Bluetooth™ Protocol Stack

Application Programming Interface Reference Manual

Protocol Version: 4.0

Release: 4.0.1
January 10, 2014

Louisville, KY www.stonestreetone.com

Table of Contents

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/eSCO (Synchronous Connection-Oriented) Link layers. In addition to basic functionality at these layers, Bluetopia 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.

The remainder of this chapter has sections on the scope of this document, other documents applicable to this documents, and a listing of acronyms and abbreviations. Chapter 2 is the API reference which contains a description of all programming interfaces for Bluetopia. Chapter 3 contains a description of the programming interfaces for the profiles contained in the core Bluetooth Protocol Stack library. And, Chapter 4 contains the header file name list for the core Bluetooth Protocol Stack library.

1.1 Scope

This reference manual provides information on the APIs identified in Figure 1-1 below. These APIs are available on the full range of platforms supported by Stonestreet One:

- Windows
- Windows Mobile
- Windows CE
- Linux
- QNX
- Other Embedded OS

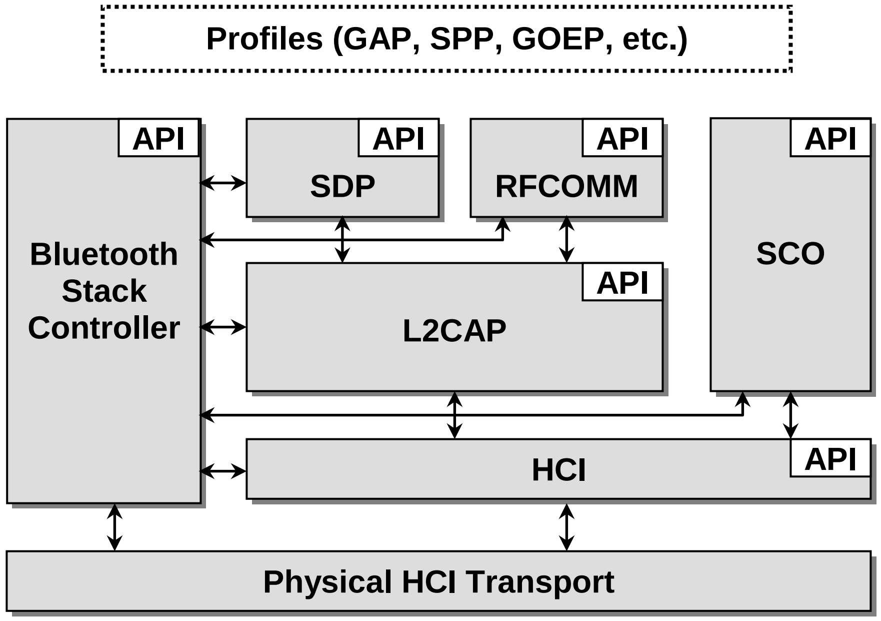

Figure 1-1 The Stonestreet One Bluetooth Protocol Stack

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 1, Core*, version 1.1, February 22, 2001.
2. *Specification of the Bluetooth System, Volume 2, Profiles*, version 1.1, February 22, 2001.
3. *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 1.2, November 5, 2003.
4. *Specification of the Bluetooth System, Volume 2, Core System Package*, version 1.2, November 5, 2003.
5. *Specification of the Bluetooth System, Volume 3, Core System Package*, version 1.2, November 5, 2003.
6. *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 2.0 + EDR, November 4, 2004.
7. *Specification of the Bluetooth System, Volume 2, Core System Package*, version 2.0 + EDR, November 4, 2004.
8. *Specification of the Bluetooth System, Volume 3, Core System Package*, version 2.0 + EDR, November 4, 2004.
9. *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 2.1+EDR, July 26, 2007.
10. *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 2.1+EDR, July 26, 2007.
11. *Specification of the Bluetooth System, Volume 2, Core System Package [Controller Volume]*, version 2.1+EDR, July 26, 2007.
12. *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 2.1+EDR, July 26, 2007.
13. *Specification of the Bluetooth System, Volume 4, Host Controller Interface [Transport Layer]*, version 2.1+EDR, July 26, 2007.
14. *Specification of the Bluetooth System, Bluetooth Core Specification Addendum 1*, June 26, 2008.
15. *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 3.0+HS, April 21, 2009.
16. *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 3.0+HS, April 21, 2009.
17. *Specification of the Bluetooth System, Volume 2, Core System Package [Controller Volume]*, version 3.0+HS, April 21, 2009.

18. *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 3.0+HS, April 21, 2009.
19. *Specification of the Bluetooth System, Volume 4, Host Controller Interface [Transport Layer]*, version 3.0+HS, April 21, 2009.
20. *Specification of the Bluetooth System, Volume 5, Core System Package [AMP Controller Volume]*, version 3.0+HS, April 21, 2009.
21. *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements*, version 4.0, June 30, 2010.
22. *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 4.0, June 30, 2010.
23. *Specification of the Bluetooth System, Volume 2, Core System Package [BR/EDR Controller Volume]*, version 4.0, June 30, 2010.
24. *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 4.0, June 30, 2010.
25. *Specification of the Bluetooth System, Volume 4, Host Controller Interface [Transport Layer]*, version 4.0, June 30, 2010.
26. *Specification of the Bluetooth System, Volume 5, Core System Package [AMP Controller Volume]*, version 4.0, June 30, 2010.
27. *Specification of the Bluetooth System, Volume 6, Core System Package [Low Energy Controller Volume]*, version 4.0, June 30, 2010.
28. *Bluetooth Assigned Numbers,* version 1.1, February 22, 2001.
29. *Digital cellular telecommunications system (Phase 2+); Terminal Equipment to Mobile Station (TE-MS) multiplexer protocol (GSM 07.10),* version 7.1.0, Release 1998; commonly referred to as: ETSI TS 07.10.
30. *Infrared Data Association, IrDA Object Exchange Protocol (IrOBEX) with Published Errata*, Version 1.2, April 1999.

The Bluetooth Protocol Stack API calls were developed to closely follow the above specifications. Note that in previous versions of this document, the Bluetooth section that was directly applicable to the specified functionality was referenced. With the advent of newer versions of the Bluetooth Specification being served by this document, multiple references would need to be given for the specified function. Because of this, the section references have been dropped from this document. The reader should therefore consult the correct Bluetooth Core specification and determine the applicable section manually. In almost all cases, the determination of the section can easily be found by examining the table of contents of the core specification.

Possible error returns are listed for each API function call. These are the *most likely* errors, but in fact programmers should allow for the possibility of any error listed in the BTerrors.h header file to occur as the value of a function return.

1.3 Acronyms and Abbreviations

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

Term	Meaning
ACL link	Asynchronous Connection-less Link – Provides a packet-switched connection. (Master to any slave)
API	Application Programming Interface
BD_ADDR	Bluetooth Device Address
BSC	Bluetooth Stack Controller
BR	Basic Rate
BR/EDR	Basic Rate/Enhanced Data Rate
BT	Bluetooth
CID	Channel Identifier
dB	Decibels
DH	Data-High Rate Data packet type for high rate data
DLCI	Data Link Connection Identifier
DM	Data - Medium Rate Data packet type for medium rate data
DUT	Device Under Test
DV	Data Valid (serial interface signal)
DV	Data Voice data packet type for data and voice
EDR	Enhanced Data Rate
ETSI	European Telecommunications Standards Institute
FC	Flow Control (serial interface signal)
FCC	Federal Communications Commission
GAP	Generic Application Profile
HCI	Host Controller Interface
HS	High Speed
HV	High quality Voice e.g. HV1 packet
IAC	Inquiry Access Code
IC	Incoming Call indicator (serial interface signal)
ID	Identifier

Term	Meaning
L2CA	Logical Link Control and Adaptation Logical Link Control And Management part of the Bluetooth protocol stack
L2CAP	Logical Link Control and Adaptation Protocol
LAP	Lower Address Part (of Bluetooth device address)
LCID	Local Channel Identifier
LE	Low Energy
LM	Link Manager
LMP	Link Manager Protocol For LM peer to peer communication
LSB	Least Significant Bit
MSB	Most Significant Bit
MSC	Message Sequence Chart
MTU	Maximum Transmission Unit
NAP	Non-significant Address Part
OCF	Opcode Command Field
OGF	Opcode Group Field
PDU	Protocol Data Unit (a message)
PIN	Personal Identification Number
PSM	Protocol/Service Multiplexer
QoS	Quality of Service
RFCOMM	Radio Frequency serial COMMunications – Serial cable emulation protocol based on ETSI TS 07.10
RSSI	Received Signal Strength Indication
RTC	Ready to Communicate (serial interface signal)
RTR	Ready to Receive (serial interface signal)
RX	Receiver
SCO link	Synchronous Connection-Oriented Link – Supports time-bounded information like voice.
eSCO link	Extended Synchronous Connection-Oriented Link – Supports time-bounded information like voice. (Version 1.2)
SDP	Service Discovery Protocol
SPP	Serial Port Protocol

Term	Meaning
SSP	Secure Simple Pairing
TBD	To Be Defined
TCS	Telephony Control protocol Specification
TEI	Terminal Endpoint Identifier
TX	Transmit
UAP	Upper Address Part
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
UUID	Universally Unique Identifier

2. Stack Application Programming Interface

The various parts of the Bluetooth Protocol Stack implementation are documented in separate sections in this chapter. The sections and their contents are:

2.1 BSC (Bluetooth Stack Controller) API

2.2 HCI API

2.3 L2CAP API

2.4 SDP API

2.5 RFCOMM API

2.6 SCO API

There is a common set of error codes that applies to all API function calls. Each function will have its allowable/expected set of error codes displayed. The set of all possible errors codes are shoen in the following list. Some error codes may occur only in a specific platform implementation. For example, the BTPS_ERROR_DLL_INITIALIZATION_ERROR is specific to a Windows or Windows CE implementation, and would not occur in an embedded stack implementation. The constant name is designed to clearly indicate the error which occurred:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_STACK_NOT_INITIALIZED
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_STACK_INITIALIZATION_ERROR
BTPS_ERROR_DLL_INITIALIZATION_ERROR
BTPS_ERROR_HCI_INITIALIZATION_ERROR
BTPS_ERROR_GAP_INITIALIZATION_ERROR
BTPS_ERROR_SCO_INITIALIZATION_ERROR
BTPS_ERROR_L2CAP_INITIALIZATION_ERROR
BTPS_ERROR_RFCOMM_INITIALIZATION_ERROR
BTPS_ERROR_SDP_INITIALIZATION_ERROR
BTPS_ERROR_SPP_INITIALIZATION_ERROR
BTPS_ERROR_GOEP_INITIALIZATION_ERROR
BTPS_ERROR_OTP_INITIALIZATION_ERROR
BTPS_ERROR_DEBUG_CALLBACK_ALREADY_INSTALLED
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_DEVICE_RESET_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR
BTPS_ERROR_HCI_TIMEOUT_ERROR
BTPS_ERROR_UNSUPPORTED_HCI_VERSION
BTPS_ERROR_UNKNOWN_SUPPORTED_FEATURES
BTPS_ERROR_UNKNOWN_HCI_BUFFER_SIZE
BTPS_ERROR_UNABLE_TO_REGISTER_EVENT_CALLBACK
BTPS_ERROR_UNABLE_TO_REGISTER_ACL_CALLBACK
BTPS_ERROR_UNABLE_TO_REGISTER_SCO_CALLBACK
BTPS_ERROR_SIGNALLING_MTU_EXCEEDED
BTPS_ERROR_UNABLE_TO_REGISTER_PSM
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_UNABLE_TO_UNREGISTER_PSM

BTPS_ERROR_PSM_NOT_REGISTERED
BTPS_ERROR_ATTEMPTING_CONNECTION_TO_DEVICE
BTPS_ERROR_ACCEPTING_CONNECTION_FROM_DEVICE
BTPS_ERROR_INVALID_FLUSH_TIMEOUT_VALUE
BTPS_ERROR_INVALID_STATE_FOR_CONFIG
BTPS_ERROR_ADDING_CID_INFORMATION
BTPS_ERROR_ADDING_CONNECTION_INFORMATION
BTPS_ERROR_ADDING_IDENTIFIER_INFORMATION
BTPS_ERROR_INVALID_CONNECTION_STATE
BTPS_ERROR_CHANNEL_NOT_IN_OPEN_STATE
BTPS_ERROR_INVALID_CID
BTPS_ERROR_WRITING_DATA_TO_DEVICE
BTPS_ERROR_MEMORY_ALLOCATION_ERROR
BTPS_ERROR_NEGOTIATED_MTU_EXCEEDED
BTPS_ERROR_CONECTIONLESS_MTU_EXCEEDED
BTPS_ERROR_CID_NOT_GROUP_CID
BTPS_ERROR_GROUP_MEMBER_ALREADY_EXISTS
BTPS_ERROR_GROUP_MEMBER_NOT_FOUND
BTPS_ERROR_CONNECTION_TO_DEVICE_LOST
BTPS_ERROR_INVALID_CID_TYPE
BTPS_ERROR_SDP_DATA_ELEMENT_EXPECTED
BTPS_ERROR_SDP_INVALID_DATA_ELEMENT_LENGTH
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_SDP_INVALID_DATA_ELEMENT
BTPS_ERROR_ADDING_SERVICE_ATTRIBUTE
BTPS_ERROR_DELETING_SERVICE_RECORD
BTPS_ERROR_EXPECTED_UUID_ENTRY
BTPS_ERROR_SDP_INVALID_DATA_TYPE
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_INVALID_MODE
BTPS_ERROR_ADDING_CALLBACK_INFORMATION
BTPS_ERROR_DELETING_CALLBACK_INFORMATION
BTPS_ERROR_NO_CALLBACK_REGISTERED
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_MAX_SCO_CONNECTIONS
BTPS_ERROR_INTERNAL_ERROR
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_ADDING_SERVER_INFORMATION
BTPS_ERROR_RFCOMM_REMOVING_SERVER_INFORMATION
BTPS_ERROR_RFCOMM_UNABLE_TO_ADD_CONNECTION_INFORMATION
BTPS_ERROR_RFCOMM_UNABLE_TO_ADD_CHANNEL_INFORMATION
BTPS_ERROR_RFCOMM_UNABLE_TO_CONNECT_TO_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI
BTPS_ERROR_RFCOMM_DISC_ALREADY_PENDING
BTPS_ERROR_RFCOMM_TEI_IS_DISCONNECTING
BTPS_ERROR_RFCOMM_CONTROL_MESSAGE_CURRENTLY_PENDING

BTPS_ERROR_RFCOMM_FLOW_IS_DISABLED
BTPS_ERROR_RFCOMM_INVALID_MAX_FRAME_SIZE
BTPS_ERROR_RFCOMM_COMMAND_NOT_ALLOWED
BTPS_ERROR_RFCOMM_ADDING_MESSAGE_INFORMATION
BTPS_ERROR_RFCOMM_INVALID_FLOW_STATE
BTPS_ERROR_RFCOMM_MAX_FRAME_SIZE_EXCEEDED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED
BTPS_ERROR_SPP_BUFFER_FULL
BTPS_ERROR_OUTSTANDING_TRANSACTION
BTPS_ERROR_TIMER_VALUE_OUT_OF_RANGE
BTPS_ERROR_GOEP_NOT_INITIALIZED
BTPS_ERROR_GOEP_COMMAND_NOT_ALLOWED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_ALREADY_CONNECTED
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED
BTPS_ERROR_DEVICE_NOT_CONNECTED
BTPS_ERROR_ACTION_NOT_ALLOWED
BTPS_ERROR_SPP_BUFFER_EMPTY
BTPS_ERROR_UNABLE_TO_ENABLE_HC_TO_H_FLOW_CONTROL
BTPS_ERROR_VS_HCI_ERROR
BTPS_ERROR_ALREADY_OUTSTANDING
BTPS_ERROR_FEATURE_NOT_AVAILABLE
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
BTPS_ERROR_SCAN_ACTIVE
BTPS_ERROR_SLAVE_CONNECTION_PRESENT
BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
BTPS_ERROR_DEVICE_IS_SLAVE
BTPS_ERROR_INVALID_CONNECTION_HANDLE
BTPS_ERROR_READ_REMOTE_FEATURES_OUTSTANDING
BTPS_ERROR_CREATE_CONNECTION_OUTSTANDING
BTPS_ERROR_INVALID_CONNECTION_PARAMETERS
BTPS_ERROR_WHITE_LIST_SIZE_EXCEEDED
BTPS_ERROR_WHITE_LIST_IN_USE
BTPS_ERROR_INVALID_RANDOM_ADDRESS
BTPS_ERROR_RANDOM_ADDRESS_IN_USE
BTPS_ERROR_PAIRING_ACTIVE
BTPS_ERROR_PAIRING_NOT_ACTIVE
BTPS_ERROR_INVALID_STATE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

2.1 BSC (Bluetooth Stack Controller) API

The functions in this section are not defined in the Bluetooth specification, but have been added to provide some stack management and debugging aids. They are divided up into subsections on Callbacks and Commands. The actual prototypes and constants outlined in this section can be found in the **BSCAPI.H** header file in the Bluetopia distribution.

2.1.1 BSC Callbacks

BSC_Timer_Callback_t

The prototype function represents the Prototype Function for a Bluetooth Timer Callback. This function will be called whenever a timer that was registered with the BSC_StartTimer function. This function is guaranteed NOT to be invoked more than once simultaneously for the specified timer (i.e. this function DOES NOT have be reentrant). It needs to be noted however, that if the same Callback is installed more that once (for multiple timers AND they expire simultaneously) then the callbacks will be called serially. Because of this, the processing in this function should be as efficient as possible. It should also 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 another Timer Callback will not be processed while this function call is outstanding).

Prototype:

void (BTPSAPI ***BSC_Timer_Callback_t**)(unsigned int BluetoothStackID,
unsigned int TimerID, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Which device stack this packet is from.
TimerID	Timer Identifier of the timer that has expired. This value will be the same as the value returned from a successful call to the BSC_StartTimer function.
CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the timer callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_Debug_Callback_t

The following prototype function is for a Bluetooth Stack Debug Data Callback. This function will be called whenever a complete HCI Packet has been sent or received by the Bluetooth device that was opened with the Bluetooth Protocol Stack. This function passes to the caller the HCI Packet that was received and the Debug Callback Parameter that was specified when this Callback was installed. This callback is best used to simply put data into a debug viewer. One *must* not make other Bluetooth Stack calls from within this callback or the whole system may become unstable or lock-up.

Prototype:

void (BTPSAPI ***BSC_Debug_Callback_t**)(unsigned int BluetoothStackID,
Boolean_t PacketSent, HCI_Packet_t *HCIPacket, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Which device stack this packet is from.
Packetsent	TRUE if HCI packet was sent, FALSE if it was received.
HCIPacket	Pointer to packet contents
CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_Cleanup_Callback_t

The following prototype function is for a Bluetooth Stack Cleanup Function Callback. The function is called from within the context of the BSC_Shutdown function. This function is guaranteed NOT to be called more than once simultaneously (i.e. this function DOES NOT have to be reentrant). This function will be passed the Bluetooth Stack if for the device which has the function registered, and the Callback Parameter specified when the function was registered. If the same function is registered more than once, it will be called once for each time it was registered.

Prototype:

void (BTPSAPI ***BSC_Cleanup_Callback_t**)(unsigned int BluetoothStackID,
unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Which device stack this packet is from.
CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_Event_Callback_t

The following prototype function is for a Bluetooth Stack BSC Event Callback. This function is used when an upper layer module requires a specific function provided by another layer. This callback is registered with BSC_RegisterEventCallback. This function is guaranteed NOT to be called more than once simultaneously (i.e. this function DOES NOT have to be reentrant). This function will be passed the Bluetooth Stack ID for the Bluetooth Stack which has the function registered, and the Callback Parameter specified when the function was registered.

Prototype:

void (BTPSAPI ***BSC_Event_Callback_t**)(unsigned int BluetoothStackID, BSC_Event_Data_t *BSC_Event_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Bluetooth Stack ID of the Bluetooth Stack that generated the event.
BSC_Event_Data	Pointer to the BSC Event Data of the specified event.
CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_AsynchronousCallbackFunction_t

The following prototype function is for a Bluetooth Stack Asynchronous Function Callback. This function will be called whenever an asynchronous callback is registered with the BSC_ScheduleAsynchronousCallback. This function is guaranteed NOT to be called more than once simultaneously (i.e. this function DOES NOT have to be reentrant). This function will be passed the Bluetooth Stack ID for the Bluetooth Stack which has the function registered, and the Callback Parameter specified when the function was registered. If the same function is registered more than once, it will be called once for each time it was registered.

Prototype:

void (BTPSAPI ***BSC_AsynchronousCallbackFunction_t**)(unsigned int BluetoothStackID, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Bluetooth Stack ID of the Bluetooth Stack that issued the call to BSC_ScheduleAsynchronousCallback.

CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.1.2 BSC Commands

The commands in this section are summarized in the table below.

Function	Description
BSC_Initialize	Initialize a Bluetooth Protocol Stack for a device.
BSC_Shutdown	Shutdown a Bluetooth Protocol Stack for a device.
BSC_RegisterDebugCallback	Register a function to be called each time an HCI packet is sent or received.
BSC_UnRegisterDebugCallback	Deregister a previously registered debug function.
BSC_RegisterEventCallback	Allows caller to register an event callback that is called when an upper layer needs a specific function in another layer.
BSC_UnRegisterEventCallback	Removes a previously installed event callback.
BSC_LockBluetoothStack	Mutual exclusion function used to protect stack resources as appropriate. Must be paired with the following unlock mutual exclusion function call.
BSC_UnLockBluetoothStack	Mutual exclusion function used to protect stack resources as appropriate. Must be paired with the previous lock mutual exclusion function call.
BSC_StartTime	Used to implement timing mechanism to support operation timeout requirements.
BSC_StopTime	Used to implement timing mechanism to support operation timeout requirements.
BSC_AuthenticateDevice	Allows a mechanism for any layer to request that a connected device be authenticated.
BSC_EnableFeature	Allows mechanism for any layer to enable a supported feature (stack must be configured to support this).
BSC_DisableFeature	Allows mechanism for any layer to disable a currently enabled feature (stack must be configured to support

Function	Description
	this).
BSC_QueryActiveFeatures	Allows mechanism for any layer to query the currently configured (and active) features.
BSC_QueryStackIdle	Allows a mechanism to determine if the stack is currently processing any packets and/or timers.
BSC_ScheduleAsynchronousCallback	Allows a mechanism to schedule an asynchronous callback.
BSC_AcquireListLock	Acquire internal list lock for application list locking
BSC_ReleaseListLock	Release previously aquired list lock
BSC_AddGenericListEntry_Actual	Add an opaque list entry to a specified list.
BSC_AddGenericListEntry	Allocate new opaque list entry and add to a specified list.
BSC_SearchGenericListEntry	Search a specified list for a specific opaque list entry.
BSC_GetNextGenericListEntry	Search for the next opaque list entry in the specified list (given a specific opaque list entry).
BSC_DeleteGenericListEntry	Delete an opaque list entry from a specified list.
BSC_FreeGenericListEntryMemory	Delete memory that was allocated for an opaque list entry.
BSC_FreeGenericListEntryList	Delete (and free the memory of) each opaque list entry that is contained in the specified list.

BSC_Initialize

This function is responsible for Initializing a Bluetooth Protocol Stack for the specified Bluetooth device (using the specified HCI Transport). This command **must** be called (and complete successfully) before any other stack command can be called.

Prototype:

int BTPSAPI **BSC_Initialize**(HCI_DriverInformation_t *HCI_DriverInformation,
unsigned long Flags)

Parameters:

HCI_DriverInformation[1] Pointer to the driver information structure. This must be a valid transport supported by the stack. This structure is declared as follows:

```
typedef struct
{
   HCI_DriverType_t          DriverType;
      (One of the following values:  hdtCOMM, hdtUSB, hdt)
   union
   {
      HCI_COMMDriverInformation_t  COMMDriverInformation;
      HCI_USBDriverInformation_t   USBDriverInformation;
   } DriverInformation;
} HCI_DriverInformation_t;
```

where the Comm Driver Information structure is defined as follows:

```
typedef struct
{
          unsigned int     DriverInformationSize;
                           (Size (in Bytes) of this structure)
          unsigned int     COMPortNumber;
                           (Physical COM Port Number)
          unsigned int     BaudRate;
                           (Baud Rate Setting)
          HCI_COMM_Protocol_t     Protocol to use;
                           (One of the following values:
                             cpUART, cpUART_RTS_CTS,
                             cpBCSP, cpBCSP_Muzzled,
                             cpH4DS, cpH4DS_RTS_CTS,
                             cpHCILL, cpHCILL_RTS_CTS)
          unsigned int     InitializationDelay;
                           (Delay (in Milliseconds) to wait for
                           Bluetooth/Transport Initialization)
          char            *COMDeviceName;
                           (Physical Device Name to use to
                           override the device to open.  If
                           COMPortNumber is specified to be
                           the equivalent of negative 1 (-1), then
                           this value is taken as an absolute
                           name and the COM Port Number is
                           NOT appended to this value If this
                           value is NULL then the default
                           (compiled) COM Device Name is
                           used (and the COM Port Number is
                           appended to the default)
} HCI_COMMDriverInformation_t;
```

and the USB driver Information structure is defined as follows:

```
typedef struct
{
          unsigned int        DriverInformationSize;
                              (Size (in Bytes) of this structure)
          HCI_USB_Driver_t    DriverType;
                              (HCI USB driver type that is to
                              be used to communicate with the
                              USB device.  Once of the
                              following values:
                              dtStonestreetOne, dtGarmin)
          unsigned int        InitializationDelay;
```

(*Delay (in Milliseconds) to wait for Bluetooth/Transport Initialization*)

} **HCI_USBDriverInformation_t**;

Utility Macro's are defined to aid the programmer initializing the above HCI Driver Information. These utility Macro's are defined as:

HCI_DRIVER_SET_COMM_INFORMATION
HCI_DRIVER_SET_EXTENDED_COMM_INFORMATION_DELAY
HCI_DRIVER_SET_EXTENDED_COMM_INFORMATION_DEVICE_NAME
HCI_DRIVER_SET_USB_INFORMATION
HCI_DRIVER_SET_EXTENDED_USB_INFORMATION
HCI_DRIVER_SET_EXTENDED_USB_INFORMATION_DELAY

Consult the Header files for a description of the parameters that are accepted by each of the above listed Macro's.

Flags	Should be zero (0) to load the standard/complete Bluetooth stack. Logical ORing of the following bitmask constants can be used to modify the standard/complete stack: BSC_INITIALIZE_FLAG_NO_L2CAP BSC_INITIALIZE_FLAG_NO_SCO BSC_INITIALIZE_FLAG_NO_SDP BSC_INITIALIZE_FLAG_NO_RFCOMM BSC_INITIALIZE_FLAG_NO_GAP BSC_INITIALIZE_FLAG_NO_SPP

Return: one of the following depending on whether the value is positive or negative:

BluetoothStackID[2]	[positive] A unique identifier that is used in other stack calls and callbacks. This ID remains valid for the specified Bluetooth device until the Bluetooth stack is closed via a call to the BSC_Shutdown function.
Error Code	[negative value] Possible values are: BTPS_ERROR_RFCOMM_INITIALIZATION_ERROR BTPS_ERROR_SDP_INITIALIZATION_ERROR BTPS_ERROR_DLL_INITIALIZATION_ERROR BTPS_ERROR_INVALID_PARAMETER BTPS_ERROR_DLL_INITIALIZATION_ERROR BTPS_ERROR_HCI_DRIVER_ERROR BTPS_ERROR_INSUFFICIENT_RESOURCES BTPS_ERROR_HCI_INITIALIZATION_ERROR BTPS_ERROR_GAP_INITIALIZATION_ERROR BTPS_ERROR_SCO_INITIALIZATION_ERROR BTPS_ERROR_L2CAP_INITIALIZATION_ERROR BTPS_ERROR_SPP_INITIALIZATION_ERROR

Notes:

1. The HCI_DriverInformation parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2. The return parameter in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia, will not indicate a BluetoothStackID. Instead, if a positive value is returned, this is an indication that the function was successful. The negative return value is valid across all versions of Bluetopia.

BSC_Shutdown

This function closes the Bluetooth Protocol Stack that was opened for the Bluetooth device specified via a successful call to the BSC_Initialize function (i.e., a positive return value from that call). Once this function completes, the Bluetooth device that was opened (and the Bluetooth Protocol Stack that is associated with the Device) cannot be accessed again until the Device (and a corresponding Bluetooth Protocol Stack) is re-opened by calling the BSC_Initialize function again.

Prototype:

void BTPSAPI **BSC_Shutdown**(unsigned int BluetoothStackID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_RegisterDebugCallback

This is a debugging function that allows the caller to register a Debug Callback that will be called each time an HCI Packet is sent or received. Note, because this function will be called every time a packet is sent or received, this function should only be used when debugging is required because of the performance penalty that is present when using this mechanism. This callback registration can only be removed via a call to BSC_UnRegisterDebugCallback.

Prototype:

int BTPSAPI **BSC_RegisterDebugCallback**(unsigned int BluetoothStackID,
    BSC_Debug_Callback_t BSC_DebugCallback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BSC_DebugCallback	Pointer to a user-supplied callback function which is define as above in the BSC callback section.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Zero if successful.

Negative if an Error occurred. Possible values are:

BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_DEBUG_CALLBACK_ALREADY_INSTALLED

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_UnRegisterDebugCallback

This function removes a previously installed Debug Callback for the specified Bluetooth Protocol Stack. After this function has completed, the caller will no longer be notified via the debug callback function when a debug event occurs.

Prototype

void BTPSAPI **BSC_UnRegisterDebugCallback**(unsigned int BluetoothStackID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_RegisterEventCallback

The following function is provided to allows the caller to register an Event Callback for a specified Bluetooth Protocol Stack that will be called when an upper layer requires a specific function that is provided by another layer. Once an Event Callback has been installed in can only be removed by a call to BSC_UnRegisterEventCallback.

Prototype:

int BTPSAPI **BSC_RegisterEventCallback** (unsigned int BluetoothStackID, BSC_Event_Callback_t BSC_EventCallback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BSC_EventCallback	Pointer to function that will be called when a BSC Event is dispatched.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user callback function.

Return:

Non-zero positive value if successful.

Negative if an Error occurred. Possible values are:

BTPS_ERROR_UNABLE_TO_REGISTER_EVENT_CALLBACK
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_UnRegisterEventCallback

This function removes a previously installed Event Callback for the specified Bluetooth Protocol Stack. Once this call is complete the caller will no longer be notified via the Event Callback Function when a BSC event occurs.

Prototype:

void BTPSAPI **BSC_UnRegisterEventCallback** (unsigned int BluetoothStackID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Return:

None

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_LockBluetoothStack

This function exists to aid Profile programmers by providing a Mutex/Semaphore Lock that is completely thread safe. This lock is the same lock that the Bluetooth Protocol Stack uses to guard against re-entrancy problems. Using this mechanism allows atomic operations to be performed (on the specified Bluetooth Protocol Stack) and guarantees that no other thread can cause an operation to be performed (on the specified Bluetooth Protocol Stack ONLY). This is a very low-level primitive and it's use is really only applicable to Profiles and/or Stack extensions. Applications should never need to call this function (or it's converse unlock function). Please see the documentation contained in the header file (**BSCAPI.h**) for more information on this function. It is very important to note that if this function is called, the **BSC_UnLockBluetoothStack** is required to be called for every successful call to this function. Failure to comply with the preceding statement can and will lead to erratic behavior. This function can be called more than once (in the same thread), however the programmer **MUST** call the unlock function the same number of times that this function is successfully called.

Prototype

int BTPSAPI **BSC_LockBluetoothStack**(unsigned int BluetoothStackID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Return:

Zero if successful.

Negative if an Error occurred. Possible values are:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_UnLockBluetoothStack

This function is provided to allow the programmer a mechanism to release a previous lock that was successfully acquired with the **BSC_LockBluetoothStack** function. The locking/unlocking mechanism exists to aid Profile programmers by providing a Mutex/Semaphore Lock that is completely thread safe. This lock is the same lock that the Bluetooth Protocol Stack uses to guard against re-entrancy problems. Using this mechanism allows atomic operations to be performed (on the specified Bluetooth Protocol Stack) and guarantees that no other thread can cause an operation to be performed (on the specified Bluetooth Protocol Stack ONLY). This is a very low-level primitive and it's use is really only applicable to Profiles and/or Stack extensions. Applications should never need to call this function (or it's converse unlock function). Please see the documentation contained in the header file (**BSCAPI.h**) for more information on this function.

Prototype

void BTPSAPI **BSC_UnLockBluetoothStack**(unsigned int BluetoothStackID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_StartTimer

The following function is a utility function that exists to allow the programmer a mechanism for installing an asynchronous Bluetooth timer (of the specified timeout value). The registered timer callback function will be called when the timeout period expires (in milliseconds), passing the user supplied callback parameter to the caller. Once a callback is installed, it will be removed from the system when it expires, the stack is closed, or it is removed by the programmer via the BSC_StopTimer. Timers should be used sparingly because there are only a finite number of timers present in the system. It should be noted that all installed Timers are one-shot timers and not periodic (i.e. they will only expire once). If a periodic timer is required then the Timer must be re-registered.

Prototype:

int BTPSAPI **BSC_StartTimer**(unsigned int BluetoothStackID, unsigned int Timeout, BSC_Timer_Callback_t BSC_TimerCallback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Timeout	Timeout value (in milliseconds)
BSC_TimerCallback	Pointer to a user-supplied callback function which is defined as above in the BSC callback section.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function when the timer expires.

Return:

Positive non-zero value if successful. This is the TimerID which is used to identify the timer. This value can be passed to the BSC_StopTimer function to cancel the timer.

Negative if an Error occurred. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_StopTimer

This function removes a previously installed Bluetooth Timer that was registered with the BSC_StartTimer function. If this function returns successfully then the specified timer (via TimerID) will no longer be present in the system, and hence not expire.

Prototype

void BTPSAPI **BSC_StopTimer**(unsigned int BluetoothStackID, unsigned int TimerID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TimerID	Timer indentifier of the timer that is to be stopped. This value must be a successful return value from the BSC_StartTimer function.

Return:

Zero value if successful.

Negative if an Error occurred. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_AuthenticateDevice

The following function is provided to allow a mechanism for any layer to request that a connected device be authenticated. This function accepts as input the Bluetooth Stack ID of the Bluetooth Stack that the Device is associated with. The second parameter is the Bluetooth address of the connected device that requires Authentication. The third parameter is a pointer to a Result variable that indicates the state of the request. This function returns zero if successful, or a negative return error code if the Authentication process was not started. This function is currently utilized to perform Level 4 Security with L2CAP and Secure Simple Pairing. Currently there is no need for applications to make use of this function.

Prototype

int BTPSAPI **BSC_AuthenticateDevice**(unsigned int BluetoothStackID,
  BD_ADDR_t BD_ADDR, Byte_t *Result);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address of device that is to be authenticated
Result	Variable that is to receive the status result from the request. This value must be one of: BSC_AUTHENTICATION_REQUEST_RESULT_SUCCESS BSC_AUTHENTICATION_REQUEST_RESULT_IN_PROGRESS BSC_AUTHENTICATION_REQUEST_RESULT_REFUSED BSC_AUTHENTICATION_REQUEST_RESULT_FAILURE

Return:

Zero value if successful.

Negative if failure.

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_EnableFeature

The following function is provided to allow a mechanism for any layer to enable a preconfigured/supported feature. This is useful when Bluetooth chipset configuration is required to support a feature (e.g. Bluetooth Low Energy or Wide Band Speech) and/or only a single feature can be active at any given time.

Note:

This functionality is not normally supported by default (i.e. a custom stack configuration/build is required to enable this functionality).

Prototype

int BTPSAPI **BSC_EnableFeature**(unsigned int BluetoothStackID, unsigned long Feature)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Feature	Feature to enable. This value must be one of: BSC_FEATURE_BLUETOOTH_LOW_ENERGY BSC_FEATURE_ANT_PLUS BSC_FEATURE_WIDE_BAND_SPEECH

Return:

Zero value if successful.

