User's Guide

CC256x TI Bluetooth Stack HIDDemo App

ABSTRACT

Table of Contents

Trademarks

MSP432™ is a trademark of Ti.
Bluetooth® is a registered trademark of Bluetooth.
All trademarks are the property of their respective owners.

1 Demo Overview

The human interface device enables a host to connect and control a HID Device. There are two roles defined in this profile, host and device. The host sends control and report requests and the second is the device which responds to the host's requests. The host is a device like a computer or tablet and the Device is an I/O device like keyboard or mouse.

This application allows the user to use a console with Bluetooth Low Energy (BLE) to establish connection between two BLE devices, control the device, and get reports and protocols.

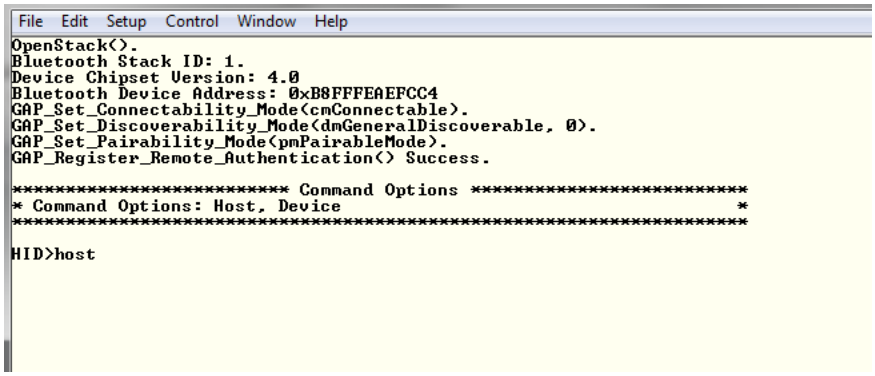
Users are advised to visit TI Dual-Mode Bluetooth® Stack on MSP432™ MCUs or Dual-Mode Bluetooth® Stack on STM32F4 MCUs pages before trying the application described on this page.

Note

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

2 Running the Bluetooth Code

Once the code is flashed, connect the board to a PC using a miniUSB or microUSB cable. Once connected, wait for the driver to install. This appears as MSP-EXP430F5438 USB -Serial Port(COM x), Tiva Virtual COM Port (COM x) , XDS110 Class Application/User UART (COM x) for MSP432, under Ports (COM & LPT) in the Device manager. Attach a Terminal program like PuTTY to the serial port x for the board. The serial parameters to use are 115200 Baud (9600 for MSP430), 8, n, 1. Once connected, reset the Device using Reset S3 button (located next to the mini USB connector for the MSP430) and observe the stack initialized on the terminal. The help screen is displayed and shows all of the commands.

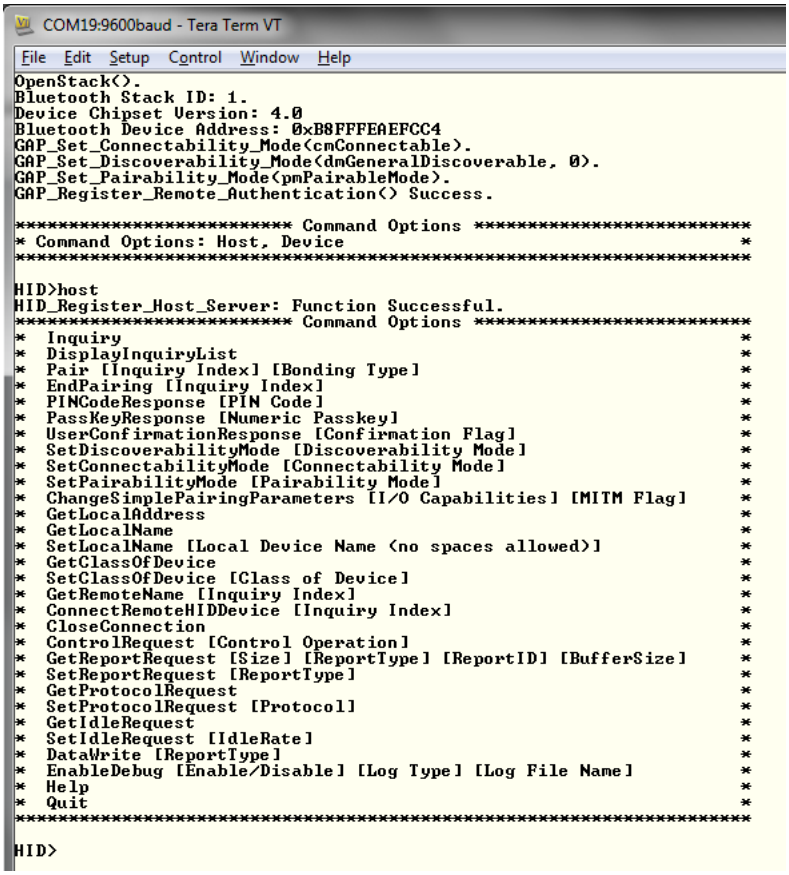


3 Demo Application

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

3.1 Device 1 (Host/HID Host) Setup on the Demo Application

1. Setup the first board as a Host. Perform the steps mentioned earlier in Running the Bluetooth Code section to initialize the application. Once initialized, note the Bluetooth address of the server. This is used later to initiate a connection from the Device or Client.
2. On the Choose mode > prompt, enter Host.
3. Observe the list of all possible commands at this time for a Host. This list is available at any time by entering Help at the Host > prompt.

