



# **HCI Transport Layer**

## **(Multi-threaded O/S)**

### **Application Programming Interface**

### **Reference Manual**

**Release: 4.0.1**  
**January 10, 2014**



Bluetooth and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc., USA and licensed to Stonestreet One, LLC. Bluetopia®, Stonestreet One™, and the Stonestreet One logo are registered trademarks of Stonestreet One, LLC, Louisville, Kentucky, USA. All other trademarks are property of their respective owners.  
Copyright © 2000-2014 by Stonestreet One, LLC. All rights reserved.

## Table Of Contents

<b><u>1. INTRODUCTION.....</u></b>	<b><u>3</u></b>
1.1 Scope .....	3
1.2 Applicable Documents .....	4
1.3 Acronyms and Abbreviations .....	5
<b><u>2. HCI TRANSPORT LAYER API.....</u></b>	<b><u>7</u></b>
2.1 HCI Transport Layer Over UART Commands.....	7
HCITR_COMOpen .....	7
HCITR_COMWrite .....	8
HCITR_COMReconfigure .....	9
HCITR_COMClose .....	9
HCITR_COMProcess .....	10
2.2 HCI Transport Driver Data Callback for UART Prototype .....	10
HCITR_COMDataCallback_t .....	10
2.3 HCI Transport Layer Over Non-UART Commands .....	11
HCITR_USBOpen .....	12
HCITR_USBWrite .....	12
HCITR_USBReconfigure .....	13
HCITR_USBChangeSCOConfiguration .....	14
HCITR_USBClose .....	14
HCITR_USBProcess .....	15
2.4 HCI Transport Driver Data Callback for Non-UART Prototype .....	15
HCITR_USBDDataCallback_t .....	16
<b><u>3. FILE DISTRIBUTIONS.....</u></b>	<b><u>17</u></b>
<b><u>4. HCI TRANSPORT HEADER FILE.....</u></b>	<b><u>18</u></b>

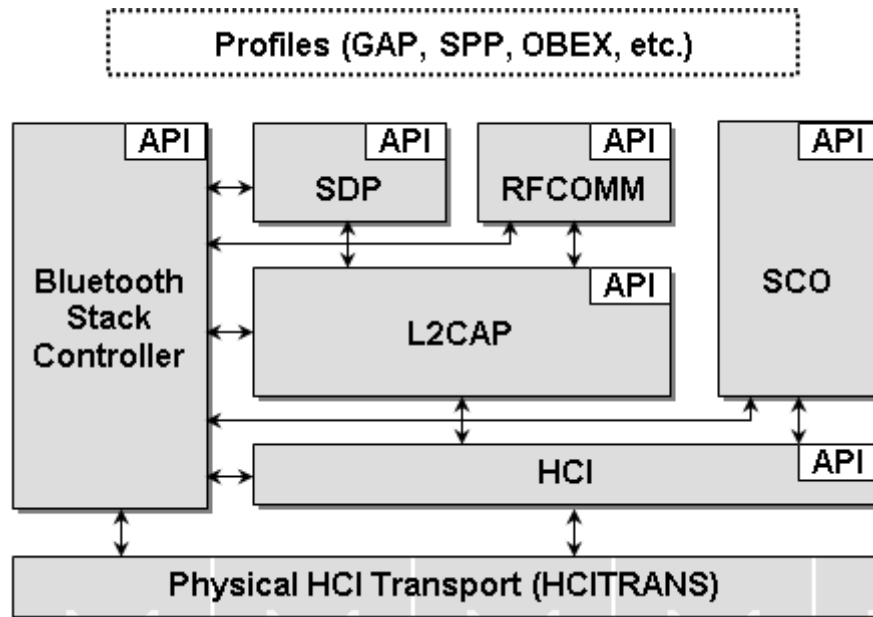
# 1. Introduction

Bluetopia®, the Bluetooth Protocol Stack by Stonestreet One provides a software architecture that encapsulates the upper functionality of the Bluetooth Protocol Stack. More specifically, this stack is a software solution that resides above the Physical HCI (Host Controller Interface) Transport Layer and extends through the L2CAP (Logical Link Control and Adaptation Protocol) and the SCO (Synchronous Connection-Oriented) Link layers. In addition to basic functionality at these layers, the Bluetooth Protocol Stack by Stonestreet One provides implementations of the Service Discovery Protocol (SDP), RFCOMM (the Radio Frequency serial COMMunications port emulator), and several of the Bluetooth Profiles. Program access to these layers, services, and profiles is handled via Application Programming Interface (API) calls.

This document focuses on the API reference that contains a description of all programming interfaces for Stonestreet One's HCI Transport Layer.

