CC256x TI's Bluetooth Stack Basic HFGAGDemo APP

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
- Running the Bluetooth Code

Demo Application
- Server setup using HFP demo application
- Client setup on the demo application
  - Example: Audio gateway with a commercial headset

Application Commands
- Generic Access Profile Commands
  - Help (DisplayHelp)
  - Inquiry
  - Pair
  - EndPairing
  - PINCodeResponse
  - PassKeyResponse
  - UserConfirmationResponse
  - SetDiscoverabilityMode
  - SetConnectabilityMode
  - SetPairabilityMode
  - ChangeSimplePairingParameters
  - GetLocalAddress
  - SetLocalName
  - GetLocalName
  - SetClassOfDevice
  - GetClassOfDevice
  - GetRemoteName
- Hands-free Profile Commands
  - ServiceDiscovery
  - OpenAudioGatewayClient
  - ManageAudio
  - UpdateControlIndicators
  - CallWaiting
  - SetVoiceRecognitionActivation
  - SetSpeakerGain
  - SetMicroPhoneGain
  - DisableRemoteSoundEnhancement
  - SendCallerIdNotification
  - SetRingIndication
  - RingIndication
  - SendIncomingCallState
  - CloseAgClient
  - SendOperatorInfo
  - SendSubNumber
  - SendCallList

Demo Overview

The Hands-free profile allows the user to demonstrate the use of Hands-free profile on 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 Audio Gateway role of the profile.

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

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

It is recommended that the user visits the kit setup 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, **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, 8, n, 1. Once connected, reset the device using Reset S3 button and you should see the stack getting initialized on the terminal and the help screen will be displayed, which shows all of the commands. This device will become the **Audio Gateway**.

Demo Application

This section provides a description of how to use the demo application to connect two configured board and communicate over Bluetooth. The Bluetooth HFP is a simple Client-Server connection process with one side operating in the Audio-Gateway role and the other operating in the Hands-free role. We will setup one of the boards as a Audio-Gateway client (HFPAG Demo acts as Audio gateway) and use an another board as Hands-free server (using HFP Demo). we Will see how to initiate connection and send indications between the two devices over Bluetooth.

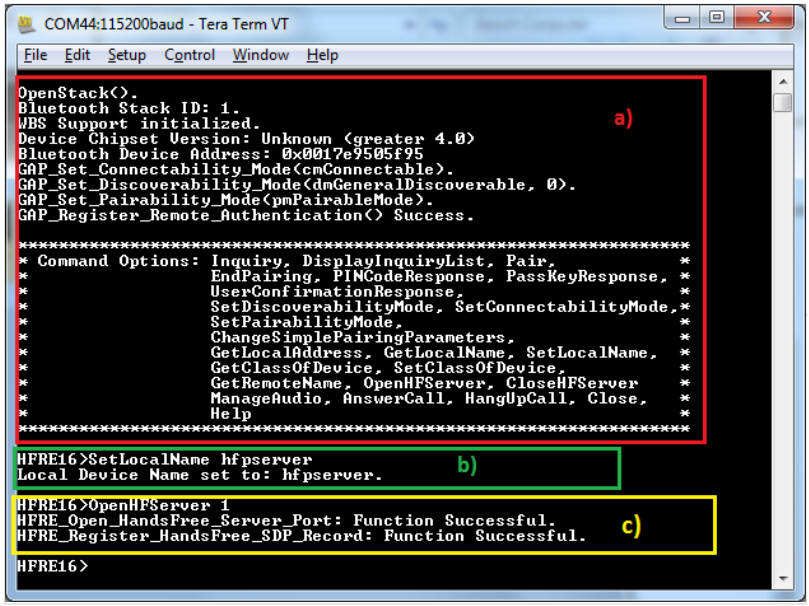
Now use the second board and follow the steps as per either HFP Demo (http://processors.wiki.ti.com/index.php/CC256x_MSP430_TI%E2%80%99s_Bluetooth_Stack_Basic_HFPDemo_APP) when using Tiva or MSP430 as the server or HFPDemo_HF (http://processors.wiki.ti.com/index.php/CC256x_STM32_TI_Bluetooth_Stack_HFPDemo_App) for MSP432 or STM32F4 as the server. Perform the steps prior to running the Audio-Gateway Bluetooth code on the first board. The second device that is connected to the computer will be the **Hands-free Server**.

Server setup using HFP demo application

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

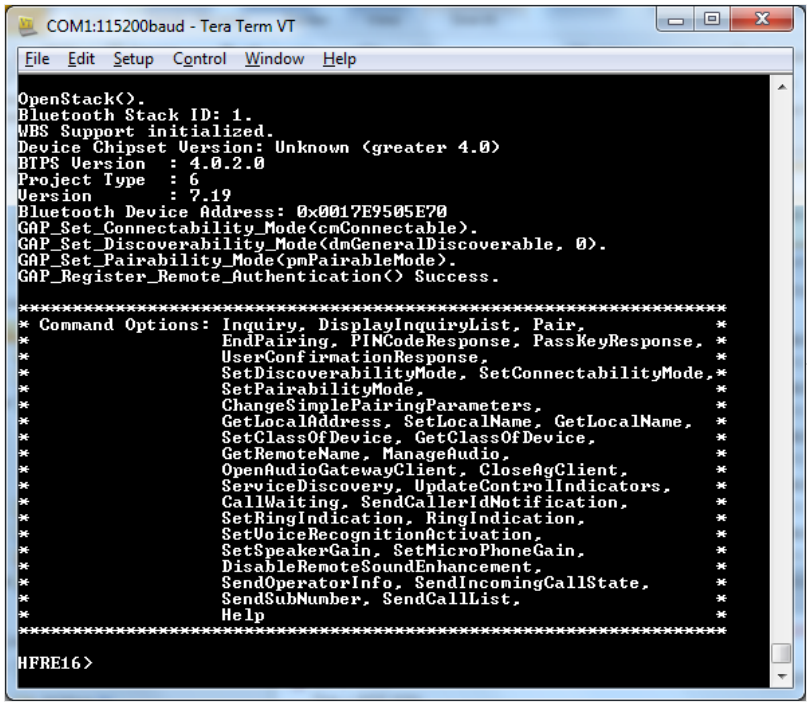
b) Give a name for the platform by issuing the **SetLocalName**. In our example we give it a name of **hfpserver**.

c) Open a HFPServer by issuing the command, **OpenHFServer**. Here we use **OpenHFServer** to open port 1, the default first port.