Negative if an Error occurred. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_FEATURE_NOT_AVAILABLE
BTPS_ERROR_INVALID_STATE
BTPS_ERROR_INVALID_MODE

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_DisableFeature

The following function is provided to allow a mechanism for any layer to disable a preconfigured/supported (and currently active) feature. This is useful when Bluetooth chipset configuration is required to support a feature (e.g. Bluetooth Low Energy or Wide Band Speech) and/or only a single feature can be active at any given time. This is also useful to turn off specific features to save power (if the chipset supports this functionality).

Note:

This functionality is not normally supported by default (i.e. a custom stack configuration/build is required to enable this functionality).

Prototype

int BTPSAPI **BSC_DisableFeature**(unsigned int BluetoothStackID, unsigned long Feature)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Feature	Feature to disable. This value must be one of: BSC_FEATURE_BLUETOOTH_LOW_ENERGY BSC_FEATURE_ANT_PLUS BSC_FEATURE_WIDE_BAND_SPEECH

Return:

Zero value if successful.

Negative if an Error occurred. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_FEATURE_NOT_AVAILABLE
BTPS_ERROR_INVALID_STATE
BTPS_ERROR_INVALID_MODE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_
ACTIVE

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_QueryActiveFeatures

The following function is provided to allow a mechanism for any layer to determine the currently active feature. This is useful when Bluetooth chipset configuration is required to support a feature (e.g. Bluetooth Low Energy or Wide Band Speech) and/or only a single feature can be active at any given time. This function allows the ability to determine if a feature is currently configured so that the appropriate action can be taken (i.e. do not use the feature and/or attempt to enable the feature so it can be used).

Note:

This functionality is not normally supported by default (i.e. a custom stack configuration/build is required to enable this functionality).

Prototype

int BTPSAPI **BSC_QueryActiveFeatures**(unsigned int BluetoothStackID,
unsigned long *Feature)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Feature	Pointer to a buffer that will contain the features that are currently enabled/active. This value will be one of: BSC_FEATURE_BLUETOOTH_LOW_ENERGY BSC_FEATURE_ANT_PLUS BSC_FEATURE_WIDE_BAND_SPEECH

Return:

Zero value if successful.

Negative if an Error occurred. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_FEATURE_NOT_AVAILABLE

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_QueryStackIdle

The following function is provided to allow a mechanism for any layer to determine if the specified protocol stack is "idle". "Idle", in this case, means there is no pending processing (e.g. no timers, packets queued for sending and/or receiving, etc). This is useful in single-threaded environments and can be used to aid in power saving schemas.

Note:

This function is only applicable in single-threaded environments. This function always returns that the stack is Idle regardless if there is on-going processing (due to the multi-threaded nature, it is not possible to ascertain this information).

Prototype

Boolean_t BTPSAPI **BSC_QueryStackIdle**(unsigned int BluetoothStackID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Return:

BOOLEAN value, TRUE if the stack is currently "idle" (i.e. no processing), or FALSE if the stack is not currently "idle".

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_ScheduleAsynchronousCallback

The following function is provided to allow a mechanism of scheduling a one-shot asynchronous callback that will be called once for each function invocation.

Prototype:

int BTPSAPI **BSC_ScheduleAsynchronousCallback**(unsigned int BluetoothStackID,
    unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function when it is called.

Return:

Non zero if successful.

Zero if an error occurred.

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

BSC_AcquireListLock

The following function is provided to allow a mechanism to acquire a global lock that can be used to search lists that are maintained by modules (for resource tracking). This Lock CANNOT be held while holding or acquiring any other lock. This functionality is provided to allow a mechanism on smaller (embedded) systems so that individual modules (such as the HCI Drivers and profiles) to do not have to waste resources for locks to protect their internal lists. The caller *MUST* call the **BSC_ReleaseListLock**() function to release the lock when finished.

<u>Note:</u>

This function is only applicable in multi-threaded environments. This function always returns that the stack TRUE in single threaded environments.

Prototype

Boolean_t BTPSAPI **BSC_AcquireListLock**(void);

Parameters:

Return:

BOOLEAN value, TRUE if the list lock was obtained successfully, FALSE if the lock was unable to be obtained (or an error occurred).

Notes:

BSC_ReleaseListLock

The following function is provided to allow a mechansim for the caller to release the acquired list lock (previously acquired via a successful call to the **BSC_AcquireListLock**() function).

Prototype

void BTPSAPI **BSC_ReleaseListLock**(void);

Parameters:

Return:

Notes:

BSC_AddGenericListEntry_Actual

The following function is a utility function that adds the actual specified opaque list entry to the specified opaque list entry list.

Notes:

Opaque lists and list entries are a schema that allows any data structure to be added to a list (of like structures). This schema is possible because all the routines that operate on the list (including this one) are told the necessary structure offsets (and sizes) of each entry. The utility functions also define the concept of a "key" that can be used for searching through the list.

Valid values must be specified for the following parameters (or the function will fail):

- ListHead - parameter cannot be NULL, but the value that it points to can be NULL
- ListEntryToAdd - parameter cannot be NULL, and it must point to the List Entry Data that is to be added (of size ListEntrySize)

If the GenericListEntryKey value is anything other than ekNone, then this function does not insert duplicate entries into the list. An item is considered a duplicate if the Key value of the entry being added matches any Key value of an entry already in the list. When this parameter is ANYTHING OTHER THAN ekNone, then the following parameters must be specified:

- ListEntryKeyOffset - specifies the byte offset of the Generic List Entry Key Element (in each individual List Entry)

In all cases, the ListEntryNextPointerOffset parameter MUST specify the byte offset of the element that represents a pointer to the next element that is present in the list (for each individual List Entry)

Prototype

Boolean_t BTPSAPI **BSC_AddGenericListEntry_Actual(**
BSC_Generic_List_Entry_Key_t GenericListEntryKey, unsigned int ListEntryKeyOffset, unsigned ListEntryNextPointerOffset, void **ListHead, void *ListEntryToAdd);

Parameters:

GenericListEntryKey	Key value type that is used to search for duplicates (see notes above). This value must be one of: ekNone ekBoolean_t ekByte_t ekWord_t ekDWord_t ekBD_ADDR_t ekEntryPointer ekUnsignedInteger

ListEntryKeyOffset	Offset (specified in bytes) from the beginning of the list entry where the list entry key is located
ListEntryNextPointerOffset	Offset (specified in bytes) from the beginning of the list entry where the list entry next pointer is located
ListHead	Pointer to the location that holds a pointer to the first entry in the list (the value at this location can be NULL for an empty list, but this parameter cannot be NULL)
ListEntryToAdd	Pointer to the actual list entry that is to be added to the specified list (note that the offsets specified in the prior parameters are applied to this address to resolve the correct locations)

Return:

BOOLEAN value, TRUE if the specified list entry was added to the specified list, or FALSE if the entry was unable to be added (either invalid parameter or there was a duplicate entry in the list).

BSC_AddGenericListEntry

The following function is a utility function that adds an opaque list entry (with the specified opaque list entry information) to the specified opaque list entry list. This function does NOT add the specified entry directly to the list. This function allocates an entry (of the correct sizes) and copies the data from the specified entry into this newly allocated entry. This newly allocated entry is then added to the specified list.

Notes:

Opaque lists and list entries are a schema that allows any data structure to be added to a list (of like structures). This schema is possible because all the routines that operate on the list (including this one) are told the necessary structure offsets (and sizes) of each entry. The utility functions also define the concept of a “key” that can be used for searching through the list.

Valid values must be specified for the following parameters (or the function will fail):

- ListEntrySizeToAllocate - cannot be zero and MUST be greater than or equal to the ListEntrySize parameter
- ListEntrySize - cannot be zero and MUST be less than or equal to the ListEntrySizeToAllocate parameter
- ListHead - parameter cannot be NULL, but the value that it points to can be NULL
- ListEntryToAdd - parameter cannot be NULL, and it must point to the List Entry Data that is to be added (of size ListEntrySize)

If the GenericListEntryKey value is anything other than ekNone, then this function does not insert duplicate entries into the list. An item is considered a duplicate if the Key value of the entry being added matches any Key value of an entry already in the list. When this parameter is ANYTHING OTHER THAN ekNone, then the following parameters must be specified:

- ListEntryKeyOffset - specifies the byte offset of the Generic List Entry Key Element (in each individual List Entry)

In all cases, the ListEntryNextPointerOffset parameter MUST specify the byte offset of the element that represents a pointer to the next element that is present in the list (for each individual List Entry)

Prototype

Boolean_t BTPSAPI **BSC_AddGenericListEntry(**unsigned int ListEntrySizeToAllocate, BSC_Generic_List_Entry_Key_t GenericListEntryKey, unsigned int ListEntryKeyOffset, unsigned int ListEntrySize, unsigned ListEntryNextPointerOffset, void **ListHead, void *ListEntryToAdd);

Parameters:

ListEntrySizeToAllocate	Entire size (in bytes) of the entry to allocate. Note that this is note the size of the list entry itself. This value must be AT-LEAST the size of ListEntrySize, but can be specified larger. This allows the ability to allocate extra space immediately after the list entry.
GenericListEntryKey	Key value type that is used to search for duplicates (see notes above). This value must be one of: ekNone ekBoolean_t ekByte_t ekWord_t ekDWord_t ekBD_ADDR_t ekEntryPointer ekUnsignedInteger
ListEntryKeyOffset	Offset (specified in bytes) from the beginning of the list entry where the list entry key is located
ListEntrySize	Specifies the size (in bytes) of the list entry size. This size is used to copy the specified list entry information (final parameter) to the newly allocated list entry.
ListEntryNextPointerOffset	Offset (specified in bytes) from the beginning of the list entry where the list entry next pointer is located
ListHead	Pointer to the location that holds a pointer to the first entry in the list (the value at this location can be NULL for an empty list, but this parameter cannot be NULL)
ListEntryToAdd	Pointer to the actual list entry that is to be added to the specified list (note that the offsets specified in the prior parameters are applied to this address to resolve the correct locations)

Return:

BOOLEAN value, TRUE if a new list entry was added to the specified list, or FALSE if the entry was unable to be added (either invalid parameter or there was a duplicate entry in the list).

BSC_SearchGenericListEntry

The following function is a utility function that allows the ability to search for a specific opaque list entry (located in the specified opaque list entry list).

Notes:

Opaque lists and list entries are a schema that allows any data structure to be added to a list (of like structures). This schema is possible because all the routines that operate on the list (including this one) are told the necessary structure offsets (and sizes) of each entry. The utility functions also define the concept of a "key" that can be used for searching through the list.

Prototype

void *BTPSAPI **BSC_SearchGenericListEntry(**
    BSC_Generic_List_Entry_Key_t GenericListEntryKey,
    void *GenericListEntryKeyValue, unsigned int ListEntryKeyOffset,
    unsigned ListEntryNextPointerOffset, void **ListHead);

Parameters:

GenericListEntryKey	Key value type that is used to search the entries. This value must be one of: ekBoolean_t ekByte_t ekWord_t ekDWord_t ekBD_ADDR_t ekEntryPointer ekUnsignedInteger
ListEntryKeyValue	Pointer to the key value that is to matched for the search. The actual data type that this value points to depends upon the value of the previous parameter. Note that this value CANNOT be NULL.
ListEntryKeyOffset	Offset (specified in bytes) from the beginning of each list entry where the list entry key is located
ListEntryNextPointerOffset	Offset (specified in bytes) from the beginning of each list entry where the list entry next pointer is located
ListHead	Pointer to the location that holds a pointer to the first entry in the list (the value at this location can be NULL for an empty list, but this parameter cannot be NULL)

Return:

Non NULL value indicating success (a pointer to the entry that was found).

NULL value indicating that an entry was not located in the specified list.

BSC_GetNextGenericListEntry

The following function is a utility function that allows the ability to find the next opaque list entry give the specified opaque list entry list.

Notes:

Opaque lists and list entries are a schema that allows any data structure to be added to a list (of like structures). This schema is possible because all the routines that operate on the list (including this one) are told the necessary structure offsets (and sizes) of each entry. The utility functions also define the concept of a "key" that can be used for searching through the list.

Prototype

```
void *BTPSAPI BSC_GetNextGenericListEntry(
    BSC_Generic_List_Entry_Key_t GenericListEntryKey,
    void *GenericListEntryKeyValue, unsigned int ListEntryKeyOffset,
    unsigned ListEntryNextPointerOffset, void **ListHead);
```

Parameters:

GenericListEntryKey	Key value type that is used to search the entries. This value must be one of: ekEntryPointer
ListEntryKeyValue	Pointer to the key value that is to matched for the search. The actual data type that this value points to depends upon the value of the previous parameter. Note that this value CANNOT be NULL.
ListEntryNextPointerOffset	Offset (specified in bytes) from the beginning of each list entry where the list entry next pointer is located
ListHead	Pointer to the location that holds a pointer to the first entry in the list (the value at this location can be NULL for an empty list, but this parameter cannot be NULL)

Return:

Non NULL value indicating success (a pointer to the entry that was found).

NULL value indicating that an entry was not located in the specified list.

BSC_DeleteGenericListEntry

The following function is a utility function that allows the ability to remove a specific opaque list entry from the specified opaque list entry list. This function does NOT delete the memory for the entry, it simply removes it from the list and returns a pointer to the newly removed entry.

Notes:

Opaque lists and list entries are a schema that allows any data structure to be added to a list (of like structures). This schema is possible because all the routines that operate on the list (including this one) are told the necessary structure offsets (and sizes) of each entry. The utility functions also define the concept of a "key" that can be used for searching through the list.

This function does not free the resources of the element that was deleted from the List, it only removes it from the list and returns a pointer to the element. The Next Pointer element of the returned element will have it's value set to NULL.

It is the callers responsibility to free the memory that is occuppied by the specified list (when finished) by calling the **BSC_FreeGenericListEntryMemory**() function.

Prototype

void *BTPSAPI **BSC_DeleteGenericListEntry(**
    BSC_Generic_List_Entry_Key_t GenericListEntryKey,
    void *GenericListEntryKeyValue, unsigned int ListEntryKeyOffset,
    unsigned ListEntryNextPointerOffset, void **ListHead);

Parameters:

GenericListEntryKey	Key value type that is used to search the entries. This value must be one of: ekBoolean_t ekByte_t ekWord_t ekDWord_t ekBD_ADDR_t ekEntryPointer ekUnsignedInteger
ListEntryKeyValue	Pointer to the key value that is to matched for the search. The actual data type that this value points to depends upon the value of the previous parameter. Note that this value CANNOT be NULL.
ListEntryKeyOffset	Offset (specified in bytes) from the beginning of each list entry where the list entry key is located
ListEntryNextPointerOffset	Offset (specified in bytes) from the beginning of each list entry where the list entry next pointer is located
ListHead	Pointer to the location that holds a pointer to the first entry in the list (the value at this location can be NULL for an empty list, but this parameter cannot be NULL)

Return:

Non NULL value indicating success (a pointer to the entry that was removed).

NULL value indicating that an entry was not located in the specified list.

BSC_FreeGenericListEntryMemory

The following function is a utility function that allows the ability to free the memory for an opaque list entry that was allocated via the **BSC_FreeGenericListEntryMemory()** function.

Notes:

Opaque lists and list entries are a schema that allows any data structure to be added to a list (of like structures). This schema is possible because all the routines that operate on the list (including this one) are told the necessary structure offsets (and sizes) of each entry. The utility functions also define the concept of a "key" that can be used for searching through the list.

This function does not free any resources contained with the entry, it simply frees the memory of the entry that passed in.

Prototype

void *BTPSAPI **BSC_FreeGenericListEntryMemory(**void *EntryToFree);

Parameters:

EntryToFree	Pointer to the the actual opaque list entry memory that is be freed.

Return:

BSC_DeleteGenericListEntryList

The following function is a utility function that removes every list entry (and frees each list entry element) from the specified list.

<u>Notes:</u>

Opaque lists and list entries are a schema that allows any data structure to be added to a list (of like structures). This schema is possible because all the routines that operate on the list (including this one) are told the necessary structure offsets (and sizes) of each entry. The utility functions also define the concept of a "key" that can be used for searching through the list.

This function does not free the resources of the element that was deleted from the List, it only removes it from the list and frees the memory of each entry itself.

When this function returns, the list head will be set to NULL (indicating an empty list).

Prototype

void *BTPSAPI **BSC_DeleteGenericListEntryList(**void **ListHead
unsigned ListEntryNextPointerOffset);

Parameters:

ListHead	Pointer to the location that holds a pointer to the first entry in the list (the value at this location can be NULL for an empty list, but this parameter cannot be NULL)
ListEntryNextPointerOffset	Offset (specified in bytes) from the beginning of each list entry where the list entry next pointer is located

Return:

2.2 HCI API

The Host Controller Interface (HCI) layer API of the Bluetooth Protocol Stack provides software access to the HCI command interface to the baseband controller and link manager. This allows access to hardware status and control registers. This API provides a uniform method of accessing the Bluetooth baseband capabilities.

This API is organized into separate subsections primarily by the seven command groups as specified in the Bluetooth Core Specification. In addition, there is a section on miscellaneous commands/parameters and a section on the HCI events and the HCI LE meta events. Therefore, the subsections that follow are:

2.2.2 Link Control Commands

2.2.3 Link Policy Commands

2.2.4 Host Controller & Baseband Commands

2.2.5 Informational Parameters

2.2.6 Status Parameters

2.2.7 Testing Commands

2.2.8 LE Controller Commands

2.2.9 Miscellaneous Commands/Parameters

2.2.10 HCI Event/Data Callbacks and Registration

2.2.11 HCI Events

2.2.12 HCI LE Meta Event Sub-events

Every API function has a return that is zero when no error occurs in processing the request, and is one of the error conditions listed in the BTErrors.h Header File. In addition, the StatusResult value returned with every HCI command is only valid if the API function return is zero. Possible values for StatusResult are any of the HCI Error Codes listed below. The actual prototypes and constants outlined in this section can be found in the **HCIAPI.H** header file in the Bluetopia distribution.

2.2.1 HCI Error Codes

Bluetooth Version 1.0B

HCI_ERROR_CODE_NO_ERROR
HCI_ERROR_CODE_UNKNOWN_HCI_COMMAND
HCI_ERROR_CODE_NO_CONNECTION
HCI_ERROR_CODE_HARDWARE_FAILURE
HCI_ERROR_CODE_PAGE_TIMEOUT
HCI_ERROR_CODE_AUTHENTICATION_FAILURE
HCI_ERROR_CODE_KEY_MISSING
HCI_ERROR_CODE_MEMORY_FULL
HCI_ERROR_CODE_CONNECTION_TIMEOUT
HCI_ERROR_CODE_MAX_NUMBER_OF_CONNECTIONS
HCI_ERROR_CODE_MAX_NUMBER_OF_SCO_CONNECTIONS_TO_A_DEVICE
HCI_ERROR_CODE_ACL_CONNECTION_ALREADY_EXISTS
HCI_ERROR_CODE_COMMAND_DISALLOWED
HCI_ERROR_CODE_HOST_REJECTED_DUE_TO_LIMITED_RESOURCES
HCI_ERROR_CODE_HOST_REJECTED_DUE_TO_SECURITY_REASONS

HCI_ERROR_CODE_HOST_REJECTED_DUE_TO_REMOTE_DEVICE_IS_PERSONAL
HCI_ERROR_CODE_HOST_TIMEOUT
HCI_ERROR_CODE_UNSUPPORTED_FEATURE_OR_PARAMETER_VALUE
HCI_ERROR_CODE_INVALID_HCI_COMMAND_PARAMETERS
HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_USER_ENDED
HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_LOW_RESOURCES
HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_ABOUT_TO_PWR_OFF
HCI_ERROR_CODE_CONNECTION_TERMINATED_BY_LOCAL_HOST
HCI_ERROR_CODE_REPEATED_ATTEMPTS
HCI_ERROR_CODE_PAIRING_NOT_ALLOWED
HCI_ERROR_CODE_UNKNOWN_LMP_PDU
HCI_ERROR_CODE_UNSUPPORTED_REMOTE_FEATURE
HCI_ERROR_CODE_SCO_OFFSET_REJECTED
HCI_ERROR_CODE_SCO_INTERVAL_REJECTED
HCI_ERROR_CODE_SCO_AIR_MODE_REJECTED
HCI_ERROR_CODE_INVALID_LMP_PARAMETERS
HCI_ERROR_CODE_UNSPECIFIED_ERROR
HCI_ERROR_CODE_UNSUPPORTED_LMP_PARAMETER_VALUE
HCI_ERROR_CODE_ROLE_CHANGE_NOT_ALLOWED
HCI_ERROR_CODE_LMP_RESPONSE_TIMEOUT
HCI_ERROR_CODE_LMP_ERROR_TRANSACTION_COLLISION

Bluetooth Version 1.1

HCI_ERROR_CODE_LMP_PDU_NOT_ALLOWED
HCI_ERROR_CODE_ENCRYPTION_MODE_NOT_ACCEPTABLE
HCI_ERROR_CODE_UNIT_KEY_USED
HCI_ERROR_CODE_QOS_NOT_SUPPORTED
HCI_ERROR_CODE_INSTANT_PASSED
HCI_ERROR_CODE_PAIRING_WITH_UNIT_KEY_NOT_SUPPORTED

Bluetooth Version 1.2

HCI_ERROR_CODE_SUCCESS
HCI_ERROR_CODE_UNKNOWN_CONNECTION_IDENTIFIER
HCI_ERROR_CODE_PIN_MISSING
HCI_ERROR_CODE_MEMORY_CAPACITY_EXCEEDED
HCI_ERROR_CODE_CONNECTION_LIMIT_EXCEEDED
HCI_ERROR_CODE_SYNCHRONOUS_CONNECTION_LIMIT_TO_A_DEVICE_EXCEEDED
HCI_ERROR_CODE_CONNECTION_REJECTED_DUE_TO_LIMITED_RESOURCES
HCI_ERROR_CODE_CONNECTION_REJECTED_DUE_TO_SECURITY_REASONS
HCI_ERROR_CODE_CONNECTION_REJECTED_DUE_TO_UNACCEPTABLE_BD_ADDR
HCI_ERROR_CODE_CONNECTION_ACCEPT_TIMEOUT_EXCEEDED
HCI_ERROR_CODE_REMOTE_USER_TERMINATED_CONNECTION
HCI_ERROR_CODE_REMOTE_DEVICE_TERMINATED_CONNECTION_LOW_RESOURCES
HCI_ERROR_CODE_REMOTE_DEVICE_TERMINATED_CONNECTION_DUE_TO_PWR_OFF
HCI_ERROR_CODE_LINK_KEY_CANNOT_BE_CHANGED
HCI_ERROR_CODE_REQUESTED_QOS_NOT_SUPPORTED
HCI_ERROR_CODE_DIFFERENT_TRANSACTION_COLLISION
HCI_ERROR_CODE_QOS_UNACCEPTABLE_PARAMETER
HCI_ERROR_CODE_QOS_REJECTED
HCI_ERROR_CODE_CHANNEL_CLASSIFICATION_NOT_SUPPORTED
HCI_ERROR_CODE_INSUFFICIENT_SECURITY

HCI_ERROR_CODE_PARAMETER_OUT_OF_MANDATORY_RANGE
HCI_ERROR_CODE_ROLE_SWITCH_PENDING
HCI_ERROR_CODE_RESERVED_SLOT_VIOLATION
HCI_ERROR_CODE_ROLE_SWITCH_FAILED

Bluetooth Version 2.1

HCI_ERROR_CODE_EXTENDED_INQUIRY_RESPONSE_TOO_LARGE
HCI_ERROR_CODE_SECURE_SIMPLE_PAIRING_NOT_SUPPORTED_BY_HOST
HCI_ERROR_CODE_HOST_BUSY_PAIRING

Bluetooth Version 3.0

HCI_ERROR_CODE_CONNECTION_REJECTED_NO_SUITABLE_CHANNEL_FOUND

Bluetooth Version 4.0

HCI_ERROR_CODE_CONTROLLER_BUSY
HCI_ERROR_CODE_UNACCEPTABLE_CONNECTION_INTERVAL
HCI_ERROR_CODE_DIRECTED_ADVERTISING_TIMEOUT
HCI_ERROR_CODE_CONNECTION_FAILED_DUE_TO_MIC_FAILURE
HCI_ERROR_CODE_CONNECTION_FAILED_TO_BE_ESTABLISHED
HCI_ERROR_CODE_MAC_CONNECTION_FAILED

2.2.2 Link Control Commands

The Link Control commands are used to control the connections to other Bluetooth devices. These commands direct the Link Manager (LM) portion of the HCI to create and modify the link layer connections, and perform inquiries of other devices. Commands included in this section are listed in the table below.

Command	Description
HCI_Inquiry	Discover other nearby Bluetooth devices.
HCI_Inquiry_Cancel	Stop the current Inquiry.
HCI_Periodic_Inquiry_Mode	Perform an automatic Inquiry based on a specified period range.
HCI_Exit_Periodic_Inquiry_Mode	End the Periodic Inquiry mode.
HCI_Create_Connection	Create an ACL connection to a Bluetooth device.
HCI_Disconnect	Terminate a connection.
HCI_Add_SCO_Connection	Create an SCO connection using an existing ACL connection.
HCI_Accept_Connection_Request	Accept a new incoming connection request.
HCI_Reject_Connection_Request	Decline a new incoming connection request.

Command	Description
HCI_Link_Key_Request_Reply	Reply to a Link Key Request event from the Host Controller if the Host has a stored Link Key for the connection.
HCI_Link_Key_Request_Negative_Reply	Reply to a Link Key Request event from the Host Controller if the Host does not have a stored Link Key for the connection.
HCI_PIN_Code_Request_Reply	Reply to a PIN Code Request event from the Host Controller with the PIN code to use for the connection.
HCI_PIN_Code_Request_Negative_Reply	Reply to a PIN Code Request event from the Host Controller when the Host cannot specify a PIN code to use for a connection.
HCI_Change_Connection_Packet_Type	Change which packet types can be used for a connection.
HCI_Authentication_Requested	Establish authentication between the two devices associated in a connection.
HCI_Set_Connection_Encryption	Enable and disable the link level encryption.
HCI_Change_Connection_Link_Key	Force both devices in a connection to generate a new Link Key.
HCI_Master_Link_Key	Force both devices in a connection to use the temporary link key of the Master device or the regular Link Keys.
HCI_Remote_Name_Request	Obtain the user-friendly name of another device.
HCI_Read_Remote_Supported_Features	Obtain a list of the supported features of a remote device.
HCI_Read_Remote_Version_Information	Obtain the version information for the remote device.
HCI_Read_Clock_Offset	Read the clock offset of a remote device.
HCI_Create_Connection_Cancel	Cancel an ongoing connection process.
HCI_Remote_Name_Request_Cancel	Cancel an ongoing remote name request process.
HCI_Read_Remote_Extended_Features	Get the extended features from the remote device.

Command	Description
HCI_Read_LMP_Handle	Read the remote LMP handle of the remote device.
HCI_Setup_Synchronous_Connection	Setup a synchronous connection.
HCI_Accept_Synchronous_Connection_Request	Accept a synchronous connection request.
HCI_Reject_Synchronous_Connection_Request	Reject a synchronous connection request.
HCI_IO_Capability_Request_Reply	Reply to the IO capability request
HCI_User_Confirmation_Request_Reply	Reply to the user confirmation request
HCI_User_Confirmation_Request_Negative_Reply	A negative reply to the user confirmation request
HCI_User_Passkey_Request_Reply	Reply to the user passkey request
HCI_User_Passkey_Request_Negative_Reply	Negative reply to the user passkey request
HCI_Remote_OOB_Data_Request_Reply	Reply to the out of band (OOB) data request
HCI_Remote_OOB_Data_Request_Negative_Reply	Negative reply to the OOBdata request
HCI_IO_Capability_Request_Negative_Reply	Negative reply to the IO capability request
HCI_Create_Physical_Link	Issues HCI_Create_Physical_Link command to Bluetooth device.
HCI_Accept_Physical_Link_Request	Issues HCI_Accept_Physical_Link_Request command to Bluetooth device.
HCI_Disconnect_Physical_Link	Issues HCI_Disconnect_Physical_Link command to Bluetooth device.
HCI_Create_Logical_Link	Issues HCI_Create_Logical_Link command to Bluetooth device.
HCI_Accept_Logical_Link	Issues HCI_Accept_Logical_Link command to Bluetooth device.
HCI_Disconnect_Logical_Link	Issues HCI_Disconnect_Logical_Link command to Bluetooth device.
HCI_Logical_Link_Cancel	Issues HCI_Logical_Link_Cancel command to Bluetooth device.
HCI_Flow_Spec_Modify	Issues HCI_Flow_Spec_Modify command to Bluetooth device.

HCI_Inquiry

This command directs the Bluetooth device to go into Inquiry Mode in order to discover other nearby Bluetooth devices. The device stays in the Inquiry Mode until the specified length of time (Inquiry_Length) is reached or the maximum number of devices (Num_Responses) is found.

Prototype:

int BTPSAPI **HCI_Inquiry**(unsigned int BluetoothStackID, LAP_t LAP,
    Byte_t Inquiry_Length, Byte_t Num_Responses, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
LAP	Lower address part of the Bluetooth device address.
Inquiry_Length	Amount of time before the inquiry is halted. Values are in increments of 1.28 seconds, with a range of 1.28 sec. (0x01) to 61.44 sec. (0x30).
Num_Responses	Maximum number of Bluetooth devices to find before the inquiry is halted. A value of zero (0) means unlimited.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etInquiry_Result_Event
etInquiry_Result_With_RSSI_Event[2]
etInquiry_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2. This event is only possible on Bluetooth devices that adhere to the Bluetooth version 1.2 specification. Further, the inquiry mode has to be enabled via the **HCI_Write_Inquiry_Mode** command.

HCI_Inquiry_Cancel

This command directs the Bluetooth device to stop the current Inquiry if the Bluetooth device is in Inquiry Mode. The command should only be issued after the Inquiry command has been issued, a Command Status event has been received for the Inquiry command, and before the Inquiry Complete event occurs.

Prototype:

int BTPSAPI **HCI_Inquiry_Cancel**(unsigned int BluetoothStackID, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Periodic_Inquiry_Mode

This command directs the Bluetooth device to go into Periodic Inquiry Mode in which it automatically tries to discover other nearby Bluetooth devices at random intervals as bounded by the provided min and max period parameters. The device stays in the Inquiry Mode each time it is started (at the end of the next random interval) until the specified length of time (Inquiry_Length) is reached or the maximum number of devices (Num_Responses) is found.

Prototype:

int BTPSAPI **HCI_Periodic_Inquiry_Mode**(unsigned int BluetoothStackID,
Word_t Max_Period_Length, Word_t Min_Period_Length, LAP_t LAP,
Byte_t Inquiry_Length, Byte_t Num_Responses, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Max_Period_Length	Upper bound on random interval between inquiries. Values are in increments of 1.28 seconds, with a range of 3.84 sec. (0x03) to ~23.3 hrs. (0xFFFE)
Min_Period_Length	Lower bound on random interval between inquiries. Values are in increments of 1.28 seconds, with a range of 2.56 sec. (0x02) to ~23.3 hrs. (0xFFFE)
LAP	Lower address part of the Bluetooth device address. Range: 0x9E8B00–0x9E8B3F
Inquiry_Length	Amount of time before *each* inquiry is halted. Values are in increments of 1.28 seconds, with a range of 1.28 sec. (0x01) to 61.44 sec. (0x30).
Num_Responses	Maximum number of Bluetooth devices to find before *each* inquiry is halted. A value of zero (0) means unlimited.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etInquiry_Result_Event
etInquiry_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Exit_Periodic_Inquiry_Mode

Command the Bluetooth device to exit the Periodic Inquiry Mode. If the device is currently performing an inquiry, that inquiry is also cancelled.

Prototype:

int BTPSAPI **HCI_Exit_Periodic_Inquiry_Mode**(unsigned int BluetoothStackID,
 Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Create_Connection

This command directs the Link Manager to create a connection to the Bluetooth device specified by the command parameters. This command causes the local Bluetooth device to start the Page process to create a link level connection (ACL link).

Prototype:

int BTPSAPI **HCI_Create_Connection**(unsigned int BluetoothStackID,
 BD_ADDR_t BD_ADDR, Word_t Packet_Type, Byte_t Page_Scan_Repetition_Mode,
 Byte_t Page_Scan_Mode, Word_t Clock_Offset, Byte_t Allow_Role_Switch,
 Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address to connect to.
Packet_Type	Which packet types the Link Manager shall use for the ACL link. This can be an ORing of multiple packet types. The currently defined packet types are:

HCI_PACKET_ACL_TYPE_DM1

	HCI_PACKET_ACL_TYPE_DH1 HCI_PACKET_ACL_TYPE_DM3 HCI_PACKET_ACL_TYPE_DH3 HCI_PACKET_ACL_TYPE_DM5 HCI_PACKET_ACL_TYPE_DH5 **Bluetooth Version 2.0** HCI_PACKET_ACL_TYPE_2_DH1_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_3_DH1_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_2_DH3_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_3_DH3_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_2_DH5_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_3_DH5_MAY_NOT_BE_USED
Page_Scan_Repetition_Mode	Part of the supported Page Scan Modes that the device being connected to supports. This information is discovered during the Inquiry mode. The currently defined values are: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2
Page_Scan_Mode	The other part of the supported Page Scan Modes that the device being connected to supports. This information is discovered during the Inquiry mode. The currently defined values are: **Bluetooth Version 1.1** HCI_PAGE_SCAN_MODE_MANDATORY HCI_PAGE_SCAN_MODE_OPTIONAL_I HCI_PAGE_SCAN_MODE_OPTIONAL_II HCI_PAGE_SCAN_MODE_OPTIONAL_III **Bluetooth Version 1.2** HCI_PAGE_SCAN_MODE_MANDATORY_STANDARD_SCAN HCI_PAGE_SCAN_MODE_OPTIONAL_INTERLACED_SCAN
Clock_Offset	Bits 16 to 2 of the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid.
Allow_Role_Switch	Whether the local device will accept a role switch and become a slave device or not. The currently defined values are: HCI_ROLE_SWITCH_LOCAL_MASTER_NO_ROLE_SWITCH HCI_ROLE_SWITCH_LOCAL_MASTER_ACCEPT_ROLE_SWITCH
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etConnection_Complete_Event

etLink_Key_Request_Event

etPIN_Code_Request_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Disconnect

This command terminates an existing connection. All SCO connections on a physical link should be disconnected before the ACL connection on the same physical connection is disconnected.

Prototype:

int BTPSAPI **HCI_Disconnect**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t Reason, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Reason	The reason for ending the connection. Subset of HCI Status Codes. Possible values are: HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_USER_ENDED HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_LOW_RESOURCES HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_ABOUT_TO_PWR_OFF HCI_ERROR_CODE_UNSUPPORTED_REMOTE_FEATURE

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etDisconnection_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Add_SCO_Connection

This command adds an SCO connection to the ACL link connection indicated (Connection_Handle parameter).

Prototype:

int BTPSAPI **HCI_Add_SCO_Connection**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Word_t Packet_Type, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Handle for the ACL connection from which to base the SCO link to the same remote device.
Packet_Type	Which packet types the Link Manager shall use for the SCO connection. This can be an ORing of multiple packet types. The currently defined packet types are: HCI_PACKET_SCO_TYPE_HV1 HCI_PACKET_SCO_TYPE_HV2 HCI_PACKET_SCO_TYPE_HV3
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etConnection_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Accept_Connection_Request

This command accepts a new incoming connection request after a Connection Request event has been received. The Connection Request event provides the BD_ADDR of the device which is requesting the connection. This address is then passed back to the Link Manager in this command to create a connection to the device.

Prototype:

int BTPSAPI **HCI_Accept_Connection_Request**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, Byte_t Role, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address for the device to connect to.
Role	Designate the master-slave role to take on in this connection. Possible Values are: HCI_ROLE_SWITCH_BECOME_MASTER HCI_ROLE_SWITCH_REMAIN_SLAVE
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etConnection_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Reject_Connection_Request

This command rejects a new incoming connection request after a Connection Request event has been received. The Connection Request event provides the BD_ADDR of the device which is requesting the connection.