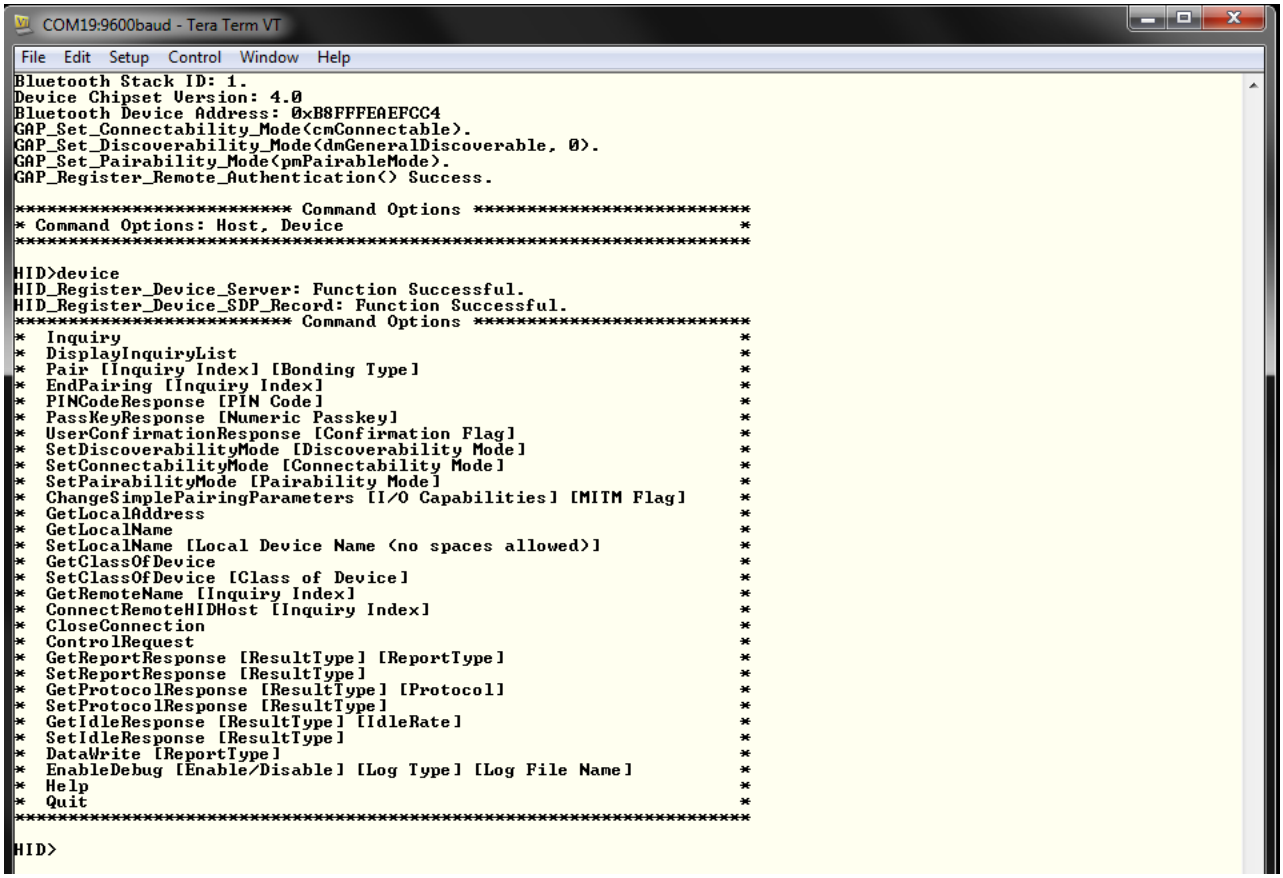


4. At the Host > prompt, enter Inquiry. This initiates the Inquiry process. Once complete, observe the list of all discovered devices.
5. This list is available at any time by choosing DisplayInquiryList at the Host > prompt.

```
HID>inquiry
Return Value is 0 GAP_Perform_Inquiry() SUCCESS.

HID>
GAP Inquiry Entry Result: 0xB8FFFEA92617.

HID>
GAP Inquiry Entry Result: 0x000272212389.

HID>
GAP Inquiry Entry Result: 0x30144A39DDB7.

HID>
GAP Inquiry Entry Result: 0x0007808031B7.

HID>
GAP Inquiry Entry Result: 0xBC0DA5F8D90E.

HID>
GAP Inquiry Entry Result: 0x0002724614C5.

HID>
GAP Inquiry Entry Result: 0x0007808031D1.

HID>
GAP Inquiry Entry Result: 0x00190E05483B.

HID>
GAP Inquiry Entry Result: 0xCDABDEADBEEF.

HID>
GAP Inquiry Entry Result: 0x001638392F2F.

HID>
GAP Inquiry Entry Result: 0x00190E0D47A3.

HID>
GAP Inquiry Entry Result: 0x7C8EE4001E72.

HID>
GAP Inquiry Entry Result: 0x000272D69F2D.

HID>
GAP Inquiry Entry Result: 0xB8FFFEAEFCC4.

HID>
GAP Inquiry Entry Result: 0x000272D69F2E.

HID>
GAP Inquiry Entry Result: 0x10BF48ED2C24.

HID>
GAP Inquiry Entry Result: 0x000272D69F33.

HID>
GAP Inquiry Entry Result: 0x000272D6A6E2.

HID>
GAP Inquiry Entry Result: 0x000780803205.

HID>
GAP Inquiry Entry Result: 0x000272D6A563.

HID>
GAP_Inquiry_Result: 20 Found.
GAP Inquiry Result: 1, 0xB8FFFEA92617.
GAP Inquiry Result: 2, 0x000272212389.
GAP Inquiry Result: 3, 0x30144A39DDB7.
GAP Inquiry Result: 4, 0x0007808031B7.
GAP Inquiry Result: 5, 0xBC0DA5F8D90E.
GAP Inquiry Result: 6, 0x0002724614C5.
GAP Inquiry Result: 7, 0x0007808031D1.
GAP Inquiry Result: 8, 0x00190E05483B.
GAP Inquiry Result: 9, 0xCDABDEADBEEF.
GAP Inquiry Result: 10, 0x001638392F2F.
GAP Inquiry Result: 11, 0x00190E0D47A3.
GAP Inquiry Result: 12, 0x7C8EE4001E72.
GAP Inquiry Result: 13, 0x000272D69F2D.
GAP Inquiry Result: 14, 0xB8FFFEAEFCC4.
GAP Inquiry Result: 15, 0x000272D69F2E.
```

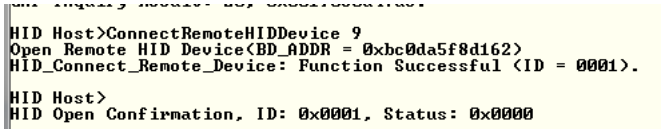
3.2 Device 2 (Client/HID Device Setup on the Demo Application

1. Setup the second board as a Device. Perform the steps mentioned earlier in the Running the Bluetooth Code section to initialize the application. On the Choose mode > prompt, enter Device.
2. Observe the list of all possible commands at this time for a Device. This list is available any time by entering Help at the Device > prompt.

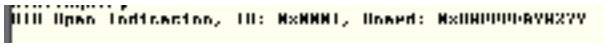


3.3 Initiating Connection from the HID Host

1. Note the index number of the second board that was configured as a HID Device. [If the list is not on the screen, issue the DisplayInquiryList command on the client to display the list of discovered devices.]
2. Issue a ConnectRemoteHIDDevice <Inquiry Index> command from the Device.
3. Wait for HID Open confirmation.



4. When a client successfully connects to a server, the server sees the open indication.



Note

When connecting for the first time, the connection can be initiated from the HID Host Device. If not the connection is rejected as the HIDHost cannot allow a connection from an unknown HID Device.

Note

A status of 0x0000 means the connection was successful. Any other status implies that the connection did not succeed.