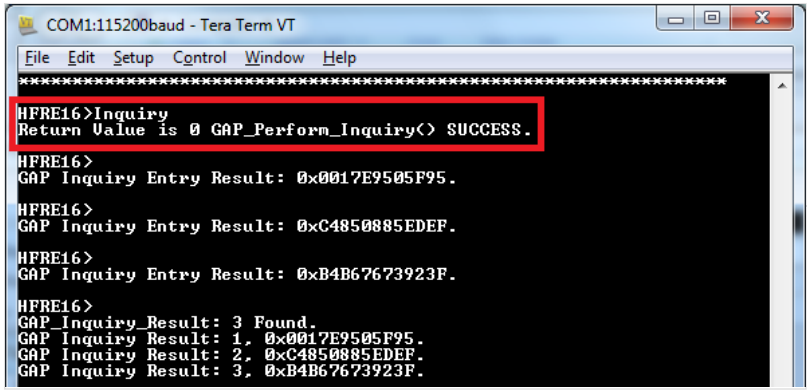


Client setup on the demo application

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

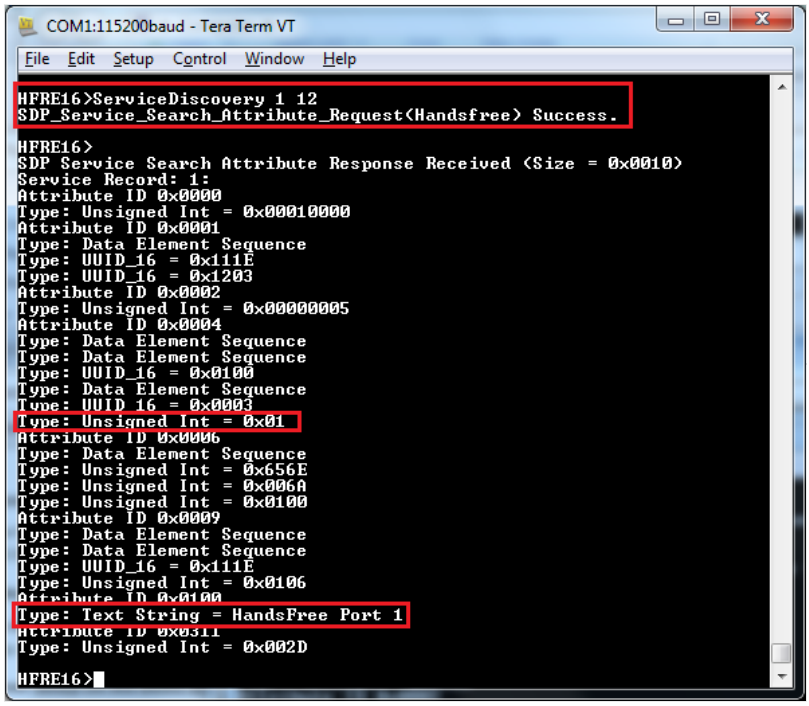


b) Issue the **Inquiry** command for the HFP server.

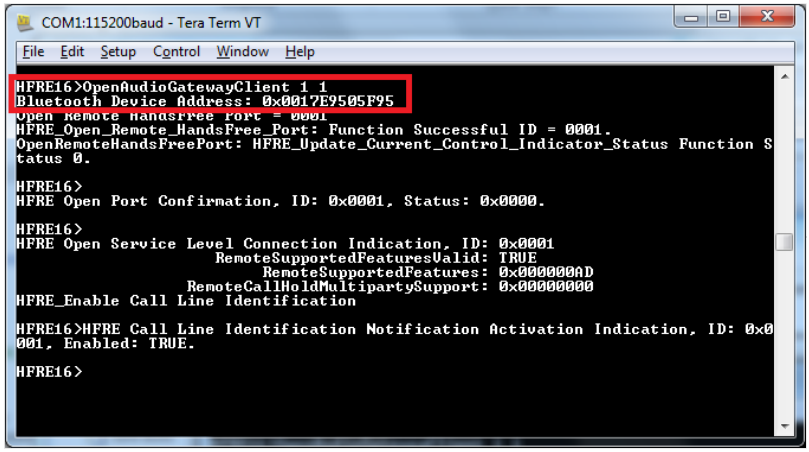


c) Discover servers of the remote HFP server by issuing the **ServiceDiscovery 1 12** , command to get the port number.

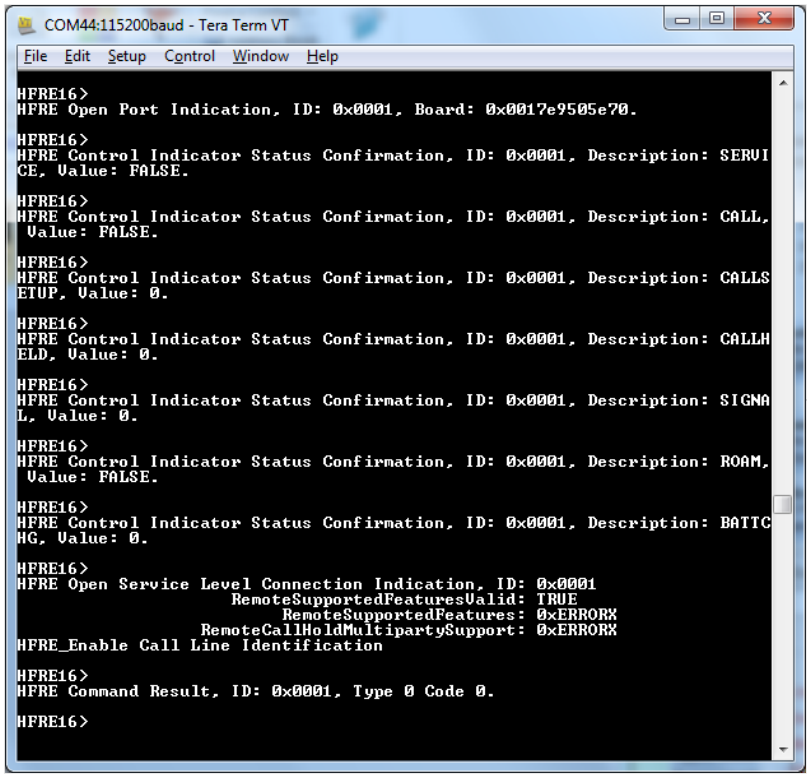
Note: **The port ID on the remote Hands free device is 0x01 (The Unsigned int), as highlighted in the image below from the Attribute ID 0x0004. This port ID is used in the following OpenAudioGatewayClient command as its second parameter.**