## 1.1 Scope

This reference manual provides information on the HCI Transport Layer API identified in Figure 1-1 below. These APIs are used by Bluetopia® to physically communicate with an attached Bluetooth Device. The implementation of these functions can be changed to allow the underlying transport mechanism for the Bluetooth device to change without impact on the Bluetopia® library. This mechanism allows the programmer the opportunity to change transport parameters as needed to support the transport required.



**Figure 1-1 The Stonestreet One Bluetooth Protocol Stack**

## 1.2 Applicable Documents

The following documents may be used for additional background and technical depth regarding the Bluetooth technology.

- 1 *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 2.1+EDR, July 26, 2007.
- 2 *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 2.1+EDR, July 26, 2007.
- 3 *Specification of the Bluetooth System, Volume 2, Core System Package [Controller Volume]*, version 2.1+EDR, July 26, 2007.
- 4 *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 2.1+EDR, July 26, 2007.
- 5 *Specification of the Bluetooth System, Volume 4, Host Controller Interface*, version 2.1+EDR, July 26, 2007.
- 6 *Specification of the Bluetooth System, Bluetooth Core Specification Addendum 1*, June 26, 2008.
- 7 *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 3.0+HS, April 21, 2009.
- 8 *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 3.0+HS, April 21, 2009.

- 9 *Specification of the Bluetooth System, Volume 2, Core System Package [Controller Volume]*, version 3.0+HS, April 21, 2009.
- 10 *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 3.0+HS, April 21, 2009.
- 11 *Specification of the Bluetooth System, Volume 4, Host Controller Interface [Transport Layer]*, version 3.0+HS, April 21, 2009.
- 12 *Specification of the Bluetooth System, Volume 5, Core System Package [AMP Controller Volume]*, version 3.0+HS, April 21, 2009.
- 13 *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 4.0, June 30, 2010.
- 14 *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 4.0, June 30, 2010.
- 15 *Specification of the Bluetooth System, Volume 2, Core System Package [BR/EDR Controller Volume]*, version 4.0, June 30, 2010.
- 16 *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 4.0, June 30, 2010.
- 17 *Specification of the Bluetooth System, Volume 4, Host Controller Interface [Transport Layer]*, version 4.0, June 30, 2010.
- 18 *Specification of the Bluetooth System, Volume 5, Core System Package [AMP Controller Volume]*, version 4.0, June 30, 2010.
- 19 *Specification of the Bluetooth System, Volume 6, Core System Package [Low Energy Controller Volume]*, version 4.0, June 30, 2010.
- 20 *Bluetopia® Protocol Stack, System Call Requirements*, version 4.0, June 30, 2011
- 21 *Bluetopia® Protocol Stack, Application Programming Interface Reference Manual*, version 4.0, June 30, 2011.

### 1.3 Acronyms and Abbreviations

Acronyms and abbreviations used in this document and other Bluetooth specifications are listed in the table below.

Term	Meaning
API	Application Programming Interface
BD_ADDR	Bluetooth Device Address
BT	Bluetooth
HCI	Host Controller Interface
HS	High Speed

Term	Meaning
LSB	Least Significant Bit
LE	Low Energy
MSB	Most Significant Bit
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

## 2. HCI Transport Layer API

The HCI Transport Layer programming interface defines the protocols and procedures to be used to implement the HCI transport layer. The HCI Transport Layer commands are listed in section 2.1 through 2.4. There are two sections of HCI Transport Layer commands. Section 2.1 and 2.2 are for HCI Transport over a UART/Serial connection and Section 2.3 and 2.4 is for HCI Transport over other transport mechanisms (such as USB). The actual prototypes and constants outlined in this section can be found in the **HCITRANS.H** header file in the Bluetopia distribution. Note that usually only a single transport is specified for a system (i.e. either UART style or USB style transport). It is not a requirement that a given system support both styles of transport.

### 2.1 HCI Transport Layer Over UART Commands

The available HCI Transport Layer over UART functions are listed in the table below and are described in the text that follows.

Function	Description
HCITR_COMOpen	Open the HCI Transport Layer.
HCITR_COMWrite	Send data through the HCI Transport Layer.
HCITR_COMReconfigure	Reconfigure the HCI Transport Layer.
HCITR_COMClose	Close the HCI Transport Layer.
HCITR_COMProcess	Instruct module to read/process COM data (non-threaded environment only)

#### HCITR\_COMOpen

The following function is responsible for opening the HCI Transport layer that will be used by Bluetopia to send and receive data. This function must be successfully issued in order for Bluetopia to function. This function accepts as its parameters the HCI COM Transport COM Information that is to be used to open the port as it's first parameter. The final two parameters specify the HCI Transport callback that will be called whenever HCI Data is received by the HCI Transport Driver as well as a caller defined callback parameter that will be passed to the caller when data is received. The second parameter must point to valid HCI Transport Driver Callback and CANNOT BE NULL. A successful call to this function will return a non-zero, positive value which specifies the HCITransportID that is used with the remaining transport functions in this module. This function returns a negative return value to signify an error.