Prototype:

int BTPSAPI **HCI_Reject_Connection_Request**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t Reason, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address for the device to connect to.
Reason	The reason for the refusal. Possible values: HCI_ERROR_CODE_HOST_REJECTED_DUE_ TO_LIMITED_RESOURCES HCI_ERROR_CODE_HOST_REJECTED_DUE_ TO_SECURITY_REASONS HCI_ERROR_CODE_HOST_REJECTED_DUE_ TO_REMOTE_DEVICE_IS_PERSONAL
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etConnection_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Link_Key_Request_Reply

This command is one of two ways to respond to a Link Key Request event, specifying a link key to use for the connection. The Link Key Request event is generated when the Host Controller needs a Link Key for the connection.

Prototype:

int BTPSAPI **HCI_Link_Key_Request_Reply**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, Link_Key_t Link_Key, Byte_t *StatusResult,
    BD_ADDR_t *BD_ADDRResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device that the link key is for.
Link_Key	16-Byte Link Key to use to make the connection.
StatusResult	Returned HCI status code.
BD_ADDRResult	Pointer for return value of Bluetooth device for which the link key request reply was completed.

Return: Zero if successful. An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Link_Key_Request_Negative_Reply

This command is one of two ways to respond to a Link Key Request event, indicating that the local host does not have the link key for the remote device. The Link Key Request event is generated when the Host Controller needs a Link Key for the connection.

Prototype:

int BTPSAPI **HCI_Link_Key_Request_Negative_Reply**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device that the link key is for.
StatusResult	Returned HCI status code.
BD_ADDRResult	Pointer for return value of Bluetooth device for which the link key request negative reply was completed.

Return: Zero if successful. An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etPIN_Code_Request_Reply

etAuthentication_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_PIN_Code_Request_Reply

This command is one of two ways to respond to a PIN Code Request event, specifying a PIN Code to use for the connection. The PIN Code Request event is generated when a connection with a remote device requests a pairing.

Prototype:

int BTPSAPI **HCI_PIN_Code_Request_Reply**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t PIN_Code_Length, PIN_Code_t PIN_Code,
Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device which the PIN Code is for.

PIN_Code_Length	The length in bytes of the PIN Code in the range of 0x01 to 0x10.
PIN_Code	The PIN Code for the device being connected, with the MSB in byte zero.
StatusResult	Returned HCI status code.
BD_ADDRResult	Pointer for return value of Bluetooth device for which the PIN Code request reply was completed.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etLink_Key_Notification_Event

etAuthentication_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_PIN_Code_Request_Negative_Reply

This command is one of two ways to respond to a PIN Code Request event, indicating that local host does not have the PIN Code for the remote device. This causes the pairing request from the remote device to fail.

Prototype:

int BTPSAPI **HCI_PIN_Code_Request_Negative_Reply**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device which the PIN Code is for.
StatusResult	Returned HCI status code.
BD_ADDRResult	Pointer for return value of Bluetooth device for which the PIN Code request negative reply was completed.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etAuthentication_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Change_Connection_Packet_Type

This command changes which packet types can be used on an established connection. This function is used to dynamically modify a connection to support different user data types.

Prototype:

int BTPSAPI **HCI_Change_Connection_Packet_Type**(unsigned int BluetoothStackID, Word_t Connection_Handle, Word_t Packet_Type, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Handle for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Packet_Type	Which packet types the Link Manager shall use for the ACL link. This can be an ORing of multiple packet types. The currently defined packet types are – For ACL Links: HCI_PACKET_ACL_TYPE_DM1 HCI_PACKET_ACL_TYPE_DH1 HCI_PACKET_ACL_TYPE_DM3 HCI_PACKET_ACL_TYPE_DH3 HCI_PACKET_ACL_TYPE_DM5 HCI_PACKET_ACL_TYPE_DH5 **Bluetooth Version 2.0** HCI_PACKET_ACL_TYPE_2_DH1_MAY_NOT_BE_USED

HCI_PACKET_ACL_TYPE_3_DH1_MAY_NOT_BE_USED
HCI_PACKET_ACL_TYPE_2_DH3_MAY_NOT_BE_USED
HCI_PACKET_ACL_TYPE_3_DH3_MAY_NOT_BE_USED
HCI_PACKET_ACL_TYPE_2_DH5_MAY_NOT_BE_USED
HCI_PACKET_ACL_TYPE_3_DH5_MAY_NOT_BE_USED

For SCO Links:

HCI_PACKET_SCO_TYPE_HV1
HCI_PACKET_SCO_TYPE_HV2
HCI_PACKET_SCO_TYPE_HV3

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etConnection_Packet_Type_Changed_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Authentication_Requested

This command attempts to authenticate the remote device associated with the specified Connection Handle for an ACL link. This command must not be used with a Connection_Handle corresponding to an encrypted link.

Prototype:

int BTPSAPI **HCI_Authentication_Requested**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etAuthentication_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Set_Connection_Encryption

This command enables or disables link level encrytion for an ACL link. All ACL link traffic for the connection must be turned off while the encrytion is changed.

Prototype:

int BTPSAPI **HCI_Set_Connection_Encryption**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t Encryption_Enable, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Encryption_Enable	Flag indicating whether the encryption should be turned on or off. Possible values are: HCI_ENCRYPTION_ENABLE_LINK_LEVEL_OFF HCI_ENCRYPTION_ENABLE_LINK_LEVEL_ON
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES

BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etEncryption_Change_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Change_Connection_Link_Key

This command forces both sides of a connection to generate a new link key for an ACL link.

Prototype:

int BTPSAPI **HCI_Change_Connection_Link_Key**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etLink_Key_Notification_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Master_Link_Key

This command forces the device that is master to use either the temporary link key of the master device, or the semi-permanent link keys.

Prototype:

int BTPSAPI **HCI_Master_Link_Key**(unsigned int BluetoothStackID, Byte_t Key_Flag, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Key_Flag	Indicator of which link key to change to. Possible values are: HCI_MASTER_LINK_KEY_USE_SEMI_PERMANENT_ LINK_KEYS HCI_MASTER_LINK_KEY_USE_TEMPORARY_ LINK_KEYS
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etMaster_Link_Key_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Remote_Name_Request

This command obtains the user-friendly name for the remote Bluetooth device.

Prototype:

int BTPSAPI **HCI_Remote_Name_Request**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Byte_t Page_Scan_Repetition_Mode, Byte_t Page_Scan_Mode, Word_t Clock_Offset, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
Page_Scan_Repetition_Mode	Part of the supported Page Scan Modes that the device being connected to supports. This information is discovered during the Inquiry mode. Possible values are: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2
Page_Scan_Mode	The other part of the supported Page Scan Modes that the device being connected to supports. This information is discovered during the Inquiry mode. Possible values are: **Bluetooth Version 1.1** HCI_PAGE_SCAN_MODE_MANDATORY HCI_PAGE_SCAN_MODE_OPTIONAL_I HCI_PAGE_SCAN_MODE_OPTIONAL_II HCI_PAGE_SCAN_MODE_OPTIONAL_III **Bluetooth Version 1.2** HCI_PAGE_SCAN_MODE_MANDATORY_STANDARD_SCAN HCI_PAGE_SCAN_MODE_OPTIONAL_INTERLACED_SCAN
Clock_Offset	Bits 16 to 2 of the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etRemote_Name_Request_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Remote_Supported_Features

This command requests a list of the supported features for the remote device, via the ACL link to that device.

Prototype:

int BTPSAPI **HCI_Read_Remote_Supported_Features**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Handle for the ACL connection.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etRead_Remote_Supported_Features_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Remote_Version_Information

This command obtains the version information for the remote device.

Prototype:

int BTPSAPI **HCI_Read_Remote_Version_Information**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etRead_Remote_Version_Information_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Clock_Offset

This command reads the clock offset of the remote device connected via an ACL link. This offset is used for frequency hopping and as an input into other functions.

Prototype:

int BTPSAPI **HCI_Read_Clock_Offset**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Handle for the ACL connection to the remote device.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etRead_Clock_Offset_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Create_Connection_Cancel

This command is used to request cancellation of an ongoing connection creation process, which was started by a **HCI_Create_Connection** command issued to the local device.

Prototype:

int BTPSAPI **HCI_Create_Connection_Cancel**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
StatusResult	Returned HCI status code.
BD_ADDRResult	Pointer for return value of Bluetooth device for which the create connection cancel reply was completed.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Remote_Name_Request_Cancel

This command is used to request cancellation of the ongoing remote name request process, which was started by the **HCI_Remote_Name_Request** command.

Prototype:

int BTPSAPI **HCI_Remote_Name_Request_Cancel**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
StatusResult	Returned HCI status code.
BD_ADDRResult	Pointer for return value of Bluetooth device for which the create connection cancel reply was completed.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Remote_Extended_Features

This command returns the requested page of the extended LMP features for the remote device identified by the specified connection handle. The connection handle must be a connection handle for an ACL connection. This command is only available if the extended features feature is implemented by the remote device. The **etRead_Remote_Extended_Features_Complete** event will return the requested information.

Prototype:

int BTPSAPI **HCI_Read_Remote_Extended_Features**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t Page_Number, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Handle for the ACL connection to the remote device.
Page_Number	The Page Number of the Extended Features Mask that is to be returned. Passing zero for this parameter returns the normal LMP features mask.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etRead_Remote_Extended_Features_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_LMP_Handle

This command will read the current LMP Handle associated with the specified connection handle. The connection handle must be a SCO or eSCO Handle. If the connection handle is a SCO connection handle, then this command will read the LMP SCO handle for this connection. If the connection handle is an eSCO connection handle, then this command will read the LMP eSCO Handle for the specified connection.

Prototype:

int BTPSAPI **HCI_Read_LMP_Handle**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult,
Byte_t *LMP_HandleResult, DWord_t *ReservedResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Handle for the ACL connection to the remote device.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the read LMPhandle was done.
LMP_HandleResult	LMP handle from the remote device.
ReservedResult	Reserved result from the remote device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Setup_Synchronous_Connection

This command adds a new, or modifies an existing, synchronous logical transport (SCO or eSCO) on a physical link depending on the Connection_Handle parameter specified. If the connection handle refers to an ACL link a new synchronous logical transport will be added. If the connection handle refers to an already existing synchronous logical transport (eSCO only), then the link will be modified. The parameters are specified per connection. This synchronous connection can be used to transfer synchronous voice at 64kbps or transparent synchronous data.

Prototype:

int BTPSAPI **HCI_Setup_Synchronous_Connection**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, DWord_t Transmit_Bandwidth,
    DWord_t Receive_Bandwidth, Word_t Max_Latency, Word_t Voice_Setting,
    Byte_t Retransmission_Effort, Word_t Packet_Type, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Handle for the ACL connection to the remote device.
Transmit_Bandwidth	Amount of bandwidth available for transmit.
Receive_Bandwidth	Amount of bandwidth available for receive.
Max_Latency	Upper limit of the time (in milliseconds) between the eSCO (or SCO) instants, plus the size of the retransmission window, plus the length of the reserved synchronous slots for this logical transport. This must fall in the range defined by the following constants: HCI_SYNCHRONOUS_CONNECTION_MAX_LATENCY_MINIMUM HCI_SYNCHRONOUS_CONNECTION_MAX_LATENCY_MAXIMUM or be the following defined value: HCI_SYNCHRONOUS_CONNECTION_MAX_LATENCY_DONT_CARE
Voice_Setting	Indicates if this connection is for voice or transparent data. This is the Logical OR'ing of bits in five categories as defined by the following masks: HCI_VOICE_SETTING_INPUT_CODING_MASK HCI_VOICE_SETTING_INPUT_DATA_FORMAT_MASK HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_MASK HCI_VOICE_SETTING_LINEAR_PCM_BIT_POS_NUM_MASK HCI_VOICE_SETTING_AIR_CODING_FORMAT_MASK the Input Coding bits which may be set are: HCI_VOICE_SETTING_INPUT_CODING_LINEAR HCI_VOICE_SETTING_INPUT_CODING_U_LAW HCI_VOICE_SETTING_INPUT_CODING_A_LAW the Input Data Format bits which may set are: HCI_VOICE_SETTING_INPUT_DATA_FORMAT_1_COMPLEMENT HCI_VOICE_SETTING_INPUT_DATA_FORMAT_2_COMPLEMENT HCI_VOICE_SETTING_INPUT_DATA_FORMAT_SIGN_MAGNITUDE HCI_VOICE_SETTING_INPUT_DATA_FORMAT_UNSIGNED the Input Sample Size which may set are: HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_8_BIT HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_16_BIT

	the Linear PCM Bit Position Shift Value bits which may be set are: HCI_VOICE_SETTING_LINEAR_PCM_BIT_POS_NUM_SHIFT_VALUE the Air Coding Format bits which may be set are: **Bluetooth Version 1.1** HCI_VOICE_SETTING_AIR_CODING_FORMAT_CVSD HCI_VOICE_SETTING_AIR_CODING_FORMAT_U_LAW HCI_VOICE_SETTING_AIR_CODING_FORMAT_A_LAW HCI_VOICE_SETTING_AIR_CODING_FORMAT_NONE **Bluetooth Version 1.2** HCI_VOICE_SETTING_AIR_CODING_FORMAT_TRANSPARENT_DATA
Retransmission_Effort	The extra resources that are allocated to this connection if a packet needs to be retransmitted. The Retransmission_Effort parameter shall be set to indicate the required behaviour, or to don't care. Possible values are: HCI_SYNCHRONOUS_CONNECTION_RETRANSMISSION_EFFORT_NONE HCI_SYNCHRONOUS_CONNECTION_RETRANSMISSION_EFFORT_ONE_OPTIMIZE_POWER HCI_SYNCHRONOUS_CONNECTION_RETRANSMISSION_EFFORT_ONE_OPTIMIZE_QUALITY HCI_SYNCHRONOUS_CONNECTION_RETRANSMISSION_EFFORT_DONT_CARE
Packet_Type	A bitmask specifying which packet types the LM shall accept in the negotiation of the link parameters. This is a Logical OR'ing of bit values for the packet types as defined by the following values: HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_HV1 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_HV2 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_HV3 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_EV1 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_EV2 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_EV3
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etConnection_Complete_Event
etSynchronous_Connection_Complete_Event
etSynchronous_Connection_Changed_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Accept_Synchronous_Connection_Request

This command is used to accept an incoming request for a synchronous connection and to inform the local Link Manager about the acceptable parameter values for the synchronous connection. This Command shall only be issued after an **etConnection_Request_Event** event with link type SCO or eSCO has been received. The connection request event contains the BD_ADDR of the device requesting the connection. The decision to accept an incoming connection must be taken before the connection accept timeout expires on the local device.

Prototype:

int BTPSAPI **HCI_Accept_Synchronous_Connection_Request**(
unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR,
DWord_t Transmit_Bandwidth, DWord_t Receive_Bandwidth, Word_t Max_Latency,
Word_t Content_Format, Byte_t Retransmission_Effort, Word_t Packet_Type,
Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
Transmit_Bandwidth	Amount of bandwidth available for transmit. This must fall in the range defined by the following constants: HCI_SYNCHRONOUS_CONNECTION_ACCEPT_TRANSMIT_BANDWIDTH_MINIMUM

	HCI_SYNCHRONOUS_CONNECTION_ACCEPT_TRANSMIT_BANDWIDTH_MAXIMUM or be the following defined value: HCI_SYNCHRONOUS_CONNECTION_ACCEPT_TRANSMIT_BANDWIDTH_DONT_CARE
Receive_Bandwidth	Amount of bandwidth available for receive. This must fall in the range defined by the following constants: HCI_SYNCHRONOUS_CONNECTION_ACCEPT_RECEIVE_BANDWIDTH_MINIMUM HCI_SYNCHRONOUS_CONNECTION_ACCEPT_RECEIVE_BANDWIDTH_MAXIMUM or be the following defined value: HCI_SYNCHRONOUS_CONNECTION_ACCEPT_RECEIVE_BANDWIDTH_DONT_CARE
Max_Latency	Upper limit of the time (in milliseconds) between the eSCO (or SCO) instants, plus the size of the retransmission window, plus the length of the reserved synchronous slots for this logical transport. This must fall in the range defined by the following constants: HCI_SYNCHRONOUS_CONNECTION_MAX_LATENCY_MINIMUM HCI_SYNCHRONOUS_CONNECTION_MAX_LATENCY_MAXIMUM or be the following defined value: HCI_SYNCHRONOUS_CONNECTION_MAX_LATENCY_DONT_CARE
Content_Format	Indicates if this connection is for voice or transparent data. This is a Logical OR'ing of bits in five categories as defined by the following bit masks: HCI_VOICE_SETTING_INPUT_CODING_MASK HCI_VOICE_SETTING_INPUT_DATA_FORMAT_MASK HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_MASK HCI_VOICE_SETTING_LINEAR_PCM_BIT_POS_NUM_MASK HCI_VOICE_SETTING_AIR_CODING_FORMAT_MASK the Input Coding bit values which may be set are: HCI_VOICE_SETTING_INPUT_CODING_LINEAR HCI_VOICE_SETTING_INPUT_CODING_U_LAW HCI_VOICE_SETTING_INPUT_CODING_A_LAW the Input Data Format bit values which may set are: HCI_VOICE_SETTING_INPUT_DATA_FORMAT_1_COMPLEMENT

	HCI_VOICE_SETTING_INPUT_DATA_FORMAT_2_COMPLEMENT HCI_VOICE_SETTING_INPUT_DATA_FORMAT_SIGN_MAGNITUDE HCI_VOICE_SETTING_INPUT_DATA_FORMAT_UNSIGNED the Input Sample Size values which may set are: HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_8_BIT HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_16_BIT the Linear PCM Bit Position Shift Value bits which may be set are: HCI_VOICE_SETTING_LINEAR_PCM_BIT_POS_NUM_SHIFT_VALUE the Air Coding Format bit values which may be set are: **Bluetooth Version 1.1** HCI_VOICE_SETTING_AIR_CODING_FORMAT_CVSD HCI_VOICE_SETTING_AIR_CODING_FORMAT_U_LAW HCI_VOICE_SETTING_AIR_CODING_FORMAT_A_LAW HCI_VOICE_SETTING_AIR_CODING_FORMAT_NONE **Bluetooth Version 1.2** HCI_VOICE_SETTING_AIR_CODING_FORMAT_TRANSPARENT_DATA
Retransmission_Effort	Specifies the extra resources that are allocated to this connection if a packet may need to be retransmitted. The Retransmission_Effort parameter shall be set to indicate the required behaviour, or to don't care. Possible values are: HCI_SYNCHRONOUS_CONNECTION_RETRANSMISSION_EFFORT_NONE HCI_SYNCHRONOUS_CONNECTION_RETRANSMISSION_EFFORT_ONE_OPTIMIZE_POWER HCI_SYNCHRONOUS_CONNECTION_RETRANSMISSION_EFFORT_ONE_OPTIMIZE_QUALITY HCI_SYNCHRONOUS_CONNECTION_RETRANSMISSION_EFFORT_DONT_CARE
Packet_Type	A bitmask specifying which packet types the LM shall accept in the negotiation of the link parameters. This is a Logical OR'ing of bit values for the packet types as defined by the following values: HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_HV1 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_HV2 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_HV3

	HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_EV1 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_EV2 HCI_PACKET_SYNCHRONOUS_CONNECTION_TYPE_EV3
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etConnection_Complete_Event
etSynchronous_Connection_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Reject_Synchronous_Connection_Request

This command is used to decline an incoming request for a synchronous link. It shall only be issued after a **etConnection_Request_Event** has been received with Link Type equal to the SCO or eSCO type.

Prototype:

int BTPSAPI **HCI_Reject_Synchronous_Connection_Request**(
    unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Byte_t Reason,
    Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
Reason	Host reject error code returned to the initiating host in the Status parameter of the Synchronous connection complete event on the remote side. Possible values:

	HCI_ERROR_CODE_HOST_REJECTED_DUE_TO_LIMITED_RESOURCES HCI_ERROR_CODE_HOST_REJECTED_DUE_TO_SECURITY_REASONS HCI_ERROR_CODE_HOST_REJECTED_DUE_TO_REMOTE_DEVICE_IS_PERSONAL
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etConnection_Complete_Event
etSynchronous_Connection_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_IO_Capability_Request_Reply

This function issues the HCI_IO_Capability_Request_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_IO_Capability_Request_Reply**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Byte_t IO_Capability, Byte_t OOB_Data_Present, Byte_t Authentication_Requirements, Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
IO_Capability	IO Capabilities of the local device. Possible values: HCI_IO_CAPABILITY_DISPLAY_ONLY HCI_IO_CAPABILITY_DISPLAY_YES_NO HCI_IO_CAPABILITY_KEYBOARD_ONLY HCI_IO_CAPABILITY_NO_INPUT_NO_OUTPUT

OOB_Data_Present	Specifies whether or not OOB Data for the remote Bluetooth device is present (zero signifies not present).
Authentication_Requirements	Authentication Requirements of the local device. Possible values: HCI_AUTHENTICATION_REQUIREMENTS_MITM_PROTECTION_NOT_REQUIRED_NO_BONDING HCI_AUTHENTICATION_REQUIREMENTS_MITM_PROTECTION_NOT_REQUIRED_DEDICATED_BONDING HCI_AUTHENTICATION_REQUIREMENTS_MITM_PROTECTION_NOT_REQUIRED_GENERAL_BONDING
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDR Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_User_Confirmation_Request_Reply

This function issues the HCI_User_Confirmation_Request_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stackspecified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_User_Confirmation_Request_Reply**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDR Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_User_Confirmation_Request_Negative_Reply

This function issues the HCI_User_Confirmation_Request_Negative_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_User_Confirmation_Request_Negative_Reply**(
    unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Byte_t *StatusResult,
    BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDR Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_User_Passkey_Request_Reply

This function issues the HCI_User_Passkey_Request_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_User_Passkey_Request_Reply**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, DWord_t Numeric_Value, Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
Numeric_Value	Actual passkey value. This value must be between 0 and 999999.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDR Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_User_Passkey_Request_Negative_Reply

This function issues the HCI_User_Passkey_Request_Negative_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_User_Passkey_Request_Negative_Reply**(
unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDR Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Remote_OOB_Data_Request_Reply

This function issues the HCI_Remote_OOB_Data_Request_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Remote_OOB_Data_Request_Reply**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Simple_Pairing_Hash_t Simple_Pairing_Hash,
Simple_Pairing_Randomizer_t Simple_Pairing_Randomizer, Byte_t *StatusResult,
BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
Simple_Pairing_Hash	Simple pairing of the OOB data that was received for the remote device (C).
Simple_Pairing_Randomizer	Simple pairing randomizer of the OOB data that was received for the remote device (R)
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDR Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Remote_OOB_Data_Request_Negative_Reply

This function issues the HCI_Remote_OOB_Data_Request_Negative_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Remote_OOB_Data_Request_Negative_Reply**(
unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Byte_t *StatusResult,
BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDR Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_IO_Capability_Request_Negative_Reply

This function issues the HCI_IO_Capability_Request_Negative_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_IO_Capability_Request_Negative_Reply**(
unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Byte_t Reason,
Byte_t *StatusResult, BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the remote Bluetooth device.
Reason	Reason code for the IO Capability rejection. Possible values are the HCI Status Codes.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDR Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Create_Physical_Link

Issues the HCI_Create_Physical_Link command to the Bluetooth device that is associated to the specified Bluetooth Protocol Stack (which is specified with the BluetoothStackID parameter). Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Create_Physical_Link**(unsigned int BluetoothStackID,
Byte_t Physical_Link_Handle, Byte_t Dedicated_AMP_Key_Length, Byte_t Dedicated_AMP_Key_Type, Byte_t Dedicated_AMP_Key[], Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Physical_Link_Handle	Physical Link Handle indentifing the physical link to be created.
Dedicated_AMP_Key_Length	The number of valid octets (bytes) in the Dedicated_AMP_Key parameter.
Dedicated_AMP_Key_Type	Indicates the type of key that the parameter Dedicated_AMP_Key[] is. Valid values are:

	HCI_PHYSICAL_LINK_LINK_KEY_TYPE_DEBUG_COMBINATION_KEY HCI_PHYSICAL_LINK_LINK_KEY_TYPE_UNAUTHENTICATED_COMBINATION_KEY HCI_PHYSICAL_LINK_LINK_KEY_TYPE_AUTHENTICATED_COMBINATION_KEY All other values are reserved.
Dedicated_AMP_Key[]	Byte array with Dedicated_AMP_Key_Length valid bytes that will be used to generate a session key in order to encrypt all data on the physical link specified by Physical_Link_Handle.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etPhysical_Link_Complete_Event

etChannel_Selected_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Accept_Physical_Link_Request

Issues the HCI_Accept_Physical_Link_Request to the Bluetooth device that is associated with the Bluetooth Protocol stack (which itself is specified with the BluetoothStackID parameter. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Accept_Physical_Link_Request**(unsigned int BluetoothStackID, Byte_t Physical_Link_Handle, Byte_t Dedicated_AMP_Key_Length, Byte_t Dedicated_AMP_Key_Type, Byte_t Dedicated_AMP_Key[], Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Dedicated_AMP_Key_Length	The number of valid octets (bytes) in the Dedicated_AMP_Key parameter.
Dedicated_AMP_Key_Type	Indicates the type of key that the parameter Dedicated_AMP_Key[] is. Valid values are: HCI_PHYSICAL_LINK_LINK_KEY_TYPE_ DEBUG_COMBINATION_KEY HCI_PHYSICAL_LINK_LINK_KEY_TYPE_ UNAUTHENTICATED_COMBINATION_ KEY HCI_PHYSICAL_LINK_LINK_KEY_TYPE_ AUTHENTICATED_COMBINATION_KEY All other values are reserved.
Dedicated_AMP_Key[]	Byte array with Dedicated_AMP_Key_Length valid bytes that will be used to generate a session key in order to encrypt all data on the physical link specified by Physical_Link_Handle.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etPhysical_Link_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Disconnect_Physical_Link

Issues the HCI_Disconnect_Physical_Link command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Disconnect_Physical_Link**(unsigned int BluetoothStackID,
Byte_t Physical_Link_Handle, Byte_t Reason, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Physical_Link_Handle	Physical Link Handle identifying the physical link which has been created.
Reason	Byte value indicating the reason that the specified physical link is being disconnected. The remote controller will receive this parameter in the etDisconnection_Physical_Link_Complete_Event event. Possible values are: HCI_ERROR_CODE_AUTHENTICATION_FAILURE HCI_ERROR_CODE_REMOTE_USER_TERMINATED_CONNECTION HCI_ERROR_CODE_REMOTE_DEVICE_TERMINATED_CONNECTION_LOW_RESOURCES HCI_ERROR_CODE_REMOTE_DEVICE_TERMINATED_CONNECTION_DUE_TO_PWR_OFF HCI_ERROR_CODE_CONNECTION_TERMINATED_BY_LOCAL_HOST HCI_ERROR_CODE_UNSUPPORTED_REMOTE_FEATURE
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etDisconnection_Physical_Link_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Create_Logical_Link

Issues the HCI_Create_Logical_Link command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Create_Logical_Link**(unsigned int BluetoothStackID,
    Byte_t Physical_Link_Handle, HCI_Extended_Flow_Spec_Data_t *Tx_Flow_Spec,
    HCI_Extended_Flow_Spec_Data_t *Rx_Flow_Spec, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Physical_Link_Handle	Handle of the physical link over which the logical link will be created.
Tx_Flow_Spec	Extended flow specification value that defines the transmitted traffic.
Rx_Flow_Spec	Extended flow specification value that defines the received traffic.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etLogical_Link_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Accept_Logical_Link

Issues the HCI_Accept_Logical_Link command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Accept_Logical_Link**(unsigned int BluetoothStackID,
  Byte_t Physical_Link_Handle, HCI_Extended_Flow_Spec_Data_t *Tx_Flow_Spec,
  HCI_Extended_Flow_Spec_Data_t *Rx_Flow_Spec, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Physical_Link_Handle	Handle of the physical link over which the logical link will be created.
Tx_Flow_Spec	Extended flow specification value that defines the transmitted traffic.
Rx_Flow_Spec	Extended flow specification value that defines the received traffic.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etLogical_Link_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Disconnect_Logical_Link

Issues the HCI_Disconnect_Logical_Link command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Disconnect_Logical_Link**(unsigned int BluetoothStackID,
Word_t Logical_Link_Handle, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Logical_Link_Handle	Handle of the logical link that is to be disconnected.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etDisconnection_Logical_Link_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Logical_Link_Cancel

Issues the HCI_Logical_Link_Cancel command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Logical_Link_Cancel**(unsigned int BluetoothStackID,
Byte_t Physical_Link_Handle, Byte_t Tx_Flow_Spec_ID, Byte_t *StatusResult,
Byte_t *Physical_Link_HandleResult, Byte_t *Tx_Flow_Spec_IDResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Physical_Link_Handle	Physical link handle for the physical link over which the logical link was being established.
Tx_Flow_Spec_ID	Flow Spec ID identifying th logical link whose creation is being cancelled.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.
Physical_Link_HandleResult	If this function returns zero (success) then the variable pointed to by Physical_Link_HandleResult will contain the Physical Link Handle returned from the Bluetooth device.
Tx_Flow_Spec_IDResult	If this function returns zero (success) then the variable pointed to by Tx_Flow_Spec_IDResult will contain the Tx Flow Spec ID returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etLogical_Link_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Flow_Spec_Modify

Issues the HCI_Flow_Spec_Modify command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Flow_Spec_Modify**(unsigned int BluetoothStackID,
    Word_t Handle, HCI_Extended_Flow_Spec_Data_t *Tx_Flow_Spec,
    HCI_Extended_Flow_Spec_Data_t *Rx_Flow_Spec, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Handle	Logical Handle of the logical connection whose Flow Spec will be modified.
Tx_Flow_Spec	Extended flow specification value that defines the transmitted traffic.
Rx_Flow_Spec	Extended flow specification value that defines the received traffic.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etFlow_Spec_Modify_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.3 Link Policy Commands

The Link Policy Commands provides a means to affect the Link Manager's (LM) operation. Commands included in this section are listed in the table below.

Command	Description
HCI_Hold_Mode	Direct the Link Manager to place the local or remote device into the hold mode.
HCI_Sniff_Mode	Direct the Link Manager to place the local or remote device into the sniff mode.
HCI_Exit_Sniff_Mode	End the sniff mode
HCI_Park_Mode	Direct the Link Manager to place the local or remote device into the Park mode.
HCI_Exit_Park_Mode	Switch the Bluetooth device from park mode back

Command	Description
	to active mode.
HCI_QoS_Setup	Specify the Quality of Service parameters for a connection.
HCI_Role_Discovery	Determine which role a Bluetooth device is performing for a particular connection.
HCI_Switch_Role	Switch the current role that a Bluetooth device is performing for a particular connection.
HCI_Read_Link_Policy_Settings	Read the Link Policy settings for the specified Connection.
HCI_Write_Link_Policy_Settings	Write the Link Policy settings for the specified Connection.
HCI_Read_Default_Link_Policy_Settings	Read the default Link Policy settings for the specified connection.
HCI_ Write_Default_Link_Policy_Settings	Write the default Link Policy settings for the specified connection.
HCI_ Flow_Specification	Specify the flow parameters for the traffic carried over the specified ACL connection.
HCI_Sniff_Subrating	Set the sniff subrating

HCI_Hold_Mode

This command places the specified connection into Hold Mode as per the specified parameters.

Prototype:

int BTPSAPI **HCI_Hold_Mode**(unsigned int BluetoothStackID, Word_t Connection_Handle,
Word_t Hold_Mode_Max_Interval, Word_t Hold_Mode_Min_Interval,
Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Hold_Mode_Max_Interval	Maximum time to stay in Hold Mode. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).

Hold_Mode_Min_Interval	Minimum time to stay in Hold Mode. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF)
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etMode_Change_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Sniff_Mode

This command places the specified connection into Sniff Mode as per the specified parameters.

Prototype:

int BTPSAPI **HCI_Sniff_Mode**(unsigned int BluetoothStackID, Word_t Connection_Handle, Word_t Sniff_Max_Interval, Word_t Sniff_Min_Interval, Word_t Sniff_Attempt, Word_t Sniff_Timeout, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Sniff_Max_Interval	Maximum time between each sniff period. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).
Sniff_Min_Interval	Minimum time between each sniff period. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).

Sniff_Attempt	Amount of time for each sniff attempt. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).
Sniff_Timeout	Amount of time for sniff timeout. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etMode_Change_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Exit_Sniff_Mode

This command terminates the Sniff Mode for a connection.

Prototype:

int BTPSAPI **HCI_Exit_Sniff_Mode**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etMode_Change_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Park_Mode

This command places a connection into Park Mode.

Prototype:

int BTPSAPI **HCI_Park_Mode**(unsigned int BluetoothStackID, Word_t Connection_Handle, Word_t Beacon_Max_Interval, Word_t Beacon_Min_Interval, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Beacon_Max_Interval	Maximum time between consecutive beacons. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).
Beacon_Min_Interval	Minimum time between consecutive beacons. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etMode_Change_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Exit_Park_Mode

This command terminates Park Mode for a connection.

Prototype:

int BTPSAPI **HCI_Exit_Park_Mode**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etMode_Change_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_QoS_Setup

This command specifies the Quality of Service parameters for a connection.

Prototype:

int BTPSAPI **HCI_QoS_Setup**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t Flags, Byte_t Service_Type, DWord_t Token_Rate, DWord_t Peak_Bandwidth, DWord_t Latency, DWord_t Delay_Variation, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Flags	(reserved for future use)
Service_Type	The type of service to establish. Possible values are: HCI_QOS_SERVICE_TYPE_NO_TRAFFIC HCI_QOS_SERVICE_TYPE_BEST_EFFORT HCI_QOS_SERVICE_TYPE_GUARANTEED
Token_Rate	Token Rate in bytes per second.
Peak_Bandwidth	Peak Bandwidth in bytes per second.
Latency	Latency in microseconds.
Delay_Variation	Delay Variation in microseconds.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etQoS_Setup_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Role_Discovery

This command determines what role a device is playing in a connection.

Prototype:

int BTPSAPI **HCI_Role_Discovery**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult,
Byte_t *Current_RoleResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the role discovery was done.
Current_RoleResult	The current role for the Connection_HandleResult. Possible values are: HCI_CURRENT_ROLE_MASTER HCI_CURRENT_ROLE_SLAVE

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Switch_Role

This command switches the current role a device is playing in a connection.

Prototype:

int BTPSAPI **HCI_Switch_Role**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR,
Byte_t Role, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
BD_ADDR	Address of the Bluetooth device.
Role	Role for this device to take on. Possible values are: HCI_CURRENT_ROLE_MASTER HCI_CURRENT_ROLE_SLAVE
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etRole_Change_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Link_Policy_Settings

This command reads the link policy settings for the specified connection.

Prototype:

int BTPSAPI **HCI_Read_Link_Policy_Settings**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult,
Word_t *Link_Policy_SettingsResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the policy reading was done.
Link_Policy_SettingsResult	The current link policy settings for the Connection_HandleResult connection. Bits in this word are a possible ORing of the following bit masks: HCI_LINK_POLICY_SETTINGS_DISABLE_ALL_LM_MODES HCI_LINK_POLICY_SETTINGS_ENABLE_MASTER_SLAVE_SWITCH HCI_LINK_POLICY_SETTINGS_ENABLE_HOLD_MODE HCI_LINK_POLICY_SETTINGS_ENABLE_SNIFF_MODE HCI_LINK_POLICY_SETTINGS_ENABLE_PARK_MODE

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Link_Policy_Settings

This command will write the link policy settings for the specified connection.

Prototype:

int BTPSAPI **HCI_Write_Link_Policy_Settings**(unsigned int BluetoothStackID, Word_t Connection_Handle, Word_t Link_Policy_Settings, Byte_t *StatusResult, Word_t *Connection_HandleResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Link_Policy_Settings	The link policy settings for the Connection_HandleResult connection to write. Bits in this word are a possible ORing of the following bit masks: HCI_LINK_POLICY_SETTINGS_DISABLE_ALL_ LM_MODES HCI_LINK_POLICY_SETTINGS_ENABLE_MASTER_ SLAVE_SWITCH HCI_LINK_POLICY_SETTINGS_ENABLE_HOLD_MODE HCI_LINK_POLICY_SETTINGS_ENABLE_SNIFF_MODE HCI_LINK_POLICY_SETTINGS_ENABLE_PARK_MODE
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the policy writing was done.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Default_Link_Policy_Settings

This command will read the Default Link Policy settings for all new connections.