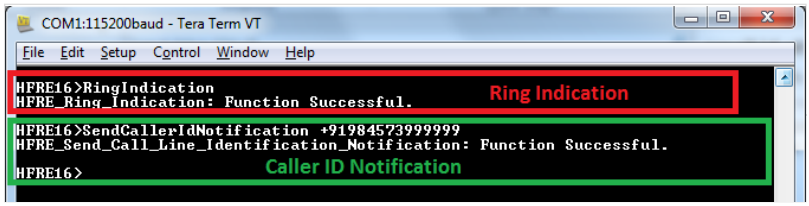
d) Initiate connection to the remote HFP server by isssuing the **OpenAudioGatewayClient 1 1** command.



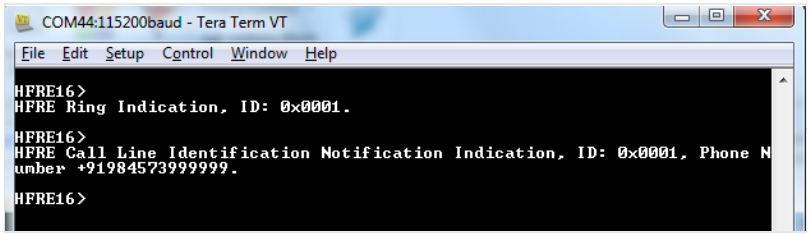
You will see the output below from the HFP server



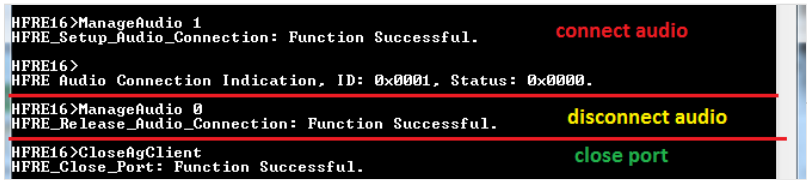
e) Sending indications: Issue the **RingIndication** or **SendCallerIdNotification +919878899889** commands.



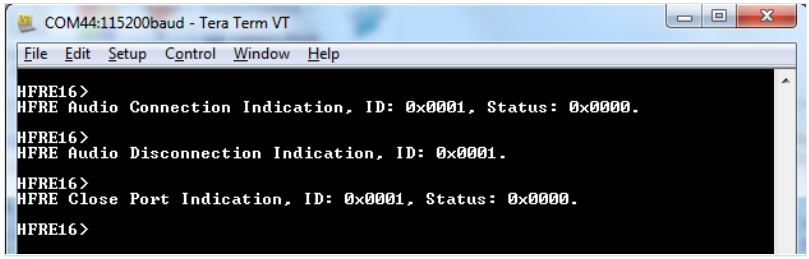
You will see the below output from the HFP server



f) Initiate audio connection/Disconnection and closing the HFP connection by issuing the: **ManageAudio <STATE>** and **CloseAgClient** commands.



You will see the below output from the HFP server



Example: Audio gateway with a commercial headset

This demonstrates setting up the client to connect to a commercial headset.

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

```
OpenStack().
Bluetooth Stack ID: 1
WBS support enabled.
Device Chipset: Unknown (greater 4.1)
BTPS Version  : 4.2.1.0
Project Type  : 6
FW Version    : 12.12
App Name      : HFPDemo_AG
App Version   : 0.3
LOCAL BD_ADDR: 0x88C255D1D645
GAP_Set_Connectability_Mode(cmConnectable).
GAP_Set_Discoverability_Mode(dmGeneralDiscoverable, 0).
GAP_Set_Pairability_Mode(pmPairableMode).
GAP_Register_Remote_Authentication() Success.

******************************************************************
* Command Options: Inquiry, DisplayInquiryList, Pair,            *
*                  EndPairing, PINCodeResponse, PassKeyResponse, *
*                  UserConfirmationResponse,                     *
*                  SetDiscoverabilityMode, SetConnectabilityMode,*
*                  SetPairabilityMode,                           *
*                  ChangeSimplePairingParameters,                *
*                  GetLocalAddress, SetLocalName, GetLocalName,  *
*                  SetClassOfDevice, GetClassOfDevice,           *
*                  GetRemoteName, ManageAudio,                   *
*                  OpenAudioGatewayClient, CloseAgClient,        *
*                  ServiceDiscovery, UpdateControlIndicators,    *
*                  CallWaiting, SendCallerIdNotification,        *
*                  SetRingIndication, RingIndication,            *
*                  SetVoiceRecognitionActivation,                *
*                  SetSpeakerGain, SetMicroPhoneGain,            *
*                  DisableRemoteSoundEnhancement,                *
*                  SendOperatorInfo, SendIncomingCallState,      *
*                  SendSubNumber, SendCallList,                  *
*                  Help                                          *
******************************************************************
```

b) Issue the **Inquiry** command for the HFP server.

```
HFP AG>Inquiry
Return Value is 0 GAP_Perform_Inquiry() SUCCESS.

HFP AG>
GAP Inquiry Entry Result: 0x244B03F712D3.

HFP AG>
GAP Inquiry Entry Result: 0x3402862CCAE9.

HFP AG>
GAP Inquiry Entry Result: 0x340286605044.

HFP AG>
GAP Inquiry Entry Result: 0x484520902A4E.

HFP AG>
GAP Inquiry Entry Result: 0x08DF1F99F8D0.

HFP AG>
GAP Inquiry Entry Result: 0x002500F84FAB.

HFP AG>
GAP_Inquiry_Result: 6 Found.
GAP Inquiry Result: 1, 0x244B03F712D3.
GAP Inquiry Result: 2, 0x3402862CCAE9.
GAP Inquiry Result: 3, 0x340286605044.
GAP Inquiry Result: 4, 0x484520902A4E.
GAP Inquiry Result: 5, 0x08DF1F99F8D0.
GAP Inquiry Result: 6, 0x002500F84FAB.
```

c) Discover services of the remote HFP server by issuing the **ServiceDiscovery 5 12** , command to get the port number.