**Prototype:**

```
int BTPSAPI HCITR_COMOpen(  
    HCI_COMMDriverInformation_t *COMMDriverInformation,  
    HCITR_COMDataCallback_t COMDataCallback, unsigned long CallbackParameter)
```

**Parameters:**

COMMDriverInformation	COM configuration information used to open the HCI transport (passed at stack initialization).
COMDataCallback	HCI transport callback function (see section 2.2) that is called whenever data is received over the UART transport.
CallbackParameter	HCI transport callback function parameter (see section 2.2) that is passed to the HCI transport callback function whenever data is received over the UART transport.

**Return:**

Non-Zero for success which represents the HCI Transport ID that is used to specify the open COM port (used when closing, reconfiguring, reading, and writing to the port).

Negative value to signify an error

**HCITR\_COMWrite**

The following function is responsible for actually sending data through the opened HCI Transport layer. Bluetopia® uses this function to send formatted HCI packets to the attached Bluetooth Device. This function **MUST NOT** return until all of the data is sent (or an error condition occurs). Bluetopia® **WILL NOT** attempt to call this function repeatedly if data fails to be delivered. This function must block until it has either buffered the specified data into the transport or sent all of the specified data to the Bluetooth device. The type of data (Command, ACL, SCO, etc.) is NOT passed to this function because it is assumed that this information is contained in the Data Stream being passed to this function.

**Prototype:**

```
int BTPSAPI HCITR_COMWrite(unsigned int HCITransportID, unsigned int Length,  
    unsigned char *Buffer)
```

**Parameters:**

HCITransportID	HCI transport ID of the transport that is to send the data. This value must be obtained via a successful call to the <b>HCITR_COMOpen()</b> function.
Length	The number of bytes that are to be sent, pointed to by the second parameter, Buffer.
Buffer	Pointer to buffer that contains the data to be written.



**Return:**

Zero if all data was transferred successfully.

Negative value if an error occurred.

**HCITR\_COMReconfigure** .

The following function is responsible for instructing the specified HCI Transport layer (first parameter) that was opened via a successful call to the **HCITR\_COMOpen()** function to reconfigure itself with the specified information. This information is completely opaque to the upper layers and is passed through the HCI Driver layer to the transport untouched. It is the responsibility of the HCI Transport driver writer to define the contents of this member (or completely ignore it). This function is provided to allow a mechanism for applications to change UART parameters when the stack is open, for instance to change the baud rate that is used to communicate with the locally connected device. Note that Bluetopia does not call this function from anywhere internally, rather, this function can be called by user defined code in the vendor specific HCI functions (if required).

**Prototype:**

```
void BTPSAPI HCITR_COMReconfigure(unsigned int HCITransportID,  
    void *TransportDriverContext)
```

**Parameters:**

HCITransportID	HCI transport ID of the transport that is to be reconfigured. This value must be obtained via a successful call to the <b>HCITR_COMOpen()</b> function.
TransportDriverContext	Platform specific transport configuration/context information. The format of this parameter is completely caller defined.

**Return:****HCITR\_COMClose** .

The following function is responsible for closing the HCI Transport layer that was opened via a successful call to the **HCITR\_COMOpen()** function. Bluetopia makes a call to this function whenever an error occurs during initialization and the driver has been opened OR when the stack is closed. Once this function completes, the transport layer that was closed will no longer process received data until the transport layer is Re-Opened by calling the **HCITR\_COMOpen()** function.

**Prototype:**

```
void BTPSAPI HCITR_COMClose(unsigned int HCITransportID)
```

**Parameters:**

HCITransportID	HCI transport ID of the transport that is to be closed. This value must be obtained via a successful call to the <b>HCITR_COMOpen()</b> function.
----------------	---

**Return:****HCITR\_COMProcess**

The following function is responsible for forcing the specified COM Transport to process incoming COM Data. This function **IS ONLY APPLICABLE** for single/non-threaded operating environments. This function **IS NEVER CALLED** in multi-threaded Bluetopia implementations. This function should read any incoming COM Data that has been received and dispatch the data through the callback that was registered with the **HCITR\_COMOpen()** function.

**Prototype:**

```
void BTPSAPI HCITR_COMProcess(unsigned int HCITransportID)
```

**Parameters:**

HCITransportID	HCI transport ID of the transport that is process COM data. This value must be obtained via a successful call to the <b>HCITR_COMOpen()</b> function.
----------------	---

**Return:**

## 2.2 HCI Transport Driver Data Callback for UART Prototype

The following declared type represents the Prototype Function for an HCI Transport Driver UART Data Callback.