Prototype:

int BTPSAPI **HCI_Read_Default_Link_Policy_Settings**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Word_t *Link_Policy_SettingsResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.

Link_Policy_SettingsResult	The current default link policy settings for all new connections. Bits in this word are a Logical OR'ing of the following bit values: HCI_LINK_POLICY_SETTINGS_ENABLE_MASTER_ SLAVE_SWITCH HCI_LINK_POLICY_SETTINGS_ENABLE_HOLD_MODE HCI_LINK_POLICY_SETTINGS_ENABLE_SNIFF_MODE HCI_LINK_POLICY_SETTINGS_ENABLE_PARK_MODE

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Default_Link_Policy_Settings

This command will write the Default Link Policy configuration value. The Default_Link_Policy_Settings parameter determines the initial value of the Link_Policy_Settings for all new connections..

Prototype:

int BTPSAPI **HCI_Write_Default_Link_Policy_Settings**(unsigned int BluetoothStackID, Word_t Link_Policy_Settings, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Link_Policy_Settings	The updated default link policy settings for all new connections. Bits in this word are a Logical OR'ing of the following bit values: HCI_LINK_POLICY_SETTINGS_ENABLE_MASTER_ SLAVE_SWITCH HCI_LINK_POLICY_SETTINGS_ENABLE_HOLD_MODE HCI_LINK_POLICY_SETTINGS_ENABLE_SNIFF_MODE HCI_LINK_POLICY_SETTINGS_ENABLE_PARK_MODE

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Flow_Specification

This command is used to specify the flow parameters for the traffic carried over the ACL connection identified by the specified connection handle.

Prototype:

int BTPSAPI **HCI_Flow_Specification**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t Flags, Byte_t Flow_Direction, Byte_t Service_Type,
DWord_t Token_Rate, DWord_t Token_Bucket_Size, DWord_t Peak_Bandwidth,
DWord_t Access_Latency, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Flags	Reserved for future use and shall be set to 0 and ignored by the receiver.
Flow_Direction	Determines if the parameters refer to the outgoing or incoming traffic of the ACL link. Possible values are: HCI_FLOW_SPECIFICATION_FLOW_DIRECTION_OUTGOING_FLOW HCI_FLOW_SPECIFICATION_FLOW_DIRECTION_INCOMING_FLOW
Service_Type	Indicates the level of service required. Possible values are:

HCI_FLOW_SPECIFICATION_SERVICE_TYPE_NO_TRAFFIC
HCI_FLOW_SPECIFICATION_SERVICE_TYPE_BEST_EFFORT
HCI_FLOW_SPECIFICATION_SERVICE_TYPE_GUARANTEED

Token_Rate	The average data rate with which the application transmits data.
Token_Bucket_Size	Specifies a limit on the 'burstiness' with which the application may transmit data.
Peak_Bandwidth	Limits how fast packets from applications may be sent back-to-back.
Access_Latency	The maximum acceptable delay of an L2CAP packet to the air-interface.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etFlow_Specification_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Sniff_Subrating

This function issues the HCI_Sniff_Subrating Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Sniff_Subrating**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, Word_t Maximum_Latency,
    Word_t Minimum_Remote_Timeout, Word_t Minimum_Local_Timeout,
    Byte_t *StatusResult, Word_t *Connection_HandleResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Maximum_Latency	Used to calculate the maximum sniff subrate that the remote device may use. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFE)
Minimum_Remote_Timeout	Minimum base sniff subrate timeout that the remote device may use. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFE)
Minimum_Local_Timeout	Minimum base sniff subrate timeout that the local device may use. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFE)
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Connection_HandleResult	If function returns zero (success) this variable will contain the Connection_Handle Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.4 Host Controller & Baseband Commands

These commands provide access and control over parts of the Bluetooth hardware. The commands available are listed in the table below.

Command	Description
HCI_Set_Event_Mask	Control which events are generated by the HCI for the Host.
HCI_Reset	Reset the Bluetooth Host Controller, Link Manager, and the radio module.
HCI_Set_Event_Filter	Specify different event filters.

Command	Description
HCI_Flush	Discard all data that is currently pending for transmission in the Host Controller for the specified connection handle, even if there currently are chunks of data that belong to more than one L2CAP packet in the Host Controller.
HCI_Read_PIN_Type	Read whether the Host supports variable PIN or only fixed PINs.
HCI_Write_PIN_Type	Specify whether the Host supports variable PIN or only fixed PINs.
HCI_Create_New_Unit_Key	Create a new unit key.
HCI_Read_Stored_Link_Key	Read one or more link keys stored in the Bluetooth Host Controller.
HCI_Write_Stored_Link_Key	Write one or more link keys to be stored in the Bluetooth Host Controller.
HCI_Delete_Stored_Link_Key	Remove one or more of the link keys stored in the Bluetooth Host Controller.
HCI_Change_Local_Name	Modify the user-friendly name for the Bluetooth device.
HCI_Read_Local_Name	Read the stored user-friendly name for the Bluetooth device.
HCI_Read_Connection_Accept_Timeout	Read the Connection_Accept_Timeout configuration parameter.
HCI_Write_Connection_Accept_Timeout	Write the Connection_Accept_Timeout configuration parameter
HCI_Read_Page_Timeout	Read the Page_Reply_Timeout configuration parameter.
HCI_Write_Page_Timeout	Write the Page_Reply_Timeout configuration parameter.
HCI_Read_Scan_Enable	Read the the Scan_Enable configuration parameter.
HCI_Write_Scan_Enable	Write the Scan_Enable configuration parameter.
HCI_Read_Page_Scan_Activity	Read the Page_Scan_Interval and

Command	Description
	Page_Scan_Window configuration parameters.
HCI_Write_Page_Scan_Activity	Write the Page_Scan_Interval and Page_Scan_Window configuration parameters.
HCI_Read_Inquiry_Scan_Activity	Read the Inquiry_Scan_Interval and Inquiry_Scan_Window configuration parameters.
HCI_Write_Inquiry_Scan_Activity	Write the Inquiry_Scan_Interval and Inquiry_Scan_Window configuration parameters.
HCI_Read_Authentication_Enable	Read the Authentication_Enable parameter.
HCI_Write_Authentication_Enable	Write the Authentication_Enable parameter.
HCI_Read_Encryption_Mode	Read the value for the Encryption_Mode parameter.
HCI_Write_Encryption_Mode	Write the value for the Encryption_Mode parameter.
HCI_Read_Class_of_Device	Read the Class_of_Device parameter.
HCI_Write_Class_of_Device	Write the Class_of_Device parameter.
HCI_Read_Voice_Setting	Read the Voice_Setting parameter.
HCI_Write_Voice_Setting	Write the Voice_Setting parameter.
HCI_Read_Automatic_Flush_Timeout	Read the Flush_Timeout parameter for the specified connection.
HCI_Write_Automatic_Flush_Timeout	Write the Flush_Timeout parameter for the specified connection.
HCI_Read_Num_Broadcast_Retransmissions	Read the Number of Broadcast Retransmissions parameter for the device.
HCI_Write_Num_Broadcast_Retransmissions	Write the Number of Broadcast Retransmissions parameter for the device.
HCI_Read_Hold_Mode_Activity	Read the Hold_Mode_Activity parameter.
HCI_Write_Hold_Mode_Activity	Write the Hold_Mode_Activity parameter.

Command	Description
HCI_Read_Transmit_Power_Level	Read the Transmit_Power_Level parameter values for the specified connection.
HCI_Read_SCO_Flow_Control_Enable	Read the SCO_Flow_Control_Enable setting.
HCI_Write_SCO_Flow_Control_Enable	Write the SCO_Flow_Control_Enable setting.
HCI_Set_Host_Controller_To_Host_Flow_ Control	Turn flow control on or off in the direction from the Host Controller to the Host.
HCI_Host_Buffer_Size	Notify the Host Controller about the Host’s buffer sizes for ACL and SCO data. The Host Controller will segment the data to be transmitted from the Host Controller to the Host, so that data contained in HCI Data Packets will not exceed these sizes.
HCI_Host_Number_Of_Completed_Packets	Notify the Host Controller when the Host is ready to receive more HCI packets for a connection.
HCI_Read_Link_Supervision_Timeout	Read the Link_Supervision_Timeout parameter for the device.
HCI_Write Link_Supervision_Timeout	Write the Link_Supervision_Timeout parameter for the device.
HCI_Read_Number_Of_Supported_IAC	Read the value for the number of Inquiry Access Codes (IAC) that the local Bluetooth device can simultaneously listen for during an Inquiry Scan.
HCI_Read_Current_IAC_LAP	Read the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans.
HCI_Write_Current_IAC_LAP	Write the LAP(s) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans.

Command	Description
HCI_Read_Page_Scan_Period_Mode	Read the mandatory Page_Scan_Period_Mode of the local Bluetooth device.
HCI_Write_Page_Scan_Period_Mode	Write the mandatory Page_Scan_Period_Mode of the local Bluetooth device.
HCI_Read_Page_Scan_Mode	Read the default Page_Scan_Mode of the local Bluetooth device.
HCI_Write_Page_Scan_Mode	Write the default Page_Scan_Mode of the local Bluetooth device.
HCI_Set_AFH_Host_Channel_Classification	Set the AFH host channel classification.
HCI_ Read_Inquiry_Scan_Type	Read the inquiry scan type of the local device.
HCI_ Write_Inquiry_Scan_Type	Write the inquiry scan type to the local device.
HCI_ Read_Inquiry_Mode	Read the inquiry mode of the local device.
HCI_ Write_Inquiry_Mode	Write the inquiry mode to the local device.
HCI_ Read_Page_Scan_Type	Read the page scan type of the local device.
HCI_ Write_Page_Scan_Type	Write the page scan type to the local device.
HCI_Read_AFH_Channel_Assessment_Mode	Read the AFH channel assessment mode of the local device.
HCI_Write_AFH_Channel_Assessment_Mode	Write the AFH channel assessment mode to the local device.
HCI_Read_Extended_Inquiry_Response	Read the extended inquiry response for the local device
HCI_Write_Extended_Inquiry_Response	Write the extended inquiry response
HCI_Refresh_Encryption_Key	Refresh the encryption key
HCI_Read_Simple_Pairing_Mode	Read simple pairing mode
HCI_Write_Simple_Pairing_Mode	Write simple pairing mode
HCI_Read_Local_OOB_Data	Read local Out of Band (OOB) data
HCI_Read_Inquiry_Response_Transmit_Power_Level	Read inquiry response transmit power

Command	Description
	level
HCI_Write_Inquiry_Transmit_Power_Level	Write inquiry transmit power level
HCI_Send_Keypress_Notification	Send keypress notification
HCI_Read_Default_Erroneous_Data_Reporting	Read default erroneous data reporting
HCI_Write_Default_Erroneous_Data_Reporting	Write default erroneous data reporting
HCI_Enhanced_Flush	Perform the enhanced flush function
HCI_Read_Logical_Link_Accept_Timeout	Reads the Logical_Link_Accept_Timeout configuration parameter.
HCI_Write_Logical_Link_Accept_Timeout	Writes the Logical_Link_Accept_Timeout configuration parameter.
HCI_Set_Event_Mask_Page_2	Used to control which events are generated by the HCI for the host.
HCI_Read_Location_Data	Reads stored knowledge of environment or regulations in use.
HCI_Write_Location_Data	Writes information of environment or regulations.
HCI_Read_Flow_Control_Mode	Reads value of Flow_Control_Mode configuration parameter.
HCI_Write_Flow_Control_Mode	Writes the value of Flow_Control_Mode configuration parameter.
HCI_Read_Enhanced_Transmit_Power_Level	Reads the values of the Enhanced_Transmit_Power_Level configuration parameters.
HCI_Read_Best_Effort_Flush_Timeout	Reads the Best Effor Flush Timeout for a specified Logical Link.
HCI_Write_Best_Effort_Flush_Timeout	Writes the Best Effor Flush Timeout for a specified Logical Link.
HCI_Short_Range_Mode	Configures Short Range Mode parameter for specified physical link.
HCI_Read_LE_Host_Supported	Reads currently configured value of LE Host support from LMP/LE features
HCI_Write_LE_Host_Supported	Writes LE Host support to LMP/LE

Command	Description
	features

HCI_Set_Event_Mask

This command controls which events are generated by the HCI layer.

Note:

This function uses MACRO's to set/clear bits in an event mask structure. Constants are provided that specify the actual bit numbers that are to be used with the MACRO (see below).

Prototype:

int BTPSAPI **HCI_Set_Event_Mask**(unsigned int BluetoothStackID,
Event_Mask_t Event_Mask, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Event_Mask	Eight-byte bit mask of events to allow. Setting a bit to one enables the corresponding event. The bit mask is constructed via the following API macros:
	SET_EVENT_MASK_BIT(Mask, BitNumber)
	RESET_EVENT_MASK_BIT(Mask, BitNumber)
	TEST_EVENT_MASK_BIT(Mask, BitNumber)
	HCI_ENABLE_ALL_HCI_EVENTS_IN_EVENT_MASK(Mask)
	HCI_DISABLE_ALL_HCI_EVENTS_IN_EVENT_MASK(Mask)
	The bit number constants defined in the API for use with these macros are:

Bluetooth Version 1.1

HCI_EVENT_MASK_INQUIRY_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_INQUIRY_RESULT_BIT_NUMBER
HCI_EVENT_MASK_CONNECTION_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_CONNECTION_REQUEST_BIT_NUMBER
HCI_EVENT_MASK_DISCONNECTION_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_AUTHENTICAITION_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_REMOTE_NAME_REQUEST_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_ENCRYPTION_CHANGE_BIT_NUMBER
HCI_EVENT_MASK_CHANGE_CONNECTION_LINK_KEY_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_MASTER_LINK_KEY_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_READ_REMOTE_SUPPORTED_FEATURES_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_READ_REMOTE_VERSION_INFORMATION_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_QOS_SETUP_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_COMMAND_COMPLETE_BIT_NUMBER

HCI_EVENT_MASK_STATUS_COMMAND_BIT_NUMBER
HCI_EVENT_MASK_HARDWARE_ERROR_BIT_NUMBER
HCI_EVENT_MASK_FLUSH_OCCURRED_BIT_NUMBER
HCI_EVENT_MASK_ROLE_CHANGE_BIT_NUMBER
HCI_EVENT_MASK_NUMBER_OF_COMPLETED_PACKETS_BIT_NUMBER
HCI_EVENT_MASK_MODE_CHANGE_BIT_NUMBER
HCI_EVENT_MASK_RETURN_LINK_KEYS_BIT_NUMBER
HCI_EVENT_MASK_PIN_CODE_REQUEST_BIT_NUMBER
HCI_EVENT_MASK_LINK_KEY_REQUEST_BIT_NUMBER
HCI_EVENT_MASK_LINK_KEY_NOTIFICATION_BIT_NUMBER
HCI_EVENT_MASK_LOOPBACK_COMMAND_BIT_NUMBER
HCI_EVENT_MASK_DATA_BUFFER_OVERFLOW_BIT_NUMBER
HCI_EVENT_MASK_MAX_SLOTS_CHANGE_BIT_NUMBER
HCI_EVENT_MASK_READ_CLOCK_OFFSET_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_CONNECTION_PACKET_TYPE_CHANGED_BIT_NUMBER
HCI_EVENT_MASK_QOS_VIOLATION_BIT_NUMBER
HCI_EVENT_MASK_PAGE_SCAN_MODE_CHANGE_BIT_NUMBER
HCI_EVENT_MASK_PAGE_SCAN_REPETITION_MODE_CHANGE_BIT_NUMBER

Bluetooth Version 1.2

HCI_EVENT_MASK_FLOW_SPECIFICATION_BIT_NUMBER
HCI_EVENT_MASK_INQUIRY_RESULT_WITH_RSSI_BIT_NUMBER
HCI_EVENT_MASK_READ_REMOTE_EXTENDED_FEATURES_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_SYNCHRONOUS_CONNECTION_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_SYNCHRONOUS_CONNECTION_CHANGED_BIT_NUMBER

Bluetooth Version 2.1

HCI_EVENT_MASK_SNIFF_SUBRATING_BIT_NUMBER
HCI_EVENT_MASK_EXTENDED_INQUIRY_RESULT_BIT_NUMBER
HCI_EVENT_MASK_ENCRYPTION_REFRESH_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_IO_CAPABILITY_REQUEST_BIT_NUMBER
HCI_EVENT_MASK_IO_CAPABILITY_REQUEST_REPLY_BIT_NUMBER
HCI_EVENT_MASK_USER_CONFIRMATION_REQUEST_BIT_NUMBER
HCI_EVENT_MASK_USER_PASSKEY_REQUEST_BIT_NUMBER
HCI_EVENT_MASK_REMOTE_OOB_DATA_REQUEST_BIT_NUMBER
HCI_EVENT_MASK_SIMPLE_PAIRING_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_LINK_SUPERVISION_TIMEOUT_CHANGED_BIT_NUMBER
HCI_EVENT_MASK_ENHANCED_FLUSH_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_USER_PASSKEY_NOTIFICATION_BIT_NUMBER
HCI_EVENT_MASK_USER_KEYPRESS_NOTIFICATION_BIT_NUMBER
HCI_EVENT_MASK_REMOTE_HOST_SUPPORTED_FEATURES_NOTIFICATION_
BIT_NUMBER

Bluetooth Version 4.0

HCI_EVENT_MASK_LE_META_BIT_NUMBER

In addition, to aid in quickly enabling all events, the API provides the following macro which enables all events:

HCI_ENABLE_ALL_HCI_EVENTS_IN_EVENT_MASK(Mask)

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Reset

This command resets the Bluetooth Host Controller, Link Manager, and the radio module. The current operational state and all queued packets will be lost. After the reset is completed, the Bluetooth device will enter standby mode, reverting to the default values for parameters which have defaults.

Prototype:

int BTPSAPI **HCI_Reset**(unsigned int BluetoothStackID, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR
BTPS_ERROR_VS_HCI_ERROR

Possible Events:

etDevice_Reset_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Set_Event_Filter

This command allows the Host to specify the various conditions under which each particular event is returned to the Host. This command may be called multiple times to set multiple filters for the same event, and can also be used to clear all filters from an events or from all events. Only a few of the HCI events allow filters, as specified below.

Prototype:

int BTPSAPI **HCI_Set_Event_Filter**(unsigned int BluetoothStackID, Byte_t Filter_Type, Byte_t Filter_Condition_Type, Condition_t Condition, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Filter_Type	The type of filter that the condition is being set for. Possible values are: HCI_FILTER_TYPE_CLEAR HCI_FILTER_TYPE_INQUIRY_RESULT HCI_FILTER_TYPE_CONNECTION_SETUP Actually, the first value is not a true filter type, but a flag to indicate that **all** event filters are to be cleared.
Filter_Condition_Type	The filter condition to be set for the specified Filter_Type. This field is ignored for the Clear type. For the Inquiry Result type filter, the possible values are (the first type, clears the others): HCI_FILTER_CONDITION_TYPE_RESULT_FILTER_NEW_DEVICE HCI_FILTER_CONDITION_TYPE_RESULT_FILTER_CLASS_OF_DEVICE HCI_FILTER_CONDITION_TYPE_RESULT_FILTER_BD_ADDR For the Connection Setup type filter, the possible values are (the first type, clears the others): HCI_FILTER_CONDITION_TYPE_CONNECTION_SETUP_NEW_DEVICE HCI_FILTER_CONDITION_TYPE_CONNECTION_SETUP_CLASS_OF_DEVICE HCI_FILTER_CONDITION_TYPE_CONNECTION_SETUP_BD_ADDR

Condition	This is a overlayed structure which permits specifying the filter condition for the later two Condition Types for each Filter Type. This structure is declared as follows:

```
typedef struct
{
  union
  {
    Inquiry_Result_Filter_Type_Class_of_Device_Condition_t
             Inquiry_Result_Filter_Type_Class_of_Device_Condition;
    Inquiry_Result_Filter_Type_BD_ADDR_Condition_t
             Inquiry_Result_Filter_Type_BD_ADDR_Condition;

    Connection_Setup_Filter_Type_All_Devices_Condition_t
             Connection_Setup_Filter_Type_All_Devices_Condition;
    Connection_Setup_Filter_Type_Class_of_Device_Condition_t
             Connection_Setup_Filter_Type_Class_of_Device_Condition;
    Connection_Setup_Filter_Type_BD_ADDR_Condition_t
             Connection_Setup_Filter_Type_BD_ADDR_Condition;

    Raw_Condition_Bytes_t
             Raw_Condition_Bytes;
  } Condition;
} Condition_t;
```

The various structures used in the Condition_t are defined below. For Inquiry Result Filter Type setting:

```
typedef struct
{
  Class_of_Device_t  Class_of_Device;
  Class_of_Device_t  Class_of_Device_Mask;
} Inquiry_Result_Filter_Type_Class_of_Device_Condition_t;
```

(see HCI_Read_Class_of_Device command for info on Class_of_Device.)

For Inquiry Result BD_ADDR setting:

```
typedef struct
{
  BD_ADDR_t  BD_ADDR;
} Inquiry_Result_Filter_Type_BD_ADDR_Condition_t;
```

For Connection Setup All Devices setting:

```
typedef struct
{
  Byte_t  Auto_Accept_Flag;
} Connection_Setup_Filter_Type_All_Devices_Condition_t;
```

For Connection Setup Class of Device setting:

```
typedef struct
{
  Class_of_Device_t  Class_of_Device;
  Class_of_Device_t  Class_of_Device_Mask;
  Byte_t  Auto_Accept_Flag;
} Connection_Setup_Filter_Type_Class_of_Device_Condition_t;
```
(see HCI_Read_Class_of_Device command for info on Class_of_Device.)

For Connection Setup BD_ADDR setting:

```
typedef struct
{
  BD_ADDR_t  BD_ADDR;
  Byte_t  Auto_Accept_Flag;
} Connection_Setup_Filter_Type_BD_ADDR_Condition_t;
```

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Flush

This command discards all data that is currently pending for transmission in the Host Controller for the specified connection handle, even if there currently are chunks of data that belong to more than one L2CAP packet in the Host Controller.

Prototype:

int BTPSAPI **HCI_Flush**(unsigned int BluetoothStackID, Word_t Connection_Handle,
    Byte_t *StatusResult, Word_t *Connection_HandleResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etFlush_Occurred_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_PIN_Type

This command reads whether the Link Manager thinks the Host supports variable PIN or only fixed PINs.

Prototype:

int BTPSAPI **HCI_Read_PIN_Type**(unsigned int BluetoothStackID, Byte_t *StatusResult, Byte_t *PIN_TypeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
PIN_TypeResult	The type of PIN supported by the Host. Possible values are: HCI_PIN_TYPE_VARIABLE HCI_PIN_TYPE_FIXED

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_PIN_Type

This command tells the Link Manager what type of PINs are supported by the Host.

Prototype:

int BTPSAPI **HCI_Write_PIN_Type**(unsigned int BluetoothStackID, Byte_t PIN_Type, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
PIN_TypeResult	The type of PIN supported by the Host. Possible values are: HCI_PIN_TYPE_VARIABLE HCI_PIN_TYPE_FIXED
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Create_New_Unit_Key

This command causes the Bluetooth hardware to generate a new (random) unit key. This key only applies to new connections, not any existing ones.

Prototype:

int BTPSAPI **HCI_Create_New_Unit_Key**(unsigned int BluetoothStackID,
Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Stored_Link_Key

This command initiates a read of one or more Link Keys stored in the Host Controller. The actual Link Keys will be returned in events.

Prototype:

int BTPSAPI **HCI_Read_Stored_Link_Key**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t Read_All_Flag, Byte_t *StatusResult,
Word_t *Max_Num_KeysResult, Word_t *Num_Keys_ReadResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the Bluetooth device.
Read_All_Flag	Flag to indicate whether only the Link Key for the specified Bluetooth device should be returned or all Link Keys. Possible values are: HCI_READ_LINK_KEY_BD_ADDR HCI_READ_LINK_KEY_ALL_STORED

StatusResult	Returned HCI status code.
Max_Num_KeysResult	Maximum number of Link Keys that can be stored in the Host Controller.
Num_Keys_ReadResult	Number of Link Keys being read. The Link Keys will be returned in this number of etReturn_Link_Keys_Event events.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

etReturn_Link_Keys_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Stored_Link_Key

This command writes one or more Link Keys to the Host Controller.

Prototype:

int BTPSAPI **HCI_Write_Stored_Link_Key**(unsigned int BluetoothStackID,
Byte_t Num_Keys_To_Write, HCI_Stored_Link_Key_Info_t HCI_Stored_Link_Key_Info[],
Byte_t *StatusResult, Byte_t *Num_Keys_Written)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Num_Keys_To_Write	Number of Keys in the array to be written.
HCI_Stored_Link_Key_Info	Array of structures which pair up Bluetooth devices and Link Keys. This structure is defined as follows:

```
typedef struct
{
  BD_ADDR_t  BD_ADDR;
  Link_Key_t  Link_Key;
} HCI_Stored_Link_Key_Info_t
```

StatusResult	Returned HCI status code.

Num_Keys_Written	Number of Link Keys actually written.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Delete_Stored_Link_Key

This command removes one or more Link Keys that are stored in the Host Controller.

Prototype:

int BTPSAPI **HCI_Delete_Stored_Link_Key**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t Delete_All_Flag, Byte_t *StatusResult,
Word_t *Num_Keys_DeletedResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the Bluetooth device. This field is ignored, if the Delete_All_Flag is set to indicate deleting all.
Delete_All_Flag	A flag to indicate whether all the stored Link Keys should be deleted or not. Possible values are: HCI_DELETE_LINK_KEY_BD_ADDR HCI_DELETE_LINK_KEY_ALL_STORED
StatusResult	Returned HCI status code.
Num_Keys_DeletedResult	Returned number of Link Keys deleted.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Change_Local_Name

This command is used to change the user-friendly name of the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Change_Local_Name**(unsigned int BluetoothStackID, char *Name, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Name	Pointer to null-terminated name (up to 249 bytes including the NULL character)
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Local_Name

The command reads back the user-friendly name of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Local_Name**(unsigned int BluetoothStackID, Byte_t *StatusResult, char *NameResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
NameResult	Returned NULL-terminated character string, up to 249 bytes.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Connection_Accept_Timeout

This command reads the Connection_Accept_Timeout configuration parameter, which is the parameter that allows the Bluetooth hardware to automatically deny a connection request after a specified time period has occurred and the new connection is not accepted.

Prototype:

int BTPSAPI **HCI_Read_Connection_Accept_Timeout**(unsigned int BluetoothStackID, Byte_t *StatusResult, Word_t *Conn_Accept_TimeoutResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Conn_Accept_TimeoutResult	Current timeout value. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Connection_Accept_Timeout

This command writes the Connection_Accept_Timeout configuration parameter, which is the parameter that allows the Bluetooth hardware to automatically deny a connection request after a specified time period has occurred and the new connection is not accepted.

Prototype:

int BTPSAPI **HCI_Write_Connection_Accept_Timeout**(unsigned int BluetoothStackID,
Word_t Conn_Accept_Timeout, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Conn_Accept_Timeout	New Timeout value. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Page_Timeout

This command reads the Page_Timeout configuration parameter, which defines the maximum time the local Link Manager will wait for a baseband page response from the remote device. If this time expires without a response, the connection attempt fails.

Prototype:

int BTPSAPI **HCI_Read_Page_Timeout**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Word_t *Page_TimeoutResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Page_TimeoutResult	Current timeout value. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Page_Timeout

This command writes the Page_Timeout configuration parameter, which defines the maximum time the local Link Manager will wait for a baseband page response from the remote device. If this time expires without a response, the connection attempt fails.

Prototype:

int BTPSAPI **HCI_Write_Page_Timeout**(unsigned int BluetoothStackID,
Word_t Page_Timeout, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Page_Timeout	New timeout value. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Scan_Enable

This command reads the Scan_Enable parameter, which controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices.

Prototype:

int BTPSAPI **HCI_Read_Scan_Enable**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Scan_EnableResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Scan_EnableResult	Current setting of this parameter. Possible values are: HCI_SCAN_ENABLE_NO_SCANS_ENABLED

HCI_SCAN_ENABLE_INQUIRY_SCAN_ENABLED_
PAGE_SCAN_DISABLED
HCI_SCAN_ENABLE_INQUIRY_SCAN_DISABLED_
PAGE_SCAN_ENABLED
HCI_SCAN_ENABLE_INQUIRY_SCAN_ENABLED_
PAGE_SCAN_ENABLED

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Scan_Enable

This command writes the Scan_Enable parameter, which controls whether or not the Bluetooth device will periodically scan for page attempts and/or inquiry requests from other Bluetooth devices.

Prototype:

int BTPSAPI **HCI_Write_Scan_Enable**(unsigned int BluetoothStackID,
Byte_t Scan_Enable, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Scan_Enable	Desired setting of this parameter. Possible values are: HCI_SCAN_ENABLE_NO_SCANS_ENABLED HCI_SCAN_ENABLE_INQUIRY_SCAN_ENABLED_ PAGE_SCAN_DISABLED HCI_SCAN_ENABLE_INQUIRY_SCAN_DISABLED_ PAGE_SCAN_ENABLED HCI_SCAN_ENABLE_INQUIRY_SCAN_ENABLED_ PAGE_SCAN_ENABLED
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Page_Scan_Activity

This command reads the Page_Scan_Activity configuration parameters.

Prototype:

int BTPSAPI **HCI_Read_Page_Scan_Activity**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Word_t *Page_Scan_IntervalResult,
Word_t *Page_Scan_WindowResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Page_Scan_IntervalResult	Amount of time between consecutive page scans. Values are number of baseband slots (0.625 msec), with a range of 11.25 msec (0x0012) to 2560 msec (0x1000).
Page_Scan_WindowResult	Amount of time for the duration of the page scan. This parameter will be less than or equal to the Page_Scan_Interval. Values are number of baseband slots (0.625 msec), with a range of 11.25 msec (0x0012) to 2560 msec (0x1000).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR

BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Page_Scan_Activity

This command writes the Page_Scan_Activity configuration parameters.

Prototype:

int BTPSAPI **HCI_Write_Page_Scan_Activity**(unsigned int BluetoothStackID,
Word_t Page_Scan_Interval, Word_t Page_Scan_Window, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Page_Scan_Interval	Defines the amount of time between consecutive page scans. Values are number of baseband slots (0.625 msec), with a range of 11.25 msec (0x0012) to 2560 msec (0x1000). Default value is 1.28 sec (0x0800).
Page_Scan_Window	Defines the amount of time for the duration of the page scan. This parameter must be less than or equal to the Page_Scan_Interval. Values are number of baseband slots (0.625 msec), with a range of 11.25 msec (0x0012) to 2560 msec (0x1000). Default value is 11.25 msec (0x0012).
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Inquiry_Scan_Activity

This command reads the Inquiry_Scan_Activity configuration parameters.

Prototype:

int BTPSAPI **HCI_Read_Inquiry_Scan_Activity**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Word_t *Inquiry_Scan_IntervalResult,
    Word_t *Inquiry_Scan_WindowResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Inquiry_Scan_IntervalResult	Amount of time between consecutive inquiry scans. Values are number of baseband slots (0.625 msec), with a range of 11.25 msec (0x0012) to 2560 msec (0x1000).
Inquiry_Scan_WindowResult	Amount of time for the duration of the inquiry scan. This parameter will be less than or equal to the Inquiry_Scan_Interval. Values are number of baseband slots (0.625 msec), with a range of 11.25 msec (0x0012) to 2560 msec (0x1000).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Inquiry_Scan_Activity

This command writea the Inquiry_Scan_Activity configuration parameters.

Prototype:

int BTPSAPI **HCI_Write_Inquiry_Scan_Activity**(unsigned int BluetoothStackID, Word_t Inquiry_Scan_Interval, Word_t Inquiry_Scan_Window, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Inquiry_Scan_Interval	Defines the amount of time between consecutive inquiry scans. Values are number of baseband slots (0.625 msec), with a range of 11.25 msec (0x0012) to 2560 msec (0x1000). Default value is 1.28 sec (0x0800).
Inquiry_Scan_Window	Defines the amount of time for the duration of the inquiry scan. This parameter must be less than or equal to the Inquiry_Scan_Interval. Values are number of baseband slots (0.625 msec), with a range of 11.25 msec (0x0012) to 2560 msec (0x1000). Default value is 11.25 msec (0x0012).
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Authentication_Enable

This command reada the Authentication_Enable parameter, which controls if the local device requires to authenticate the remote device at connection setup. At connection setup, only the device(s) with the Authentication_Enable parameter set to enabled will try to authenticate the other device.

Prototype:

int BTPSAPI **HCI_Read_Authentication_Enable**(unsigned int BluetoothStackID, Byte_t *StatusResult, Byte_t *Authentication_EnableResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Authentication_EnableResult	Current value of this parameter. Possible values are: HCI_AUTHENTICATION_ENABLE_AUTHENTICATION_ DISABLED HCI_AUTHENTICATION_ENABLE_AUTHENTICATION_ ENABLED_ALL_CONNECTIONS

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Authentication_Enable

This command writea the Authentication_Enable parameter, which controls if the local device requires to authenticate the remote device at connection setup. At connection setup, only the device(s) with the Authentication_Enable parameter set to enabled will try to authenticate the other device. Note, changing this parameter will only affect future connections, not any existing connections.

Prototype:

int BTPSAPI **HCI_Write_Authentication_Enable**(unsigned int BluetoothStackID, Byte_t Authentication_Enable, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Authentication_Enable	Desired value of this parameter. Possible values are: HCI_AUTHENTICATION_ENABLE_AUTHENTICATION_ DISABLED

	HCI_AUTHENTICATION_ENABLE_AUTHENTICATION_ ENABLED_ALL_CONNECTIONS
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Encryption_Mode

This command reads the Encryption_Mode parameter, which controls if the local device requires encryption to the remote device at connection setup. At connection setup, only the device(s) with the Authentication_Enable parameter enabled and Encryption_Mode parameter enabled will try to encrypt the connection to the other device.

Prototype:

int BTPSAPI **HCI_Read_Encryption_Mode**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Byte_t *Encryption_ModeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Encryption_ModeResult	Current value of this parameter. Possible values are: HCI_ENCRYPTION_MODE_ENCRYPTION_DISABLED HCI_ENCRYPTION_MODE_ENCRYPTION_POINT_TO_ POINT_PACKETS HCI_ENCRYPTION_MODE_ENCRYPTION_POINT_TO_ POINT_BROADCAST_PACKETS

Return:

Zero if successful.

An error code if negative; one of the following values:

	BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID BTPS_ERROR_INVALID_PARAMETER BTPS_ERROR_INSUFFICIENT_RESOURCES BTPS_ERROR_HCI_DRIVER_ERROR BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Encryption_Mode

This command writes the Encryption_Mode parameter, which controls if the local device requires encryption to the remote device at connection setup. At connection setup, only the device(s) with the Authentication_Enable parameter enabled and Encryption_Mode parameter enabled will try to encrypt the connection to the other device. Note, changing this parameter will only affect future connections, not any existing connections.

Prototype:

int BTPSAPI **HCI_Write_Encryption_Mode**(unsigned int BluetoothStackID,
    Byte_t Encryption_Mode, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Encryption_Mode	Desired value of this parameter. Possible values are: HCI_ENCRYPTION_MODE_ENCRYPTION_DISABLED HCI_ENCRYPTION_MODE_ENCRYPTION_POINT_TO_ POINT_PACKETS HCI_ENCRYPTION_MODE_ENCRYPTION_POINT_TO_ POINT_BROADCAST_PACKETS
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

	BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID BTPS_ERROR_INVALID_PARAMETER BTPS_ERROR_INSUFFICIENT_RESOURCES BTPS_ERROR_HCI_DRIVER_ERROR BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Class_of_Device

This command reads the Class_of_Device parameter, which indicates the capabilities of the local device to other devices.