Note: **The port ID on the remote Hands free device is 0x0A (The Unsigned int), from the Attribute ID 0x0004. This port ID is used in the following OpenAudioGatewayClient command as its second parameter after being converted to its decimal equivilant (10).**

```
HFP AG>ServiceDiscovery
Usage: SERVICEDISCOVERY [Inquiry Index] [Profile Index] [16/32 bit UUID (Manual only)].

   Profile Index:
       0) Manual (MUST specify 16/32 bit UUID)
       1) L2CAP
       2) Advanced Audio
       3) A/V Remote Control
       4) Basic Imaging
       5) Basic Printing
       6) Dial-up Networking
       7) FAX
       8) File Transfer
       9) Hard Copy Cable Repl.
      10) Health Device
      11) Headset
      12) Handsfree
      13) HID
      14) LAN Access
      15) Message Access
```

```
      16) Object Push
      17) Personal Area Network
      18) Phonebook Access
      19) SIM Access
      20) Serial Port
      21) IrSYNC

Function Error.

HFP AG>ServiceDiscovery 5 12
SDP_Service_Search_Attribute_Request(Handsfree) Success.

HFP AG>
SDP Service Search Attribute Response Received (Size = 0x0010)
Service Record: 1:
Attribute ID 0x0000
Type: Unsigned Int = 0x00010000
Attribute ID 0x0001
Type: Data Element Sequence
Type: UUID_16 = 0x111E
Type: UUID_16 = 0x1203
Attribute ID 0x0004
Type: Data Element Sequence
Type: Data Element Sequence
Type: UUID_16 = 0x0100
Type: Data Element Sequence
Type: UUID_16 = 0x0003
Type: Unsigned Int = 0x0A
Attribute ID 0x0006
Type: Data Element Sequence
Type: Unsigned Int = 0x656E
Type: Unsigned Int = 0x006A
Type: Unsigned Int = 0x0100
Attribute ID 0x0009
Type: Data Element Sequence
Type: Data Element Sequence
Type: UUID_16 = 0x111E
Type: Unsigned Int = 0x0106
Attribute ID 0x0100
Type: Text String = Hands-Free unit
Attribute ID 0x0311
Type: Unsigned Int = 0x003F
```

d) Initiate connection to the remote HFP server by isssuing the **OpenAudioGatewayClient 5 10** command.

```
HFP AG>OpenAudioGatewayClient
Usage: OPENAUDIOGATEWAYCLIENT [Inquiry Index] [Port Number].
Function Error.

HFP AG>OpenAudioGatewayClient 5 10
Bluetooth Device Address: 0x08DF1F99F8D0
Open Remote HandsFree Port = 000A
HFRE_Open_Remote_HandsFree_Port: Function Successful ID = 0001.
OpenRemoteHandsFreePort: HFRE_Update_Current_Control_Indicator_Status Function Status 0.
```

You will see the output below from the HFP server

```
HFP AG>
atLinkKeyRequest: 0x08DF1F99F8D0
GAP_Authentication_Response() Success.

HFP AG>
atPINCodeRequest: 0x08DF1F99F8D0

Respond with the command: PINCodeResponse

HFP AG>PINCodeResponse 0000
PINCodeResponse.
GAP_Authentication_Response(), Pin Code Response Success.

HFP AG>
atLinkKeyCreation: 0x08DF1F99F8D0
Link Key: 0x4AF49AD7072771919BAC62840F1F985D
Link Key Stored locally.

HFP AG>
HFRE Open Port Confirmation, ID: 0x0001, Status: 0x0000.

HFP AG>
HFRE Available Codec List Indiocation, ID: 0x0001 NumCodecs: 2 [  1  2 ]

HFP AG>
HFRE Open Service Level Connection Indication, ID: 0x0001
          RemoteSupportedFeaturesValid:    TRUE
          RemoteSupportedFeatures:         0x000000BF
          RemoteCallHoldMultipartySupport: 0x00000000
HFRE_Enable Call Line Identification

HFP AG>
HFRE Disable Sound_Enhancement Indication, ID: 0x0001
HFRE_Send_Terminating_Response (erOK) :: Res = 0

HFP AG>HFRE Speaker Gain Indication, ID: 0x0001, Speaker Gain 0x000A.

HFP AG>HFRE Call Line Identification Notification Activation Indication, ID: 0x0001, Enabled: TRUE.

HFP AG>HFRE Call Waiting Notification Activation Indication, ID: 0x0001, Enabled: TRUE.

HFP AG>
HFRE Response Hold Status Indication, ID: 0x0001
```

```
HFRE_Send_Incoming_Call_State (csNone) :: Res = 0

HFP AG>
HFRE_Current_Calls_List_Indication

HFRE_Send_Terminating_Response (erOK) :: Res = 0
```

e) Initiate audio connection by issuing the: **ManageAudio <STATE>** command which, by defaut chooses to uses Modified sub-band coding (MSBC).

```
HFP AG>ManageAudio
Usage: Audio [Release = 0, Setup = 1].
Function Error.

HFP AG>ManageAudio 1
HFRE_Send_Select_Codec:: Codec = 2, Res = 0
```

You will see the below output from the HFP server

```
HFP AG>
HFRE Codec Select Confirmation, ID: 0x0001  AcceptedCodec=2
Setup WBS with Audio for ACL handle 0x0001
HFRE_Setup_Audio_Connection: Function Successful.
HFRE_Setup_Audio_Connection:: Res = 0

HFP AG>
HFRE Audio Connection Indication, ID: 0x0001, BDADDR=0x08DF1F99F8D0, Status: 0x0000.
```

Application Commands

TI's Bluetooth stack is an 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 a Bluetooth controller chip.