3.4 Initiating Connection from HID Device

Initiating a connection from the HID Device is the same as the procedure as the Host except that we run the inquiry (mentioned above in Step 3 of Device 1) on the HID Device instead and issue a ConnectRemoteHIDHost <Inquiry Index> where the Inquiry Index is the number that corresponds to the BD-ADDR of the Host when we run the inquiry.

```
HID>ConnectRemoteHIDHost 12
Open Remote HID Host<BD_ADDR = 0xB8FFFEAEFCC4>
HID_Connect_Remote_Host: Function Successful (ID = 0001).

HID>
HID Open Confirmation, ID: 0x0001, Status: 0x0000
```

3.5 Communication Between Host and Device

1. Now that a connection is established, the host and device can communicate with each other.
2. We can send a control operation from either the host or the device using the ControlRequest command. For the host we issue the Controlrequest < parameter-number > command. The options for control request are 0= hcNop, 1= hcHardReset, 2= hcSoftReset, 3= hcSuspend, 4=hcExitSuspend, 5=hcVirtualCableUnplug. When we type Control Request 5 on the host we get the following message:

   ```
   HID>ControlRequest 5
   HID_Control_Request: Function Successful.

   HID>
   HID Close Indication, ID: 0x0002
   ```

   On the device side, a hcVirtualCableUnplug indication appears.

   ```
   HID>
   HID Control Indication, ID: 0x0001, Control Operation: hcVirtualCableUnplug

   HID>
   HID Close Indication, ID: 0x0001
   ```

   For the DeviceControlRequest has no parameters. This does hcVirtualCableUnplug by default.

   ```
   HID>ControlRequest
   HID_Control_Request: Function Successful.

   HID>
   HID Close Indication, ID: 0x0001
   ```

   On the host side, a hcVirtualCableUnplug indication appears.

   ```
   HID>
   HID Control Indication, ID: 0x0003, Control Operation: hcVirtualCableUnplug

   HID>
   HID Close Indication, ID: 0x0003
   ```

3. Make Report Requests by issuing the GetReportRequest command. This needs either 3 or 4 parameters. The first one is Size which is 0 for using the size of the Report or 1 for using a custom buffer size. The second is ReportType which is 0 for rtOther, 1 for rtInput, 2 for rtOutput, and 3 for rtFeature. The third is ReportID. If a custom buffer for size is used in the first parameter, specify that here. For example, send a Report Request with size of Report, rtInput and ReportID of 2. Observe a Report Request Success indication.

   ```
   HID>GetReportRequest 0 1 2
   HID_Get_Report_Request: Function Successful.
   ```

   On the device, observe a report indication with the report type, ID, size, and buffer size.

   ```
   HID>
   HID Get Report Indication, ID: 0x0003, ReportType: rtInput, ReportID: 2, Size: grSizeOfReport, BufferSize: 0
   ```

   The device can respond to GetReportRequest using GetReportResponse. This needs the Result type (0 for rtSuccessful, 1 for rtNotReady, 2 for rtErrInvalidReportID, 3 forrtErrUnsupportedRequest, 4 for rtErrInvalidParameter, 5 for rtErrUnknown, 6 for rtErrFatal, 7 for rtData) and ReportType (0 for rtOther, 1 for rtInput, 2 for rtOutput, 3 for rtFeature) as parameters. For example, respond to the above rtInput request from thehost with rtData as Result type and rtInput as Report Type.

   ```
   HID>GetReportResponse 7 1
   HID Get Report Response: Function Successful.
   ```

   The host gets a report confirmation back.

   ```
   HID>
   HID Get Report Confirmation, ID: 0x0004, Status: rtData, ReportType: rtInput, ReportLength: 4,
   Report: 0x02 0x80 0x50 0x00
   ```

4. Run SetReportRequest from the Host. The only parameter needed is the report type which is 0 for rtOther, 1 for rtInput, 2 for rtOutput, and 3 for rtFeature.

```
HID>SetReportRequest 1
HID_Set_Report_Request: Function Successful.
```

The device gets a SetReportIndication with the Report type.

```
HID>
HID Set Report Indication, ID: 0x0004, ReportType: rtInput, ReportLength: 4,
Report: 0x02 0x80 0x50 0x00
```

The device can respond to SetReportReques t by issuing the SetReportResponse command. The only parameter needed is Result type (0 for rtSuccessful, 1 for rtNotReady, 2 forrtErrInvalidReportID, 3 for rtErrUnsupportedRequest, 4 for rtErrInvalidParameter, 5 for rtErrUnknown, 6 for rtErrFatal, 7 for rtData). For example, respond to the above rtInputReport request using rtSuccesful.

```
HID>SetReportResponse 0
HID_Set_Report_Response: Function Successful.
```

The host receives a SetReportConfirmation indication with the Result type.

```
HID>
HID Set Report Confirmation, ID: 0x0004, Status: rtSuccessful
```

5. Send a Protocol Request using GetProtocolRequest. This requires no parameters.

```
HID>GetProtocolRequest
HID_Get_Protocol_Request: Function Successful.
```

The device gets a Protocol Indication.

```
HID>
HID Get Protocol Indication, ID: 0x0003
```

The device can respond to the Protocol Request by issuing the GetProtocolResponse command. This requires two parameters: Result type (0 for rtSuccessful, 1 for rtNotReady, 2 forrtErrInvalidReportID, 3 for rtErrUnsupportedRequest, 4 for rtErrInvalidParameter, 5 for rtErrUnknown, 6 for rtErrFatal, and 7 for rtData) and Protocol (0 for ptBoot and 1 forptReport). For example, respond to the previous Request with rtData and ptBoot.

```
HID>GetProtocolResponse 7 0
HID_Get_Protocol_Response: Function Successful.

HID>
```

The host gets a protocol confirmation with the result type and protocol.

```
HID>
HID Get Protocol Confirmation, ID: 0x0003, Status: rtData, Protocol: ptBoot
```

6. Issue SetProtocolRequest from the host. The only parameter needed is the protocol(0 for ptBoot and 1 for ptReport). For example, send a request with ptReport.

```
HID>SetProtocolResponse 0
HID_Set_Protocol_Response: Function Successful.
```

The host gets a Set Protocol Indication along with the protocol. This can respond by issuing the SetProtocolResponse command which requires Result type as the parameter (0 forrtSuccessful, 1 for rtNotReady, 2 for rtErrInvalidReportID, 3 for rtErrUnsupportedRequest, 4 for rtErrInvalidParameter, 5 for rtErrUnknown, 6 for rtErrFatal, and 7 for rtData).

```
HID>
HID Set Protocol Indication, ID: 0x0001, Protocol: ptReport

HID>SetProtocolResponse 0
HID_Set_Protocol_Response: Function Successful.
```

A Protocol Confirmation with Result type in the host.