Prototype:

int BTPSAPI **HCI_Read_Class_of_Device**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Class_of_Device_t *Class_of_DeviceResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Class_of_DeviceResult	Bit mask list of features that determine the class of device for this Bluetooth device. The class is divided into the following fields: Format Type; Major Service Class; Major Device Class; Minor Device Class. The bit number constants defined for each field are listed below. These bit numbers can be used with the following macros to set the fields in a CoD (Class of Device bit list): GET_CLASS_OF_DEVICE_FORMAT_TYPE(CoD); SET_CLASS_OF_DEVICE_FORMAT_TYPE (CoD, bitnumb); GET_MAJOR_SERVICE_CLASS(CoD); SET_MAJOR_SERVICE_CLASS(CoD, bitnumb); GET_MAJOR_DEVICE_CLASS(CoD); SET_MAJOR_DEVICE_CLASS(CoD, bitnumb); GET_MINOR_DEVICE_CLASS(CoD); SET_MINOR_DEVICE_CLASS(CoD, bitnumb). Possible values for Format Type bit numbers are: HCI_LMP_CLASS_OF_DEVICE_FORMAT_TYPE_1. Possible values for Major Service Class bit numbers are: HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_ LIMITED_DISCOVER_MODE_BIT

HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_
POSITIONING_BIT
HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_
NETWORKING_BIT
HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_
RENDERING_BIT
HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_
CAPTURING_BIT
HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_
OBJECT_TRANSFER_BIT
HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_
AUDIO_BIT
HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_
TELEPHONY_BIT0
HCI_LMP_CLASS_OF_DEVICE_SERVICE_CLASS_
INFORMATION_BIT

Possible values for Major Device Class bit numbers are:

HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_
MISCELLANEOUS
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_
COMPUTER
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_P
HONE
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_
LAN_ACCESS_POINT
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_
AUDIO
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_P
ERIPHERAL
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_I
MAGING
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_
WEARABLE
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_
TOY
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_
HEALTH
HCI_LMP_CLASS_OF_DEVICE_MAJOR_DEVICE_CLASS_
UNCLASSIFIED

The Minor Device Class bit numbers depend upon the Major Device Class. Possible values are:

For the Computer Major Device Class:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_
COMPUTER_UNCLASSIFIED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_
COMPUTER_DESKTOP
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_
COMPUTER_SERVER

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_COMPUTER_LAPTOP
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_COMPUTER_HANDHELD
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_COMPUTER_PALM_PC
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_COMPUTER_WEARABLE

For the Phone Major Device Class:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PHONE_UNCLASSIFIED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PHONE_CELLULAR
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PHONE_CORDLESS
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PHONE_SMARTPHONE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PHONE_WIRED_MODEM
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PHONE_VOICE_GATEWAY
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PHONE_VOICE_ISDN_ACCESS

For the LAN Access Point Major Class, the masks are:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_LOAD_FACTOR_MASK
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_SUB_FIELD_MASK

For the LAN Access Point Major Class, the bits for the Load Factor subfield are:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_FULLY_AVAILABLE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_1_17_UTILIZED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_17_33_UTILIZED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_33_50_UTILIZED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_50_67_UTILIZED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_67_83_UTILIZED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_83_99_UTILIZED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_NO_SERVICE

For the LAN Access Point Major Class, the bits for the reserved subfield are:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_LAN_SUB_FELD_UNCLASSIFIED

For the Audio/Video Major Class:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_UNCLASSIFIED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_HEADSET
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_HANDS_FREE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_MICROPHONE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_LOUD_SPEAKER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_HEADPHONES
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_PORTABLE_AUDIO
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_CAR_AUDIO
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_SET_TOP_BOX
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_HIFI_AUDIO_DEVICE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_VCR
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_VIDEO_CAMERA
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_CAMCORDER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_VIDEO_MONITOR
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_VIDEO_DISPLAY_LOUD_SPEAKER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_CONFERENCING
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_AUDIO_VIDEO_GAMING_TOY

For the Peripheral Major Class:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_UNCLASSIFIED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_JOYSTICK
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_GAMEPAD
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_REMOTE_CONTROL

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_SENSING_DEVICE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_DIGITIZER_TABLET
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_CARD_READER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_DIGITAL_PEN
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_HANDHELD_SCANNER_RFID
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_HANDHELD_GESTURAL_INPUT
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_KEYBOARD_MASK
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_POINTING_DEVICE_MASK
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_PERIPHERAL_KEYBOARD_POINTING_DEVICE_MASK

For the Imaging Major Class:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_IMAGING_UNCLASSIFIED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_IMAGING_DISPLAY_MASK
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_IMAGING_CAMERA_MASK
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_IMAGING_SCANNER_MASK
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_IMAGING_PRINTER_MASK

For the Wearable Major Class:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_WEARABLE_UNCLASSIFIED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_WEARABLE_WRIST_WATCH
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_WEARABLE_PAGER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_WEARABLE_JACKET
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_WEARABLE_HELMET
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_WEARABLE_GLASSES

For the Toy Major Class:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_TOY_UNCLASSIFIED

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_TOY_ROBOT
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_TOY_VEHICLE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_TOY_DOLL_ACTION_FIGURE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_TOY_CONTROLLER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_CLASS_TOY_GAME

For the Health Major Class:

HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_UNCLASSIFIED
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_BLOOD_PRESSURE_MONITOR
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_THERMOMETER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_WEIGHING_SCALE
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_GLUCOSE_METER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_PULSE_OXIMETER
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_HEART_PULSE_RATE_MONITOR
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_HEALTH_DATA_DISPLAY
HCI_LMP_CLASS_OF_DEVICE_MINOR_DEVICE_HEALTH_STEP_COUNTER

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Class_of_Device

This command writes the Class_of_Device parameter, which indicates the capabilities of the local device to other devices.

Prototype:

int BTPSAPI **HCI_Write_Class_of_Device**(unsigned int BluetoothStackID,
    Class_of_Device_t Class_of_Device, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Class_of_Device	Bit mask list of features that determine the class of device for this Bluetooth device. See the HCI_Read_Class_of_Device command for a complete listing of feature bits.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Voice_Setting

This command reads the Voice_Setting parameter, which controls all the various settings for voice connections. These settings apply to all voice connections, and cannot be set for individual voice connections.

Prototype:

int BTPSAPI **HCI_Read_Voice_Setting**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Word_t *Voice_SettingResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

StatusResult	Returned HCI status code.
Voice_SettingResult	Current voice settings. To test these bits, the result must first be masked with one of the following masks. Then the bits listed below can be tested for on the result of each of the five maskings.

HCI_VOICE_SETTING_INPUT_CODING_MASK
HCI_VOICE_SETTING_INPUT_DATA_FORMAT_MASK
HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_MASK
HCI_VOICE_SETTING_LINEAR_PCM_BIT_POS_NUM_MASK
HCI_VOICE_SETTING_AIR_CODING_FORMAT_MASK

The Input Coding bits to test for are:

HCI_VOICE_SETTING_INPUT_CODING_LINEAR
HCI_VOICE_SETTING_INPUT_CODING_U_LAW
HCI_VOICE_SETTING_INPUT_CODING_A_LAW

The Input Data Format bits to test for are:

HCI_VOICE_SETTING_INPUT_DATA_FORMAT_
1_COMPLEMENT
HCI_VOICE_SETTING_INPUT_DATA_FORMAT_
2_COMPLEMENT
HCI_VOICE_SETTING_INPUT_DATA_FORMAT_
SIGN_MAGNITUDE
HCI_VOICE_SETTING_INPUT_DATA_FORMAT_
UNSIGNED

The Input Sample Size bits to test for are:

HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_8_BIT
HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_16_BIT

The Linear PCM Bit Position Shift Value bits to test for are:

HCI_VOICE_SETTING_LINEAR_PCM_BIT_POS_
NUM_SHIFT_VALUE

The Air Coding Format bits to test for are:

Bluetooth Version 1.1

HCI_VOICE_SETTING_AIR_CODING_FORMAT_CVSD
HCI_VOICE_SETTING_AIR_CODING_FORMAT_U_LAW
HCI_VOICE_SETTING_AIR_CODING_FORMAT_A_LAW
HCI_VOICE_SETTING_AIR_CODING_FORMAT_NONE

Bluetooth Version 1.2

HCI_VOICE_SETTING_AIR_CODING_FORMAT_
TRANSPARENT_DATA

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Voice_Setting

This command writes the Voice_Setting parameter, which controls all the various settings for voice connections. These settings apply to all voice connections, and cannot be set for individual voice connections.

Prototype:

int BTPSAPI **HCI_Write_Voice_Setting**(unsigned int BluetoothStackID,
    Word_t Voice_Setting, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Voice_Setting	Desired voice settings. This is an ORing of bits in five categories as defined by the following masks: HCI_VOICE_SETTING_INPUT_CODING_MASK HCI_VOICE_SETTING_INPUT_DATA_FORMAT_MASK HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_MASK HCI_VOICE_SETTING_LINEAR_PCM_BIT_POS_NUM_MASK HCI_VOICE_SETTING_AIR_CODING_FORMAT_MASK The Input Coding bits which may be set are: HCI_VOICE_SETTING_INPUT_CODING_LINEAR HCI_VOICE_SETTING_INPUT_CODING_U_LAW HCI_VOICE_SETTING_INPUT_CODING_A_LAW The Input Data Format bits which may set are: HCI_VOICE_SETTING_INPUT_DATA_FORMAT_ 1_COMPLEMENT HCI_VOICE_SETTING_INPUT_DATA_FORMAT_ 2_COMPLEMENT HCI_VOICE_SETTING_INPUT_DATA_FORMAT_ SIGN_MAGNITUDE HCI_VOICE_SETTING_INPUT_DATA_FORMAT_ UNSIGNED

The Input Sample Size which may set are:

HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_8_BIT
HCI_VOICE_SETTING_INPUT_SAMPLE_SIZE_16_BIT

The Linear PCM Bit Position Shift Value bits which may be set are:

HCI_VOICE_SETTING_LINEAR_PCM_BIT_POS_
NUM_SHIFT_VALUE

The Air Coding Format bits which may be set are:

Bluetooth Version 1.1

HCI_VOICE_SETTING_AIR_CODING_FORMAT_CVSD
HCI_VOICE_SETTING_AIR_CODING_FORMAT_U_LAW
HCI_VOICE_SETTING_AIR_CODING_FORMAT_A_LAW
HCI_VOICE_SETTING_AIR_CODING_FORMAT_NONE

Bluetooth Version 1.2

HCI_VOICE_SETTING_AIR_CODING_FORMAT_
TRANSPARENT_DATA

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Automatic_Flush_Timeout

This command reads the Flush_Timeout parameter for the specified connection (ACL link only), which defines the amount of time before all chunks of the L2CAP packet, of which a baseband packet is currently being transmitted, are automatically flushed by the Host Controller.

Prototype:

int BTPSAPI **HCI_Read_Automatic_Flush_Timeout**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult, Word_t *Flush_TimeoutResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.
Flush_TimeoutResult	Current timeout value. A zero indicates that there is no timeout defined (or infinite timeout). Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to almost 1.28 sec (0x07FF).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Automatic_Flush_Timeout

This command writes the Flush_Timeout parameter for the specified connection (ACL link only), which defines the amount of time before all chunks of the L2CAP packet, of which a baseband packet is currently being transmitted, are automatically flushed by the Host Controller.

Prototype:

int BTPSAPI **HCI_Write_Automatic_Flush_Timeout**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Word_t Flush_Timeout, Byte_t *StatusResult,
Word_t *Connection_HandleResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Flush_Timeout	Current timeout value. A zero indicates that there is no timeout defined (or infinite timeout). Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to almost 1.28 sec (0x07FF).
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Num_Broadcast_Retransmissions

This command reads the device's Number of Broadcast Retransmissions parameter, which defines the number of times the device will retransmit a broadcast data packet to increase reliability.

Prototype:

int BTPSAPI **HCI_Read_Num_Broadcast_Retransmissions**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Num_Broadcast_RetranResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Num_Broadcast_RetranResult	Current parameter value in the range of 0x00 to 0xFF.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Num_Broadcast_Retransmissions

This command reads the device's Number of Broadcast Retransmissions parameter, which defines the number of times the device will retransmit a broadcast data packet to increase reliability. This parameter should be adjusted as the link quality measurement changes.

Prototype:

int BTPSAPI **HCI_Write_Num_Broadcast_Retransmissions**(
    unsigned int BluetoothStackID,
    Byte_t Num_Broadcast_Retran, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Num_Broadcast_Retran	Desired parameter value in the range of 0x00 to 0xFF.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Hold_Mode_Activity

This command reads the Hold_Mode_Activity parameter, which determines what activities should be suspended when the device is in hold mode.

Prototype:

int BTPSAPI **HCI_Read_Hold_Mode_Activity**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Hold_Mode_ActivityResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Hold_Mode_ActivityResult	Current parameter value. This is a bitwise ORing of the following defined bits: HCI_HOLD_MODE_ACTIVITY_MAINTAIN_CURRENT_POWER_STATE HCI_HOLD_MODE_ACTIVITY_SUSPEND_PAGE_STATE HCI_HOLD_MODE_ACTIVITY_SUSPEND_INQUIRY_STATE HCI_HOLD_MODE_ACTIVITY_SUSPEND_PERIODIC_INQUIRIES

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Hold_Mode_Activity

This command writes the Hold_Mode_Activity parameter, which determines what activities should be suspended when the device is in hold mode.

Prototype:

int BTPSAPI **HCI_Write_Hold_Mode_Activity**(unsigned int BluetoothStackID,
Byte_t Hold_Mode_Activity, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Hold_Mode_Activity	Current parameter value. This is a bitwise ORing of the following defined bits: HCI_HOLD_MODE_ACTIVITY_MAINTAIN_CURRENT_POWER_STATE HCI_HOLD_MODE_ACTIVITY_SUSPEND_PAGE_STATE HCI_HOLD_MODE_ACTIVITY_SUSPEND_INQUIRY_STATE HCI_HOLD_MODE_ACTIVITY_SUSPEND_PERIODIC_INQUIRIES
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Transmit_Power_Level

This command reads the Transmit_Power_Level parameters for the specified (ACL Link) Connection.

Prototype:

int BTPSAPI **HCI_Read_Transmit_Power_Level**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t Type, Byte_t *StatusResult,
Word_t *Connection_HandleResult, Byte_t *Transmit_Power_LevelResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Type	Flag to indicate whether to read the current or maximum power level. The possible values are: HCI_TRANSMIT_POWER_LEVEL_TYPE_CURRENT HCI_TRANSMIT_POWER_LEVEL_TYPE_MAXIMUM
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.
Transmit_Power_LevelResult	The current/maximum power level in the range of -30 dBm to +20 dBm.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_SCO_Flow_Control_Enable

This command reads the SCO_Flow_Control_Enable parameter, which enables and disables SCO flow control.

Prototype:

int BTPSAPI **HCI_Read_SCO_Flow_Control_Enable**(unsigned int BluetoothStackID, Byte_t *StatusResult, Byte_t *SCO_Flow_Control_EnableResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
SCO_Flow_Control_EnableResult	Current parameter setting. Possible values are: HCI_SCO_FLOW_CONTROL_DISABLE HCI_SCO_FLOW_CONTROL_ENABLE

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_SCO_Flow_Control_Enable

This command writes the SCO_Flow_Control_Enable parameter, which enables and disables SCO flow control. Note, changing this parameter will only affect future connections, not any existing connections.

Prototype:

int BTPSAPI **HCI_Write_SCO_Flow_Control_Enable**(unsigned int BluetoothStackID, Byte_t SCO_Flow_Control_Enable, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

SCO_Flow_Control_Enable	Current parameter setting. Possible values are: HCI_SCO_FLOW_CONTROL_DISABLE HCI_SCO_FLOW_CONTROL_ENABLE
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Set_Host_Controller_To_Host_Flow_Control

This command allows the Host to turn flow control on or off in the direction from the Host Controller to the Host. If flow control is turned off, the Host should not send the Host_Number_Of_Completed_Packets command. That command will be ignored by the Host Controller if it is sent by the Host and flow control is off.

Prototype:

int BTPSAPI **HCI_Set_Host_Controller_To_Host_Flow_Control**(
unsigned int BluetoothStackID, Byte_t Flow_Control_Enable, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Flow_Control_Enable	Desired setting of this parameter. Possible values are: HCI_HOST_FLOW_CONTROL_ENABLE_OFF HCI_HOST_FLOW_CONTROL_ENABLE_ON HCI_HOST_FLOW_CONTROL_ENABLE_ACL_ON_SCO_OFF HCI_HOST_FLOW_CONTROL_ENABLE_ACL_OFF_SCO_ON HCI_HOST_FLOW_CONTROL_ENABLE_ACL_ON_SCO_ON
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Host_Buffer_Size

This command allows the Host to notify the Host Controller of the maximum size of the data portion of HCI ACL and SCO Data Packets sent from the Host Controller to the Host and the total number of HCI ACL and SCO Data Packets that can be stored in the data buffers of the Host. The Host Controller will break up data into packets no bigger than the limits specified. The number of data packets parameters are only relevant when flow control is turned on (command above).

Prototype:

int BTPSAPI **HCI_Host_Buffer_Size**(unsigned int BluetoothStackID,
  Word_t Host_ACL_Data_Packet_Length, Byte_t Host_SCO_Data_Packet_Length,
  Word_t Host_Total_Num_ACL_Data_Packets,
  Word_t Host_Total_Num_SCO_Data_Packets, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Host_ACL_Data_Packet_Length	Maximum length of ACL data packets, up to 0xFFFF
Host_SCO_Data_Packet_Length	Maximum length of SCO data packets, up to 0xFF
Host_Total_Num_ACL_Data_Packets	Maximum number of ACL packets that can be held in the host, up to 0xFFFF.
Host_Total_Num_SCO_Data_Packets	Maximum number of SCO packets that can be held in the host, up to 0xFFFF.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Host_Number_Of_Completed_Packets

This command is used by the Host to indicate to the Host Controller the number of HCI Data Packets that have been completed (processed) for each connection since the last time this command was sent. This tells the Host Controller that the corresponding buffer space has been freed in the Host. This command should only be used when flow control is on (command above).

Prototype:

int BTPSAPI **HCI_Host_Number_Of_Completed_Packets**(unsigned int BluetoothStackID,
    Byte_t Number_Of_Handles,
    HCI_Host_Completed_Packets_Info_t HCI_Host_Completed_Packets_Info[],
    Byte_t WaitForResponse, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Number_Of_Handles	Number of packets in the provided array. Must not be zero.
HCI_Host_Completed_Packets_Info	Array of structures which pair up a connection handle and the number of packets which have been completed for that handle. The definition of the structures in this array is: `typedef struct { Word_t Connection_Handle; Word_t Host_Num_Of_Completed_Packets; } HCI_Host_Completed_Packets_Info_t;`
WaitForResponse	Boolean flag indicating whether this function call should wait until it gets a response from the Host Controller. Note, there is no response unless there is invalid data. Therefore, when the data is good this function will stall until the timeout is reached. If the

	Host knows it is passing good data, it should probably set this flag to FALSE.
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Link_Supervision_Timeout

This command reads the Link_Supervision_Timeout parameter, which is used by the master or slave Bluetooth device to monitor link loss. If, for any reason, no Baseband packets are received for a duration longer than the Link_Supervision_Timeout, the connection is disconnected. The same timeout value applies to both the SCO and ACL connections for the device specified by the ACL Connection Handle passed in this command.

Prototype:

int BTPSAPI **HCI_Read_Link_Supervision_Timeout**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult, Word_t *Link_Supervision_TimeoutResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.

Link_Supervision_TimeoutResult	Current timeout value. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Link_Supervision_Timeout

This command writes the Link_Supervision_Timeout parameter, which is used by the master or slave Bluetooth device to monitor link loss. If, for any reason, no Baseband packets are received for a duration longer than the Link_Supervision_Timeout, the connection is disconnected. The same timeout value applies to both the SCO and ACL connections for the device specified by the ACL Connection Handle passed in this command.

Setting the Link_Supervision_Timeout parameter to No Link_ Supervision_Timeout (0x0000) will disable the check for the specified connection. This makes it unnecessary for the master of the piconet to unpark and then park each Bluetooth device every ~40 seconds. By using this setting, the scalability of the Park mode is not limited.

Prototype:

int BTPSAPI **HCI_Write_Link_Supervision_Timeout**(unsigned int BluetoothStackID, Word_t Connection_Handle, Word_t Link_Supervision_Timeout, Byte_t *StatusResult, Word_t *Connection_HandleResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.

Link_Supervision_Timeout	Current timeout value. A value of zero disables this timeout. Values are number of baseband slots (0.625 msec), with a range of 0.625 msec (0x0001) to 40.9 sec (0xFFFF).
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Number_Of_Supported_IAC

This command reads the number of Inquiry Access Codes (IAC) that the local Bluetooth device can simultaneous listen for during an Inquiry Scan.

Prototype:

int BTPSAPI **HCI_Read_Number_Of_Supported_IAC**(unsigned int BluetoothStackID, Byte_t *StatusResult, Byte_t *Num_Support_IACResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Num_Support_IACResult	Current setting in the range of 0x01 to 0x40.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR

BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Current_IAC_LAP

This command reads the LAP(s) (lower address part of Bluetooth device address) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans.

Prototype:

int BTPSAPI **HCI_Read_Current_IAC_LAP**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Num_Current_IACResult, IAC_LAP_t *IAC_LAPResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Num_Current_IACResult	Number of IACs currently in use by the local Bluetooth device.

IAC_LAPResult	Array of LAPs (3-Byte structures) for in-use IACs. MACRO's exist for the two most commonly used IAC LAP's:
	HCI_ASSIGN_GIAC_LAP(lapvar)
	HCI_ASSIGN_LIAC_LAP(lapvar)
	Both MACRO's accept a variable of type LAP_t.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Current_IAC_LAP

This command writes the LAP(s) (lower address part of Bluetooth device address) used to create the Inquiry Access Codes (IAC) that the local Bluetooth device is simultaneously scanning for during Inquiry Scans. This command writes over the current IACs used by the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Current_IAC_LAP**(unsigned int BluetoothStackID,
Byte_t Num_Current_IAC, IAC_LAP_t IAC_LAP[], Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Num_Current_IAC	Number of IAC LAPs provided in this command.

IAC_LAPResult	Array of LAPs (3-Byte structures) for in-use IACs. MACRO's exist for the two most commonly used IAC LAP's:
	HCI_ASSIGN_GIAC_LAP(lapvar)
	HCI_ASSIGN_LIAC_LAP(lapvar)
	Both MACRO's accept a variable of type LAP_t.

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Page_Scan_Period_Mode

This command reads the mandatory Page_Scan_Period_Mode of the local Bluetooth device. Every time an inquiry response message is sent, the Bluetooth device will start a timer, the value of which is dependent on the Page_Scan_Period_Mode. As long as this timer has not expired, the Bluetooth device will use the Page_Scan_Period_Mode parameter for all future page scans.

Prototype:

int BTPSAPI **HCI_Read_Page_Scan_Period_Mode**(unsigned int BluetoothStackID, Byte_t *StatusResult, Byte_t *Page_Scan_Period_ModeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Page_Scan_Period_ModeResult	Current setting of this parameter. Possible values are: HCI_PAGE_SCAN_PERIOD_MODE_P0 HCI_PAGE_SCAN_PERIOD_MODE_P1 HCI_PAGE_SCAN_PERIOD_MODE_P2

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Page_Scan_Period_Mode

This command writes the mandatory Page_Scan_Period_Mode of the local Bluetooth device. Every time an inquiry response message is sent, the Bluetooth device will start a timer, the value of which is dependent on the Page_Scan_Period_Mode. As long as this timer has not expired, the Bluetooth device will use the Page_Scan_Period_Mode parameter for all future page scans.

Prototype:

int BTPSAPI **HCI_Write_Page_Scan_Period_Mode**(unsigned int BluetoothStackID,
Byte_t Page_Scan_Period_Mode, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Page_Scan_Period_Mode	Current setting of this parameter. Possible values are: HCI_PAGE_SCAN_PERIOD_MODE_P0 HCI_PAGE_SCAN_PERIOD_MODE_P1 HCI_PAGE_SCAN_PERIOD_MODE_P2
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Page_Scan_Mode

This command reads the default page scan mode of the local Bluetooth device, which is a parameter that indicates the page scan mode that is used for default page scan.

Prototype:

int BTPSAPI **HCI_Read_Page_Scan_Mode**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Page_Scan_ModeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Page_Scan_ModeResult	Current parameter setting. Possible values are:

Bluetooth Version 1.1

HCI_PAGE_SCAN_MODE_MANDATORY
HCI_PAGE_SCAN_MODE_OPTIONAL_I
HCI_PAGE_SCAN_MODE_OPTIONAL_II
HCI_PAGE_SCAN_MODE_OPTIONAL_III

Bluetooth Version 1.2

HCI_PAGE_SCAN_MODE_MANDATORY_STANDARD_SCAN
HCI_PAGE_SCAN_MODE_OPTIONAL_INTERLACED_SCAN

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Page_Scan_Mode

This command writes the default page scan mode of the local Bluetooth device, which is a parameter that indicates the page scan mode that is used for default page scan.

Prototype:

int BTPSAPI **HCI_Write_Page_Scan_Mode**(unsigned int BluetoothStackID, Byte_t Page_Scan_Mode, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Page_Scan_Mode	Current parameter setting. Possible values are:

Bluetooth Version 1.1

HCI_PAGE_SCAN_MODE_MANDATORY
HCI_PAGE_SCAN_MODE_OPTIONAL_I
HCI_PAGE_SCAN_MODE_OPTIONAL_II
HCI_PAGE_SCAN_MODE_OPTIONAL_III

Bluetooth Version 1.2

HCI_PAGE_SCAN_MODE_MANDATORY_STANDARD_SCAN
HCI_PAGE_SCAN_MODE_OPTIONAL_INTERLACED_SCAN

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Set_AFH_Host_Channel_Classification

This command command allows the Bluetooth host to specify a channel classification based on its "local information".

Prototype:

int BTPSAPI **HCI_Set_AFH_Host_Channel_Classification**(
    unsigned int BluetoothStackID,
    AFH_Channel_Map_t AFH_Host_Channel_Classification,
    Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
AFH_Host_Channel_Classification	Host channel classification. This is simply a bitmask of the available channels (numbered 0 through 79). Useful macros defined for manipulation of AFH Channel Maps are:

COMPARE_AFH_CHANNEL_MAP(map1, map2)

ASSIGN_AFH_CHANNEL_MAP(map, MSByte, …, LSByte)

SET_AFH_CHANNEL_MAP_CHANNEL(map, channum)

RESET_AFH_CHANNEL_MAP_CHANNEL(map, channum)

TEST_AFH_CHANNEL_MAP_CHANNEL(map, channum)

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Inquiry_Scan_Type

This command is used to read the Inquiry_Scan_Type configuration parameter of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Inquiry_Scan_Type**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Inquiry_Scan_TypeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Inquiry_Scan_TypeResult	Returned inquiry scan type of the local device. Possible values are: HCI_INQUIRY_SCAN_TYPE_MANDATORY_STANDARD_SCAN HCI_INQUIRY_SCAN_TYPE_OPTIONAL_INTERLACED_SCAN

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES

BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Inquiry_Scan_Type

This command is used to write the Inquiry Scan Type configuration parameter of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Inquiry_Scan_Type**(unsigned int BluetoothStackID,
    Byte_t Scan_Type, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Scan_Type	Indicates standard scan or interlaced scan. Possible values are: HCI_INQUIRY_SCAN_TYPE_MANDATORY_STANDARD_SCAN HCI_INQUIRY_SCAN_TYPE_OPTIONAL_INTERLACED_SCAN
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Inquiry_Mode

This command is used to read the Inquiry_Mode configuration parameter of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Inquiry_Mode**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Byte_t *Inquiry_ModeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Inquiry_ModeResult	Returned inquiry mode setting. Possible values are: HCI_INQUIRY_MODE_STANDARD_INQUIRY_RESULT HCI_INQUIRY_MODE_INQUIRY_RESULT_FORMAT_WITH_RSSI

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Inquiry_Mode

This command is used to write the Inquiry_Mode configuration parameter of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Inquiry_Mode**(unsigned int BluetoothStackID,
    Byte_t Inquiry_Mode, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Inquiry_Mode	Indicates standard inquiry result mode or inquiry result with RSSI mode. Possible values are: HCI_INQUIRY_MODE_STANDARD_INQUIRY_RESULT HCI_INQUIRY_MODE_INQUIRY_RESULT_FORMAT_WITH_RSSI
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Page_Scan_Type

This command is used to read the Page Scan Type configuration parameter of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Page_Scan_Type**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Page_Scan_TypeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Page_Scan_TypeResult	Returned page scan type setting. Possible values are: HCI_PAGE_SCAN_TYPE_MANDATORY_STANDARD_SCAN HCI_PAGE_SCAN_TYPE_OPTIONAL_INTERLACED_SCAN

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Page_Scan_Type

This command is used to write the Page Scan Type configuration parameter of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Page_Scan_Type**(unsigned int BluetoothStackID,
Byte_t Page_Scan_Type, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Page_Scan_Type	Indicates standard scan or interlaced scan. Possible values are: HCI_PAGE_SCAN_TYPE_MANDATORY_STANDARD_SCAN HCI_PAGE_SCAN_TYPE_OPTIONAL_INTERLACED_SCAN
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_AFH_Channel_Assessment_Mode

This command is used to read the AFH_Channel_Assessment_Mode configuration parameter of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_AFH_Channel_Assessment_Mode**(
unsigned int BluetoothStackID,
Byte_t *StatusResult,
Byte_t *AFH_Channel_Assessment_ModeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
AFH_Channel_Assessment_ModeResult	Returned AFH channel assessment mode setting. Possible values are:

HCI_AFH_CHANNEL_ASSESSMENT_MODE_CONTROLLER_
ASSESSMENT_DISABLED
HCI_AFH_CHANNEL_ASSESSMENT_MODE_CONTROLLER_
ASSESSMENT_ENABLED

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_AFH_Channel_Assessment_Mode

This command is used to write the AFH_Channel_Assessment_Mode configuration parameter of the local Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_AFH_Channel_Assessment_Mode**(
unsigned int BluetoothStackID,

Byte_t AFH_Channel_Assessment_Mode,
Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
AFH_Channel_Assessment_Mode	Indicates whether the controller channel assessment is enabled or disabled. Possible values are:

HCI_AFH_CHANNEL_ASSESSMENT_MODE_CONTROLLER_
ASSESSMENT_DISABLED
HCI_AFH_CHANNEL_ASSESSMENT_MODE_CONTROLLER_
ASSESSMENT_ENABLED

StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Extended_Inquiry_Response

This function issues the HCI_Read_Extended_Inquiry_Response Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Read_Extended_Inquiry_Response**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *FEC_RequiredResult, Extended_Inquiry_Response_Data_t *Extended_Inquiry_Response_DataResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
FEC_RequiredResult	If function returns zero (success) this variable will contain the FEC Required parameter returned from the Bluetooth device
Extended_Inquiry_Response_DataResult	If function returns zero (success) this variable variable will the contain the Extended Inquiry Response Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Extended_Inquiry_Response

This function issues the HCI_Write_Extended_Inquiry_Response Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Write_Extended_Inquiry_Response**(unsigned int BluetoothStackID,
Byte_t FEC_Required, Extended_Inquiry_Response_Data_t
*Extended_Inquiry_Response_Data, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
FEC_Required	Specifies whether FEC is required
Extended_Inquiry_Response_Data	Pointer to the actual formatted Extended Inquiry Response Data (must be 240 bytes in length).

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Refresh_Encryption_Key

This function issues the HCI_Refresh_Encryption_Key Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Refresh_Encryption_Key**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Simple_Pairing_Mode

This function issues the HCI_Read_Simple_Pairing_Mode Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Read_Simple_Pairing_Mode**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Byte_t *Simple_Pairing_ModeResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Simple_Pairing_ModeResult	If function returns zero (success) this variable will contain the Simple Pairing Mode returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Simple_Pairing_Mode

This function issues the HCI_Write_Simple_Pairing_Mode Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Write_Simple_Pairing_Mode**(unsigned int BluetoothStackID,
    Byte_t Simple_Pairing_Mode, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Simple_Pairing_Mode	Flags whether not simple pairing mode is enabled. Possible values: HCI_SIMPLE_PAIRING_MODE_NOT_ENABLED HCI_SIMPLE_PAIRING_MODE_ENABLED
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Local_OOB_Data

This function issues the HCI_Read_Local_OOB_Data Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Read_Local_OOB_Data**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Simple_Pairing_Hash_t *Simple_Pairing_HashResult,
Simple_Pairing_Randomizer_t *Simple_Pairing_RandomizerResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Simple_Pairing_HashResult	If function returns zero (success) this variable will contain the Simple_Pairing_HashResult returned from the Bluetooth device
Simple_Pairing_RandomizerResult	If function returns zero (success) this variable will contain the Simple_Pairing_RandomizerResult returned from the Bluetooth device

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Inquiry_Response_Transmit_Power_Level

This function issues the HCI_Read_Inquiry_Response_Transmit_Power_Level Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Read_Inquiry_Response_Transmit_Power_Level**(
unsigned int BluetoothStackID, Byte_t *StatusResult, SByte_t *TX_PowerResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
TX_PowerResult	If function returns zero (success) this variable will contain the TX_PowerResult returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Inquiry_Transmit_Power_Level

This function issues the HCI_Write_Inquiry_Transmit_Power_Level Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Write_Inquiry_Transmit_Power_Level**(
    unsigned int BluetoothStackID, SByte_t TX_Power, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TX_Power	Transmit power level. This is a signed value that specifies dBm. Range must be between -70 dB and 20 dBm.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Send_Keypress_Notification

This function issues the HCI_Send_Keypress_Notification Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI HCI**_Send_Keypress_Notification**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, Byte_t KeyPress, Byte_t *StatusResult,
    BD_ADDR_t *BD_ADDRResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address of the remote Bluetooth device to receive the Keypress notification.
KeyPress	Keypress Notification value. Possible values: HCI_KEYPRESS_NOTIFICATION_TYPE_PASSKEY_ENTRY_STARTED HCI_KEYPRESS_NOTIFICATION_TYPE_PASSKEY_DIGIT_ENTERED HCI_KEYPRESS_NOTIFICATION_TYPE_PASSKEY_DIGIT_ERASED HCI_KEYPRESS_NOTIFICATION_TYPE_PASSKEY_CLEARED HCI_KEYPRESS_NOTIFICATION_TYPE_PASSKEY_ENTRY_COMPLETED
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
BD_ADDRResult	If function returns zero (success) this variable will contain the BD_ADDRResult returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Default_Erroneous_Data_Reporting

This function issues the HCI_Read_Default_Erroneous_Data_Reporting Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Read_Default_Erroneous_Data_Reporting**(
unsigned int BluetoothStackID, Byte_t *StatusResult,
Byte_t *Erroneous_Data_ReportingResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Erroneous_Data_ReportingResult	If function returns zero (success) this variable will contain the Connection_Handle Result returned from the Bluetooth device.

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Default_Erroneous_Data_Reporting

This function issues the HCI_Write_Default_Erroneous_Data_Reporting Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Write_Default_Erroneous_Data_Reporting**(
unsigned int BluetoothStackID, Byte_t Erroneous_Data_Reporting,
Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Erroneous_Data_Reporting	Specifies whether Erroneous Data Repoirting is enabled. Possible values:

	HCI_ERRONEOUS_DATA_REPORTING_NOT_ENABLED HCI_ERRONEOUS_DATA_REPORTING_ENABLED
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Enhanced_Flush

This function issues the HCI_Enhanced_Flush Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter.

