Authentication_Information);

API Prototype

*int BTPSAPI GAP_Authentication_Response(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Authentication_Information_t *GAP_Authentication_Information);*

Description of API

This function is provided to allow a mechanism for the local device to respond to GAP 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).

UserConfirmationResponse

Description

The UserConfirmationResponse command is responsible for issuing a GAP Authentication Response with a User Confirmation value via the input parameter. 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 function. The device must also be in the middle of an on-going Pairing operation that was started by the local device or a remote device.

- **Parameters**

  The UserConfirmationResponse command requires one parameter which is the User Confirmation value used for authenticating the connection. This is an integer value that must be either 1, to confirm the connection, or 0 to NOT confirm the Authentication and stop the Pairing Procedure.

  Command Call Examples

  "UserConfirmationResponse 0" Attempts to decline the connection made with a remote Bluetooth Device and cancels the Authentication Procedure.

  "UserConfirmationResponse 1" Attempts to accept the connection made with a remote Bluetooth Device and confirm the Authentication Procedure.

  Possible Return Values

  (0) Successful User Confirmation 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
  (-57) BTPS_ERROR_DEVICE_HCI_ERROR

  API Call

  GAP_Authentication_Response(BluetoothStackID, CurrentRemoteBD_ADDR, &GAP_Authentication_Information);

  API Prototype

  *int BTPSAPI GAP_Authentication_Response(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Authentication_Information_t *GAP_Authentication_Information);*

  Description of API

  This function is provided to allow a mechanism for the local device to respond to GAP 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).

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. A Bluetooth Stack ID must exist before attempting to call this function. If setting the device as Limited Discoverable, the device will be discoverable for 60 seconds; a General Discoverable device will always be discoverable.

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
(-4) FUNCTION_ERROR
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-5) BTPS_ERROR_GAP_NOT_INITIALIZED
(-58) BTPS_ERROR_INVALID_MODE
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-64) BTPS_ERROR_INTERNAL_ERROR
(-1) BTPS_ERROR_INVALID_PARAMETER

API Call

GAP_Set_Discoverability_Mode(BluetoothStackID, DiscoverabilityMode, (DiscoverabilityMode == dmLimitedDiscoverableMode)?60:0);

API Prototype

int BTPSAPI GAP_Set_Discoverability_Mode(unsigned int BluetoothStackID, GAP_Discoverability_Mode_t GAP_Discoverability_Mode, unsigned int Max_Discoverable_Time);

Description of API

This function is provided to set the discoverability mode of the local Bluetooth device specified by the Bluetooth Protocol Stack that is specified by the Bluetooth protocol stack ID. The second parameter specifies the discoverability mode to place the local Bluetooth device into, and the third parameter species the length of time (in seconds) that the local Bluetooth device is to be placed into the specified discoverable mode (if mode is not specified as non-discoverable). At the end of this time (provided the time is not infinite), the local Bluetooth device will return to non-discoverable mode.

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. A Bluetooth Stack ID must exist before attempting to call this function.

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-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
(-4) FUNCTION_ERROR
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-5) BTPS_ERROR_GAP_NOT_INITIALIZED
(-58) BTPS_ERROR_INVALID_MODE
(-57) BTPS_ERROR_DEVICE_HCI_ERROR

API Call

GAP_Set_Connectability_Mode(BluetoothStackID, ConnectableMode);

API Prototype

int BTPSAPI GAP_Set_Connectability_Mode(unsigned int BluetoothStackID, GAP_Connectability_Mode_t GAP_Connectability_Mode);

Description of API

This function is provided to set the connectability mode of the local Bluetooth device specified by the Bluetooth protocol stack that is specified by the Bluetooth protocol stack ID. The second parameter specifies the connectability mode to place the local Bluetooth device into.

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. A Bluetooth Stack ID must exist before attempting to call this function.

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 Secure Simple Pairing).

Command Call Examples

"SetPairabilityMode 0" Attempts to set the Pairability Mode of the Local Device to Non-Pairable.
"SetPairabilityMode 1" Attempts to set the Pairability Mode of the Local Device to Pairable.
"SetPairabilityMode 2" Attempts to set the Pairability Mode of the Local Device to Secure Simple Pairing.