7. Set the Idle request issuing the GetIdleRequest command. This requires no parameters.

```
HID>GetIdleRequest
HID_Get_Idle_Request: Function Successful.
```

The device gets a GetIdleIndication.

```
HID>
HID Get Idle Indication, ID: 0x0001
```

This can respond with a GetIdleResponse which requires Result type (0 for rtSuccessful, 1 for rtNotReady, 2 for rtErrInvalidReportID, 3 for rtErrUnsupportedRequest, 4 forrtErrInvalidParameter, 5 for rtErrUnknown, 6 for rtErrFatal, and 7 for rtData) and Idle Rate as parameters. For example, respond with a Result type of rtData and Idle Rate of 50.

```
HID>GetIdleResponse 7 50
HID_Get_Idle_Response: Function Successful.
```

The host gets an Idle Confirmation.

```
HID>
HID Get Idle Confirmation, ID: 0x0002, Status: rtData, IdleRate: 50
```

8. Set the Idle Rate using SetIdleRequest from the Host which requires Idle rate as the only parameter. For example we set the Idle rate to 50 from here.

```
HID>SetIdleRequest 50
HID_Set_Idle_Request: Function Successful.
```

The device receives a Set Idle Indication. This can respond using Set Idle Response which requires Result type (0 for rtSuccessful, 1 for rtNotReady, 2 for rtErrInvalidReportID, 3 forrtErrUnsupportedRequest, 4 for rtErrInvalidParameter, 5 for rtErrUnknown, 6 for rtErrFatal, and 7 for rtData) as the one parameter.

```
HID>
HID Set Idle Indication, ID: 0x0001, IdleRate: 50

HID>SetIdleResponse 0
HID_Set_Idle_Response: Function Successful.
```

The host receives a SetIdleConfirmation with the result type.

```
HID>
HID Set Idle Confirmation, ID: 0x0002, Status: rtSuccessful
```

9. Write data between devices using DataWrite. The one parameter needed is ReportType (0 for rtOther, 1 for rtInput, 2 for rtOutput, and 3 for rtFeature). Data is written from the device in the below example.

```
HID>DataWrite 1
HID_Data_Write: Function Successful.
```

The Host receives a Data indication.

```
HID>
HID Data Indication, ID: 0x0002, ReportType: rtInput, ReportLength: 4,
Report: 0x02 0x80 0x50 0x00
```

4 Application Commands

Send a Protocol Request using GetProtocolRequest. This requires no parameters. The device receives a protocol indication. The device can respond to the protocol request by issuing the GetProtocolRequest command. This requires two parameters, Result Type (0 for rtSuccessful, 1 for rtNotReady, 2 for rtErrInvalidReportID, 3 for rtErrUnsupportedRequest, 4 for rtErrInvalidParameter, 5 for rtErrUnknown, 6 for rtErrFatal, and 7 for rtData) and Protocol (0 for ptBoot and 1 for ptReport.) In the example below, the response to the previous request is rtData and ptBoot. The host receives a protocol confirmation with the result type and protocol.

An overview of the application and other applications can be read at TI Dual-Mode Bluetooth® Stack on MSP432™ MCUs and Dual-Mode Bluetooth® Stack on STM32F4 MCUs.

This page describes the various commands that a user of the application can use. Each command is a wrapper over a TI's Bluetooth stack API which gets invoked with the parameters selected by the user. This is a subset of the APIs available to the user. TI's Bluetooth stack API documentation (TI_Bluetooth_Stack_Version-Number\Documentationor forSTM32F4,TI_Bluetooth_Stack_Version-Number\RTOS_VERSION\Documentation) describes all of the API's in detail.

5 Gap Commands

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

5.1 Help (DisplayHelp)

Description

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

Parameters

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

Possible Return Values

This command always returns 0.

5.2 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 lasts 10 seconds unless 20 devices (MAX_INQUIRY_RESULTS) are found before that time limit.

Parameters

Including parameters is not necessary when using this command. A parameter has 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 intMaximumPeriodLength, 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 if 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 fails if an outstanding inquiry is in progress. 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 must set BOTH of these values to zero if a simple InquiryProcedure is used (Non-Periodic). If these two parameters are specified, then these two parameters must satisfy the following formula:

MaximumPeriodLength > MinimumPeriodLength > InquiryLength

5.3 Display Inquiry List

Description

The DisplayInquiryList command exists to display the current Inquiry List with indexes. This command is useful for when a user has forgotten the Inquiry Index for a particular Bluetooth Device the user wants to interact with. This function returns zero on a successful execution and a negative value on all errors. The command requires that a valid BluetoothStack ID exists before running and must be called after using the Inquiry command, since the list is empty without already discovering devices.

Parameters

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

Possible Return Values

- (0) Successful Display of the Inquiry List
- (-8) INVALID_STACK_ID_ERROR

5.4 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 the device tries to pair with). Please 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 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, the pair cannot take place. The second parameter is the bonding type used for the pairing procedure. This is an optional parameter which is only required if General Bonding is needed 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 is 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 is 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_tGAP_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 performing 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 is used to handle Authentication Events that follow if this function is successful. If this function is successful, then all further information is returned through the Registered GAP Event Callback. Please note that if this function returns successfully, 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 DisplayInquiryList Pair 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 and the link is maintained, then the connection is NOT terminated until the GAP_End_Bonding function is called. This allows any higher level initialization that is needed on he 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 informs 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 Connectionis established over the Bluetooth Link that was initiated with this function, the Bluetooth Protocol Stack CAN or CAN NOT terminate the Physical Link when (and if) an L2CAPDisconnect Request (or Response) is issued. If this occurs, then calling the GAP_End_Bonding function has no effect (the GAP_End_Bonding function returns an error code in this case).

5.5 End Pairing

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. Please 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 commandDisplayInquiryList is used. This is 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_tGAP_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 BluetoothDevice address of the remote Bluetooth Device that was specified to be bonded with (general bonding). This function terminates the ACL connection that was established. NO GAP Event Callbacks are issued to the GAP Event Callback that was specified in the original GAP_Initiate_Bonding function call (if this function returns successfully).

5.6 PIN Code Response

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 sees a message displayed during the Pairing Procedure to call this command. A responder receives 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).

5.7 Pass Key Response

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).

5.8 User Confirmation Response

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).

5.9 Set Discoverability Mode

Description

The SetDiscoverabilityMode command is responsible for setting the Discoverability Mode of the local Device. This command returns zero on successful execution and a negative value on all errors. A Bluetooth Stack ID must exist before attempting to call this function. If setting the device as Limited Discoverable, the device is discoverable for 60 seconds; a GeneralDiscoverable device is always 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 specifies the length of time (in seconds) that the local Bluetooth Device is 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 returns to non-discoverable mode.

5.10 Set Connectability Mode

Description

The SetConnectabilityMode command is responsible for setting the Connectability Mode of the local Device. This command returns zero on successful execution and a negative value on all errors. 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.

5.11 Set Pairability Mode

Description

The SetPairabilityMode command is responsible for setting the Pairability Mode of the local Device. This command returns zero on successful execution and a negative value on all errors. 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, than the device *MUST* be set to pairable with SSP enabled.