The basic bluetooth application included with MSP432 (http://www.ti.com/tool/msp-exp432p401r) and STM32F4 (http://www.st.com/en/evaluation-tools/stm3240g-eval.html) is a Hands-free audio gateway Application.

An overview of the application and other applications can be read at the 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.

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).

Hands-free Profile Commands

ServiceDiscovery

Description
The following function is responsible for issuing a Service Search Attribute Request to a Remote SDP Server. This function returns zero if successful and a negative value if an error occurred.

Parameters
The command requires two parameter. first parameter is the Inquiry Index and the second is Profile Index.

Command Call Examples
"ServiceDiscovery 1 12" Attempts to Open a Audio Gateway client port on inquiry index #1 and Profile Index Number #12 (Hands-free).
"ServiceDiscovery 3 12" Attempts to Open a Audio Gateway client port on inquiry index #3 and Profile Index Number #12 (Hands-free).
Possible Return Values
(-1) BTPS_ERROR_INVALID_PARAMETER (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
(-4) FUNCTION_ERROR
(-8) INVALID_STACK_ID_ERROR
(-40) BTPS_ERROR_MEMORY_ALLOCATION_ERROR (-54) BTPS_ERROR_EXPECTED_UUID_ENTRY (-64) BTPS_ERROR_INTERNAL_ERROR (-103) BTPS_ERROR_FEATURE_NOT_AVAILABLE

API Call
SDP_Service_Search_Attribute_Request(BluetoothStackID, InquiryResultList[(TempParam->Params[0].intParam - 1)], 1, &SDPUUIDEntry, 1, &AttributeID, SDP_Event_Callback, (unsigned long)0)
API Prototype
BTPSAPI_DECLARATION int BTPSAPI SDP_Service_Search_Attribute_Request(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, unsigned int NumberServiceUUID, SDP_UUID_Entry_t SDP_UUID_Entry[], unsigned int NumberAttributeListElements, SDP_Attribute_ID_List_Entry_t AttributeIDList[], SDP_Response_Callback_t SDP_Response_Callback, unsigned long CallbackParameter)
Description of API
The following function is responsible for issuing an SDP Service Search Attribute Request to the specified BD_ADDR. This function will return the result of the Service Search Attribute Request in the SDP Response Callback that is specified in the calling of this function. This function accepts as input, the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Client resides on, the Bluetooth Board Address to remotely connect to (the Remote SDP Server will reside on this BD_ADDR), the Number of Service UUID's that are to be searched for, the Servcee UUID's to actually search for, the Number of Entries in the Attribute List that are to be queried, the Attribute List to actually use in the Query, the SDP Response Callback Function, and the SDP Response Callback Function Callback Parameter. This function will return a positive, non-zero, return code if successful, or a negative return error code if there was an error. If this function is successful, the user can call the SDP_Cancel_Service_Request() function to cancel a the SDP Service Search Request prematurely.It should be noted that the Number of UUID Parameter must be at least one, and the Service UUID Parameter must point to a List of at least the Number of UUID's that have been specified. Finally, the BD_ADDR Parameter and the SDP_Reponse_Callback Parameter MUST be valid or the call to this function will be unsuccessful. It should also be noted that the Number of Attributes that are in the Attribute List must be at least one, and the Attribute ID List Parameter must point to a List of Attribute ID's that contains at least the Number of Attribute List Entries that have been specified. Finally, the BD_ADDR Parameter and the SDP_Reponse_Callback Parameter MUST be valid or the call to this function will be unsuccessful.

OpenAudioGatewayClient

Description
The following function is responsible for opening an Audio Gateway client port. This function returns zero on successful execution and a negative value on all errors.
Parameters
The command requires two parameter. first parameter is the Inquiry Index and the second is the remote device Port Number

Command Call Examples
"OpenAudioGatewayClient 1 1" Attempts to Open a Audio Gateway client port on inquiry index #1 and remote Port Number #1.
"OpenAudioGatewayClient 2 3" Attempts to Open a Audio Gateway client port on inquiry index #2 and remote Port Number #3.
Possible Return Values
(0) AG Client 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_Remote_Hands-free_Port(BluetoothStackID, InquiryResultList[(TempParam->Params[0].intParam-1)], TempParam->Params[1].intParam, DEFAULT_AG_SUPPORTED_FEATURES, DEFAULT_CALL_HOLDING_SUPPORT, 0, NULL, HFRE_Event_Callback, (unsigned long)0)
API Prototype
BTPSAPI_DECLARATION int BTPSAPI HFRE_Open_Remote_Hands-free_Port(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, unsigned int RemoteServerPort, unsigned long SupportedFeaturesMask, unsigned long CallHoldSupportMask, unsigned int NumberAdditionalIndicators, HFRE_Control_Indicator_Entry_t AdditionalSupportedIndicators[], HFRE_Event_Callback_t EventCallback, unsigned long CallbackParameter)
Description of API
The following function is responsible for Opening a Remote Hands-Free Port on the specified Remote Device. This function accepts the Bluetooth Stack ID of the Bluetooth Stack which is to open the HFRE Connection as the first parameter. The second parameter specifies the Board Address (NON NULL) of the Remote Bluetooth Device to connect with. The third parameter specifies the features the local Audio Gateway supports. The next parameter is a bit mask which specifies the Call Hold and Multiparty Handling Features that are supported. The fifth parameter to this function is the number of indicator names which appear in the list represented by the previous parameter. The next parameter is a list of additional indicators in which this Audio Gateway will support. If the Additional Indicators parameter is NULL and Number Additional Indicators is zero no additional parameters will be supported. The final two parameters specify the HFRE Event Callback function, and callback parameter, respectively, of the HFRE Event Callback that is to process any further interaction with the specified

Remote Port (Opening Status, Close Status,etc). This function returns a non-zero, positive, value if successful, or a negative return error code if this function is unsuccessful. If this function is successful, the return value will represent the HFRE Port ID that can be passed to all other functions that require it. Once a Remote Hands-Free unit is opened, it can only be closed via a call to the HFRE_Close_Port() function (passing the return value from this function). **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.

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) Command sent 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, CurrentClientPortID)

or HFRE_Release_Audio_Connection(BluetoothStackID, CurrentClientPortID)

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.