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
(-5) BTPS_ERROR_GAP_NOT_INITIALIZED
(-58) BTPS_ERROR_INVALID_MODE

API Call

GAP_Set_Pairability_Mode(BluetoothStackID, PairabilityMode);

API Prototype

int BTPSAPI GAP_Set_Pairability_Mode(unsigned int BluetoothStackID, GAP_Pairability_Mode_t GAP_Pairability_Mode);

Description of API

This function is provided to set the pairability mode of the local Bluetooth device. The second parameter specifies the pairability mode to place the local Bluetooth device into. If secure simple pairing (SSP) pairing mode is specified, then SSP *MUST* be used for all pairing operations. The device can be placed into non pairable mode after this, however, if pairing is re-enabled, it *MUST* be set to pairable with SSP enabled.

ChangeSimplePairingParameters

Description

The ChangeSimplePairingParameters command is responsible for changing the Secure Simple Pairing Parameters that are exchanged during the Pairing procedure whenSecure Simple Pairing (Security Level 4) is used. 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 function. The IOCapability and MITMProtection values are stored in static global variables which are used for Secure Simple Pairing.

Parameters

This command requires two parameters which are the I/O Capability and the MITM Requirement. The first parameter must be specified as 0 (for Display Only), 1 (for Display Yes/No), 2 (for Keyboard Only), or 3 (for No Input/Output). The second parameter must be specified as 0 (for No MITM) or 1 (for MITM required).

Command Call Examples

"ChangeSimplePairingParameters 3 0" Attempts to set the I/O Capability to No Input/Output and turns off MITM Protection.
"ChangeSimplePairingParameters 2 1" Attempts to set the I/O Capability to Keyboard Only and activates MITM Protection.
"ChangeSimplePairingParameters 1 1" Attempts to set the I/O Capability to Display Yes/No and activates MITM Protection.

Possible Return Values

(0) Successfully Pairing Parameters Change
(-6) INVALID_PARAMETERS_ERROR
(-8) INVALID_STACK_ID_ERROR

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 function.

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
(-8) INVALID_STACK_ID_ERROR
(-4) FUNCTION_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).

SetLocalName

Description

The SetLocalName command is responsible for setting the name of the local Bluetooth Device to a specified name. 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 function.

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.

GetLocalName

Description
This function is responsible for querying the name 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 function.

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).

SetClassOfDevice

Description
The SetClassOfDevice command is responsible for setting the Class of Device of the local Bluetooth Device to a Class of Device value. 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 function.

Parameters
The only parameter needed is the new Class of Device value. It is preferred to start the value with “0x” and use a six digit value after that. Without doing this, the Class of Device written will be assumed decimal and will be converted to hexadecimal format and change the values given.

Command Call Examples
"SetClassOfDevice 0x123456" Attempts to set the Class of Device for the local Bluetooth Device to "0x123456."
"SetClassOfDevice 123456" Attempts to set the Class of Device for the local Bluetooth Device to "0x01E240" which is equivalent to the decimal value of 123456.

Possible Return Values
(0) Successfully Set Local Class of Device
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

(-8) INVALID_STACK_ID_ERROR
(-4) FUNCTION_ERROR
(-5) BTPS_ERROR_GAP_NOT_INITIALIZED

API Call
GAP_Set_Class_of_Device(BluetoothStackID, Class_of_Device);

API Prototype
int BTPSAPI GAP_Set_Class_Of_Device(unsigned int BluetoothStackID, Class_of_Device_t Class_of_Device);

Description of API
This function is provided to allow the changing of the class of device of the local Bluetooth device. The Class_of_Device parameter represents the class of device value that is to be written to the local Bluetooth device. This function will return zero if the class of device was successfully changed, or a negative return error code if there was an error condition.

GetClassOfDevice

Description
The GetClassOfDevice command is responsible for querying the Bluetooth Class of Device 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 function.

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 Class of Device
(-57) BTPS_ERROR_DEVICE_HCI_ERROR
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-8) INVALID_STACK_ID_ERROR
(-4) FUNCTION_ERROR
(-1) BTPS_ERROR_INVALID_PARAMETER

API Call
GAP_Query_Class_Of_Device(BluetoothStackID, &Class_of_Device);

API Prototype
*int BTPSAPI GAP_Query_Class_Of_Device(unsigned int BluetoothStackID, Class_of_Device_t *Class_of_Device);*

Description of API
This function is responsible for querying (and reporting) the class of device of the local Bluetooth device. The second parameter is a pointer to a class of device buffer that is to receive the Bluetooth class of device of the local device. If this function is successful, this function returns zero, and the buffer that Class_Of_Device points to will be filled with the Class of Device read from the local Bluetooth device. If there is an error, this function returns a negative value, and the class of device of the local Bluetooth device is NOT copied into the specified input buffer.