5.12 Change Simple Pairing Parameters

Description

The ChangeSimplePairingParameters command is responsible for changing the Secure Simple Pairing Parameters that are exchanged during the Pairing procedure when Secure SimplePairing (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

5.13 Get Local Address

Description

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

Parameters

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

Possible Return Values

- (0) Successfully Query Local Address
- (-1) BTPS_ERROR_INVALID_PARAMETER
- (-2) BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-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 is filled with the device address read from the local Bluetooth Device. If this function returns a negative value, then the device address of the local Bluetooth Device was NOT able to be queried (error condition).

5.14 Set Local Name

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.

Note

Do not add spaces to the name or only the first section of the name is set.

Command Call Examples

- "SetLocalName New_Bluetooth_Device_Name" Attempts to set the Local Device Name to "New_Bluetooth_Device_Name."
- "SetLocalName New Bluetooth Device Name" attempts to set the Local Device Name to "New Bluetooth Device Name" but only sets the first parameter, which makes the LocalDevice 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 returns zero if the local device name was successfully changed, or a negative return error code if there was an error condition.

5.15 Get Local Name

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 StackID must exist before attempting to call this function.

Parameters

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

Possible Return Values

- (0) Successfully Queried 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 must 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 is filled with a NULL terminatedASCII 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).

5.16 Set Class of Device

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. Start the value with “0x” and use a six digit value after that. If this is not done, the class of device written is assumed decimal and is converted to hexadecimal format and changes 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 returns zero if the class of device was successfully changed, or a negative return error code if there was an error condition.

5.17 Get Class of Device

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 anegative value on all errors. A Bluetooth Stack ID must exist before attempting to call this function.

Parameters

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

Possible Return Values

- (0) Successfully Queried 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 is 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.

5.18 Get Remote Name

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 is called after using the Inquiry command. The DisplayInquiryList command is useful in this situation to find which remote device goes with which Inquiry Index.

Parameters

The GetRemoteName command requires one parameter which is the Inquiry Index of the Remote Bluetooth Device. This value can be found after an Inquiry or displayed when the command DisplayInquiryList is used.

Command Call Examples

- "GetRemoteName 5" Attempts to query the Device Name for the Remote Device that is at the fifth Inquiry Index.
- "GetRemoteName 8" Attempts to query the Device Name for the Remote Device that is at the eighth Inquiry Index.

Possible Return Values

- (0) Successfully Queried Remote Name
- (-6) INVALID_PARAMETERS_ERROR
- (-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 longCallbackParameter)

Description of API

This function is provided to allow a mechanism to query the user-friendly Bluetooth Device name of the specified remote Bluetooth Device. This function accepts as input the BluetoothDevice 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 is 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 localBluetooth Device. The GAP_Query_Local_Name function cannot 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 notifies the caller of the result via the specified callback. The caller is free to cancel the remote name request at any time by issuing the GAP_Cancel_Query_Remote_Name function and specifying the Bluetooth Device address of the Bluetooth Device that was specified in the original call to this function. Please note that when the callback is cancelled, the operation issues a cancellation attempt, then the callback is cancelled (i.e. the GAP module can still perform the remote name request, but no callback is ever issued).

6 Human Interface Demo Profile

6.1 Host

6.1.1 Connect Remote HID Device

Description

The following function is responsible for Connecting to a Remote HID Device. This function returns zero on successful execution and a negative value on all errors.

Parameters

This command takes Inquiry Index number to work which can be found using the DisplayInquiryList command after an Inquiry has been completed.

Possible Return Values

- (0)HID_Connect_Remote_Device: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1004)BTHID_ERROR_INSUFFICIENT_RESOURCES

API Call

HID_Connect_Remote_Device(BluetoothStackID, InquiryResultList[(TempParam->Params->intParam-1)], &HIDConfiguration, HID_Event_Callback, 0)

API Prototype

int BTPSAPI HID_Connect_Remote_Device(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, HID_Configuration_t *HIDConfiguration, HID_Event_Callback_tEventCallback, unsigned long CallbackParameter)

Description of API

The following function is responsible for opening a connection to a Remote HID Device on the Specified Bluetooth Device. This function accepts as the first parameter the Bluetooth StackID of the Bluetooth Stack which opens the HID Connection. The second parameter specifies the Board Address (NON NULL) of the Remote Bluetooth Device to connect with. The third parameter to this function is the HID Configuration Specification to be used in the negotiation of the L2CAP Channels associated with this Device Client. The final two parameters specify the HID Event Callback function and Callback Parameter, respectively, of the HID Event Callback that is to process any further events associated with this Device Client. 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 represents the HID ID that can be passed to all other functions that require this. Once a Connection is opened to a Remote Device, this can only be closed via a call to the HID_Close_Connection() function (passing in the return value from a successful call to this function as the HID ID input parameter.)

6.1.2 Close Connection

Description

The following function is responsible for closing any ongoing connection. This function returns zero on successful execution and a negative value on all errors.

Parameters

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

Possible Return Values

- (0) HID_Close_Connection: Function Successful
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1004)BTHID_ERROR_INSUFFICIENT_RESOURCES
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Close_Connection(BluetoothStackID, HIDID)

API Prototype

int BTPSAPI HID_Close_Connection(unsigned int BluetoothStackID, unsigned int HIDID)

Description of API

The following function is responsible for closing a HID connection established through a connection made to a Registered Server or a connection that was made by calling either the HID_Open_Remote_Device() or HID_Open_Remote_Host() functions. This function accepts as input the Bluetooth Stack ID of the Bluetooth Protocol Stack that the HID ID specified by the Second Parameter is valid for. This function returns zero if successful, or a negative return error code if an error occurred. Note that if this function is called with the HID ID of a Local Server, the Server remains registered but the connection associated with the specified HID ID is closed.

6.1.3 Control Request

Description

The following function is responsible for sending a HID_CONTROL Transaction to the remote entity. This function returns zero on successful execution and a negative value on all errors.

Parameters

The Control Operation, 0= hcNop, 1= hcHardReset, 2= hcSoftReset, 3= hcSuspend, 4=hcExitSuspend,5=hcVirtualCableUnplug parameter is necessary if the device is a host. Paramaters are not necessary when using this command if the device is not a host because a parameter has no effect on the outcome of ControlRequest.

Possible Return Values

- (0) HID_Control_Request: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Control_Request(BluetoothStackID, HIDID, (HID_Control_Operation_Type_t)((!IsHost)? (hcVirtualCableUnplug):(TempParam->Params->intParam)))

API Prototype

int BTPSAPI HID_Control_Request (unsigned int BluetoothStackID, unsigned int HIDID, HID_Control_Operation_Type_t ControlOperation)

Description of API

The following function is responsible for Sending a HID_CONTROL transaction to the remote side. This function accepts as input the Bluetooth Stack ID of the Bluetooth Stack which is to send the request and the HID ID for which the Connection has been established. The third parameter is the Control Operation . This function returns zero if successful,or a negative return error code if there as an error.

Note

Control Channel Transfers normally consist of two phases, a request by the host and a response by the device. However, HID Control transactions require no response phase. Note that HID Control Requests are not allowed while other transactions are being processed unless the Control Operation Type is hcVirtualCableUnplug which can be sent at any time.