**HCITR\_COMDataCallback\_t**

This function will be called whenever HCI Packet Information has been received by the HCI Transport Driver. This function passes to the caller the actual data that was received (length of data followed by a pointer to the data). The caller is free to use the contents of the HCI data **ONLY** in the context of this callback. If the caller requires the data for a longer period of time, then the callback function **MUST** copy the data into another data Buffer. This function is guaranteed **NOT** to be invoked more than once simultaneously for the specified installed callback (i.e. this function **DOES NOT** have to be reentrant). It should be noted that this function is called in the thread context of a thread that the user does **NOT** own. Therefore, processing in this function should be as efficient as possible (this argument holds anyway because data Processing (receiving) will be suspended while this function call is outstanding). The type of data (Event, ACL, SCO, etc.) is not passed to the caller in this callback because it is assumed that this information is contained in the data stream being passed to the caller.

**Prototype:**

```
void (BTPSAPI *HCITR_COMDataCallback_t)( unsigned int HCITransportID,  
    unsigned int DataLength, unsigned char *DataBuffer , unsigned long CallbackParameter)
```

**Parameters:**

HCITransportID	HCI transport ID of the HCI transport that has received the data.
DataLength	Length of data in the DataBuffer.
DataBuffer	Pointer to the data buffer that contains the data received.
CallbackParameter	Callback parameter that was registered with the callback function when the callback function was registered.

**Return:**

## 2.3 HCI Transport Layer Over Non-UART Commands

The available HCI Transport Layer Over Non-UART command functions are listed in the table below and are described in the text that follows.

Function	Description
HCITR_USBOpen	Open the HCI Transport Layer.
HCITR_USBWrite	Send data through the HCI Transport Layer.
HCITR_USBREconfigure	Reconfigure the HCI Transport Layer.
HCITR_USBChangeSCOConfiguration	Change USB Isochronous Endpoint Configuration (for SCO data).
HCITR_USBClose	Close the HCI Transport Layer.
HCITR_USBProcess	Instruct module to read/process USB data (non-threaded environment only)

## HCITR\_USBOpen

The following function is responsible for opening the HCI Transport layer that will be used by Bluetopia to send and receive data. This function must be successfully issued in order for Bluetopia to function. This function accepts as its parameters the HCI USB Transport USB Information that is to be used to open the port as it's first parameter. The final two parameters specify the HCI Transport callback that will be called whenever HCI Data is received by the HCI Transport Driver as well as a caller defined callback parameter that will be passed to the caller when data is received. The second parameter must point to valid HCI Transport Driver Callback and CANNOT BE NULL. A successful call to this function will return a non-zero, positive value which specifies the HCITransportID that is used with the remaining transport functions in this module. This function returns a negative return value to signify an error.

### Prototype:

```
int BTPSAPI HCITR_USBOpen(  
    HCI_USBDriverInformation_t *USBDriverInformation,  
    HCITR_USBDataCallback_t USVDataCallback, unsigned long CallbackParameter)
```

### Parameters:

USBDriverInformation	USB configuration information used to open the HCI transport (passed at stack initialization).
USBDataCallback	HCI transport callback function (see section 2.2) that is called whenever data is received over the USB transport.
CallbackParameter	HCI transport callback function parameter (see section 2.2) that is passed to the HCI transport callback function whenever data is received over the USB transport.

### Return:

Non-Zero for success which represents the HCI Transport ID that is used to specify the open USB port (used when closing, reconfiguring, reading, and writing to the port).

Negative value to signify an error

## HCITR\_USBWrite

The following function is responsible for actually sending data through the opened HCI Transport layer. Bluetopia® uses this function to send formatted HCI packets to the attached Bluetooth Device. This function **MUST NOT** return until all of the data is sent (or an error condition occurs). Bluetopia® **WILL NOT** attempt to call this function repeatedly if data fails to be delivered. This function must block until it has either buffered the specified data into the transport or sent all of the specified data to the Bluetooth device. The first parameter to this function specifies the HCI transport that is to be used to send the specified data. The second parameter specifies the HCI packet type of the data that is to be sent. The final two parameters specify the length and the actual data (respectively) to send to the device.

**Prototype:**

```
int BTPSAPI HCITR_USBWrite(unsigned int HCITransportID, HCI_PacketType_t
    HCIPacketType, unsigned int Length, unsigned char *Buffer)
```

**Parameters:**

HCITransportID	HCI transport ID of the transport that is to send the specified data. This value must be obtained via a successful call to the <b>HCITR_USBOpen()</b> function.
HCIPacketType	The packet type of the data that is to be sent over the HCI transport. This value must be one of the following: ptHCICommandPacket ptHCIACLDDataPacket ptHCISCODDataPacket ptHCIEventPacket
Length	The number of bytes that are to be sent, pointed to by the second parameter, Buffer.
Buffer	Pointer to buffer that contains the data to be written.

**Return:**

Zero if all data was transferred successfully.