GetRemoteName

Description
The GetRemoteName command is responsible for querying the Bluetooth Device Name of a Remote Device. This function 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).

Handsfree Profile Commands

OpenHFServer

Description
The following function is responsible for opening a Serial Port Server on the Local Device. This function opens the Serial Port Server on the specified RFCOMM Channel. This function returns zero if successful, or a negative return value if an error occurred.
Parameters
The command only requires one parameter. This parameter is the Port Number.

Command Call Examples
"OpenHFServer 4" Attempts to Open a Handsfree Server at Port Number #4.
"OpenHFServer 1" Attempts to Open a Handsfree Server at Port Number #1.
Possible Return Values
(0) A3DP Endpoint opened successfully
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-1000) BTHFRE_ERROR_INVALID_PARAMETER
(-1001) BTHFRE_ERROR_NOT_INITIALIZED
(-1002) BTHFRE_ERROR_INVALID_BLUETOOTH_STACK_ID
(-1003) BTHFRE_ERROR_LIBRARY_INITIALIZATION_ERROR
(-1004) BTHFRE_ERROR_INSUFFICIENT_RESOURCES
(-1005) BTHFRE_ERROR_INVALID_OPERATION
(-1006) BTHFRE_ERROR_INVALID_CODEC_ID
API Call
HFRE_Open_HandsFree_Server_Port(BluetoothStackID, TempParam->Params[0].intParam, HFRE_SUPPORTED_FEATURES, 0, NULL, HFRE_Event_Callback, (unsigned long)0)
API Prototype
int BTPSAPI HFRE_Open_HandsFree_Server_Port(unsigned int BluetoothStackID, unsigned int ServerPort, unsigned long SupportedFeaturesMask, unsigned int NumberAdditionalIndicators, char *AdditionalSupportedIndicators[], HFRE_Event_Callback_t EventCallback, unsigned long CallbackParameter)
Description of API
The following function is responsible for Opening a Hands-Free Server on the specified Bluetooth SPP Serial Port. This function accepts as input the Bluetooth Stack ID of the Bluetooth Stack Instance to use for the Hands-Free Server, the Local Serial Port Server Number to use, a bit mask that specifies the features in which the Hands-Free unit supports, the Number of Additional Indicators, a list of Additional Indicators to Support, and the HFRE Event Callback function (and parameter) to associate with the specified Hands-Free Port. The ServerPort parameter *MUST* be between SPP_PORT_NUMBER_MINIMUM and SPP_PORT_NUMBER_MAXIMUM. This function returns a positive, non-zero, value if successful or a negative return error code if an error occurs. A successful return code will be a HFRE Port ID that can be used to reference the Opened HFRE Port in ALL other functions in this module except for the HFRE_Register_Audio_Gateway_SDP_Record() function which is specific to an Audio Gateway Server NOT a Hands-Free Server. Once a Server HFRE Port is opened, it can only be Un-Registered via a call to the HFRE_Close_Server_Port() function (passing the return value from this function). The HFRE_Close_Port() function can be used to Disconnect a Client from the Server Port (if one is connected, it will NOT Un-Register the Server Port however).

- NOTE * The Mandatory Hands-Free Indicators (call, service, and call_setup) are automatically added to the list and need not be specified as additional indicators.

CloseHFServer

Description
The following function is responsible for closing a Serial Port Server that was previously opened via a successful call to the OpenServer() function. This function returns zero if successful or a negative return error code if there was an error.
Parameters
It is not necessary to include parameters when using this command. A parameter will have no effect on the outcome of CloseHFServer.