Prototype:

int BTPSAPI **HCI_Enhanced_Flush**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, Byte_t Packet_Type, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

Non zero if failure

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Logical_Link_Accept_Timeout

Issues the HCI_Read_Logical_Link_Accept_Timeout command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). The purpose of sending this command is to read the Logical_Link_Accept_Timeout configuration parameter. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Logical_Link_Accept_Timeout** (unsigned int BluetoothStackID, Byte_t *StatusResult, Word_t *Logical_Link_Accept_TimeoutResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Logical_Link_Accept_TimeoutResult	If function returns zero (success) the variable pointed to by this parameter will contain the Logical Link Accept Timeout returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Logical_Link_Accept_Timeout

Issues the HCI_Write_Logical_Link_Accept_Timeout command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). The purpose of sending this command is to write the Logical_Link_Accept_Timeout configuration parameter. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Logical_Link_Accept_Timeout** (unsigned int BluetoothStackID, Word_t Logical_Link_Accept_Timeout, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Logical_Link_Accept_Timeout	Contains the Logical Link Accept Timeout that will be written to the Logical Link Accept Timeout configuration parameter.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Set_Event_Mask_Page_2

Issues the HCI_Set_Event_Mask_Page_2 command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). The purpose of this command is to control which events are generated by the HCI for the host. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Note:

This function uses MACRO's to set/clear bits in an event mask structure. Constants are provided that specify the actual bit numbers that are to be used with the MACRO (see below).

Prototype:

int BTPSAPI **HCI_Set_Event_Mask_Page_2** (unsigned int BluetoothStackID,
  Event_Mask_t Event_Mask, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Event_Mask	Eight-byte bit mask of events to allow. Setting a bit to one enables the corresponding event. The bit mask is constructed via the following API macros:

SET_EVENT_MASK_BIT(Mask, BitNumber)

RESET_EVENT_MASK_BIT(Mask, BitNumber)

TEST_EVENT_MASK_BIT(Mask, BitNumber)

HCI_ENABLE_ALL_HCI_EVENTS_IN_EVENT_
MASK_PAGE_2(Mask)

HCI_DISABLE_ALL_HCI_EVENTS_IN_EVENT_
MASK_PAGE_2(Mask)

The bit number constants defined in the API for use with these macros are:

HCI_EVENT_MASK_PHYSICAL_LINK_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_CHANNEL_SELECTED_BIT_NUMBER
HCI_EVENT_MASK_DISCONNECTION_PHYSICAL_LINK_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_PHYSICAL_LINK_LOSS_EARLY_WARNING_BIT_NUMBER
HCI_EVENT_MASK_PHYSICAL_LINK_RECOVERY_BIT_NUMBER
HCI_EVENT_MASK_LOGICAL_LINK_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_DISCONNECTION_LOGICAL_LINK_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_FLOW_SPEC_MODIFY_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_NUMBER_OF_COMPLETED_DATA_BLOCKS_BIT_NUMBER
HCI_EVENT_MASK_AMP_START_TEST_BIT_NUMBER
HCI_EVENT_MASK_AMP_TEST_END_BIT_NUMBER
HCI_EVENT_MASK_AMP_RECEIVER_REPORT_BIT_NUMBER
HCI_EVENT_MASK_SHORT_RANGE_MODE_CHANGE_COMPLETE_BIT_NUMBER
HCI_EVENT_MASK_AMP_STATUS_CHANGE_BIT_NUMBER

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Location_Data

Issues the HCI_Read_Location_Data command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command provides the ability to read any stored knowledge of environment or regulations that are currently in use. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Location_Data**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Byte_t *Location_Domain_AwareResult,
    Word_t *Location_DomainResult, Byte_t *Location_Domain_OptionsResult,
    Byte_t *Location_OptionsResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Location_Domain_AwareResult	If function returns zero (success) the variable pointed to by this parameter will contain the Location Domain Aware Result returned from the device. This value is one of the following: HCI_LOCATION_DOMAIN_REGULATORY_ DOMAIN_UNKNOWN HCI_LOCATION_DOMAIN_REGULATORY_ DOMAIN_KNOWN
Location_DomainResult	If function returns zero (success) the variable pointed to by this parameter will contain the Location Domain Result returned from the device.
Location_Domain_OptionsResult	If function returns zero (success) the variable pointed to by this parameter will contain the Location Domain Options result returned from the device.
Location_OptionsResult	If function returns zero (success) the variable pointed to by this parameter will contain the Location Options Result returned from the device. This value is one of the following: HCI_LOCATION_DOMAIN_OPTIONS_NOT_ MAINS_POWERED HCI_LOCATION_DOMAIN_OPTIONS_MAINS_ POWERED

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Location_Data

Issues the HCI_Write_Location_Data command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command provides the ability to write information about the environment or regulations in use. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Location_Data**(unsigned int BluetoothStackID,
    Byte_t Location_Domain_Aware, Word_t Location_Domain,
    Byte_t Location_Domain_Options, Byte_t Location_Options, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Location_Domain_Aware	Location Domain Aware result to write. This value is one of the following: HCI_LOCATION_DOMAIN_REGULATORY_ DOMAIN_UNKNOWN HCI_LOCATION_DOMAIN_REGULATORY_ DOMAIN_KNOWN
Location_Domain	Location Domain result to write.
Location_Domain_Options	Location Domain Options to write.
Location_Options	Location Options to write. This value is one of the following: HCI_LOCATION_DOMAIN_OPTIONS_NOT_ MAINS_POWERED HCI_LOCATION_DOMAIN_OPTIONS_MAINS_ POWERED
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Flow_Control_Mode

Issues the HCI_Read_Flow_Control_Mode command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command reads the Flow_Control_Mode configuration parameter. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Flow_Control_Mode**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Flow_Control_ModeResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Flow_Control_ModeResult	If function returns zero (success) the variable pointed to by this parameter will contain the Flow Control Mode Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Flow_Control_Mode

Issues the HCI_Write_Flow_Control_Mode command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command writes the Flow_Control_Mode configuration parameter. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Flow_Control_Mode**(unsigned int BluetoothStackID,
    Byte_t Flow_Control_Mode, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Flow_Control_Mode	Flow Control Mode to write to Flow_Control_Mode configuration parameter.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Enhanced_Transmit_Power_Level

Issues the HCI_Read_Enhanced_Transmit_Power_Level command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Reads the values for the Enhanced_Transmit_Power_Level configuration parameters. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Enhanced_Transmit_Power_Level** (
    unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult,
    Word_t *Connection_HandleResult, SByte_t *Transmit_Power_Level_GFSKResult,

```
        SByte_t *Transmit_Power_Level_DQPSKResult,
        SByte_t *Transmit_Power_Level_8DPSKResult);
```

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
ConnectionHandle	Connection handle used to identify the connection to be used, must be a Connection_Handle for an ACL connection.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Connection_HandleResult	If function returns zero (success) the variable pointed to by this parameter will contain the Connection Handle Result returned from the Bluetooth device.
Transmit_Power_Level_GFSKResult	If function returns zero (success) the variable pointed to by this parameter will contain the GFSK Transmit Power level returned from the Bluetooth device
Transmit_Power_Level_DQPSKResult	If function returns zero (success) the variable pointed to by this parameter will contain DQPSK Transmit Power level returned from the Bluetooth device.
Transmit_Power_Level_8DQPSKResult	If function returns zero (success) the variable pointed to by this parameter will contain the 8DQPSK Transmit Power Level returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Best_Effort_Flush_Timeout

Issues the HCI_Read_Best_Effort_Flush_Timeout command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Reads the values of the Best Effort Flush Timeout. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Best_Effort_Flush_Timeout** (unsigned int BluetoothStackID,
    Word_t Logical_Link_Handle, Byte_t *StatusResult,
    DWord_t *Best_Effort_Flush_TimeoutResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Logical_Link_Handle	Handle of Logical Link to which the command applies.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
Best_Effort_Flush_TimeoutResult	If function returns zero (success) the variable pointed to by this parameter will contain the Best Effort Flush Timeout read from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Best_Effort_Flush_Timeout

Issues the HCI_Write_Best_Effort_Flush_Timeout command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). Writes the values of the Best Effort Flush Timeout. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device

Prototype:

int BTPSAPI **HCI_Write_Best_Effort_Flush_Timeout** (unsigned int BluetoothStackID,
Word_t Logical_Link_Handle, DWord_t Best_Effort_Flush_Timeout,
Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Logical_Link_Handle	Handle of Logical Link to which the command applies.
Best_Effort_Flush_Timeout	Value to write to the Best Effort Flush Timeout of the specified logical link.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Short_Range_Mode

Issues the HCI_Short_Range command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command configures the Short Range Mode value. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device

Prototype:

int BTPSAPI **HCI_Short_Range_Mode** (unsigned int BluetoothStackID,
Byte_t Physical_Link_Handle, Byte_t Short_Range_Mode, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Physical_Link_Handle	Handle of the physical link to which the command applies.
Short_Range_Mode	Configuration setting of Short Range Mode configuration parameter. Possible values are (all others are reserverd):

	HCI_SHORT_RANGE_MODE_DISABLED HCI_SHORT_RANGE_MODE_ENABLED
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_LE_Host_Supported

Issues the HCI_Read_LE_Host_Supported command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_LE_Host_Supported**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Byte_t *LE_Supported_HostResult,
    Byte_t *Simultaneous_LE_HostResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device
LE_Supported_HostResult	If function is successful, this will contain the LE supported host result. Possible values are HCI_LE_SUPPORTED_HOST_LE_SUPPORTED_HOST_ENABLED HCI_LE_SUPPORTED_HOST_LE_SUPPORTED_HOST_DISABLED
Simultaneous_LE_HostResult	If function is successful, this will contain the simultaneous LE host result. Possible values are

HCI_LE_SIMULTANEOUS_LE_HOST_
SUPPORTED_ENABLED
HCI_LE_SIMULTANEOUS_LE_HOST_
SUPPORTED_DISABLED

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_LE_Host_Supported

Issues the HCI_Read_LE_Write_Supported command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter followed by the Host supported LE parameters. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_LE_Host_Supported**(unsigned int BluetoothStackID,
Byte_t *LE_Supported_HostResult, Byte_t *Simultaneous_LE_HostResult,
Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
LE_Supported_HostResult	Used to set the LE supported feature bit on the host device. Possible values are HCI_LE_SUPPORTED_HOST_LE_SUPPORTED_HOST_ENABLED HCI_LE_SUPPORTED_HOST_LE_SUPPORTED_HOST_DISABLED
Simultaneous_LE_HostResult	Used to set the simultaneous LE and BR/EDR to same device capable feature bit on the host device. Possible values are HCI_LE_SIMULTANEOUS_LE_HOST_SUPPORTED_ENABLED HCI_LE_SIMULTANEOUS_LE_HOST_SUPPORTED_DISABLED

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.5 Informational Parameters

The API functions in this section provide access to the Informational Parameters which are settings established by the Bluetooth hardware manufacturer and which provide information about the Bluetooth device and the capabilities of the Host Controller, Link Manager, and Baseband sections. These parameters cannot be modified. The commands in this section are listed in the table below.

Command	Description
HCI_Read_Local_Version_Information	Read the version information for the local device.
HCI_Read_Local_Supported_Features	Read a list of the supported features for the local device.
HCI_Read_Buffer_Size	Read the size of the HCI buffers (used for transmissions).
HCI_Read_Country_Code	Read the Country Code status parameter, which defines which range of frequency band of the ISM 2.4 GHz band will be used by the device.
HCI_Read_BD_ADDR	Read the BD_ADDR, which is a 48-bit unique identifier for a Bluetooth device.
HCI_Read_Local_Supported_Commands	Read the list of HCI commands supported for the local device.
HCI_ Read_Local_Extended_Features	Read the requested page of the extended LMP features.
HCI_Read_Data_Block_Size	Reads information pertaining to the maximum permitted data transfer over the controller and the data buffering available in the controller.

HCI_Read_Local_Version_Information

This command reads the version information for the local Bluetooth device (several components).

Prototype:

int BTPSAPI **HCI_Read_Local_Version_Information**(unsigned int BluetoothStackID, Byte_t *StatusResult, Byte_t *HCI_VersionResult, Word_t *HCI_RevisionResult, Byte_t *LMP_VersionResult, Word_t *Manufacturer_NameResult, Word_t *LMP_SubversionResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
HCI_VersionResult	Major version for the Bluetooth hardware. Corresponds to changes in the released specifications only. Possible values are: HCI_VERSION_SPECIFICATION_1_0B HCI_VERSION_SPECIFICATION_1_1 HCI_VERSION_SPECIFICATION_1_2 HCI_VERSION_SPECIFICATION_2_0 HCI_VERSION_SPECIFICATION_2_1 HCI_VERSION_SPECIFICATION_3_0 HCI_VERSION_SPECIFICATION_4_0
HCI_RevisionResult	The HCI revision number
LMP_VersionResult	The Link Manager Protocol version number. Possible values are: HCI_LMP_VERSION_BLUETOOTH_1_0 HCI_LMP_VERSION_BLUETOOTH_1_1 HCI_LMP_VERSION_BLUETOOTH_1_2 HCI_LMP_VERSION_BLUETOOTH_2_0 HCI_LMP_VERSION_BLUETOOTH_2_1 HCI_LMP_VERSION_BLUETOOTH_3_0 HCI_LMP_VERSION_BLUETOOTH_4_0
Manufacturer_NameResult	Manufacturer code. Possible values are: HCI_LMP_COMPID_MANUFACTURER_NAME_ ERICSSON_MOBILE_COMMUNICATIONS HCI_LMP_COMPID_MANUFACTURER_NAME_ NOKIA_MOBILE_PHONES HCI_LMP_COMPID_MANUFACTURER_NAME_ INTEL_CORPORATION HCI_LMP_COMPID_MANUFACTURER_NAME_ IBM_CORPORATION HCI_LMP_COMPID_MANUFACTURER_NAME_ TOSHIBA_CORPORATION

HCI_LMP_COMPID_MANUFACTURER_NAME_3COM
HCI_LMP_COMPID_MANUFACTURER_NAME_MICROSOFT
HCI_LMP_COMPID_MANUFACTURER_NAME_LUCENT
HCI_LMP_COMPID_MANUFACTURER_NAME_MOTOROLA
HCI_LMP_COMPID_MANUFACTURER_NAME_INFINEON_TECHNOLOGIES_AG
HCI_LMP_COMPID_MANUFACTURER_NAME_CAMBRIDGE_SILICON_RADIO
HCI_LMP_COMPID_MANUFACTURER_NAME_SILICON_WAVE
HCI_LMP_COMPID_MANUFACTURER_NAME_DIGIANSWER
HCI_LMP_COMPID_MANUFACTURER_NAME_TEXAS_INSTRUMENTS
HCI_LMP_COMPID_MANUFACTURER_NAME_PARTHUS_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_BROADCOM
HCI_LMP_COMPID_MANUFACTURER_NAME_MITEL_SEMICONDUCTOR
HCI_LMP_COMPID_MANUFACTURER_NAME_WIDCOMM
HCI_LMP_COMPID_MANUFACTURER_NAME_TELENCOMM
HCI_LMP_COMPID_MANUFACTURER_NAME_ATMEL
HCI_LMP_COMPID_MANUFACTURER_NAME_MITSUBISHI
HCI_LMP_COMPID_MANUFACTURER_NAME_RTX_TELECOM
HCI_LMP_COMPID_MANUFACTURER_NAME_KC_TECHNOLOGY
HCI_LMP_COMPID_MANUFACTURER_NAME_NEWLOGIC
HCI_LMP_COMPID_MANUFACTURER_NAME_TRANSILICA
HCI_LMP_COMPID_MANUFACTURER_NAME_ROHDE_AND_SCHWARTZ
HCI_LMP_COMPID_MANUFACTURER_NAME_TTPCOM
HCI_LMP_COMPID_MANUFACTURER_NAME_SIGNIA_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_CONEXANT_SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_QUALCOMM

HCI_LMP_COMPID_MANUFACTURER_NAME_
INVENTEL
HCI_LMP_COMPID_MANUFACTURER_NAME_
AVM_BERLIN
HCI_LMP_COMPID_MANUFACTURER_NAME_
BANDSPEED
HCI_LMP_COMPID_MANUFACTURER_NAME_
MANSELLA
HCI_LMP_COMPID_MANUFACTURER_NAME_
NEC
HCI_LMP_COMPID_MANUFACTURER_NAME_
WAVEPLUS_TECHNOLOGY
HCI_LMP_COMPID_MANUFACTURER_NAME_
ALCATEL
HCI_LMP_COMPID_MANUFACTURER_NAME_
PHILIPS_SEMICONDUCTORS
HCI_LMP_COMPID_MANUFACTURER_NAME_
C_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_
OPEN_INTERFACE
HCI_LMP_COMPID_MANUFACTURER_NAME_
RF_MICRO_DEVICES
HCI_LMP_COMPID_MANUFACTURER_NAME_
HITACHI
HCI_LMP_COMPID_MANUFACTURER_NAME_
SYMBOL_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_
TENOVIS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MACRONIX_INTERNATIONAL
HCI_LMP_COMPID_MANUFACTURER_NAME_
GCT_SEMICONDUCTOR
HCI_LMP_COMPID_MANUFACTURER_NAME_
NORWOOD_SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MEWTEL_TECHNOLOGY
HCI_LMP_COMPID_MANUFACTURER_NAME_
ST_MICROELECTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_
SYNOPSYS
HCI_LMP_COMPID_MANUFACTURER_NAME_
RED_M_COMMUNICATIONS
HCI_LMP_COMPID_MANUFACTURER_NAME_
COMMIL_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
CATC
HCI_LMP_COMPID_MANUFACTURER_NAME_
ECLIPSE_SL
HCI_LMP_COMPID_MANUFACTURER_NAME_
RENESAS_TECHNOLOGY_CORP

HCI_LMP_COMPID_MANUFACTURER_NAME_
MOBILIAN_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
TERAX
HCI_LMP_COMPID_MANUFACTURER_NAME_
INTEGRATED_SYSTEM_SOLUTION
HCI_LMP_COMPID_MANUFACTURER_NAME_
MATSUSHITA
HCI_LMP_COMPID_MANUFACTURER_NAME_
GENNUM_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
RESEARCH_IN_MOTION
HCI_LMP_COMPID_MANUFACTURER_NAME_
IPEXTREME
HCI_LMP_COMPID_MANUFACTURER_NAME_
SYSTEMS_AND_CHIPS
HCI_LMP_COMPID_MANUFACTURER_NAME_
BLUETOOTH_SIG
HCI_LMP_COMPID_MANUFACTURER_NAME_
SEIKO_EPSON_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
INTEGRATED_SILICON_SOLUTION
HCI_LMP_COMPID_MANUFACTURER_NAME_
CONWISE_TECHNOLOGY_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
PARROT_SA
HCI_LMP_COMPID_MANUFACTURER_NAME_
SOCKET_MOBILE
HCI_LMP_COMPID_MANUFACTURER_NAME_
ATHEROS_COMMUNICATIONS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MEDIATEK_INCORPORATED
HCI_LMP_COMPID_MANUFACTURER_NAME_
BLUEGIGA
HCI_LMP_COMPID_MANUFACTURER_NAME_
MARVELL_TECHNOLOGY_GROUP
HCI_LMP_COMPID_MANUFACTURER_NAME_
3DSP_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
ACCEL_SEMICONDUCTOR
HCI_LMP_COMPID_MANUFACTURER_NAME_
CONTINENTAL_AUTOMOTIVE_SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_
APPLE_INCORPORATED
HCI_LMP_COMPID_MANUFACTURER_NAME_
STACCATO_COMMUNICATIONS
HCI_LMP_COMPID_MANUFACTURER_NAME_
AVAGO_TECHONOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_APT_
LIMITED

HCI_LMP_COMPID_MANUFACTURER_NAME_SIRF_
TECHONOLIGY
HCI_LMP_COMPID_MANUFACTURER_NAME_
TZERO_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_J_
AND_M_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
FREE2MOVE_AB
HCI_LMP_COMPID_MANUFACTURER_NAME_3DIJOY_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
PLANTRONICS_INCORPORATED
HCI_LMP_COMPID_MANUFACTURER_NAME_SONY_
ERICSSON_MOBILE_COMM
HCI_LMP_COMPID_MANUFACTURER_NAME_
HARMAN_INTERNATIONAL_IND
HCI_LMP_COMPID_MANUFACTURER_NAME_
VIZIO_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_NORDIC_S
EMICONDUCTOR_ASA
HCI_LMP_COMPID_MANUFACTURER_NAME_EM_
MICROELECTRONIC_MARIN_SA
HCI_LMP_COMPID_MANUFACTURER_NAME_RALINK_T
ECHNOLOGY_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_BELKIN_
INTERNATIONAL_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
REALTEK_SEMICONDUCTOR_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_
STONESTREET_ONE_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
WICENTRIC_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_RIVIERA_
WAVES_SAS
HCI_LMP_COMPID_MANUFACTURER_NAME_RDA_
MICROELECTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_GIBSON_G
UITARS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MICOMMAND_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_BAND_
XI_INTERNATIONAL_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
HEWLETT_PACKARD_COMPANY
HCI_LMP_COMPID_MANUFACTURER_NAME_
9SOLUTIONS_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_GN_
NETCOM_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_
GENERAL_MOTORS

HCI_LMP_COMPID_MANUFACTURER_NAME_A_
AND_D_ENGINEERING_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
MINDTREE_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_POLAR_
ELECTRO_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_
BEAUTIFUL_ENTERPRISE_COMPANY
HCI_LMP_COMPID_MANUFACTURER_NAME_
BRIARTEK_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_SUMMIT_
DATA_COMMUNICATIONS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_SOUND_ID
HCI_LMP_COMPID_MANUFACTURER_NAME_
MONSTER_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
CONNECT_BLUE_AB
HCI_LMP_COMPID_MANUFACTURER_NAME_
SHANGHAI_SUPER_SMART_ELECTRON
HCI_LMP_COMPID_MANUFACTURER_NAME_GROUP_
SENSE_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_ZOMM_
LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
SAMSUNG_ELECTRONICS_CO_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
CREATIVE_TECHNOLOGY_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_LAIRD_
TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_NIKE_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
LESSWIRE_AG
HCI_LMP_COMPID_MANUFACTURER_NAME_MSTAR_
SEMICONDUCTOR_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
HANLYNN_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_A_AND_R_
CAMBRIDGE
HCI_LMP_COMPID_MANUFACTURER_NAME_SEERS_
TECHNOLOGY_CO_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_SPORTS_
TRACKING_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_
AUTONET_MOBILE
HCI_LMP_COMPID_MANUFACTURER_NAME_
DELORME_PUBLISHING_COMPANY_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_WUXI_
VIMICRO
HCI_LMP_COMPID_MANUFACTURER_NAME_
SENNHEISER_COMMUNICATIONS_AS

HCI_LMP_COMPID_MANUFACTURER_NAME_
TIMEKEEPING_SYSTEMS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_LUDUS_
HELSINKI_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
BLUERADIOS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
EQUINUX_AG
HCI_LMP_COMPID_MANUFACTURER_NAME_GARMIN_
INTERNATIONAL_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ECOTEST
HCI_LMP_COMPID_MANUFACTURER_NAME_GN_
RESOUND_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_JAWBONE
HCI_LMP_COMPID_MANUFACTURER_NAME_TOPCON_
POSITIONING_SYSTEMS_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUALCOMM_LABS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ZSCAN_
SOFTWARE
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUINTIC_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_
STOLLMANN_E_V_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_FUNAI
_ELECTRIC_COMPANY_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
ADVANCED_PANMOBIL_SYSTEMS_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
THINKOPTICS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
UNIVERSAL_ELECTRONICS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_AIROHA_
TECHNOLOGY_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_NEC_
LIGHTING_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_ODM_
TECHNOLOGY_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
BLUETREK_TECHNOLOGIES_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_ZERO_1_
TV_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_I_TECH_
DYNAMIC_GLOBAL_DIST_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_ALPWISE
HCI_LMP_COMPID_MANUFACTURER_NAME_JIANGSU_
TOPPOWER_AUTOMOTIVE
HCI_LMP_COMPID_MANUFACTURER_NAME_COLORFY_
INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
GEOFORCE_INC

HCI_LMP_COMPID_MANUFACTURER_NAME_BOSE_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_SUUNTO_
OY
HCI_LMP_COMPID_MANUFACTURER_NAME_
KENSINGTON_COMPUTER_PROD_GROUP
HCI_LMP_COMPID_MANUFACTURER_NAME_SR_
MEDIZINELEKTRONIK
HCI_LMP_COMPID_MANUFACTURER_NAME_VERTU_
CORPORATION_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_META_
WATCH_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_LINAK_
A_S
HCI_LMP_COMPID_MANUFACTURER_NAME_OTL_
DYNAMICS_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_PANDA_
OCEAN_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_VISTEON_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_ARP_
DEVICES_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_MAGNETI_
MARELLI_S_P_A
HCI_LMP_COMPID_MANUFACTURER_NAME_CAEN_
RFID_SRL
HCI_LMP_COMPID_MANUFACTURER_NAME_
INGENIEUR_SYSTEMGRUPPE_ZAHN
HCI_LMP_COMPID_MANUFACTURER_NAME_GREEN_
THROTTLE_GAMES
HCI_LMP_COMPID_MANUFACTURER_NAME_
PETER SYSTEMTECHNIK_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
OMEGAWAVE_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_CINETIX
HCI_LMP_COMPID_MANUFACTURER_NAME_PASSIF_
SEMICONDUCTOR_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_SARIS_
CYCLING_GROUP_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_BEKEY_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_
CLARINOX_TECHNOLOGIES_PTY_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_BDE_
TECHNOLOGY_CO_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_SWIRL_
NETWORKS
HCI_LMP_COMPID_MANUFACTURER_NAME_MESO_
INTERNATIONAL
HCI_LMP_COMPID_MANUFACTURER_NAME_TRELAB_
LTD

HCI_LMP_COMPID_MANUFACTURER_NAME_
QUALCOMM_INNOVATION_CENTER_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_JOHNSON_
CONTROLS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_STARKEY_
LABORATORIES_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_S_POWER_
ELECTRONICS_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_ACE_
SENSOR_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_APLIX_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_AAMP_OF_
AMERICA
HCI_LMP_COMPID_MANUFACTURER_NAME_
STALMART_TECHNOLOGY_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_
AMICCOM_ELECTRONICS_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_
SHENZHEN_EXCELSECU_DATA_TECH
HCI_LMP_COMPID_MANUFACTURER_NAME_
GENEQ_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
ADIDAS_AG
HCI_LMP_COMPID_MANUFACTURER_NAME_LG_
ELECTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_ONSET_
COMPUTER_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
SELFLY_BV
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUUPPA_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_GELO_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_EVLUMA
HCI_LMP_COMPID_MANUFACTURER_NAME_MC10
HCI_LMP_COMPID_MANUFACTURER_NAME_
BINAURIC_SE
HCI_LMP_COMPID_MANUFACTURER_NAME_BEATS_
ELECTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MICROCHIP_TECHNOLOGY_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ELGATO_
SYSTEMS_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
ARCHOS_SA
HCI_LMP_COMPID_MANUFACTURER_NAME_
DEXCOM_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_POLAR_
ELECTRO_EUROPE_BV
HCI_LMP_COMPID_MANUFACTURER_NAME_DIALOG_
SEMICONDUCTOR_BV

HCI_LMP_COMPID_MANUFACTURER_NAME_
TAIXINGBANG_TECHNOLOGY_HK_CO
HCI_LMP_COMPID_MANUFACTURER_NAME_
KAWANTECH
HCI_LMP_COMPID_MANUFACTURER_NAME_AUSTCO_
COMMUNICATION_SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_TIMEX_
GROUP_USA_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUALCOMM_TECHNOLOGIES_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUALCOMM_CONNECTED_EXPERIENCES
HCI_LMP_COMPID_MANUFACTURER_NAME_
VOYETRA_TURTLE_BEACH
HCI_LMP_COMPID_MANUFACTURER_NAME_
TXTR_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
BIOSENTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_PROCTER_
AND_GAMBLE
HCI_LMP_COMPID_MANUFACTURER_NAME_HOSIDEN_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
MUZIK_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_MISFIT_
WEARABLES_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_GOOGLE
HCI_LMP_COMPID_MANUFACTURER_NAME_
DANLERS_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
SEMILINK_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_INMUSIC_
BRANDS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_LS_
RESEARCH_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_EDEN_
SOFTWARE_CONSULTANTS_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
FRESHTEMP
HCI_LMP_COMPID_MANUFACTURER_NAME_KS_
TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_ACTS_
TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_VTRACK_
SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_NIELSEN_
KELLERMAN_COMPANY
HCI_LMP_COMPID_MANUFACTURER_NAME_SERVER_
TECHNOLOGY_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
BIORESEARCH_ASSOCIATES

HCI_LMP_COMPID_MANUFACTURER_NAME_JOLLY_
LOGIC_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_ABOVE_
AVERAGE_OUTCOMES_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
BITSPLITTERS_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
PAYPAL_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_WITRON_
TECHNOLOGY_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_MORSE_
PROJECT_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_KENT_
DISPLAYS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
NAUTILUS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
SMARTIFIER_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_
ELCOMETER_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_VSN_
TECHNOLOGIES_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ACEUNI_
CORP_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
STICKNFIND
HCI_LMP_COMPID_MANUFACTURER_NAME_CRYSTAL_
CODE_AB
HCI_LMP_COMPID_MANUFACTURER_NAME_
KOUKAAM_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_
DELPHI_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
VALENCETECH_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_
RESERVED
HCI_LMP_COMPID_MANUFACTURER_NAME_TYPO_
PRODUCTS_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_TOMTOM_
INTERNATIONAL_BV
HCI_LMP_COMPID_MANUFACTURER_NAME_
FUGOO_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_KEISER_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_BANG_
AND_OLUFSEN_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_PLUS_
LOCATIONS_SYSTEMS_PTY_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
UBIQUITOUS_COMPUTING_TECH_CORP

	HCI_LMP_COMPID_MANUFACTURER_NAME_ INNOVATIVE_YACHTTER_SOLUTIONS
LMP_SubversionResult	The LMP sub-version number. These are defined by each manufacturer.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Local_Supported_Features

This command reads a list of the local supported features of the Bluetooth hardware.

Note:

Each Page of the LMP Features is 64 bits (0 - 0x3F). If a Feature bit number is larger than 64 bits (0 - 0x3F) then it exists as an "Extended Feature" and exists on a non-zero page. The actual LMP Features page can be found by dividing the bit number by 64 (or (sizeof(LMP_Feature_t)*8).

Note:

Constants are provided below to determine the actual bit number within a Page (HCI_LMP_FEATURE_PAGE_BIT_NUMBER_MASK) and the divisor to apply to the bit numbers to determine the correct page (HCI_LMP_FEATURE_PAGE_NUMBER_DIVISOR).

Prototype:

int BTPSAPI **HCI_Read_Local_Supported_Features**(unsigned int BluetoothStackID, Byte_t *StatusResult, LMP_Features_t *LMP_FeaturesResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.

LMP_FeaturesResult	Bit mask list of supported features. Defined bit numbers are (note that are all on Page 0 which is only applicable to this function):

Bluetooth Version 1.1

HCI_LMP_FEATURE_THREE_SLOT_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_FIVE_SLOT_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_ENCRYPTION_BIT_NUMBER
HCI_LMP_FEATURE_SLOT_OFFSET_BIT_NUMBER
HCI_LMP_FEATURE_TIMING_ACCURACY_BIT_NUMBER
HCI_LMP_FEATURE_SWITCH_BIT_NUMBER
HCI_LMP_FEATURE_HOLD_MODE_BIT_NUMBER
HCI_LMP_FEATURE_SNIFF_MODE_BIT_NUMBER
HCI_LMP_FEATURE_PARK_MODE_BIT_NUMBER
HCI_LMP_FEATURE_RSSI_BIT_NUMBER
HCI_LMP_FEATURE_CHANNEL_QUALITY_DRIVEN_
DATA_RATE_BIT_NUMBER
HCI_LMP_FEATURE_SCO_LINK_BIT_NUMBER
HCI_LMP_FEATURE_HV2_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_HV3_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_U_LAW_LOG_BIT_NUMBER
HCI_LMP_FEATURE_A_LAW_LOG_BIT_NUMBER
HCI_LMP_FEATURE_CVSD_BIT_NUMBER
HCI_LMP_FEATURE_PAGING_SCHEME_BIT_NUMBER
HCI_LMP_FEATURE_POWER_CONTROL_BIT_NUMBER

Bluetooth Version 1.2

HCI_LMP_FEATURE_ROLE_SWITCH_BIT_NUMBER
HCI_LMP_FEATURE_PARK_STATE_BIT_NUMBER
HCI_LMP_FEATURE_POWER_CONTROL_REQUESTS_
BIT_NUMBER
HCI_LMP_FEATURE_PAGING_PARAMETER_
NEGOTIATION_BIT_NUMBER
HCI_LMP_FEATURE_TRANSPARENT_SYNCHRONOUS_
DATA_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_LEAST_
SIGNIFICANT_BIT_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_MIDDLE_
BIT_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_MOST_
SIGNIFICANT_BIT_BIT_NUMBER
HCI_LMP_FEATURE_BROADCAST_ENCRYPTION_BIT_
NUMBER
HCI_LMP_FEATURE_ENHANCED_INQUIRY_SCAN_BIT_N
UMBER
HCI_LMP_FEATURE_INTERLACED_INQUIRY_SCAN_
BIT_NUMBER
HCI_LMP_FEATURE_INTERLACED_PAGE_SCAN_BIT_
NUMBER
HCI_LMP_FEATURE_RSSI_WITH_INQUIRY_RESULTS_
BIT_NUMBER

HCI_LMP_FEATURE_EXTENDED_SCO_LINKS_EV3_
PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_EV4_PACKETS_BIT_
NUMBER
HCI_LMP_FEATURE_EXTENDED_EV5_PACKETS_BIT_
NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_CAPABLE_
SLAVE_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_
CLASSIFICATION_SLAVE_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_CAPABLE_
MASTER_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_
CLASSIFICATION_MASTER_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_FEATURES_BIT_
NUMBER

Bluetooth Version 2.0

HCI_LMP_FEATURE_ENHANCED_DATA_RATE_
ACL_2_MBPS_MODE_BIT_NUMBER
HCI_LMP_FEATURE_ENHANCED_DATA_RATE_
ACL_3_MBPS_MODE_BIT_NUMBER
HCI_LMP_FEATURE_3_SLOT_ENHANCED_DATA_RATE_
ACL_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_5_SLOT_ENHANCED_DATA_RATE_
ACL_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_ENHANCED_DATA_RATE_ESCO_
2_MBPS_MODE_BIT_NUMBER
HCI_LMP_FEATURE_ENHANCED_DATA_RATE_ESCO_
3_MBPS_MODE_BIT_NUMBER
HCI_LMP_FEATURE_3_SLOT_ENHANCED_DATA_RATE_
ESCO_PACKETS_BIT_NUMBER

Bluetooth Version 2.1

HCI_LMP_FEATURE_SNIFF_SUBRATING_BIT_NUMBER
HCI_LMP_FEATURE_PAUSE_ENCRYPTION_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_INQUIRY_RESPONSE_
BIT_NUMBER
HCI_LMP_FEATURE_SECURE_SIMPLE_PAIRING_BIT_
NUMBER
HCI_LMP_FEATURE_ENCAPSULATED_PDU_BIT_NUMBER
HCI_LMP_FEATURE_ERRONEOUS_DATA_REPORTING_
BIT_NUMBER
HCI_LMP_FEATURE_NON_FLUSHABLE_PACKET_
BOUNDARY_FLAG_BIT_NUMBER
HCI_LMP_FEATURE_LINK_SUPERVISION_TIMEOUT_
CHANGED_EVENT_BIT_NUMBER
HCI_LMP_FEATURE_INQUIRY_RESPONSE_TX_POWER_
LEVEL_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_FEATURES_BIT_NUMBER

Bluetooth Version 3.0

HCI_LMP_FEATURE_ENHANCED_POWER_CONTROL_
BIT_NUMBER

Bluetooth Version 4.0

HCI_LMP_FEATURE_BR_EDR_NOT_SUPPORTED_BIT_
NUMBER
HCI_LMP_FEATURE_LE_SUPPORTED_BIT_NUMBER
HCI_LMP_FEATURE_SIMULTANEOUS_LE_BR_EDR_
TO_SAME_DEVICE_SUPPORTED_BIT_NUMBER

Useful macros defined for manipulation of LMP Features are:

COMPARE_LMP_FEATURES(feats1, feats2)

ASSIGN_LMP_FEATURES(feats, MSByte, … LSByte)

SET_FEATURES_BIT(feats, bitnumb)

RESET_FEATURES_BIT(feats, bitnum)

TEST_FEATURES_BIT(feats, bitnum)

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Buffer_Size

This command reads the maximum size of the data portion of HCI ACL and SCO Data Packets sent from the Host to the Host Controller (i.e., the Host Controller's size limits), and the total number of HCI ACL and SCO Data Packets that can be stored in the data buffers of the Host Controller. The Host must segment the data to be transmitted according to these sizes, so that the HCI Data Packets will contain data with up to these sizes. This command must be issued by the Host before it sends any data to the Host Controller.