Negative value if an error occurred.

**HCITR\_USBReconfigure**

The following function is responsible for instructing the specified HCI Transport layer (first parameter) that was opened via a successful call to the **HCITR\_USBOpen()** function to reconfigure itself with the specified information. This information is completely opaque to the upper layers and is passed through the HCI Driver layer to the transport untouched. It is the responsibility of the HCI Transport driver writer to define the contents of this member (or completely ignore it). This function is provided to allow a mechanism for applications to change UART parameters when the stack is open, for instance to change the baud rate that is used to communicate with the locally connected device. Note that Bluetopia does not call this function from anywhere internally, rather, this function can be called by user defined code in the vendor specific HCI functions (if required).

**Prototype:**

```
void BTPSAPI HCITR_USBReconfigure(unsigned int HCITransportID,
    void *TransportDriverContext)
```

**Parameters:**

HCITransportID	HCI transport ID of the transport that is to be reconfigured. This value must be obtained via a successful call to the <b>HCITR_USBOpen()</b> function.
----------------	---

TransportDriverContext      Platform specific transport configuration/context information.  
The format of this parameter is completely caller defined.

**Return:**

## HCITR\_USBChangeSCOConfiguration

The following function changes the settings for the specified USB HCI Transport regarding HCI SCO Bandwidth Configuration. This function is applicable to USB Devices in particular because the specification supports dynamic Bandwidth Allocation on the USB Interface (Isochronous Endpoint configuration).

**Prototype:**

```
int BTPSAPI HCITR_USBChangeSCOConfiguration(unsigned int HCITransportID,
      HCI_SCOConfiguration_t SCOConfiguration)
```

**Parameters:**

HCITransportID	HCI transport ID of the transport that is to be reconfigured. This value must be obtained via a successful call to the <b>HCITR_USBOpen()</b> function.
SCOConfiguration	Specifies the new SCO configuration (Isochronous Endpoint configuration) that is be used. This value must be one of:  <div style="margin-left: 40px;"> hscNoChannels,  hscOneChannel8BitVoice,  hscOneChannel16BitVoice  hscTwoChannel8BitVoice  hscTwoChannel16BitVoice  hscThreeChannel8BitVoice  hscThreeChannel16BitVoice </div>

**Return:**

Zero if specified SCO configuration was set successfully.  
Negative value if an error occurred.

## HCITR\_USBClose

The following function is responsible for closing the HCI Transport layer that was opened via a successful call to the **HCITR\_USBOpen()** function. Bluetopia makes a call to this function whenever an error occurs during initialization and the driver has been opened OR when the stack is closed. Once this function completes, the transport layer that was closed will no longer process received data until the transport layer is Re-Opened by calling the **HCITR\_USBOpen()** function.

**Prototype:**

```
void BTPSAPI HCITR_USBClose(unsigned int HCITransportID)
```

**Parameters:**

HCITransportID	HCI transport ID of the transport that is to be closed. This value must be obtained via a successful call to the <b>HCITR_USBOpen()</b> function.
----------------	---

**Return:****HCITR\_USBProcess**

The following function is responsible for forcing the specified USB Transport to process incoming USB Data. This function **IS ONLY APPLICABLE** for single/non-threaded operating environments. This function **IS NEVER CALLED** in multi-threaded Bluetopia implementations. This function should read any incoming USB Data that has been received and dispatch the data through the callback that was registered with the **HCITR\_USBOpen()** function.

**Prototype:**

void BTPSAPI **HCITR\_USBProcess**(unsigned int HCITransportID)

**Parameters:**

HCITransportID	HCI transport ID of the transport that is process USB data. This value must be obtained via a successful call to the <b>HCITR_USBOpen()</b> function.
----------------	---

**Return:**

## 2.4 HCI Transport Driver Data Callback for Non-UART Prototype

The following declared type represents the Prototype Function for an HCI Transport Driver Data Callback for non UART transports.

## HCITR\_USBDataCallback\_t

This function will be called whenever the HCI transport driver has received HCI packet information. This function passes to the caller the type of Packet Information that was received (Event, ACL, SCO, etc.), followed by the actual data that was received (Length of Data followed by a pointer to the data). The caller is free to use the contents of the HCI Data ONLY in the context of this callback. If the caller requires the Data for a longer period of time, then the callback function MUST copy the data into another Data Buffer. This function is guaranteed NOT to be invoked more than once simultaneously for the specified installed callback (i.e. this function DOES NOT have to be reentrant). It should be noted that this function is called in the Thread Context of a Thread that the User does NOT own. Therefore, processing in this function should be as efficient as possible (this argument holds anyway because Packet Processing (Receiving) will be suspended while this function call is outstanding). The type of data (Event, ACL, SCO, etc.) is passed to the caller in this callback because it is assumed that this information is NOT contained in the Data Stream being passed to the caller.