- **Possible Return Values**

(0) A3DP Endpoint opened successfully
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-1000) BTHFRE_ERROR_INVALID_PARAMETER
(-1001) BTHFRE_ERROR_NOT_INITIALIZED
(-1002) BTHFRE_ERROR_INVALID_BLUETOOTH_STACK_ID
(-1003) BTHFRE_ERROR_LIBRARY_INITIALIZATION_ERROR
(-1004) BTHFRE_ERROR_INSUFFICIENT_RESOURCES
(-1005) BTHFRE_ERROR_INVALID_OPERATION
(-1006) BTHFRE_ERROR_INVALID_CODEC_ID

API Call

HFRE_Close_Server_Port(BluetoothStackID, HFServerPortID)

API Prototype

int BTPSAPI HFRE_Close_Server_Port(unsigned int BluetoothStackID, unsigned int HFREPortID)

Description of API

The following function is responsible for Un-Registering a HFRE Port Server (which was Registered by a successful call to either the HFRE_Open_HandsFree_Server_Port() or the HFRE_Open_Audio_Gateway_Server_Port() function). This functionaccepts as input the Bluetooth Stack ID of the Bluetooth Protocol Stack that the HFRE Port specified by the Second Parameter is valid for. This function returns zero if successful, or a negative return error code if an error occurred (see BTERRORS.H). Note that this function does NOT delete any SDP Service Record Handles.

ManageAudio

Description

The following function is responsible for setting up or releasing an audio connection. This function returns zero on successful execution and a negative value on all errors.

Parameters

The Manage Audio command requires only one parameter to which is is an integer value that represents the ManageAudio mode. This value must be specified as 0 (for Release) or 1 (for Setup)

Command Call Examples

"ManageAudio 0" Attempts to Release the Audio Connection.
"ManageAudio 1" Attempts to Setup the Audio Connection.

Possible Return Values

(0) A3DP Endpoint opened successfully
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-1000) BTHFRE_ERROR_INVALID_PARAMETER
(-1001) BTHFRE_ERROR_NOT_INITIALIZED
(-1002) BTHFRE_ERROR_INVALID_BLUETOOTH_STACK_ID
(-1003) BTHFRE_ERROR_LIBRARY_INITIALIZATION_ERROR
(-1004) BTHFRE_ERROR_INSUFFICIENT_RESOURCES
(-1005) BTHFRE_ERROR_INVALID_OPERATION
(-1006) BTHFRE_ERROR_INVALID_CODEC_ID

API Call

HFRE_Setup_Audio_Connection(BluetoothStackID, HFServerPortID)
or HFRE_Release_Audio_Connection(BluetoothStackID, HFServerPortID)

API Prototype

int BTPSAPI HFRE_Setup_Audio_Connection(unsigned int BluetoothStackID, unsigned int HFREPortID)
or int BTPSAPI HFRE_Release_Audio_Connection(unsigned int BluetoothStackID, unsigned int HFREPortID)

Description of API

This function is responsible for Setting Up an Audio Connection between the Local and Remote Device. This function may be used by either an Audio Gateway or a Hands-Free unit for which a valid Service Level Connection Exists. This function accepts as its input parameters the Bluetooth Stack ID for which the HFRE Port ID is valid as well as the HFRE Port ID. This function returns zero if successful or a negative return error code if there was an error.

(or) This function is responsible for Releasing an Audio Connection which was previously established by the Remote Device or by a call to the HFRE_Setup_Audio_Connection() function. This function may be used by either an Audio Gateway or a Hands-Free unit. This function accepts as its input parameters the Bluetooth Stack ID for which the HFRE Port ID is valid as well as the HFRE Port ID. This function returns zero if successful or a negative return error code if there was an error.

AnswerCall

Description

The following function is responsible for sending the command to Anwser an Incoming Call on the Remote Audio Gateway. This function 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 AnswerCall .

Possible Return Values

(0) A3DP Endpoint opened successfully

- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
  (-4) FUNCTION_ERROR
  (-8) INVALID_STACK_ID_ERROR
  (-1000) BTHFRE_ERROR_INVALID_PARAMETER
  (-1001) BTHFRE_ERROR_NOT_INITIALIZED
  (-1002) BTHFRE_ERROR_INVALID_BLUETOOTH_STACK_ID
  (-1003) BTHFRE_ERROR_LIBRARY_INITIALIZATION_ERROR
  (-1004) BTHFRE_ERROR_INSUFFICIENT_RESOURCES
  (-1005) BTHFRE_ERROR_INVALID_OPERATION
  (-1006) BTHFRE_ERROR_INVALID_CODEC_ID

API Call

HFRE_Answer_Incoming_Call(BluetoothStackID, HFServerPortID)

API Prototype

int BTPSAPI HFRE_Answer_Incoming_Call(unsigned int BluetoothStackID, unsigned int HFREPortID)

Description of API

This function is responsible for sending the command to Answer an Incoming call on a Remote Audio Gateway. This function may only be performed by Hands-Free units for which a valid Service Level Connection exists. This function accepts as its input parameters the Bluetooth Stack ID for which the HFRE Port ID is valid as well as the HFRE Port ID. This function return zero if successful or a negative return error code if there was an error.