UpdateControlIndicators

Description

The following function is responsible for updating the current state of the Control Indicators on the Remote Hands-Free unit. This function returns zero on successful execution and a negative value on all errors. indicators name : Call Status, Call Setup, Service availability, Signal indicator, Roam indicator, battchg indicator and Call Held

Parameters

The command requires two parameter. first parameter is the indicators name and the second is the value you want to set.

Command Call Examples

"UpdateControlIndicators Call Setup 1"

"UpdateControlIndicators Call Held 3"

Possible Return Values

(0) Command sent 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_Update_Current_Control_Indicator_Status(BluetoothStackID, PortToUse, 1, &HFREIndicatorUpdate)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Update_Current_Control_Indicator_Status(unsigned int BluetoothStackID, unsigned int HFREPortID, unsigned int NumberUpdateIndicators, HFRE_Indicator_Update_t UpdateIndicators[])

Description of API

The following function is responsible for Updating the Current Control Indicator Status. This function may only be performed by Audio Gateways that have a valid Service Level Connection or by Audio Gateways that have received the etHFRE_Control_Indicator_Request_Indication event. 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. The third parameter to this function is the number of name/value pairs that are present in the list. The final parameter to this function is a list of name/value pairs for the indicators to be updated. This function returns zero if successful or a negative return error code if there was an error.

CallWaiting

NOTE : This function will only work if call waiting feature is supported in the remote HF device. **Description**

The following function is responsible for sending a Call Waiting Notification to the Remote Hands-Free unit. This function returns zero on successful execution and a negative value on all errors. If the remote device callwaiting feature is supported in the HF device than this will work.

Parameters

The Call Waiting command takes only one parameter, which is the phone number to be sent as part of this response.

Command Call Examples

"CallWaiting +9198787899889"

Possible Return Values

(0) Command sent 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_Send_Call_Waiting_Notification(BluetoothStackID, CurrentClientPortID, TempParam->Params->strParam)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Send_Call_Waiting_Notification(unsigned int BluetoothStackID, unsigned int HFREPortID, char *PhoneNumber)

Description of API

This function is responsible for Sending Call Waiting Notifications to the Remote Device. This function may only be performed by Audio Gateways which have Call Waiting Notification Enabled and have a valid Service Level Connection. This function accepts as its first input parameter the HFRE Port ID. The final parameter is the Phone Number required as one of the parameters within this response. This parameter should be a pointer to a NULL terminated string (if specified) and must have a length less than HFRE_PHONE_NUMBER_LENGTH_MAXIMUM. This function returns zero if successful or a negative return error code if there was an error.

NOTE : It is valid to either pass a NULL for the PhoneNumber parameter of a blank string to specify that there is no phone number present.

SetVoiceRecognitionActivation

Description

The following function is responsible for deactivating Voice Recognition Activation on the Audio Gateway and for change the Voice Recognition Activation state on the Hands-free Unit. This function returns zero on successful execution and a negative value on all errors.

Parameters

The Voice Recognition command takes only one parameter, This value must be specified as 0 (for Deactivate) or 1 (to Activate).

Command Call Examples

"SetVoiceRecognitionActivation 0"

Possible Return Values

(0) Command sent 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_Set_Remote_Voice_Recognition_Activation(BluetoothStackID, CurrentClientPortID, (Boolean_t)TempParam->Params->intParam)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Set_Remote_Voice_Recognition_Activation(unsigned int BluetoothStackID, unsigned int HFREPortID, Boolean_t VoiceRecognitionActive)

Description of API

This function is responsible for activation and deactivation of the Voice Recognition which resides on the Remote Audio Gateway when called by a Hands-Free unit. When called by an Audio Gateway this function is responsible for informing the remote Hands-Free unit of the current activation state of the local Voice Recognition function. This function may only be called by local devices that were opened with the Supported Feature Bit set for Voice Recognition. 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. The final parameter is a Boolean flag specifying if this is a call to Activate or Deactivate this function on a Remote Audio Gateway, or to specify that Voice Recoginition is locally Activated or Deactivated to a Remote Hands-Free unit. When active the Voice Recognition function on the Audio Gateway is turned on, when inactive the Voice Recognition function on the Audio Gateway is turned off. This function returns zero if successful or a negative return error code if there was an error.

SetSpeakerGain

Description

The following function is responsible for setting the Speaker Gain on Remote Device. This function returns zero on successful execution and a negative value on all errors. If the remote audio volume control feature is supported in the HF device than this will work.

Parameters

The command requires only one parameter. It would be a integer value for the Speaker Gain.

Command Call Examples

"SetSpeakerGain 5"
"SetSpeakerGain 6"

Possible Return Values

(0) Command sent 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_Set_Remote_Speaker_Gain(BluetoothStackID, CurrentClientPortID, TempParam->Params->intParam)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Set_Remote_Speaker_Gain(unsigned int BluetoothStackID, unsigned int HFREPortID, unsigned int SpeakerGain)

Description of API

This function is responsible for allowing synchronization and setting of the Remote Devices Speaker Gain. This function may only be performed if a valid Service Level Connection exists. When called by a Hands-Free unit this function is provided as a means to inform the Remote Audio Gateway of the current Speaker Gain value. When called by an Audio Gateway this function provides a means for the Audio Gateway to control the Speaker Gain of the Remote 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. The final parameter is the Speaker Gain to be sent to the Remote Device. The Speaker Gain Parameter *MUST* be between the values of HFRE_SPEAKER_GAIN_MINIMUM and HFRE_SPEAKER_GAIN_MAXIMUM. This function returns zero if successful or a negative return error code if there was an error.

SetMicroPhoneGain

Description

The following function is responsible for setting the Microphone Gain on Remote Device. This function returns zero on successful execution and a negative value on all errors. If the remote audio volume control feature is supported in the HF device than this will work.

Parameters

The command requires only one parameter. It would be a integer value for the MicroPhone Gain.