### Prototype:

```
void (BTPSAPI *HCITR_USBDataCallback_t)(unsigned int HCITransportID,  
    HCI_PacketType_t HCIPacketType, unsigned int DataLength,  
    unsigned char *DataBuffer, unsigned long CallbackParameter)
```

### Parameters:

HCITransportID	HCI transport ID of the HCI transport that has received the data.
HCIPacketType	HCIPacketType enumerated data type. This is declared as follows and indicates what type of packet is in the buffer: ptHCICommandPacket, ptHCIACLDataPacket, ptHCISCODataPacket, ptHCIEventPacket
DataLength	Number of bytes in the DataBuffer.
DataBuffer	Pointer to a data buffer containing the packet data received.
CallbackParameter	HCI transport callback function parameter (see section 2.2) that is passed to the HCI transport callback function whenever data is received over the USB transport.

### Return:



### 3. File Distributions

The source and header files required for the HCI Transport layer to be used with Bluetopia®, the Bluetooth Protocol Stack by Stonestreet One, are listed in the table below.

File	Contents/Description
HCITRANS.h	Bluetopia® HCI Transport layer header module.
HCITRANS.c	Bluetopia® HCI Transport layer source code module.

## 4. HCI Transport Header File

```

/****< hcitrans.h >*****/
/*      Copyright 2000 - 2012 Stonestreet One.      */
/*      All Rights Reserved.      */
/*      */
/*      HCITRANS - HCI Transport Layer for use with Bluetopia.      */
/*      */
/*      Author:  Rory Sledge      */
/*      */
/**** MODIFICATION HISTORY *****/
/*      mm/dd/yy  F. Lastname      Description of Modification      */
/*      -----      -----      -----      */
/*      10/25/01  R. Sledge      Initial creation.      */
/*****/
#ifndef __HCITRANSH__
#define __HCITRANSH__

#include "BTAPIType.h"      /* Bluetooth API Type Definitions.      */
#include "HCITypes.h"      /* Bluetooth HCI Type Definitions/Constants.      */

#define HCITR_ERROR_UNABLE_TO_OPEN_TRANSPORT (-1)      /* Denotes that the      */
/* there was an error      */
/* opening the      */
/* transport layer.      */

#define HCITR_ERROR_READING_FROM_PORT (-2)      /* Denotes that an      */
/* error occurred      */
/* reading from the      */
/* transport layer      */
/* port.      */

#define HCITR_ERROR_WRITING_TO_PORT (-3)      /* Denotes that an      */
/* error occurred      */
/* writing to the      */
/* transport layer      */
/* port.      */

/* The following declared type represents the Prototype Function for */
/* an HCI Transport Driver Data Callback for COM data. This function*/
/* will be called whenever HCI Packet Information has been received */
/* by the HCI Transport Driver. This function passes to the caller */
/* the actual data that was received (Length of Data followed by a */
/* pointer to the data). The caller is free to use the contents of */
/* the HCI Data ONLY in the context of this callback. If the caller */
/* requires the Data for a longer period of time, then the callback */
/* function MUST copy the data into another Data Buffer. This */
/* function is guaranteed NOT to be invoked more than once */
/* simultaneously for the specified installed callback (i.e. this */
/* function DOES NOT have be reentrant). It should be noted that */
/* this function is called in the Thread Context of a Thread that the */
/* User does NOT own. Therefore, processing in this function should */
/* be as efficient as possible (this argument holds anyway because */
/* Packet Processing (Receiving) will be suspended while this */
/* function call is outstanding). */
/* * NOTE * The type of data (Event, ACL, SCO, etc.) is not passed to */
/* the caller in this callback because it is assumed that */
/* this information is contained in the Data Stream being */
/* passed to the caller. */
typedef void (BTAPI *HCITR_COMDataCallback_t)(unsigned int HCITransportID, unsigned int
DataLength, unsigned char *DataBuffer, unsigned long CallbackParameter);

/* The following function is responsible for opening the HCI      */
/* Transport layer that will be used by Bluetopia to send and receive*/