Prototype:

int BTPSAPI **HCI_Read_Buffer_Size**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Word_t *HC_ACL_Data_Packet_Length,
    Byte_t *HC_SCO_Data_Packet_Length,
    Word_t *HC_Total_Num_ACL_Data_Packets,
    Word_t *HC_Total_Num_SCO_Data_Packets)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
HC_ACL_Data_Packet_Length	Maximum length (in bytes) of the data portion of each HCI ACL Data Packet passed to the Host Controller.
HC_SCO_Data_Packet_Length	Maximum length (in bytes) of the data portion of each HCI SCO Data Packet passed to the Host Controller.
HC_Total_Num_ACL_Data_Packets	Maximum number of ACL Data Packets that can be stored in the Host Controller.
HC_Total_Num_SCO_Data_Packets	Maximum number of SCO Data Packets that can be stored in the Host Controller.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Country_Code

This command reads the Country_Code parameter, which defines which range of frequency band of the ISM 2.4 GHz band will be used by the device since each country has local regulatory bodies regulating which ISM 2.4 GHz frequency ranges can be used.

Prototype:

int BTPSAPI **HCI_Read_Country_Code**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Country_CodeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Country_CodeResult	Returned Country Code. Possible values are: HCI_COUNTRY_CODE_NORTH_AMERICA_AND_EUROPE HCI_COUNTRY_CODE_FRANCE HCI_COUNTRY_CODE_SPAIN HCI_COUNTRY_CODE_JAPAN HCI_COUNTRY_CODE_NORTH_AMERICA_EUROPE_JAPAN_NOT_FRANCE *(ver 1.1 of Bluetooth)*

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_BD_ADDR

This command reads the BD_ADDR parameter, which is a 48-bit unique identifier for a Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_BD_ADDR**(unsigned int BluetoothStackID, Byte_t *StatusResult,
BD_ADDR_t *BD_ADDRResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.

BD_ADDRResult	The local device's address/identifier.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Local_Supported_Commands

This command reads the list of HCI commands supported for the local device.

Prototype:

int BTPSAPI HCI_Read_Local_Supported_Commands(unsigned int BluetoothStackID,
Byte_t *StatusResult, Supported_Commands_t *Supported_CommandsResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Supported_CommandsResult	Bit mask for each HCI command. The defined bit numbers are:

HCI_SUPPORTED_COMMAND_INQUIRY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_INQUIRY_CANCEL_BIT_NUMBER
HCI_SUPPORTED_COMMAND_PERIODIC_INQUIRY_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_EXIT_PERIODIC_INQUIRY_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_CREATE_CONNECTION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_DISCONNECT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_ADD_SCO_CONNECTION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_CANCEL_CREATE_CONNECTION_BIT_NUMBER

HCI_SUPPORTED_COMMAND_ACCEPT_CONNECTION_REQUEST_BIT_NUMBER
HCI_SUPPORTED_COMMAND_REJECT_CONNECTION_REQUEST_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LINK_KEY_REQUEST_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LINK_KEY_REQUEST_NEGATIVE_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_PIN_CODE_REQUEST_BIT_NUMBER
HCI_SUPPORTED_COMMAND_PIN_CODE_REQUEST_NEGATIVE_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_CHANGE_CONNECTION_PACKET_TYPE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_AUTHENTICATION_REQUEST_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SET_CONNECTION_ENCRYPTION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_CHANGE_CONNECTION_LINK_KEY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_MASTER_LINK_KEY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_REMOTE_NAME_REQUEST_BIT_NUMBER
HCI_SUPPORTED_COMMAND_CANCEL_REMOTE_NAME_REQUEST_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_REMOTE_SUPPORTED_FEATURES_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_REMOTE_EXTENDED_FEATURES_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_REMOTE_VERSION_INFORMATION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_CLOCK_OFFSET_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LMP_HANDLE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_HOLD_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_EXIT_SNIFF_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_PARK_STATE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_EXIT_PARK_STATE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_QOS_SETUP_BIT_NUMBER
HCI_SUPPORTED_COMMAND_ROLE_DISCOVERY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SWITCH_ROLE_BIT_NUMBER

HCI_SUPPORTED_COMMAND_READ_LINK_POLICY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_LINK_POLICY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_DEFAULT_LINK_POLICY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_DEFAULT_LINK_POLICY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_FLOW_SPECIFICATION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SET_EVENT_MASK_BIT_NUMBER
HCI_SUPPORTED_COMMAND_RESET_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SET_EVENT_FILTER_BIT_NUMBER
HCI_SUPPORTED_COMMAND_FLUSH_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_PIN_TYPE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_PIN_TYPE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_CREATE_NEW_UNIT_KEY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_STORED_LINK_KEY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_STORED_LINK_KEY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_DELETE_STORED_LINK_KEY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_LOCAL_NAME_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOCAL_NAME_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_CONNECTION_ACCEPT_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_CONNECTION_ACCEPT_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_PAGE_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_PAGE_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_SCAN_ENABLE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_SCAN_ENABLE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_PAGE_SCAN_ACTIVITY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_PAGE_SCAN_ACTIVITY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_INQUIRY_SCAN_ACTIVITY_BIT_NUMBER

HCI_SUPPORTED_COMMAND_WRITE_INQUIRY_SCAN_ACTIVITY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_AUTHENTICATION_ENABLE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_AUTHENTICATION_ENABLE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_ENCRYPTION_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_ENCRYPTION_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_CLASS_OF_DEVICE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_CLASS_OF_DEVICE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_VOICE_SETTING_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_VOICE_SETTING_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_AUTOMATIC_FLUSH_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_AUTOMATIC_FLUSH_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_NUM_BROADCAST_RETRANSMISSIONS_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_NUM_BROADCAST_RETRANSMISSIONS_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_HOLD_MODE_ACTIVITY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_HOLD_MODE_ACTIVITY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_TRANSMIT_POWER_LEVEL_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_SYNCHRONOUS_FLOW_CONTROL_ENABLE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_SYNCHRONOUS_FLOW_CONTROL_ENABLE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SET_HOST_CONTROLLER_TO_HOST_FLOW_CONTROL_BIT_NUMBER
HCI_SUPPORTED_COMMAND_HOST_BUFFER_SIZE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_HOST_NUMBER_OF_COMPLETED_PACKETS_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LINK_SUPERVISION_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_LINK_SUPERVISION_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_NUMBER_SUPPORTED_IAC_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_CURRENT_IAC_LAP_BIT_NUMBER

HCI_SUPPORTED_COMMAND_WRITE_CURRENT_IAC_
LAP_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_PAGE_SCAN_P
ERIOD_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_PAGE_SCAN_P
ERIOD_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_PAGE_SCAN_M
ODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_PAGE_SCAN_
MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SET_AFH_CHANNEL_
CLASSIFICATION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_INQUIRY_SCAN_T
YPE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_INQUIRY_SCAN_T
YPE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_INQUIRY_MODE_B
IT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_INQUIRY_MODE_
BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_PAGE_SCAN_T
YPE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_PAGE_SCAN_T
YPE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_AFH_CHANNEL_
ASSESSMENT_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_AFH_CHANNEL_A
SSESSMENT_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOCAL_VERSION_I
NFORMATION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOCAL_SUPPORTED_F
EATURES_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOCAL_EXTENDED_F
EATURES_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_BUFFER_SIZE_
BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_COUNTRY_CODE_B
IT_NUMBER
HCI_SUPPORTED_COMMAND_READ_BD_ADDR_BIT_
NUMBER
HCI_SUPPORTED_COMMAND_READ_FAILED_CONTACT_
COUNT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_RESET_FAILED_CONTACT_
COUNT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_GET_LINK_QUALITY_B
IT_NUMBER
HCI_SUPPORTED_COMMAND_READ_RSSI_BIT_N
UMBER
HCI_SUPPORTED_COMMAND_READ_AFH_CHANNEL_
MAP_BIT_NUMBER

HCI_SUPPORTED_COMMAND_READ_BD_CLOCK_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOOPBACK_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_LOOPBACK_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_ENABLE_DEVICE_UNDER_TEST_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SETUP_SYNCHRONOUS_CONNECTION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_ACCEPT_SYNCHRONOUS_CONNECTION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_REJECT_SYNCHRONOUS_CONNECTION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_EXTENDED_INQUIRY_RESPONSE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_EXTENDED_INQUIRY_RESPONSE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_REFRESH_ENCRYPTION_KEY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SNIFF_SUBRATING_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_SIMPLE_PAIRING_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_SIMPLE_PAIRING_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOCAL_OOB_DATA_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_INQUIRY_RESPONSE_TRANSMIT_POWER_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_INQUIRY_TRANSMIT_POWER_LEVEL_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_DEFAULT_ERRONEOUS_DATA_REPORTING_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING_BIT_NUMBER
HCI_SUPPORTED_COMMAND_IO_CAPABILITY_REQUEST_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_USER_CONFIRMATION_REQUEST_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_USER_CONFIRMATION_REQUEST_NEGATIVE_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_USER_PASSKEY_REQUEST_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_USER_PASSKEY_REQUEST_NEGATIVE_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_REMOTE_OOB_DATA_REQUEST_REPLY_BIT_NUMBER

HCI_SUPPORTED_COMMAND_WRITE_SIMPLE_PAIRING_DEBUG_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_ENHANCED_FLUSH_BIT_NUMBER
HCI_SUPPORTED_COMMAND_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SEND_KEYPRESS_NOTIFICATION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_IO_CAPABILITIES_RESPONSE_NEGATIVE_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_ENCRYPTION_KEY_SIZE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_CREATE_PHYSICAL_LINK_BIT_NUMBER
HCI_SUPPORTED_COMMAND_ACCEPT_PHYSICAL_LINK_BIT_NUMBER
HCI_SUPPORTED_COMMAND_DISCONNECT_PHYSICAL_LINK_BIT_NUMBER
HCI_SUPPORTED_COMMAND_CREATE_LOGICAL_LINK_BIT_NUMBER
HCI_SUPPORTED_COMMAND_ACCEPT_LOGICAL_LINK_BIT_NUMBER
HCI_SUPPORTED_COMMAND_DISCONNECT_LOGICAL_LINK_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LOGICAL_LINK_CANCEL_BIT_NUMBER
HCI_SUPPORTED_COMMAND_FLOW_SPEC_MODIFY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOGICAL_LINK_ACCEPT_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_LOGICAL_LINK_ACCEPT_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SET_EVENT_MASK_PAGE_2_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOCATION_DATA_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_LOCATION_DATA_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOCAL_AMP_INFO_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LOCAL_AMP_ASSOC_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_REMOTE_AMP_ASSOC_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_FLOW_CONTROL_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_FLOW_CONTROL_MODE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_DATA_BLOCK_SIZE_BIT_NUMBER

HCI_SUPPORTED_COMMAND_ENABLE_AMP_
    RECEIVER_REPORTS_BIT_NUMBER
HCI_SUPPORTED_COMMAND_AMP_TEST_END_
    BIT_NUMBER
HCI_SUPPORTED_COMMAND_AMP_TEST_COMMAND_
    BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_ENHANCED_
    TRANSMIT_POWER_LEVEL_BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_BEST_EFFORT_
    FLUSH_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_BEST_EFFORT_
    FLUSH_TIMEOUT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_SHORT_RANGE_MODE_
    BIT_NUMBER
HCI_SUPPORTED_COMMAND_READ_LE_HOST_
    SUPPORT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_WRITE_LE_HOST_
    SUPPORT_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_EVENT_MASK_
    BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_READ_BUFFER_
    SIZE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_READ_LOCAL_
    SUPPORTED_FEATURES_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_RANDOM_
    ADDRESS_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_ADVERTISING_
    PARAMETERS_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_READ_ADVERTISING_
    CHANNEL_TX_POWER_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_ADVERTISING_
    DATA_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_SCAN_
    RESPONSE_DATA_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_ADVERTISE_
    ENABLE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_SCAN_
    PARAMETERS_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_SCAN_ENABLE_
    BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_CREATE_
    CONNECTION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_CREATE_
    CONNECTION_CANCEL_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_READ_WHITE_LIST_
    SIZE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_CLEAR_WHITE_LIST_B
    IT_NUMBER
HCI_SUPPORTED_COMMAND_LE_ADD_DEVICE_TO_
    WHITE_LIST_BIT_NUMBER

HCI_SUPPORTED_COMMAND_LE_REMOVE_DEVICE_
FROM_WHITE_LIST_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_CONNECTION_
UPDATE_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_SET_HOST_
CHANNEL_CLASSIFICATION_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_READ_CHANNEL_
MAP_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_READ_REMOTE_
USED_FEATURES_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_ENCRYPT_BIT_
NUMBER
HCI_SUPPORTED_COMMAND_LE_RAND_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_START_ENCRYPTION_
BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_LONG_TERM_KEY_
REQUEST_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_LONG_TERM_KEY_
REQUEST_NEGATIVE_REPLY_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_READ_SUPPORTED_
STATES_BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_RECEIVER_TEST_BIT_
NUMBER
HCI_SUPPORTED_COMMAND_LE_TRANSMITTER_TEST_
BIT_NUMBER
HCI_SUPPORTED_COMMAND_LE_TEST_END_BIT_
NUMBER

Useful macros defined for manipulation of Supported Commands are:

COMPARE_SUPPORTED_COMMANDS(cmd1, cmd2)

SET_SUPPORTED_COMMANDS_BIT(cmd, bitnumb)

RESET_SUPPORTED_COMMANDS_BIT(cmd, bitnum)

TEST_SUPPORTED_COMMANDS_BIT(cmd, bitnum)

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Local_Extended_Features

This command returns the requested page of the extended LMP features.

Note:

Each Page of the LMP Features is 64 bits (0 - 0x3F). If a Feature bit number is larger than 64 bits (0 - 0x3F) then it exists as an "Extended Feature" and exists on a non-zero page. The actual LMP Features page can be found by dividing the bit number by 64 (or (sizeof(LMP_Feature_t)*8).

Note:

Constants are provided below to determine the actual bit number within a Page (HCI_LMP_FEATURE_PAGE_BIT_NUMBER_MASK) and the divisor to apply to the bit numbers to determine the correct page (HCI_LMP_FEATURE_PAGE_NUMBER_DIVISOR).

Prototype:

int BTPSAPI **HCI_Read_Local_Extended_Features**(unsigned int BluetoothStackID, Byte_t PageNumber, Byte_t *StatusResult, Byte_t *Page_NumberResult, Byte_t *Maximum_Page_NumberResult, LMP_Features_t *Extended_LMP_FeaturesResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
PageNumber	Requests the normal LMP features as returned by HCI_Read_Local_Supported_Features (if 0) or the corresponding page of features (non-zero).
StatusResult	Returned HCI status code.
Page_NumberResult	Returned the normal LMP features as returned by HCI_Read_Local_Supported_Features (if 0) or the corresponding page of features (non-zero).
Maximum_Page_NumberResult	The highest features page number which contains non-zero bits for the local device.
Extended_LMP_FeaturesResult	Bit map of requested page of LMP features. Defined bit numbers are (note some of these feature bit numbers are not on page zero – see note above):

Bluetooth Version 1.1

HCI_LMP_FEATURE_THREE_SLOT_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_FIVE_SLOT_PACKETS_BIT_NUMBER

HCI_LMP_FEATURE_ENCRYPTION_BIT_NUMBER
HCI_LMP_FEATURE_SLOT_OFFSET_BIT_NUMBER
HCI_LMP_FEATURE_TIMING_ACCURACY_BIT_NUMBER
HCI_LMP_FEATURE_SWITCH_BIT_NUMBER
HCI_LMP_FEATURE_HOLD_MODE_BIT_NUMBER
HCI_LMP_FEATURE_SNIFF_MODE_BIT_NUMBER
HCI_LMP_FEATURE_PARK_MODE_BIT_NUMBER
HCI_LMP_FEATURE_RSSI_BIT_NUMBER
HCI_LMP_FEATURE_CHANNEL_QUALITY_DRIVEN_
DATA_RATE_BIT_NUMBER
HCI_LMP_FEATURE_SCO_LINK_BIT_NUMBER
HCI_LMP_FEATURE_HV2_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_HV3_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_U_LAW_LOG_BIT_NUMBER
HCI_LMP_FEATURE_A_LAW_LOG_BIT_NUMBER
HCI_LMP_FEATURE_CVSD_BIT_NUMBER
HCI_LMP_FEATURE_PAGING_SCHEME_BIT_NUMBER
HCI_LMP_FEATURE_POWER_CONTROL_BIT_NUMBER

Bluetooth Version 1.2

HCI_LMP_FEATURE_ROLE_SWITCH_BIT_NUMBER
HCI_LMP_FEATURE_PARK_STATE_BIT_NUMBER
HCI_LMP_FEATURE_POWER_CONTROL_REQUESTS_
BIT_NUMBER
HCI_LMP_FEATURE_PAGING_PARAMETER_
NEGOTIATION_BIT_NUMBER
HCI_LMP_FEATURE_TRANSPARENT_SYNCHRONOUS_
DATA_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_LEAST_
SIGNIFICANT_BIT_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_MIDDLE_
BIT_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_MOST_
SIGNIFICANT_BIT_BIT_NUMBER
HCI_LMP_FEATURE_BROADCAST_ENCRYPTION_BIT_
NUMBER
HCI_LMP_FEATURE_ENHANCED_INQUIRY_SCAN_BIT_N
UMBER
HCI_LMP_FEATURE_INTERLACED_INQUIRY_SCAN_
BIT_NUMBER
HCI_LMP_FEATURE_INTERLACED_PAGE_SCAN_BIT_
NUMBER
HCI_LMP_FEATURE_RSSI_WITH_INQUIRY_RESULTS_
BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_SCO_LINKS_EV3_
PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_EV4_PACKETS_BIT_
NUMBER
HCI_LMP_FEATURE_EXTENDED_EV5_PACKETS_BIT_
NUMBER

HCI_LMP_FEATURE_EXTENDED_AFH_CAPABLE_SLAVE_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_CLASSIFICATION_SLAVE_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_CAPABLE_MASTER_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_CLASSIFICATION_MASTER_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_FEATURES_BIT_NUMBER

Bluetooth Version 2.0

HCI_LMP_FEATURE_ENHANCED_DATA_RATE_ACL_2_MBPS_MODE_BIT_NUMBER
HCI_LMP_FEATURE_ENHANCED_DATA_RATE_ACL_3_MBPS_MODE_BIT_NUMBER
HCI_LMP_FEATURE_3_SLOT_ENHANCED_DATA_RATE_ACL_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_5_SLOT_ENHANCED_DATA_RATE_ACL_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_ENHANCED_DATA_RATE_ESCO_2_MBPS_MODE_BIT_NUMBER
HCI_LMP_FEATURE_ENHANCED_DATA_RATE_ESCO_3_MBPS_MODE_BIT_NUMBER
HCI_LMP_FEATURE_3_SLOT_ENHANCED_DATA_RATE_ESCO_PACKETS_BIT_NUMBER

Bluetooth Version 2.1

HCI_LMP_FEATURE_SNIFF_SUBRATING_BIT_NUMBER
HCI_LMP_FEATURE_PAUSE_ENCRYPTION_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_INQUIRY_RESPONSE_BIT_NUMBER
HCI_LMP_FEATURE_SECURE_SIMPLE_PAIRING_BIT_NUMBER
HCI_LMP_FEATURE_ENCAPSULATED_PDU_BIT_NUMBER
HCI_LMP_FEATURE_ERRONEOUS_DATA_REPORTING_BIT_NUMBER
HCI_LMP_FEATURE_NON_FLUSHABLE_PACKET_BOUNDARY_FLAG_BIT_NUMBER
HCI_LMP_FEATURE_LINK_SUPERVISION_TIMEOUT_CHANGED_EVENT_BIT_NUMBER
HCI_LMP_FEATURE_INQUIRY_RESPONSE_TX_POWER_LEVEL_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_FEATURES_BIT_NUMBER
HCI_LMP_FEATURE_SECURE_SIMPLE_PAIRING_HOST_SUPPORT_BIT_NUMBER

Bluetooth Version 3.0

HCI_LMP_FEATURE_ENHANCED_POWER_CONTROL_BIT_NUMBER

Bluetooth Version 4.0

HCI_LMP_FEATURE_BR_EDR_NOT_SUPPORTED_BIT_
NUMBER
HCI_LMP_FEATURE_LE_SUPPORTED_BIT_NUMBER
HCI_LMP_FEATURE_SIMULTANEOUS_LE_BR_EDR_
TO_SAME_DEVICE_SUPPORTED_BIT_NUMBER
HCI_LMP_FEATURE_LE_SUPPORTED_HOST_BIT_NUMBER
HCI_LMP_FEATURE_SIMULTANEOUS_LE_AND_BR_
EDR_TO_SAME_DEVICE_CAPABILE_BIT_NUMBER

Useful macros defined for manipulation of LMP Features are:

COMPARE_LMP_FEATURES(feats1, feats2)

ASSIGN_LMP_FEATURES(feats, MSByte, … LSByte)

SET_FEATURES_BIT(feats, bitnumb)

RESET_FEATURES_BIT(feats, bitnum)

TEST_FEATURES_BIT(feats, bitnum)

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Data_Block_Size

Issues the HCI_Read_Data_Block_Size command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This commands reads information regarding maximum data transfers over the controller and the data buffering that is available. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_ Data_Block_Size** (unsigned int BluetoothStackID,
Byte_t *StatusResult, Word_t *Max_ACL_Data_Packet_LengthResult, Word_t
*Data_Block_LengthResult, Word_t *Total_Num_Data_BlocksResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.
Max_ACL_Data_Packet_LengthResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Max ACL Data Packet Length returned from the Bluetooth device.
Data_Block_LengthResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Data Block Length returned from the Bluetooth device.
Total_Num_Data_BlocksResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Total Number Data Blocks returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.6 Status Parameters

The Status Parameters retrieved via the commands in this section provide information about the current state of the Host Controller, Link Manager, and Baseband. The Host cannot modify any of these parameters other than to reset certain parameters. The API commands available in this section are listed in the table below.

Command	Description
HCI_Read_Failed_Contact_Counter	Read the Failed_Contact_Counter parameter for a particular connection to another device.
HCI_Reset_Failed_Contact_Counter	Reset the Failed_Contact_Counter parameter for a particular connection to another device.

Command	Description
HCI_Get_Link_Quality	Read the Link_Quality for the specified connection.
HCI_Read_RSSI	Read the Received Signal Strength Indication (RSSI) for a connection with another Bluetooth device.
HCI_Read_AFH_Channel_Map	Read AFH channel map.
HCI_Read_Clock	Read local or piconet Bluetooth clock.
HCI_Read_Encryption_Key_Size	Reads the current encryption key size for a specified link.
HCI_Read_Local_AMP_Info	Reads information about the amp controller.
HCI_Read_Local_AMP_ASSOC	Returns a fragment of AMP_ASSOC structure.
HCI_Write_Remote_AMP_ASSOC	Write an AMP_ASSOC fragment to AMP controller.

HCI_Read_Failed_Contact_Counter

This command reads the Failed_Contact_Counter parameter for a particular (ACL) connection to another device. The Failed_Contact_Counter records the number of consecutive incidents in which either the slave or master didn't respond before the flush timeout had expired, and the L2CAP packet that was currently being transmitted was automatically 'flushed'. This counter is reset when the connection is initiated, when the L2CAP packet is acknowledged for that connection, and when the reset command is issued (see next command).

Prototype:

int BTPSAPI **HCI_Read_Failed_Contact_Counter**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult, Word_t *Failed_Contact_CounterResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code (see table in HCI introduction).
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.
Failed_Contact_CounterResult	Number of consecutive failed contacts for this connection.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Reset_Failed_Contact_Counter

Reset the Failed_Contact_Counter parameter for the specified connection.

Prototype:

int BTPSAPI **HCI_Reset_Failed_Contact_Counter**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Get_Link_Quality

This command reads the Link_Quality for the specified connection.

Prototype:

int BTPSAPI **HCI_Get_Link_Quality**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult,
Byte_t *Link_QualityResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.
Link_QualityResult	The current quality of the link between the local and remote devices, range 0 to 255, where higher is better.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_RSSI

This command reads the difference between the measured Received Signal Strength Indication (RSSI) and the limits of the Golden Receive Power Range for an ACL connection to another Bluetooth device. The returned value is how many dB above (if positive) or how many dB below (if negative) the RSSI is relative to the limits. A reading of zero indicates that the RSSI is inside the Golden Receive Power Range.

Prototype:

int BTPSAPI **HCI_Read_RSSI**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult,
Byte_t *RSSIResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.
RSSIResult	Difference between the measured RSSI and the limits of the Golden Receive Power Range. This value may range from -128 to +127 dB.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_AFH_Channel_Map

This command will return the values for the AFH_Mode and AFH_Channel_Map for the specified Connection Handle.

Prototype:

int BTPSAPI **HCI_Read_AFH_Channel_Map**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult,
Byte_t *AFH_ModeResult, AFH_Channel_Map_t *AFH_Channel_MapResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.
AFH_ModeResult	Valued returned for AFH enabled or disabled. Possible values are: HCI_AFH_CHANNEL_ASSESSMENT_MODE_CONTROLLER_ASSESSMENT_DISABLED HCI_AFH_CHANNEL_ASSESSMENT_MODE_CONTROLLER_ASSESSMENT_ENABLED
AFH_Channel_MapResult	If enabled (AFH_ModeResult), this parameter returns a 79 bit field where each bit represents a frequency that is either used or not used in the hopping sequences.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Clock

This command will read the estimate of the value of the Bluetooth Clock.

Prototype:

int BTPSAPI **HCI_Read_Clock**(unsigned int BluetoothStackID, Byte_t Which_Clock, Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult, DWord_t *ClockResult, Word_t *AccuracyResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Which_Clock	Determines if the local clock or the piconet clock is returned. Possible values are: HCI_CLOCK_LOCAL_CLOCK HCI_CLOCK_PICONET_CLOCK
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
StatusResult	Returned HCI status code.
Connection_HandleResult	Unique identifier for the connection handle for which the operation was done.
ClockResult	Bluetooth clock of the device requested.
AccuracyResult	Bluetooth clock error.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Encryption_Key_Size

Issues the HCI_Read_Encryption_Key_Size command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command reads the size of the current encryption key for a specified connection. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Encryption_Key_Size**(unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult, Byte_t *Key_SizeResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Handle of connection that the encryption key size will be read from. This should be for an active ACL connection.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.
Connection_HandleResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Connection Handle returned from the device.
Key_SizeResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Encryption Key Size read from the device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Local_AMP_Info

Issues the HCI_Read_Local_AMP_Info command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command reads information about the AMP controller. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Local_AMP_Info** (unsigned int BluetoothStackID, Byte_t *StatusResult, Byte_t *AMP_StatusResult, DWord_t *Total_BandwidthResult, DWord_t *Max_Guaranteed_BandwidthResult, DWord_t *Min_LatencyResult, DWord_t *Max_PDU_SizeResult, Byte_t *Controller_TypeResult, Word_t *PAL_CapabilitiesResult, Word_t *Max_AMP_ASSOC_LengthResult, DWord_t *Max_Flush_TimeoutResult, DWord_t *Best_Effort_Flush_TimeoutResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.
AMP_StatusResult	If this function returns zero (success) then variable pointed to by this parameter will contain the AMP Status returned from the Bluetooth device. Valid values are 0x00 – 0x06. Consult the Host Controller Interface Function specifications for a full description of the possible meanings for each value. The following is a brief description of the possible values: HCI_AMP_STATUS_AMP_STATUS_AVAILABLE_RADIO_POWERED_DOWN HCI_AMP_STATUS_AMP_STATUS_AVAILABLE_BLUETOOTH_TECHNOLOGY_ONLY HCI_AMP_STATUS_AMP_STATUS_NO_CAPICITY_FOR_BLUETOOTH_OPERATION HCI_AMP_STATUS_AMP_STATUS_LOW_CAPICITY_FOR_BLUETOOTH_OPERATION HCI_AMP_STATUS_AMP_STATUS_MEDIUM_CAPICITY_FOR_BLUETOOTH_OPERATION HCI_AMP_STATUS_AMP_STATUS_HIGH_CAPICITY_FOR_BLUETOOTH_OPERATION HCI_AMP_STATUS_AMP_STATUS_FULL_CAPICITY_FOR_BLUETOOTH_OPERATION

Total_BandwidthResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Total Bandwidth returned from the device. This is an upper bound on the data rate that can be achieved over HCI and accounts for the total bandwidth achieved over the HCI transport. Expressed in kbps.
Max_Guaranteed_BandwidthResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Max Guaranteed Bandwidth returned from the Bluetooth device. This is the maximum bandwidth the AMP controller can quarantee for a single logical link over HCI. Expressed in kbps.
Min_LatencyResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Min Latency returned from the device. This is the minimum latency, in microsenconds, that the AMP controller can quarantee for a logical channel.
Max_PDU_SizeResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Max PDU Size returned from the Bluetooth device. This is the maximum size of an L2CAP PDU that the AMP will accept.
Controller_TypeResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Controller Type returned from the Bluetooth device. Possible values are: HCI_AMP_CONTROLLER_TYPE_CONTROLLER_ TYPE_BR_EDR HCI_AMP_CONTROLLER_TYPE_CONTROLLER_ TYPE_802_11
PAL_CapabilitiesResult	If this function returns zero (success) then variable pointed to by this parameter will contain the PAL Capabilities returned from the Bluetooth device. Possible values are: HCI_AMP_PAL_CAPABILITIES_SERVICE_TYPE_ NOT_GUARANTEED_BIT_VALUE HCI_AMP_PAL_CAPABILITIES_SERVICE_TYPE_ GUARANTEED_BIT_VALUE
Max_AMP_ASSOC_LengthResult	If this function returns zero (success) then variable pointed to by this parameter will contain the MAX Amp ASSOC Length returned from the Bluetooth device. This value will not be larger than: HCI_AMP_ASSOC_FRAGMENT_SIZE_MAXIMUM_ FRAGMENT_SIZE

Max_Flush_TimeoutResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Max Flush Timeout returned from the Bluetooth device.
Best_Effort_Flush_TimeoutResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Max Flush Timeout returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Read_Local_AMP_ASSOC

Issues the HCI_Read_Local_AMP_ASSOC command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command returns a fragment of the AMP_ASSOC structure. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Read_Local_AMP_ASSOC** (unsigned int BluetoothStackID,
    Byte_t Physical_Link_Handle, Word_t Length_So_Far,
    Word_t Max_Remote_AMP_ASSOC_Length, Byte_t
    AMP_ASSOC_Fragment_Buffer_Length, Byte_t *StatusResult,
    Byte_t *Physical_Link_HandleResult,
    Word_t *AMP_ASSOC_Remaining_LengthResult,
    Byte_t *AMP_ASSOC_FragmentLengthResult, Byte_t AMP_ASSOC_FragmentResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Physical_Link_Handle	AMP physical link handle, may be set to 0x00 if command is called outside of physical link creation context.

Length_So_Far	0 for the first AMP_ASSOC fragment, should be incremented by the length of the previous fragment for each call.
Max_Remote_AMP_ASSOC_Length	Max length in octects allowed by host for AMP_ASSOC.
AMP_ASSOC_Fragment_Buffer_Length	Defines the size of the buffer that AMP_ASSOC_FragmentResult points to. This size MUST be at least: HCI_AMP_ASSOC_FRAGMENT_SIZE_ MAXIMUM_FRAGMENT_SIZE bytes long when the calculated remaining length is greater than that value.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.
Physical_Link_HandleResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Physical Link Handle returned by the device.
AMP_ASSOC_Remaining_LengthResult	If this function returns zero (success) then variable pointed to by this parameter will contain the length in octets of the remainder of AMP_ASSOC structure including this fragment.
AMP_ASSOC_FragmentLengthResult	If this function returns zero (success) then variable pointed to by this parameter will contain the AMP_ASSOC_FragmentLength returned from the Bluetooth device.
AMP_ASSOC_FragmentResult	If this function returns zero (success) then variable pointed to by this parameter will contain a fragment of the AMP_ASSOC structure.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Remote_AMP_ASSOC

Issues the HCI_Write_Remote_AMP_ASSOC command to the Bluetooth device that is associated with the Bluetooth Protocol Stack (which itself is specified with the BluetoothStackID parameter). This command writes an AMP_ASSOC fragment to an AMP Controller. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Remote_AMP_ASSOC** (unsigned int BluetoothStackID,
    Byte_t Physical_Link_Handle, Word_t Length_So_Far,
    Word_t AMP_ASSOC_Remaining_Length, Byte_t AMP_ASSOC_Fragment_Length,
    Byte_t *AMP_ASSOC_Fragment, Byte_t *StatusResult,
    Byte_t *Physical_Link_HandleResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Physical_Link_Handle	Handle of physical link that identifies the physical link to be created with associated AMP_ASSOC.
Length_So_Far	0 for the first AMP_ASSOC fragment, should be incremented by the length of the previous fragment for each call.
AMP_ASSOC_Remaining_Length	Length in octets of remainder of AMP_ASSOC including this fragment.
AMP_ASSOC_Fragment_Length	Size of buffer pointed to by AMP_ASSOC_Fragment. This is the fragment size that will be written by this command.
AMP_ASSOC_Fragment	AMP_ASSOC fragment buffer that will be written by this command.
StatusResult	If this function returns zero (success) then variable pointed to by StatusResult will contain the status result returned from the Bluetooth device.
Physical_Link_HandleResult	If this function returns zero (success) then variable pointed to by this parameter will contain the Physical Link Handle returned by the device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.7 Testing Commands

The Testing commands provide the ability to test various functions of the Bluetooth hardware. These commands provide the ability to arrange various conditions for testing. The commands in this section are listed in the table below.

Command	Description
HCI_Read_Loopback_Mode	Read the setting of the Host Controllers Loopback Mode, which determines the path for information.
HCI_Write_Loopback_Mode	Write the setting of the Host Controllers Loopback Mode, which determines the path for information.
HCI_Enable_Device_Under_Test_Mode	Instruct the local Bluetooth module to enter test mode via LMP test commands. The Host issues this command when it wants the local device to be the DUT for the Testing scenarios as described in the Bluetooth Test Mode document.
HCI_Write_Simple_Pairing_Debug_Mode	Instruct the local Bluetooth device to go into Simple Pairing Debug mode.
HCI_Enable_AMP_Receiver_Reports	Used to enable and disable reporting of frames received.
HCI_AMP_Test_End	Used to stop a test scenario in progress.
HCI_AMP_Test_Command	Used to configure and start a test.

HCI_Read_Loopback_Mode

This command reads the setting of the Host Controller’s Loopback Mode, which determines the path of information.

Prototype:

int BTPSAPI **HCI_Read_Loopback_Mode**(unsigned int BluetoothStackID,
Byte_t *StatusResult, Byte_t *Loopback_ModeResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.
Loopback_ModeResult	Current setting of this parameter. Possible values are: HCI_LOOPBACK_MODE_NO_LOOPBACK_MODE HCI_LOOPBACK_MODE_ENABLE_LOCAL_LOOPBACK HCI_LOOPBACK_MODE_ENABLE_REMOTE_LOOPBACK

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Loopback_Mode

This command reads the setting of the Host Controller's Loopback Mode, which determines the path of information. In Non-testing Mode operation, the Loopback Mode is set to Non-testing Mode and the path of the information is as specified by the Bluetooth specifications. In Local Loopback Mode, every Data Packet (ACL and SCO) and Command Packet that is sent from the Host to the Host Controller is sent back with no modifications by the Host Controller.