Command Call Examples

"SetMicroPhoneGain 5"
"SetMicroPhoneGain 6"

Possible Return Values

(0) Command sent 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_Set_Remote_Microphone_Gain(BluetoothStackID, CurrentClientPortID, TempParam->Params->intParam)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Set_Remote_Microphone_Gain(unsigned int BluetoothStackID, unsigned int HFREPortID, unsigned int MicrophoneGain)

Description of API

This function is responsible for allowing synchronization and setting of the Remote Devices Microphone Gain. This function may only be performed if a valid Service Level Connection exists. When called by a Hands-Free unit this function is provided as a means to inform the Remote Audio Gateway of the current Microphone Gain value. When called by an Audio Gateway this function provides a means for the Audio Gateway to control the Microphone Gain of the Remote 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. The final parameter is the Microphone Gain to be sent to the Remote Device. The Microphone Gain Parameter *MUST* be between the values of HFRE_MICROPHONE_GAIN_MINIMUM and HFRE_MICROPHONE_GAIN_MAXIMUM. This function returns zero if successful or a negative return error code if there was an error.

DisableRemoteSoundEnhancement

NOTE : This function may be performed only when a valid Service Level Connection exist but no Audio Connection exists. **Description**

The following function is responsible for disabling Sound Enhancement on the Remote Device. This function returns zero on successful execution and a negative value on all errors. This function may be performed only when a valid Service Level Connection exist but no Audio Connection exists.

Parameters

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

Command Call Examples

"DisableRemoteSoundEnhancement"

Possible Return Values

(0) Command sent 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_Disable_Remote_Echo_Cancelation_Noise_Reduction(BluetoothStackID, CurrentClientPortID)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Disable_Remote_Echo_Cancelation_Noise_Reduction(unsigned int BluetoothStackID, unsigned int HFREPortID)

Description of API

This function is responsible for Disabling Echo Cancelation and Noise Reduction on the Remote Device. This function may be performed by both the Hands-Free unit and the Audio Gateway for which a valid Service Level Connection exist but no Audio 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. **NOTE :** It is not possible to enable this feature once it has been disbled because the specification provides no means to re-enable this feature. This feature will remained disabled until the current Service Level Connection has been dropped.

SendCallerIdNotification

Description

The following function is responsible for sending a Call Line Identification Notification to the Remote Hands-Free unit. This function returns zero on successful execution and a negative value on all errors.

Parameters

The command requires only one parameter. It would be a CALLER id (phone number).

Command Call Examples

"SendCallerIdNotification +9198787899889"

Possible Return Values

(0) Command sent 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_Send_Call_Line_Identification_Notification(BluetoothStackID, CurrentClientPortID, TempParam->Params->strParam)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Send_Call_Line_Identification_Notification(unsigned int BluetoothStackID, unsigned int HFREPortID, char *PhoneNumber)

Description of API

This function is responsible for sending Call Line Identification Notifications to the Remote device. This function may only be performed by Audio Gateways which have Call Line Identification Notification Enabled and have a valid Service Level Connection. 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. The final parameter is the Phone Number required as one of the parameters within this response. This parameter should be a pointer to a NULL terminated string and its length *MUST* be between the values of HFRE_PHONE_NUMBER_LENGTH_MINIMUM and HFRE_PHONE_NUMBER_LENGTH_MAXIMUM. This function return zero if successful or a negative return error code if there was an error.

SetRingIndication

Description

The following function is responsible for sending an enable or disable In-Band Ring Indication to the Remote Hands-Free unit. This function returns zero on successful execution and a negative value on all errors.

Parameters

The command requires only one parameter. This value must be specified as 0 (for disable) or 1 (for enabling).

Command Call Examples

"SetRingIndication 1" for enabling In-Band Ring Indication
"SetRingIndication 0" for disable In-Band Ring Indication

Possible Return Values

(0) Command sent 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_Enable_Remote_InBand_Ring_Tone_Setting(BluetoothStackID, CurrentClientPortID, TempParam->Params[0].intParam? TRUE : FALSE)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Enable_Remote_InBand_Ring_Tone_Setting(unsigned int BluetoothStackID, unsigned int HFREPortID, Boolean_t EnableInBandRing)

Description of API

This function is responsible for Enabling or Disabling In-Band Ring Tone Capabilities for the Local Device. This function may only be performed by Audio Gateways for which a valid Service Level Connection exists. This function may only be used to enable In-Ring Tone Capabilities if the Local Audio Gateway supports this feature. 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. The final parameter is a Boolean flag specifying if this is a call to Enable or Disable this functionality. This function returns zero if successful or a negative return error code if there was an error.

RingIndication

Description

The following function is responsible for sending a Ring Indication to the Remote Hands-Free unit. 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 RingIndication.

Command Call Examples

"RingIndication" send ring indication.

Possible Return Values

(0) Command sent 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_Ring_Indication(BluetoothStackID, CurrentClientPortID)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Ring_Indication(unsigned int BluetoothStackID, unsigned int HFREPortID)

Description of API

This function is responsible for sending a Ring Indication to the Remote Hands-Free unit. This function may only be performed by Audio Gateways 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.

SendIncomingCallState

Description

The following function is responsible for sending the Response and Hold Command (+BTRH) to the remote device. This function returns zero on successful execution and a negative value on all errors.

Parameters

This command requires only one parameter to which is is an integer value that represents the incoming call state. This value must be specified as 0 (incoming call is put on hold), 1 (held call is accept) and 2(held call is reject).

Command Call Examples

"SendIncomingCallState 1" if there is a incoming call.

Possible Return Values

(0) Command sent 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_Send_Incoming_Call_State(BluetoothStackID, CurrentClientPortID, (HFRE_Call_State_t) CallState)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Send_Incoming_Call_State(unsigned int BluetoothStackID, unsigned int HFREPortID, HFRE_Call_State_t CallState)

Description of API

The following function is responsible for sending information about the incoming call state. This function may only be performed by Audio Gateways that have a valid Service Level Connection. 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. The final parameter to this function indicates the call state to set as part of this message. This function returns zero if successful or a negative return error code if there was an error.

CloseAgClient

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) Client closed 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, CurrentClientPortID)

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_Hands-free_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_Hands-free_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.

SendOperatorInfo

NOTE : This function should be performed When received a request to query the network operator selection from the remote Hands-free device.

Description

The following function is responsible for sending the command to send an Operator Selection Response to the Remote Hands-free Device. When received a request to query the remote network operator selection from the the remote Hands-free device. This function returns zero on successful execution and a negative value on all errors.