```

```

/* COM (Serial) data. This function must be successfully issued in */
/* order for Bluetopia to function. This function accepts as its */
/* parameter the HCI COM Transport COM Information that is to be used*/
/* to open the port. The final two parameters specify the HCI */
/* Transport Data Callback and Callback Parameter (respectively) that*/
/* is to be called when data is received from the UART. A successful*/
/* call to this function will return a non-zero, positive value which*/
/* specifies the HCITransportID that is used with the remaining */
/* transport functions in this module. This function returns a */
/* negative return value to signify an error. */
int BTPSAPI HCITR_COMOpen(HCI_COMMDriverInformation_t *COMMDriverInformation,
HCITR_COMDataCallback_t COMDataCallback, unsigned long CallbackParameter);

/* The following function is responsible for closing the specific */
/* HCI Transport layer that was opened via a successful call to the */
/* HCITR_COMOpen() function (specified by the first parameter). */
/* Bluetopia makes a call to this function whenever an either */
/* Bluetopia is closed, or an error occurs during initialization and */
/* the driver has been opened (and ONLY in this case). Once this */
/* function completes, the transport layer that was closed will no */
/* longer process received data until the transport layer is */
/* Re-Opened by calling the HCITR_COMOpen() function. */
/* * NOTE * This function *MUST* close the specified COM Port. */
/* This module will then call the registered COM Data */
/* Callback function with zero as the data length and NULL */
/* as the data pointer. This will signify to the HCI */
/* Driver that this module is completely finished with the */
/* port and information and (more importantly) that NO */
/* further data callbacks will be issued. In other words */
/* the very last data callback that is issued from this */
/* module *MUST* be a data callback specifying zero and NULL*/
/* for the data length and data buffer (respectively). */
void BTPSAPI HCITR_COMClose(unsigned int HCITransportID);

/* The following function is responsible for instructing the */
/* specified HCI Transport layer (first parameter) that was opened */
/* via a successful call to the HCITR_COMOpen() function to */
/* reconfigure itself with the specified information. This */
/* information is completely opaque to the upper layers and is passed*/
/* through the HCI Driver layer to the transport untouched. It is */
/* the responsibility of the HCI Transport driver writer to define */
/* the contents of this member (or completely ignore it). */
/* * NOTE * This function does not close the HCI Transport specified */
/* by HCI Transport ID, it merely reconfigures the */
/* transport. This means that the HCI Transport specified */
/* by HCI Transport ID is still valid until it is closed */
/* via the HCITR_COMClose() function. */
void BTPSAPI HCITR_COMReconfigure(unsigned int HCITransportID, void *TransportDriverContext);

/* The following function is provided to allow a mechanism for */
/* modules to force the processing of incoming COM Data. */
/* * NOTE * This function is only applicable in device stacks that */
/* are non-threaded. This function has no effect for device*/
/* stacks that are operating in threaded environments. */
void BTPSAPI HCITR_COMProcess(unsigned int HCITransportID);

/* The following function is responsible for actually sending data */
/* through the opened HCI Transport layer (specified by the first */
/* parameter). Bluetopia uses this function to send formatted HCI */
/* packets to the attached Bluetooth Device. The second parameter to*/
/* this function specifies the number of bytes pointed to by the */
/* third parameter that are to be sent to the Bluetooth Device. This*/
/* function returns a zero if the all data was transfered successfully*/
/* or a negative value if an error occurred. This function MUST NOT */
/* return until all of the data is sent (or an error condition */
/* occurs). Bluetopia WILL NOT attempt to call this function */
/* repeatedly if data fails to be delivered. This function will */
/* block until it has either buffered the specified data or sent all */
/* of the specified data to the Bluetooth Device. */

```

```

/* * NOTE * The type of data (Command, ACL, SCO, etc.) is NOT passed */
/*          to this function because it is assumed that this          */
/*          information is contained in the Data Stream being passed */
/*          to this function.                                         */
int BTPSAPI HCITR_COMWrite(unsigned int HCITransportID, unsigned int Length, unsigned char
*Buffer);

/* Optional USB Support.                                             */

#ifdef __INCLUDE_USB_SUPPORT__

/* The following declared type represents the Prototype Function for */
/* an HCI Transport Driver Data Callback. This function will be      */
/* called whenever HCI Packet Information has been received by the   */
/* HCI Transport Driver. This function passes to the caller the type */
/* of Packet that was received, followed by the actual data that was */
/* received (Length of Data followed by a pointer to the data). The */
/* caller is free to use the contents of the HCI Data ONLY in the    */
/* context of this callback. If the caller requires the Data for a   */
/* longer period of time, then the callback function MUST copy the   */
/* data into another Data Buffer. This function is guaranteed NOT to */
/* be invoked more than once simultaneously for the specified       */
/* installed callback (i.e. this function DOES NOT have to be      */
/* reentrant). It should be noted that this function is called in   */
/* the Thread Context of a Thread that the User does NOT own.      */
/* Therefore, processing in this function should be as efficient as */
/* possible (this argument holds anyway because Packet Processing   */
/* (Receiving) will be suspended while this function call is      */
/* outstanding).                                                    */
typedef void (BTPSAPI *HCITR_USBDataCallback_t)(unsigned int HCITransportID, HCI_PacketType_t
HCIPacketType, unsigned int DataLength, unsigned char *DataBuffer, unsigned long
CallbackParameter);

/* The following function is responsible for opening the HCI         */
/* Transport layer that will be used by Bluetopia to send and receive*/
/* USB data. This function must be successfully issued in order for */
/* Bluetopia to function. This function accepts as its parameter the*/
/* HCI USB Transport USB Information that is to be used to open the */
/* port. The final two parameters specify the HCI Transport Data    */
/* Callback and Callback Parameter (respectively) that is to be     */
/* called when data is received from the UART. A successful call to */
/* this function will return a non-zero, positive value which       */
/* specifies the HCITransportID that is used with the remaining    */
/* transport functions in this module. This function returns a     */
/* negative return value to signify an error.                       */
int BTPSAPI HCITR_USBOpen(HCI_USBDriverInformation_t *USBDriverInformation,
HCITR_USBDataCallback_t USBDataCallback, unsigned long CallbackParameter);

/* The following function is responsible for closing the the specific*/
/* HCI Transport layer that was opened via a successful call to the */
/* HCITR_USBOpen() function (specified by the first parameter).    */
/* Bluetopia makes a call to this function whenever an either      */
/* Bluetopia is closed, or an error occurs during initialization and */
/* the driver has been opened (and ONLY in this case). Once this   */
/* function completes, the transport layer that was closed will no */
/* longer process received data until the transport layer is      */
/* Re-Opened by calling the HCITR_USBOpen() function.             */
/* * NOTE * This function *MUST* close the specified USB Device.   */
/*          This module will then call the registered COM Data     */
/*          Callback function with zero as the data length and NULL */
/*          as the data pointer. This will signify to the HCI      */
/*          Driver that this module is completely finished with the */
/*          port and information and (more importantly) that NO     */
/*          further data callbacks will be issued. In other words  */
/*          the very last data callback that is issued from this   */
/*          module *MUST* be a data callback specifying zero and NULL*/
/*          for the data length and data buffer (respectively).    */
void BTPSAPI HCITR_USBClose(unsigned int HCITransportID);