6.1.4 Get Report Request

Description

The following function is responsible for sending a GET_REPORT Transaction to the remote HID Device. This function returns zero on successful execution and a negative value on all errors.

Parameters

This function required three or four parameters. The first is size, 0 = grSizeOfReport, 1 = grUseBufferSize, the second is ReportType 0 = rtOther, 1 = rtInput, 2 = rtOutput, 3 = rtFeature,the third is ReportId. If the size parameter is 1, we need to specify a fourth parameter Buffersize.

Possible Return Values

- (0) HID_Get_Report_Request: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION

API Call

HID_Get_Report_Request(BluetoothStackID, HIDID, (HID_Get_Report_Size_Type_t)TempParam->Params[0].intParam, (HID_Report_Type_Type_t)TempParam->Params[1].intParam, (Byte_t)(TempParam->Params[2].intParam), (Word_t)(TempParam->Params[3].intParam))

API Prototype

HID_Get_Report_Request(unsigned int BluetoothStackID, unsigned int HIDID, HID_Get_Report_Size_Type_t Size, HID_Report_Type_Type_t ReportType, Byte_t ReportID,Word_t BufferSize)

Description of API

The following function is responsible for sending a GET_REPORT transaction to the remote Device. This function accepts as input the Bluetooth Stack ID of the Bluetooth Stack which is to send the request and the HID ID for which the connection has been established. The third parameter is the descriptor that indicates how the device is to determine the size of the buffer that the host has allocated. The fourth parameter is the type of report requested. The fifth parameter is the Report ID determined by the Device's SDP record. Passing a zero for this parameter indicated that this parameter is not used and excludes the appropriate byte from the transaction payload. The fifth parameters use is based on the parameter passed as size. If the host indicates an allocation for a smaller size of buffer than the report is requesting, this parameter is used as the size of the report returned. Otherwise, the appropriate bytes are not included in the transaction payload. This function returns zero if successful or a negative return error code if there was an error.

Note

Control Channel transfers have two phases, a Request by the host and a Response by the device. Only ONE host control channel Request can be outstanding at a time. Reception of a HID Get Report Confirmation event indicates that a Response has been received and the Control Channel is now free for further transactions.

6.1.5 Set Report Request

Description

The following function is responsible for sending a SET_REPORT Transaction to the remote HID Device. This function returns zero on successful execution and a negative value on all errors.

Parameters

SetReportRequest uses one parameter, ReportType 0 = rtOther, 1 = rtInput, 2 = rtOutput, 3 = rtFeature.

Possible Return Values

- (0)HID_Set_Report_Request: Function Successful.
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Set_Report_Request(BluetoothStackID, HIDID, (HID_Report_Type_Type_t)TempParam->Params[1].intParam, size of(GenericMouseReport), GenericMouseReport)

API Prototype

int BTPSAPI HID_Set_Report_Request(unsigned int BluetoothStackID, unsigned int HIDID, HID_Report_Type_Type_t ReportType, Word_t ReportPayloadSize, Byte_t*ReportDataPayload)

Description of API

The following function is responsible for sending a SET_REPORT request to the remote device. This function accepts as input the Bluetooth Stack ID of the Bluetooth Stack which is to send the transaction and the HID ID for which the connection has been established. The third parameter is the type of report being sent. Note that rtOther is an Invalid Report Type for use with this function. The final two parameters to this function are the

length of the Report Payload to send and a pointer to the Report Payload that is sent. This function returns zero if successful or a negative return error code if there was an error.

Note

Control Channel transfers have two phases, a Request by the host and a Response by the device. Only ONE host control channel Request can be outstanding at a time. Reception of an HID Set Report Confirmation event indicates that a Response has been received and the Control Channel is now free for further transactions.

6.1.6 Get Protocol Request

Description

The following function is responsible for sending a GET_PROTOCOL Transaction to the remote HID Device. This function returns zero on successful execution and a negative value on all errors.

Parameters

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

Possible Return Values

- (0)HID_Get_Protocol_Request: Function Successful.
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Get_Protocol_Request(BluetoothStackID, HIDID)

API Prototype

int BTPSAPI HID_Get_Protocol_Request (unsigned int BluetoothStackID, unsigned int HIDID)

Description of API

The following function is responsible for sending a GET_PROTOCOL transaction to the remote HID Device. This function accepts as input the Bluetooth Stack ID of the Bluetooth Stack which is to send the request and the HID ID for which the Connection has been established. This function returns a zero if successful or a negative return error code if there was an error.

Note

Control Channel transfers have two phases, a Request by the host and a Response by the device. Only ONE host control channel request can be outstanding at a time. Reception of a HID Get Protocol Confirmation event indicates that a response has been received and the Control Channel is now free for further transactions.

6.1.7 Set Protocol Request

Description

The following function is responsible for sending a SET_PROTOCOL Transaction to the remote HID Device. This function returns a zero on successful execution and a negative value on all errors.

Parameters

SetProtocolRequest needs one parameter which is Protocol, 0= ptReport and 1=ptBoot.

Possible Return Values

- (0) HID_Set_Protocol_Request: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Set_Protocol_Request(BluetoothStackID, HIDID, (HID_Protocol_Type_t)TempParam->Params[0].intParam)

API Prototype

int BTPSAPI HID_Set_Protocol_Request(unsigned int BluetoothStackID, unsigned int HIDID, HID_Protocol_Type_t Protocol)

Description of API

The following function is responsible for sending a SET_PROTOCOL transaction to the remote HID Device. This function accepts the Bluetooth Stack ID of the Bluetooth Stack which is to send the request and the HID ID for which the connection has been established. The last parameter is the protocol to be set. This function returns a zero if successful or a negative return error code if there was an error.

Note

Control Channel transfers have two phases, a Request by the host and a Response by the device. Only ONE host control channel Request can be outstanding at a time. Reception of a HID Set Protocol Confirmation event indicates that a response has been received and the Control Channel is now free for further transactions.

6.1.8 Get Idle Request

Description

The following function is responsible for sending a GET_IDLE Transaction to the remote HID Device. This function returns zero on successful execution and a negative value on all errors.

Parameters

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

Possible Return Values

- (0) HID_Get_Idle_Request: Function Successful.
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Get_Idle_Request(BluetoothStackID, HIDID)

API Prototype

int BTPSAPI HID_Get_Idle_Request(unsigned int BluetoothStackID, unsigned int HIDID)

Description of API

The following function is responsible for sending a GET_IDLE transaction to the remote HID Device. This function accepts the Bluetooth Stack ID of the Bluetooth Stack which is to send the request and the HID ID for which the Connection has been established. This function returns a zero if successful or a negative return error code if there was an error.

Note

Control Channel transfers have two phases, a Request by the host and a Response by the device. Only ONE host control channel request can be outstanding at a time. Reception of a HID Get Idle Confirmation event indicates that a Response has been received and the Control Channel is now free for further transactions.