Parameters

The Send Operator info command requires only one parameter which is the current Network Operator.

Command Call Examples

"SendOperatorInfo airtel"

Possible Return Values

(0) Command sent 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_Send_Network_Operator_Selection(BluetoothStackID, CurrentClientPortID, 1, TempParam->Params[0].strParam)

API Prototype

BTPSAPI_DECLARATION int BTPSAPI HFRE_Send_Network_Operator_Selection(unsigned int BluetoothStackID, unsigned int HFREPortID, unsigned int NetworkMode, char *NetworkOperator)

Description of API

The following function is responsible for sending the network operator. This function may only be performed by Audio Gateways that have received a request to query the remote network operator selection. 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. The third parameter to this function is the current Network Mode. The final parameter The final parameter is the current Network Operator. This parameter should be a pointer to a NULL terminated string (if specified) and must have a length less than HFRE_NETWORK_OPERATOR_LENGTH_MAXIMUM. This function returns zero if successful or a negative return error code if there was an error. **NOTE :** It is valid to either pass a NULL for the NetworkOperator parameter of a blank string to specify that there is no Network Operator present.

SendSubNumber

NOTE : This function should be performed when received a request to query the subscriber number information from the the remote Hands-free device.

Description

The following function is responsible for sending the command to send a subscriber number to the Remote Hands-free Device. when received a request to query the subscriber number information from the remote Hands-free device. This function returns zero on successful execution and a negative value on all errors.

Parameters

The Send Subscriber Number command requires only one parameter which is the phone number to be sent as part of this response.

Command Call Examples

"SendSubNumber +9198787899889"

Possible Return Values
(0) Command sent 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_Send_Subscriber_Number_Information(BluetoothStackID, CurrentClientPortID, TempParam->Params[0].strParam, 4, HFRE_DEFAULT_NUMBER_FORMAT, TRUE)
API Prototype
BTPSAPI_DECLARATION int BTPSAPI HFRE_Send_Subscriber_Number_Information(unsigned int BluetoothStackID, unsigned int HFREPortID, char *PhoneNumber, unsigned int ServiceType, unsigned int NumberFormat, Boolean_t FinalEntry)
Description of API
The following function is responsible for sending Subscriber Number Information. This function may only be performed by Audio Gateways that have received a request to query the subscriber number information. 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. The third parameter to this function is the phone number to be sent as part of this response. The Phone Number Parameter String Length *MUST* be between the values of HFRE_PHONE_NUMBER_LENGTH_MINIMUM and HFRE_PHONE_NUMBER_LENGTH_MAXIMUM. The fourth parameter to this function is the Service type related to the specified Phone Number. The fifth paramter to this function is the Number Format to use for this number, The final parameter to this function is a boolean indicating if this is the last subscriber number information entry to be sent therefore requiring that an OK be sent as well. This function returns zero if successful or a negative return error code if there was an error.

SendCallList

NOTE : This function should be performed when received a request to query the current calls list information from the the remote Hands-free device.
Description
The following function is responsible for sending the command(s) to send the Call Entry to a remote Hands-free Device. When received a request to query the remote current calls list from the remote Hands-free device. This function returns zero on successful execution and a negative value on all errors.
Parameters
The Send Call list command takes six parameter, they are Index, call direction, call status, call Mode, Multiparty, Phone Number (in the same order).
Command Call Examples
"SendCallList 0 1 1 0 0 5551212"
Possible Return Values
(0) Command sent 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_Send_Current_Calls_List(BluetoothStackID, CurrentClientPortID, &CurrentCallListEntry, Final)
API Prototype
BTPSAPI_DECLARATION int BTPSAPI HFRE_Send_Current_Calls_List(unsigned int BluetoothStackID, unsigned int HFREPortID, HFRE_Current_Call_List_Entry_t *CurrentCallListEntry, Boolean_t FinalEntry)
Description of API
The following function is responsible for Sending Current Calls List Entries to the Remote Device. This function may only be performed by Audio Gateways that have received a request to query the remote current calls list. 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. The third parameter to this function is the current call list entry to be sent. The final parameter to this function is a boolean indicating if this is the last call list entry to be sent therefore requiring that an OK be sent as well. This function returns zero if successful or a negative return error code if there was an error.
NOTE : If the third parameter is specified as NULL, then the Final parameter *MUST* specify that this is the Final Entry. In this case, no call list entry will be sent, however, the terminating response will be sent.
NOTE : This function does not send the Phonebook Name as part of the Call List Entry. Use HFRE_Send_Current_Calls_List_With_Phonebook_Name if you wish to also send the Phonebook Name with the entry.

{{ 1. switchcategory:MultiCore= ▪ For technical support on MultiCore devices, please	Keystone= ▪ For technical support on MultiCore devices, please post your	C2000=*For technical support on the C2000 please post your questions on*	DaVinci=*For technical support on DaVincoplease post your questions on*	MSP430=*For technical support on MSP430 please post your*	OMAP35x=*For technical support on OMAP please post your questions on*	OMAPL1=*For technical support on OMAP please post your questions on*	MAVRK=*For technical support on MAVRK please post your*	*For tech please p question http://e2 Please p commen*

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's Bluetooth Stack Basic HFGAGDemo APP** here.

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's Bluetooth Stack Basic HFGAGDemo APP** here.

*The C2000 Forum. Please post only comments about the article **CC256x TI's Bluetooth Stack Basic HFGAGDemo APP** here.*

*The DaVinci Forum. Please post only comments about the article **CC256x TI's Bluetooth Stack Basic HFGAGDemo APP** here.*

*questions on The MSP430 Forum. Please post only comments about the article **CC256x TI's Bluetooth Stack Basic HFGAGDemo APP** here.*

*The OMAP Forum. Please post only comments about the article **CC256x TI's Bluetooth Stack Basic HFGAGDemo APP** here.*

*The OMAP Forum. Please post only comments about the article **CC256x TI's Bluetooth Stack Basic HFGAGDemo APP** here.*

*questions on The MAVRK Toolbox Forum. Please post only comments about the article **CC256x TI's Bluetooth Stack Basic HFGAGDemo APP** here.*

*article **C Bluetoo Basic H APP** her*

}}

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