```

```

/* The following function is responsible for instructing the          */
/* specified HCI Transport layer (first parameter) that was opened   */
/* via a successful call to the HCITR_USBOpen() function to          */
/* reconfigure itself with the specified information. This          */
/* information is completely opaque to the upper layers and is passed*/
/* through the HCI Driver layer to the transport untouched. It is   */
/* the responsibility of the HCI Transport driver writer to define   */
/* the contents of this member (or completely ignore it).          */
/* * NOTE * This function does not close the HCI Transport specified */
/*          by HCI Transport ID, it merely reconfigures the         */
/*          transport. This means that the HCI Transport specified  */
/*          by HCI Transport ID is still valid until it is closed   */
/*          via the HCI_USBClose() function.                        */
void BTPSAPI HCITR_USBReconfigure(unsigned int HCITransportID, void *TransportDriverContext);

/* The following function is provided to allow a mechanism for      */
/* modules to force the processing of incoming USB Data.          */
/* * NOTE * This function is only applicable in device stacks that  */
/*          are non-threaded. This function has no effect for device*/
/*          stacks that are operating in threaded environments.     */
void BTPSAPI HCITR_USBProcess(unsigned int HCITransportID);

/* The following function is responsible for actually sending data  */
/* through the opened HCI Transport layer. Bluetopia uses this      */
/* function to send formatted HCI packets to the attached Bluetooth */
/* Device. The second parameter to this function specifies the type */
/* of HCI Packet that is to be send through the HCI Transport layer */
/* (Command, ACL, SCO, etc.). The third parameter specifies the     */
/* number of bytes pointed to by the fourth parameter that are to be */
/* sent to the Bluetooth Device. This function returns a zero if the */
/* all data was transfered sucessfully or a negetive value if an     */
/* error occurred. This function MUST NOT return until all of the    */
/* data is sent (or an error condition occurs). Bluetopia WILL NOT  */
/* attempt to call this function repeatedly if data fails to be     */
/* delivered. This function will block until it has either buffered */
/* the specified data or sent all of the specified data to the      */
/* Bluetooth Device.                                                */
/* * NOTE * The type of data (Command, ACL, SCO, etc.) is passed to */
/*          this function because it is assumed that this information*/
/*          is NOT contained in the Data Stream being passed to this */
/*          function.                                                */
int BTPSAPI HCITR_USBWrite(unsigned int HCITransportID, HCI_PacketType_t HCIPacketType, unsigned
int Length, unsigned char *Buffer);

/* The following function changes the settings for the specified USB */
/* HCI Transport regarding HCI SCO Bandwidth Configuration. This    */
/* function is applicable to USB Devices in particular because the  */
/* specification supports dynamic Bandwidth Allocation on the USB   */
/* Interface. This function returns zero upon successful execution   */
/* or a negative error code on failure.                              */
int BTPSAPI HCITR_USBChangeSCOConfiguration(unsigned int HCITransportID, HCI_SCOConfiguration_t
SCOConfiguration);

#endif

#endif

```