6.1.9 Set Idle Request

Description

The following function is responsible for sending a SET_IDLE Transaction to the remote HID Device. This function returns zero on successful execution and a negative value on all errors.

Parameters

SetIdleRequest requires one Parameter which is Idlerate.

Possible Return Values

- (0) HID_Set_Idle_Request: Function Successful
- (-4) FUNCTION_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1004)BTHID_ERROR_INSUFFICIENT_RESOURCES
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Set_Idle_Request(BluetoothStackID, HIDID, (Byte_t)TempParam->Params[0].intParam)

API Prototype

int BTPSAPI HID_Set_Idle_Request(unsigned int BluetoothStackID, unsigned int HIDID, Byte_t IdleRate)

Description of API

The following function is responsible for sending a SET_IDLE transaction to the remote HID Device. This function accepts the Bluetooth Stack ID of the Bluetooth Stack which is to sendthe request and the HID ID for which the Connection has been established. The last parameter is the Idle Rate to be set. The Idle Rate LSB is weighted to 4 ms (i.e. the Idle Rate resolution is 4 ms with a range from 4 ms to 1.020 s). This function returns a zero if successful or a negative return error code if there was an error.

Note

Control Channel transfers have two phases, a Request by the host and a Response by the device. Only ONE host control channel request can be outstanding at a time. Reception of an HID Set Idle Confirmation event indicates that a response has been received and the Control Channel is now free for further transactions.

6.1.10 Data Write

Description

The following function is responsible for sending a DATA Transaction on the Interrupt Channel to the remote entity. This function returns zero on successful execution and a negativevalue on all errors.

Parameters

Requires one parameter which is ReportType, 0 = rtOther, 1 = rtInput, 2 = rtOutput, 3 = rtFeature.

Possible Return Values

- (0) HID_Control_Request: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION

API Call

HID_Data_Write(BluetoothStackID, HIDID, (HID_Report_Type_Type_t)TempParam->Params[0].intParam, sizeof(GenericMouseReport), GenericMouseReport)

API Prototype

int BTPSAPI HID_Data_Write(unsigned int BluetoothStackID, unsigned int HIDID, HID_Report_Type_Type_t ReportType, Word_t ReportPayloadSize, Byte_t *ReportDataPayload)

Description of API

The following function is responsible for sending Reports over the Interrupt Channel. This function accepts the Bluetooth Stack ID of the Bluetooth Stack which is to send the ReportData and the HID ID for which the connection has been established. The third parameter is the type of report being sent. The final two parameters are the Length of the Report Payload to send and a pointer to the Report Payload that is going to be sent. Note that rtOther and rtFeature are Invalid Report Types for use with this function. Also note that rtInput Reports must be sent from the device to the host, and that rtOutput Reports must be sent from the host to the device. This function returns a zero if successful or a negative return error code if there was an error.

6.2 Client

6.2.1 Get Report Response

Description

The following function is responsible for sending a response for an outstanding GET_REPORT Transaction to the remote HID Host. This function returns zero on successful execution and a negative value on all errors.

Parameters

GetReportResponse requires two parameters, ResultType, 0= rtSuccessful,1= rtNotReady, 2= rtErrInvalidReportID, 3= rtErrUnsupportedRequest, 4= rtErrInvalidParameter, 5=rtErrUnknown, 6= rtErrFatal, 7= rtData and ReportType, 0 = rtOther, 1 = rtInput, 2 = rtOutput, 3 = rtFeature.

Possible Return Values

- (0) HID_Get_Report_Response: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Get_Report_Response(BluetoothStackID, HIDID, (HID_Result_Type_t)TempParam->Params[0].intParam, (HID_Report_Type_Type_t)TempParam->Params[1].intParam,sizeof(GenericMouseReport), GenericMouseReport)

API Prototype

int BTPSAPI HID_Get_Report_Response(unsigned int BluetoothStackID, unsigned int HIDID, HID_Result_Type_t ResultType, HID_Report_Type_Type_t ReportType, Word_tReportPayloadSize, Byte_t *ReportDataPayload)

Description of API

The following function is responsible for sending the appropriate Response to an outstanding GET_REPORT transaction. This function accepts the Bluetooth Stack ID of the BluetoothStack which is to send the response and the HID ID for which the connection has been established. The third parameter to this function is the Result Type that is to be associated withth is response. The rtSuccessful Result Type is invalid for use with this function. If the rtNotReady through rtErrFatal Result Statuses are used to respond, a HANDSHAKE response that has a Result Code parameter of the specified Error Condition is sent. If the ResultType specified is rtData, the GET_REPORT transaction is responded to with a DATA Response that has the Report (specified by the final parameter) as the Payload. The fourth parameter is the type of report being sent. Note that rtOther is an Invalid Report Type for use with this function.The final two parameters are the length of the Report Payload to send and a pointer to the Report Payload. This function returns a zero if successful, or a negative return error code if there was an error.

6.2.2 Set Report Response

Description

The following function is responsible for sending a response for an outstanding SET_REPORT Transaction to the remote HID Host. This function returns zero on successful execution and a negative value on all errors.

Parameters

SetReportResponse requires one parameter which is ResultType, 0= rtSuccessful, 1= rtNotReady, 2= rtErrInvalidReportID, 3= rtErrUnsupportedRequest, 4= rtErrInvalidParameter, 5=rtErrUnknown, 6= rtErrFatal, 7= rtData.

Possible Return Values

- (0) HID_Set_Report_Response: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION

API Call

HID_Get_Report_Response(BluetoothStackID, HIDID, (HID_Result_Type_t)TempParam->Params[0].intParam, (HID_Report_Type_Type_t)TempParam->Params[1].intParam,sizeof(GenericMouseReport), GenericMouseReport)

API Prototype

int BTPSAPI HID_Set_Report_Response(unsigned int BluetoothStackID, unsigned int HIDID, HID_Result_Type_t ResultType)

Description of API

The following function is responsible for sending the appropriate Response to an Outstanding SET_REPORT transaction. This function accepts as input the Bluetooth Stack ID of the Bluetooth Stack which is to send the response and the HID ID for which the connection has been established. The third parameter to this function is the Result Type that is to be associated with this response. The rtData Result Type is invalid for use with this function. If the rtSuccessful through rtErrFatal Result Types are specified, this function responds to the SET_REPORT request with a HANDSHAKE response that has a Result Code parameter that matches the specified Result Type. This function returns zero if successful or a negative return error code if there was an error.

6.2.3 Get Protocol Response

Description

The following function is responsible for sending a response for an outstanding GET_PROTOCOL Transaction to the remote HID Host. This function returns zero upon successful execution and a negative value on all errors.

Parameters

GetProtocolResponse requires two parameters which are ResultType, 0= rtSuccessful, 1= rtNotReady, 2= rtErrInvalidReportID, 3= rtErrUnsupportedRequest, 4=rtErrInvalidParameter, 5= rtErrUnknown, 6= rtErrFatal, 7= rtData and Protocol, 0= ptReport and 1=ptBoot.