HangupCall

Description

The following function is responsible for sending the command to HangUp ongoing calls or reject incoming calls on the Remote Audio Gateway. This function 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 HangupCall

Possible Return Values

(0) A3DP Endpoint opened successfully
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-1000) BTHFRE_ERROR_INVALID_PARAMETER
(-1001) BTHFRE_ERROR_NOT_INITIALIZED
(-1002) BTHFRE_ERROR_INVALID_BLUETOOTH_STACK_ID
(-1003) BTHFRE_ERROR_LIBRARY_INITIALIZATION_ERROR
(-1004) BTHFRE_ERROR_INSUFFICIENT_RESOURCES
(-1005) BTHFRE_ERROR_INVALID_OPERATION
(-1006) BTHFRE_ERROR_INVALID_CODEC_ID

API Call

HFRE_Hang_Up_Call(BluetoothStackID, HFServerPortID)

API Prototype

int BTPSAPI HFRE_Hang_Up_Call(unsigned int BluetoothStackID, unsigned int HFREPortID)

Description of API

This function is responsible for sending a Hang-Up Command to the Remote Audio Gateway. This function may be used to Reject an incoming call, or to terminate an ongoing call. This function may only be performed by Hands-Free units for which a valid Service Level Connection exists. This function accepts as its input parameters the Bluetooth Stack ID for which the HFRE Port ID is valid as well as the HFRE Port ID. This function returns zero if successful or a negative return error code if there was an error.

Close

Description

The following function is responsible for closing any open HFP ports. This function 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 Close.

Possible Return Values

(0) A3DP Endpoint opened successfully
(-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-1000) BTHFRE_ERROR_INVALID_PARAMETER
(-1001) BTHFRE_ERROR_NOT_INITIALIZED
(-1002) BTHFRE_ERROR_INVALID_BLUETOOTH_STACK_ID
(-1003) BTHFRE_ERROR_LIBRARY_INITIALIZATION_ERROR
(-1004) BTHFRE_ERROR_INSUFFICIENT_RESOURCES
(-1005) BTHFRE_ERROR_INVALID_OPERATION
(-1006) BTHFRE_ERROR_INVALID_CODEC_ID

API Call

HFRE_Close_Port(BluetoothStackID, HFServerPortID)

API Prototype

int BTPSAPI HFRE_Close_Port(unsigned int BluetoothStackID, unsigned int HFREPortID)

- **Description of API**

The following function exists to close a HFRE Port that was previously opened by any of the following mechanisms:

- Successful call to HFRE_Open_Remote_HandsFree_Port() function.

- Successful call to HFRE_Open_Remote_Audio_Gateway_Port() function.

- Incoming open request (Hands-Free or Audio Gateway) which the server was opened with either the HFRE_Open_HandsFree_Server_Port() or the HFRE_Open_Audio_Gateway_Server_Port() functions.

This function accepts as input the Bluetooth Stack ID of the Bluetooth Stack which the Open HFRE Port resides and the HFRE Port ID (return value from one of the above mentioned Open functions) of the Port to Close. This function returns zero if successful, or a negative return value if there was an error. This function does NOT Un-Register a HFRE Server Port from the system, it ONLY disconnects any connection that is currently active on the Server Port. The HFRE_Close_Server_Port()function can be used to Un-Register the HFRE Server Port.

{{

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 HFPDemo 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 HFPDemo 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 HFPDemo 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 HFPDemo 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 HFPDemo 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 HFPDemo 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 HFPDemo 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 HFPDemo App** here.*

*For technical su please post you questions at http://e2e.ti.com Please post onl comments abou article **CC256x Bluetooth Stac HFPDemo App***

}}

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