Possible Return Values

- (0) HID_Get_Protocol_Response: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE

- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1004)BTHID_ERROR_INSUFFICIENT_RESOURCES
- (-1005)BTHID_ERROR_INVALID_OPERATION

API Call

HID_Get_Protocol_Response(BluetoothStackID, HIDID, (HID_Result_Type_t)TempParam->Params[0].intParam, (HID_Protocol_Type_t)TempParam->Params[1].intParam)

API Prototype

int BTPSAPI HID_Get_Protocol_Response(unsigned int BluetoothStackID, unsigned int HIDID, HID_Result_Type_t ResultType, HID_Protocol_Type_t Protocol)

Description of API

The following function is responsible for sending the appropriate Response to an outstanding GET_PROTOCOL transaction. This function accepts the Bluetooth Stack ID of the Bluetooth Stack which is to send the response and the HID ID for which the connection has been established. The third parameter to this function is the Result Type that is to be associated with this response. The rtSuccessful Result Type is invalid for use with this function. If the rtNotReady through rtErrFatal Result Types are specified, this function responds to the GET_PROTOCOL request with a HANDSHAKE response that has a Result Code parameter of the specified Error Condition. If the ResultType specified is rtData, theGET_PROTOCOL transaction is responded to with a DATA Response that has the Protocol type specified as the final parameter as the Payload. This function returns zero if successful or a negative return error code if there was an error.

6.2.4 Set Protocol Response

Description

The following function is responsible for sending a response for an outstanding SET_PROTOCOL transaction to the remote HID Host. This function returns zero on successful execution and a negative value on all errors.

Parameters

SetProtocolResponse requires one parameter which is ResultType, 0= rtSuccessful, 1= rtNotReady, 2= rtErrInvalidReportID, 3= rtErrUnsupportedRequest, 4= rtErrInvalidParameter,5= rtErrUnknown, 6= rtErrFatal, 7= rtData.

Possible Return Values

- (0) HID_Set_Protocol_Response: Function Successful
- (-4) FUNCTION_ERROR
- (-6) INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Set_Protocol_Response(BluetoothStackID, HIDID, (HID_Protocol_Type_t)TempParam->Params[0].intParam)

API Prototype

int BTPSAPI HID_Set_Protocol_Response(unsigned int BluetoothStackID, unsigned int HIDID, HID_Result_Type_t ResultType)

Description of API

The following function is responsible for sending the appropriate Response to an outstanding SET_PROTOCOL transaction. This function accepts the Bluetooth Stack ID of the Bluetooth Stack which is to send the response and the HID ID for which the connection has been established. The third parameter to this function is the Result Type that is to be associated with this response. The rtData Result Type is Invalid for use with this function. If the rtSuccessful through rtErrFatal Result Types are specified then this function responds to the SET_PROTOCOL Transaction with a HANDSHAKE response that has a Result Code parameter that matches the specified Result Type. This function returns zero if successful, or a negative return error code if there was an error.

6.2.5 Get Idle Response

Description

The following function is responsible for sending a response for an outstanding GET_IDLE Transaction to the remote HID Host. This function returns zero on successful execution and a negative value on all errors.

Parameters

GetIdleResponse requires two parameters, ResultType, 0= rtSuccessful, 1= rtNotReady, 2= rtErrInvalidReportID, 3= rtErrUnsupportedRequest, 4= rtErrInvalidParameter, 5=rtErrUnknown, 6= rtErrFatal, 7= rtData and IdleRate.

Possible Return Values

- (0) HID_Set_Idle_Response: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION

API Call

HID_Get_Idle_Response(BluetoothStackID, HIDID, (HID_Result_Type_t)TempParam->Params[0].intParam, (Byte_t)TempParam->Params[1].intParam)

API Prototype

int BTPSAPI HID_Get_Idle_Response(unsigned int BluetoothStackID, unsigned int HIDID, HID_Result_Type_t ResultType, Byte_t IdleRate)

Description of API

The following function is responsible for sending the appropriate Response to an outstanding GET_IDLE transaction. This function accepts the Bluetooth Stack ID of the BluetoothStack which is to send the response and the HID ID for which the connection has been established. The third parameter to this function is the Result Type that is to be associated with this response. The rtSuccessful Result Type is invalid for use with this function. If the rtNotReady through rtErrFatal Result Types are specified, then this function responds to the GET_IDLE Transaction with a HANDSHAKE response that has a Result Code parameter of the specified error condition. If the ResultType specified is rtData the GET_IDLEtransaction is responded to with a DATA Response that has the Idle Rate specified as the final parameter as the payload. This function returns zero if successful, or a negative return error code if there was an error.

6.2.6 Set Idle Response

Description

The following function is responsible for sending a response for an outstanding SET_IDLE Transaction to the remote HID Host. This function returns zero on successful execution and a negative value on all errors.

Parameters

SetIdleResponse requires one parameter which is ResultType, 0= rtSuccessful, 1= rtNotReady, 2= rtErrInvalidReportID, 3= rtErrUnsupportedRequest, 4= rtErrInvalidParameter, 5=rtErrUnknown, 6= rtErrFatal, 7= rtData.

Possible Return Values

- (0) HID_Get_ Idle _Response: Function Successful
- (-4) FUNCTION_ERROR
- (-6)INVALID_PARAMETERS_ERROR
- (-8) INVALID_STACK_ID_ERROR
- (-103)BTPS_ERROR_FEATURE_NOT_AVAILABLE
- (-1000)BTHID_ERROR_INVALID_PARAMETER
- (-1001)BTHID_ERROR_NOT_INITIALIZED
- (-1002)BTHID_ERROR_INVALID_BLUETOOTH_STACK_ID
- (-1005)BTHID_ERROR_INVALID_OPERATION
- (-1006)BTHID_ERROR_REQUEST_OUTSTANDING

API Call

HID_Set_Idle_Response(BluetoothStackID, HIDID, (HID_Result_Type_t)TempParam->Params[0].intParam)

API Prototype

int BTPSAPI HID_Set_Idle_Response(unsigned int BluetoothStackID, unsigned int HIDID, HID_Result_Type_t ResultType)

Description of API

The following function is responsible for sending the appropriate Response to an outstanding SET_IDLE transaction. This function accepts the Bluetooth Stack ID of the BluetoothStack which is to send the response and the HID ID for which the connection has been established. The third parameter to this function is the Result Type that is to be associated with this response. The rtData Result Type is invalid for use with this function.
If the rtSuccessful through rtErrFatal Result Types are specified, then this function responds to the SET_IDLE Transaction with a HANDSHAKE response that has a Result Code parameter that matches the specified Result Type. This function returns zero if successful, or a negative return error code if there was an error.

7 References

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

8 Revision History

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

DATE	REVISION	NOTES
August 2023	*	Initial Release

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265