When the Bluetooth Host Controller enters Local Loopback Mode, it shall respond with four Connection Complete events, one for an ACL channel and three for SCO channels, so that the Host gets connection handles to use when sending ACL and SCO data. When in Local Loopback Mode the Host Controller loops back commands and data to the Host. The Loopback Command event is used to loop back commands that the Host sends to the Host Controller.

If a device is set to Remote Loopback Mode, it will send back all data (ACL and SCO) that comes over the air. In this mode it will only allow a maximum of one ACL connection and three SCO connections – and these must be all to the same remote device.

Prototype:

int BTPSAPI **HCI_Write_Loopback_Mode**(unsigned int BluetoothStackID, Byte_t Loopback_Mode, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Loopback_Mode	Current setting of this parameter. Possible values are: HCI_LOOPBACK_MODE_NO_LOOPBACK_MODE HCI_LOOPBACK_MODE_ENABLE_LOCAL_LOOPBACK HCI_LOOPBACK_MODE_ENABLE_REMOTE_LOOPBACK
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Enable_Device_Under_Test_Mode

This command allows the local Bluetooth module to enter test mode via LMP test commands. The Host issues this command when it wants the local device to be the DUT for the Testing scenarios. After receiving this command, the Host Controller functions as normal until the remote tester issues the LMP test command to place the local device into Device Under Test mode. To disable and exit the Device Under Test Mode, the Host can issue the HCI_Reset command. This command prevents remote Bluetooth devices from causing the local Bluetooth device to enter test mode without first issuing this command.

Prototype:

int BTPSAPI **HCI_Enable_Device_Under_Test_Mode**(unsigned int BluetoothStackID, Byte_t *StatusResult)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	Returned HCI status code.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Write_Simple_Pairing_Debug_Mode

The following function issues the HCI_Write_Simple_Pairing_Debug_Mode Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This command configures the controller to use a predefined Diffie Hellman private key for Simple Pairing debugging. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Write_Simple_Pairing_Debug_Mode**(unsigned int BluetoothStackID,
Byte_t Debug_Mode, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Debug_Mode	Specifies whether to enable (0x01) or disable (0x00) Simple Pairing debug mode.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Enable_AMP_Receiver_Reports

The following function issues the HCI_Enable_AMP_Receiver_Reports Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function is used to enable and disable the reporting of frames received. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_Enable_AMP_Receiver_Reports** (unsigned int BluetoothStackID, Byte_t Enable, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Enable	Specifies whether to enable (0x01) or disable (0x00) the reporting of frames sent.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_AMP_Test_End

The following function issues the HCI_AMP_Test_End Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function is used to stop any test scenario. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_AMP_Test_End** (unsigned int BluetoothStackID, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etAMP_Test_End_Event

etAMP_Receiver_Report_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_AMP_Test_Command

The following function issues the HCI_AMP_Test_Command Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function is used to start and configure a test. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_AMP_Test_Command** (unsigned int BluetoothStackID,
    Byte_t Parameter_Length, Byte_t Parameter_Data[], Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

Parameter_Length	Number of bytes to send from buffer specified by Parameter_Data parameter
Parameter_Data[]	Byte buffer containing the bytes to be sent.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etAMP_Start_Test_Event

etAMP_Test_End_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.8 LE Controller Commands

These commands provide access and control over parts of the LE Bluetooth hardware. The available commands are listed below.

Command	Description
HCI_LE_Set_Event_Mask	Determines which LE events are generated by the host controller.
HCI_LE_Read_Buffer_Size	Reads the maximum size of the data portion of LE ACL Data Packets sent from the host to the controller.
HCI_LE_Read_Local_Supported_Features	Requests the list of the supported LE features of the controller.
HCI_LE_Set_Random_Address	Used by the host to set the LE random device address to be used by the controller.
HCI_LE_Set_Advertising_Parameters	Informs controller of the advertising parameters to utilize.
HCI_LE_Read_Advertising_Channel_Tx_Power	Read the transmit power level for LE advertising packets.

Command	Description
HCI_LE_Set_Advertising_Data	Sets the data used in advertising packets that have a data field.
HCI_LE_Set_Scan_Response	Sets the data used in scanning response packets that have a data field.
HCI_LE_Set_Advertise_Enable	Requests the controller to start or stop advertising.
HCI_LE_Set_Scan_Parameters	Sets the parameters to be used for scanning.
HCI_LE_Set_Scan_Enable	Used to start scanning and find nearby advertising devices.
HCI_LE_Create_Connection	Creates an LE link layer connection to a connectable advertiser.
HCI_LE_Create_Connection_Cancel	Cancels a currently on-going LE connection attempt.
HCI_LE_Read_White_List_Size	Reads total number of entries that can be stored in the white list of the controller.
HCI_LE_Clear_White_List	Clears the white list stored in the controller.
HCI_LE_Add_Device_To_White_List	Adds a single device to the white list.
HCI_LE_Remove_Device_From_White_List	Removes devices from the white list.
HCI_LE_Connection_Update	Used to change the link layer connection parameters of a current connection.
HCI_LE_Set_Host_Channel_Classification	Specifies a channel classification for the data channels to be used.
HCI_LE_Read_Channel_Map	Returns the channel map for a specified connection.
HCI_LE_Read_Remote_Used_Features	Requests a list of the LE features from a remote device.
HCI_LE_Encrypt	Request the controller to encrypt the specified plain-text data.
HCI_LE_Rand	Requests the controller to generate an 8 octet random number.
HCI_LE_Start_Encryption	Starts encryption on a currently authenticated connection.

Command	Description
HCI_LE_Long_Term_Key_Request_Reply	Reply to a LE Long Term Key Request event from the controller.
HCI_LE_Long_Term_Key_Requested_Negative_Reply	Negative Reply to an LE Long Term Key Request event from the controller.
HCI_LE_Read_Supported_States	Reads the states and state combinations that the local link layer supports.
HCI_LE_Reciever_Test	Start a test where the the local controller is put into a mode to receive reference packets.
HCI_LE_Transmitter_Test	Start a test where the local controller generates test reference packets at a fixed interval.
HCI_LE_Test_End	Stop any test which is in currently in progress.

HCI_LE_Set_Event_Mask

The following function issues the HCI_LE_Set_Event_Mask Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter followed by the LE Event Mask to set. This function is used to control which LE events are generated by the controller for the host. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Event_Mask**(unsigned int BluetoothStackID,
Event_Mask_t LE_Event_Mask, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

LE_Event_Mask	Event mask to set for the Host. The bit mask is constructed via the following API macros:
	SET_EVENT_MASK_BIT(Mask, BitNumber)
	RESET_EVENT_MASK_BIT(Mask, BitNumber)
	TEST_EVENT_MASK_BIT(Mask, BitNumber)
	HCI_ENABLE_ALL_LE_EVENTS_IN_EVENT_MASK(Mask)
	HCI_DISABLE_ALL_LE_EVENTS_IN_EVENT_MASK(Mask)

	The bit number constants defined in the API for use with these macros are: HCI_LE_EVENT_MASK_CONNECTION_COMPLETE_BIT_NUMBER HCI_LE_EVENT_MASK_ADVERTISING_REPORT_BIT_NUMBER HCI_LE_EVENT_MASK_CONNECTION_UPDATE_COMPLETE_BIT_NUMBER HCI_LE_EVENT_MASK_READ_REMOTE_USED_FEATURES_COMPLETE_BIT_NUMBER HCI_LE_EVENT_MASK_LONG_TERM_KEY_REQUEST_BIT_NUMBER
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Read_Buffer_Size

The following function issues the HCI_LE_Read_Buffer_Size Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It returns the maximum size of the data field of an LE ACL packet as well as the maximum number of packets the controller can hold. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Read_Buffer_Size**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Word_t *HC_LE_ACL_Data_Packet_Length,
    Byte_t *HC_Total_Num_LE_ACL_Data_Packets);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize,

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
HC_LE_ACL_Data_Packet_Length	Contains the returned maximum length of ACL data packet.
HC_Total_Num_LE_ACL_Data_Packets	Contains the returned total number of data packets the can be stored in the buffers.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Read_Local_Supported_Features

The following function issues the HCI_LE_Read_Local_Supported_Features Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It fetches a list of LE features that a device supports. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Read_Local_Supported_Features**(unsigned int BluetoothStackID, Byte_t *StatusResult, LE_Features_t *LE_FeaturesResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

LE_FeaturesResult	Bit mask list of supported features. Defined bit numbers which are applicable to this function: HCI_LE_FEATURE_LE_ENCRYPTION_BIT_NUMBER
	Useful macros defined for manipulation of LE Features are:
	COMPARE_LE_FEATURES(feats1, feats2)
	ASSIGN_LE_FEATURES(feats, MSByte, … LSByte)

SET_FEATURES_BIT(feats, bitnumb)

RESET_FEATURES_BIT(feats, bitnum)

TEST_FEATURES_BIT(feats, bitnum)

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Set_Random_Address

The following function issues the HCI_LE_Set_Random_Address Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows a host to set the random device address in the Controller. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Random_Address**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Random address to use.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Set_Advertising_Parameters

The following function issues the HCI_LE_Set_Advertising_Parameters Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This allows the host to set the parameters that determine how the controller advertises. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Advertising_Parameters**(unsigned int BluetoothStackID,
    Word_t Advertising_Interval_Min, Word_t Advertising_Interval_Max,
    Byte_t Advertising_Type, Byte_t Own_Address_Type, Byte_t Direct_Address_Type,
    BD_ADDR_t Direct_Address, Byte_t Advertising_Channel_Map,
    Byte_t Advertising_Filter_Policy, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Advertising_Interval_Min	Mininum interval to advertise. Should be in terms of baseband slots (0.625 msec) and should be in the range: HCI_LE_ADVERTISING_INTERVAL_MINIMUM HCI_LE_ADVERTISING_INTERVAL_MAXIMUM
Advertising_Interval_Max	Maximum interval to advertise. Should be greater than or equal to Advertising_Interval_Min, should be in terms of baseband slots (0.625msec), and should be in the range: HCI_LE_ADVERTISING_INTERVAL_MINIMUM HCI_LE_ADVERTISING_INTERVAL_MAXIMUM Both intervals follow the rule: Time = N * 0.625msec
Advertising_Type	Type of advertising to use. Possible values are: HCI_LE_ADVERTISING_TYPE_CONNECTABLE_UNDIRECTED HCI_LE_ADVERTISING_TYPE_CONNECTABLE_DIRECTED HCI_LE_ADVERTISING_TYPE_SCANNABLE_UNDIRECTED HCI_LE_ADVERTISING_TYPE_NON_CONNECTABLE_UNDIRECTED
Own_Address_Type	Address type of local device’s address. Possible values are:

	HCI_LE_ADDRESS_TYPE_PUBLIC HCI_LE_ADDRESS_TYPE_RANDOM
Direct_Address_Type	Address type of directed address (if directed advertising). Possible values are: HCI_LE_ADDRESS_TYPE_PUBLIC HCI_LE_ADDRESS_TYPE_RANDOM
Direct_Address	Address of directed device (if directed advertising).
Advertising_Channel_Map	Indicates which advertising channels to use. Possible values include one or more of the following bit-mask values: HCI_LE_ADVERTISING_CHANNEL_MAP_ENABLE_ CHANNEL_37 HCI_LE_ADVERTISING_CHANNEL_MAP_ENABLE_ CHANNEL_38 HCI_LE_ADVERTISING_CHANNEL_MAP_ENABLE_ CHANNEL_39 Additionally, the following constant can be used to specify all Advertising channels: HCI_LE_ADVERTISING_CHANNEL_MAP_ENABLE_ ALL_CHANNELS
Advertising_Filter_Policy	Policy of which devices to allow requests from. Possible values are: HCI_LE_ADVERTISING_FILTER_POLICY_SCAN_ ANY_CONNECT_ANY HCI_LE_ADVERTISING_FILTER_POLICY_SCAN_ WHITE_LIST_CONNECT_ANY HCI_LE_ADVERTISING_FILTER_POLICY_SCAN_ ANY_CONNECT_WHITE_LIST HCI_LE_ADVERTISING_FILTER_POLICY_SCAN_ WHITE_LIST_CONNECT_WHITE_LIST
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Read_Advertising_Channel_Tx_Power

The following function issues the HCI_LE_Read_Advertising_Channel_Tx_Power Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows the host to read the power level that is used for the transmission of advertising packets. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Read_Advertising_Channel_Tx_Power**(
  unsigned int BluetoothStackID, Byte_t *StatusResult,
  Byte_t *Transmit_Power_LevelResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
Transmit_Power_LevelResult	Contains the returned transmit power level.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Set_Advertising Data

The following function issues the HCI_LE_Set_Advertising_Data to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. Allows a device to set the data it transmits in advertising packets that allows data. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Advertising_Data**(unsigned int BluetoothStackID,
  Byte_t Advertising_Data_Length, Advertising_Data_t *Advertising_Data,
  Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Advertising_Data_Length	Length of advertising data.
Advertising_Data	Actual advertising data.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Set_Scan_Response_Data

The following function issues the HCI_LE_Set_Scan_Response_Data Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows a device to specify the data used in scanning packet responses that allow data. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Scan_Response_Data**(unsigned int BluetoothStackID,
    Byte_t Scan_Response_Data_Length, Scan_Response_Data_t *Scan_Response_Data,
    Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Scan_Response_Data_Length	Length of scan response data.
Scan_Response_Data	Actual scan response data.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Set_Advertise_Enable

The following function issues the HCI_LE_Set_Advertise_Enable Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows a device the ability to enable/disable advertising. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Advertise_Enable**(unsigned int BluetoothStackID,
    Byte_t Advertising_Enable, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Advertising_Enable	Desired value to set. Possible values are: HCI_LE_ADVERTISING_DISABLE HCI_LE_ ADVERTISING_ENABLE
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Set_Scan_Parameters

The following function issues the HCI_LE_Set_Scan_Parameters Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function returns zero if successfull, or a non-zero value if there was an error. If this function returns zero (success) then the StatusResult variable will contain the Status Result returned from the Bluetooth device. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Scan_Parameters**(unsigned int BluetoothStackID,
    Byte_t LE_Scan_Type, Word_t LE_Scan_Interval, Word_t LE_Scan_Window,
    Byte_t Own_Address_Type, Byte_t Scanning_Filter_Policy, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
LE_Scan_Type	Type of scan to perform. Possible values are: HCI_LE_SCAN_TYPE_PASSIVE HCI_LE_SCAN_TYPE_ACTIVE
LE_Scan_Interval	Interval to set between LE scans. Defined as number of baseband slots (0.625 msec). Should be within the range: HCI_LE_SCAN_INTERVAL_MINIMUM to HCI_LE_SCAN_INTERVAL_MAXIMUM
LE_Scan_Window	Value to set duration of an LE scan. Should be defined as number of baseband slots (00625msec), less than or equal to scan window, and within the range as scan window. Both intervals follow the rule: Time = N * 0.625msec
Own_Address_Type	Type of local device's address. Possible values are: HCI_LE_ADDRESS_TYPE_PUBLIC HCI_LE_ADDRESS_TYPE_RANDOM
Scanning_Filter_Policy	Determines which advertising packets to accept. Possible values are: HCI_SCANNING_FILTER_POLICY_ACCEPT_ALL HCI_SCANNING_FILTER_POLICY_ACCEPT_ WHITE_LIST_ONLY
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Set_Scan_Enable

The following function issues the HCI_LE_Set_Scan_Enable Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows a device to enable or disable scanning for advertisering devices. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Scan_Enable**(unsigned int BluetoothStackID,
  Byte_t LE_Scan_Enable, Byte_t Filter_Duplicates, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
LE_Scan_Enable	Enable or disable scanning. Possible values are: HCI_LE_SCAN_ENABLE HCI_LE_SCAN_DISABLE
Filter_Duplicates	Specifies whether duplicate reports should be filtered out. Possible values are: HCI_LE_SCAN_FILTER_DUPLICATES_DISABLED HCI_LE_SCAN_FILTER_DUPLICATES_ENABLED
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Create_Connection

The following function issues the HCI_LE_Create_Connection Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows a device to open a connection to a connectable advertising device. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Create_Connection**(unsigned int BluetoothStackID,
    Word_t LE_Scan_Interval, Word_t LE_Scan_Window, Byte_t Initiator_Filter_Policy,
    Byte_t Peer_Address_Type, BD_ADDR_t Peer_Address, Byte_t Own_Address_Type,
    Word_t Conn_Interval_Min, Word_t Conn_Interval_Max, Word_t Conn_Latency,
    Word_t Supervision_Timeout, Word_t Minimum_CE_Length,
    Word_t Maximum_CE_Length, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
LE_Scan_Interval	Interval to delay between LE scans. Defined as number of baseband slots (0.625 msec). Should be within the range: HCI_LE_SCAN_INTERVAL_MINIMUM to HCI_LE_SCAN_INTERVAL_MAXIMUM
LE_Scan_Window	Value to use for the duration of an LE scan. Should be defined as number of baseband slots (0.625 msec), less than or equal to scan window, and within the range as scan window.
Initiator_Filter_Policy	Determines whether to use a white list. Possible values are: HCI_LE_INITIATOR_FILTER_POLICY_WHITE_LIST_ NOT_USED HCI_LE_INITIATOR_FILTER_POLICY_WHITE_LIST_ IS_USED
Peer_Address_Type	Type of peer address. Possible values are: HCI_LE_ADDRESS_TYPE_PUBLIC HCI_LE_ADDRESS_TYPE_RANDOM
Peer_Address	Address of advertiser to connect if white list is not enabled.
Own_Address_Type	Type of local device address. Possible values are: HCI_LE_ADDRESS_TYPE_PUBLIC HCI_LE_ADDRESS_TYPE_RANDOM

Conn_Interval_Min	Minimum value for the the connection interval. This should fall within the range: HCI_LE_CONNECTION_INTERVAL_MINIMUM HCI_LE_CONNECTION_INTERVAL_MAXIMUM
Conn_Interval_Max	This should be greater than or equal to Conn_Interval_Min and shall fall within the range: HCI_LE_CONNECTION_INTERVAL_MINIMUM HCI_LE_CONNECTION_INTERVAL_MAXIMUM Both intervals follow the rule: Time = N * 1.25 msec
Conn_Latency	Slave latency for connection. This should be in range: HCI_LE_CONNECTION_LATENCY_MINIMUM HCI_LE_CONNECTION_LATENCY_MAXIMUM
Supervision_Timeout	Supervision timeout for LE link. This should be in range: HCI_LE_SUPERVISION_TIMEOUT_MINIMUM HCI_LE_SUPERVISION_TIMEOUT_MAXIMUM The Supervision_Timeout follows the rule: Time = N * 10 msec
Minimum_CE_Length	Information about minimum length of LE connection. This should be in range: HCI_LE_LENGTH_OF_CONNECTION_MINIMUM HCI_LE_LENGTH_OF_CONNECTION_MAXIMUM
Maximum_CE_Length	Information about maximum length of LE connection. Should be in range HCI_LE_LENGTH_OF_CONNECTION_MINIMUM HCI_LE_LENGTH_OF_CONNECTION_MAXIMUM Both CE_Lengths follow the rule: Time = N * 0.625 msec
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

me_Connection_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Create_Connection_Cancel

The following function issues the HCI_LE_Create_Connection_Cancel Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It cancels a currently executing HCI_LE_Create_Connection procedure. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Create_Connection_Cancel**(unsigned int BluetoothStackID,
    Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Read_White_List_Size

The following function issues the HCI_LE_Read_White_List_Size Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows a device to read the total number of devices stored in the white list on the local controller. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Read_White_List_Size**(unsigned int BluetoothStackID,
    Byte_t *StatusResult, Byte_t *White_List_SizeResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
White_List_SizeResult	Contains the returned size of the white list (specified in number of devices).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Clear_White_List

The following function issues the HCI_LE_Clear_White_List Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It clears the white list stored on the Controller. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Clear_White_List**(unsigned int BluetoothStackID,
    Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Add_Device_To_White_List

The following function issues the HCI_LE_Add_Device_To_White_List Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It adds a device to the white list stored on the controller. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Add_Device_To_White_List**(unsigned int BluetoothStackID, Byte_t Address_Type, BD_ADDR_t Address, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Address_Type	Type of address being added. Possible values are: HCI_LE_ADDRESS_TYPE_PUBLIC HCI_LE_ADDRESS_TYPE_RANDOM
Address	Address to of device to add to the white list.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Remove-Device_From_White_List

The following function issues the HCI_LE_Remove_Device_From_White_List Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This command removes a device from the white list stored on the controller. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Remove_Device_From_White_List**(
    unsigned int BluetoothStackID, Byte_t Address_Type, BD_ADDR_t Address,
    Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Address_Type	Type of address being added. Possible values are: HCI_LE_ADDRESS_TYPE_PUBLIC HCI_LE_ADDRESS_TYPE_RANDOM
Address	Address to of device to remove from the white list.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Connection_Update

The following function issues the HCI_LE_Connection_Update Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This command allows the changing of the link layer LE connection parameters between two currently connected Bluetooth LE devices. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Connection_Update**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Word_t Conn_Interval_Min, Word_t Conn_Interval_Max,
Word_t Conn_Latency, Word_t Supervision_Timeout, Word_t Minimum_CE_Length,
Word_t Maximum_CE_Length, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Connection_Handle	Handle to the connection desired to be updated.
Conn_Interval_Min	Minimum value for the the connection interval. This should fall within the range: HCI_LE_CONNECTION_INTERVAL_MINIMUM HCI_LE_CONNECTION_INTERVAL_MAXIMUM
Conn_Interval_Max	This should be greater than or equal to Conn_Interval_Min and shall fall within the range: HCI_LE_CONNECTION_INTERVAL_MINIMUM HCI_LE_CONNECTION_INTERVAL_MAXIMUM Both intervals follow the rule: Time = N * 1.25 msec
Conn_Latency	Slave latency for connection. This should be in range: HCI_LE_CONNECTION_LATENCY_MINIMUM HCI_LE_CONNECTION_LATENCY_MAXIMUM
Supervision_Timeout	Supervision timeout for LE link. This should be in range: HCI_LE_SUPERVISION_TIMEOUT_MINIMUM HCI_LE_SUPERVISION_TIMEOUT_MAXIMUM The Supervision_Timeout follows the rule: Time = N * 10 msec
Minimum_CE_Length	Information about minimum length of LE connection. This should be in range: HCI_LE_LENGTH_OF_CONNECTION_MINIMUM HCI_LE_LENGTH_OF_CONNECTION_MAXIMUM

Maximum_CE_Length	Information about maximum length of LE connection. Should be in range HCI_LE_LENGTH_OF_CONNECTION_MINIMUM HCI_LE_LENGTH_OF_CONNECTION_MAXIMUM Both CE_Lengths follow the rule: Time = N * 0.625 msec
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

me_Connection_Update_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Set_Host_Channel_Classifaction

The following function issues the HCI_LE_Set_Host_Channel_Classification Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows a host to specify a channel classification for data channels. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Set_Host_Channel_Classification**(unsignedint BluetoothStackID,
    LE_Channel_Map_t Channel_Map, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Channel_Map	New channel map to set. It is a 37-bit field where the nth bit represents channel index n. A value of 0 represents the channel is bad (not used). A value of 1 represents the channel is unkown. At least one channel should be marked as unkown.

	Useful macros defined for manipulation of LE Channel Maps are:
	COMPARE_LE_CHANNEL_MAP(map1, map2)
	ASSIGN_LE_CHANNEL_MAP(map, MSByte, …, LSByte)
	SET_LE_CHANNEL_MAP_CHANNEL(map, channum)
	RESET_LE_CHANNEL_MAP_CHANNEL(map, channum)
	TEST_LE_CHANNEL_MAP_CHANNEL(map, channum)

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Read_Channel_Map

The following function issues the HCI_LE_Read_Channel_Map Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. It allows a device to obtain the channel map used for a specified connection. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Read_Channel_Map**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, Byte_t *StatusResult, Word_t *Connection_HandleResult,
    LE_Channel_Map_t *Channel_MapResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Connection_Handle	Handle that identifies the desired connection.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
Connection_HandleResult	Connection handle returned from Bluetooth device.

Channel_MapResult	Returned channel map. It is a 37-bit field where the nth bit represents channel index n. A value of 0 represents the channel is bad (not used). A value of 1 represents the channel is unkown.
	Useful macros defined for manipulation of LE Channel Maps are:
	COMPARE_LE_CHANNEL_MAP(map1, map2)
	ASSIGN_LE_CHANNEL_MAP(map, MSByte, …, LSByte)
	SET_LE_CHANNEL_MAP_CHANNEL(map, channum)
	RESET_LE_CHANNEL_MAP_CHANNEL(map, channum)
	TEST_LE_CHANNEL_MAP_CHANNEL(map, channum)

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Read_Remote_Used_Features

The following function issues the HCI_LE_Read_Remote_Used_Features Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function allows a device to determine the LE features being used by a remote device. The results will be returned in a meRead_Remote_Used_Features_Complete_Event. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Read_Remote_Used_Features**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Connection_Handle	Handle that identifies the desired connection.

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

meRead_Remove_Used_Features_Complete_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Encrypt

The following function issues the HCI_LE_Encrypt Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function allows a device to encrypt plain text data with a specified key. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Encrypt**(unsigned int BluetoothStackID, Encryption_Key_t Key,
Plain_Text_Data_t Plain_Text_Data, Byte_t *StatusResult,
Encrypted_Data_t *Encrypted_DataResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Key	128 bit encryption key.
Plain_Text_Data	128 bit data block to be encrypted.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
Encrypted_DataResult	128 bit encrypted data block.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Rand

The following function issues the HCI_LE_Rand Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This allows the host to request 64 bits of randomly generated data (e.g. a 64 bit random number). Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Rand**(unsigned int BluetoothStackID, Byte_t *StatusResult,
    Random_Number_t *Random_NumberResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
Random_NumberResult	64-bit random number generated from the controller.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Start_Encryption

The following function issues the HCI_LE_Start_Encryption Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function is used to authenticate the encryption key associated with the given connection. Once authenticated, it will encrypt, or re-encrypt if already encrypted, the link.

Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Start_Encryption**(unsigned int BluetoothStackID,
    Word_t Connection_Handle, Random_Number_t Random_Number,
    Word_t Encrypted_Diversifier, Long_Term_Key_t Long_Term_Key,
    Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Connection_Handle	Handle used to identify the desired connection.
Random_Number	64 bit random number to use during the encryption process.
Encrypted_Diversifier	16-bit encrypted diversifier.
Long_Term_Key	128-bit long term key.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Possible Events:

etEncyrption_Key_Refresh_Complete_Event

etEncryption_Change_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Long_Term_Key_Request_Reply

The following function issues the HCI_LE_Long_Term_Key_Request_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This command is used in response to a meLong_Term_Key_Request_Event. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Long_Term_Key_Request_Reply**(unsigned intBluetoothStackID,
Word_t Connection_Handle, Long_Term_Key_t Long_Term_Key, Byte_t *StatusResult,
Word_t *Connection_HandleResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Connection_Handle	Handle used ot identify the desired connection.
Long_Term_Key	128-bit long term key.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
Connection_HandleResult	Returned connection handle.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Long_Term_Key_Request_Negative_Key_Reply

The following function issues the HCI_LE_Long_Term_Key_Request_Negative_Reply Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function is used in reply to a meLong_Term_Key_Request_Event if the host cannot (or does not want to) provide a long term key for this connection. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Long_Term_Key_Request_Negative_Reply**(
unsigned int BluetoothStackID, Word_t Connection_Handle, Byte_t *StatusResult,
Word_t *Connection_HandleResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Connection_Handle	Handle used to identify the desired connection.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
Connection_HandleResult	Returned connection handle.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Read_Supported_States

The following function issues the HCI_LE_Read_Supported_States Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function reads the supported channels and combinations that the link layer supports. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Read_Supported_States**(unsigned int BluetoothStackID,
Byte_t *StatusResult, LE_States_t *LE_StatesResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.
LE_StatesResult	Returned supported LE states. These states are represented as a bit mask. The following macro's can be used to manipulate the LE states mask.:

ASSIGN_LE_STATES(Mask, MSByte, …, LSByte)

COMPARE_LE_STATES(Mask1, Mask2)

SET_LE_STATES_BIT (Mask, BitNumber)

RESET_LE_STATES_BIT (Mask, BitNumber)

TEST_LE_STATES_BIT(Mask, BitNumber)

The bit number constants defined in the API for use with these macros are:

HCI_LE_STATES_NON_CONNECTABLE_ADVERTISING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_SCANNABLE_ADVERTISING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_CONNECTABLE_ADVERTISING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_DIRECTED_ADVERTISING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_PASSIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_ACTIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_INITIATING_STATE_MASTER_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_CONNECTION_STATE_SLAVE_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_NON_CONNECTABLE_ADVERTISING_PASSIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_SCANNABLE_ADVERTISING_PASSIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_CONNECTABLE_ADVERTISING_PASSIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_DIRECTED_ADVERTISING_PASSIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_NON_CONNECTABLE_ADVERTISING_ACTIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER

HCI_LE_STATES_SCANNABLE_ADVERTISING_ACTIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_CONNECTABLE_ADVERTISING_ACTIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_DIRECTED_ADVERTISING_ACTIVE_SCANNING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_NON_CONNECTABLE_ADVERTISING_INITIATING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_NON_SCANNABLE_ADVERTISING_INITIATING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_NON_CONNECTABLE_ADVERTISING_STATE_MASTER_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_SCANNABLE_ADVERTISING_STATE_MASTER_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_NON_CONNECTABLE_ADVERTISING_STATE_SLAVE_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_SCANNABLE_ADVERTISING_STATE_SLAVE_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_PASSIVE_SCANNING_INITITIATING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_ACTIVE_SCANNING_INITITIATING_STATE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_PASSIVE_SCANNING_STATE_MASTER_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_ACTIVE_SCANNING_STATE_MASTER_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_PASSIVE_SCANNING_STATE_SLAVE_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_ACTIVE_SCANNING_STATE_SLAVE_ROLE_SUPPORTED_BIT_NUMBER
HCI_LE_STATES_INITIATING_STATE_MASTER_ROLE_MASTER_ROLE_MASTER_ROLE_SUPPORTED_BIT_NUMBER

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Receiver_Test_Command

The following function issues the HCI_LE_Receiver_Test Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This function starts a test in which the the local device receives packets at a fixed interval. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Receiver_Test**(unsigned int BluetoothStackID,
    Byte_t RX_Frequency, Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
RX_Frequency	Frequency to receive packets, Where N (RX_Frequency) = (F - 2402) / 2. This value should be in the range: HCI_LE_RECEIVER_TRANSMITTER_TEST_ FREQUENCY_MINIMUM HCI_LE_RECEIVER_TRANSMITTER_TEST_ FREQUENCY_MAXIMUM
StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device,

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Transmitter_Test

The following function issues the HCI_LE_Transmitter_Test Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. This command runs a test in which the local device transmits test packets at a fixed interval. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Transmitter_Test**(unsigned int BluetoothStackID,
Byte_t TX_Frequency, Byte_t Length_Of_Test_Data, Byte_t Packet_Payload,
Byte_t *StatusResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TX_Frequency	Frequency to receive packets, Where N(TX_Frequency) = (F - 2402) / 2. This value should be in the range: HCI_LE_RECEIVER_TRANSMITTER_TEST_FREQUENCY_MINIMUM HCI_LE_RECEIVER_TRANSMITTER_TEST_FREQUENCY_MAXIMUM
Length_Of_Test_Data	Length in bytes of payload data in each packet. This value should be in the range: HCI_LE_TRANSMITTER_TEST_LENGTH_OF_TEST_DATA_MINIMUM_LENGTH HCI_LE_TRANSMITTER_TEST_LENGTH_OF_TEST_DATA_MAXIMUM_LENGTH
Packet_Payload	Description of the transmitted test pattern. The possible values are: HCI_LE_TRANSMITTER_TEST_PAYLOAD_PSEUDO_RANDOM_BIT_SEQUENCE_9 HCI_LE_TRANSMITTER_TEST_PAYLOAD_PATTERN_ALTERNATING_BITS_0xF0 HCI_LE_TRANSMITTER_TEST_PAYLOAD_PATTERN_ALTERNATING_BITS_0xAA HCI_LE_TRANSMITTER_TEST_PAYLOAD_PSEUDO_RANDOM_BIT_SEQUENCE_15 HCI_LE_TRANSMITTER_TEST_PAYLOAD_PATTERN_ALL_1_BITS HCI_LE_TRANSMITTER_TEST_PAYLOAD_PATTERN_ALL_0_BITS HCI_LE_TRANSMITTER_TEST_PAYLOAD_PATTERN_ALTERNATING_BITS_0x0F HCI_LE_TRANSMITTER_TEST_PAYLOAD_PATTERN_ALTERNATING_BITS_0x55

StatusResult	If function returns zero (success) this variable will contain the Status Result returned from the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_LE_Test_End

The following function issues the HCI_LE_Test_End Command to the Bluetooth device that is associated with the Bluetooth Protocol Stack specified by the BluetoothStackID parameter. Note, this function blocks until either a result is returned from the Bluetooth device OR the function times out waiting for a response from the Bluetooth device.

Prototype:

int BTPSAPI **HCI_LE_Test_End**(unsigned int BluetoothStackID, Byte_t *StatusResult,
    Word_t *Number_Of_PacketsResult);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Number_Of_PacketsResult	Number of packets received (0x0000 for a transmitter test).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.9 Miscellaneous Commands/Parameters

These are commands and parameters which are not called out in the Bluetooth specifications, but are needed to facilitate operation of the Bluetooth Protocol Stack. The commands in this section are listed in the table below.

Command	Description
HCI_Version_Supported	Read the HCI version supported by the HCI API layer.
HCI_Command_Supported	Allows caller mechanism to determine if a specific HCI function is supported by the HCI API layer present for specified Bluetooth protocol stack.
HCI_Send_Raw_Command	Issue a raw HCI command to the specified Bluetooth device.
HCI_Send_ACL_Data	Send HCI ACL packets to a Bluetooth device.
HCI_Send_SCO_Data	Send HCI SCO packets to a Bluetooth device.
HCI_Change_SCO_Configuration	Set SCO data delivery via HCI channel enabled or disabled.
HCI_Reconfigure_Driver	Request HCI Driver reconfiguration process.
HCI_Set_Host_Flow_Control	Configures the Controller to Host Flow Control configuration.
HCI_Query_Host_Flow_Control	Queries the Controller to Host Flow Control configuration.

HCI_Version_Supported

This command reads the HCI version which is supported by the HCI API layer.

Prototype:

int BTPSAPI **HCI_Version_Supported**(unsigned int BluetoothStackID,
    HCI_Version_t *HCI_Version);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
HCI_Version	A returned enumerated type, where higher levels of Bluetooth specification revised are assured of having a higher ordinal value in the enumeration. Possible values are: hvSpecification_1_0B hvSpecification_1_1 hvSpecification_1_2 hvSpecification_2_0 hvSpecification_2_1 hvSpecification_3_0 hvSpecification_4_0

which represent ver 1.0B, ver 1.1, ver 1.2, ver 2.0, ver 2.1, ver 3.0, and ver 4.0 of the Bluetooth specification, respectively.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Command_Supported

This function allows the caller to determine if a specified HCI function is present in the HCI API layer of a specified Bluetooth protocol stack. This function should be used instead of making a call to HCI_Read_Local_Supported_Commands.

Prototype:

int BTPSAPI **HCI_Command_Supported**(unsigned int BluetoothStackID,
    unsigned int SupportedCommandBitNumber);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SupportedCommandBitNumber	Supported HCI Command bit number (defined in HCITypes.h) for the specified HCI command that is to be tested. See description of HCI_Read_Local_Supported_Commands() function for more information on this parameter.

Return:

Positive, non-zero, value if the HCI command is supported.

Zero if the HCI command is NOT supported.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Send_Raw_Command

Issue a raw HCI command to the specified Bluetooth device.

Prototype:

int BTPSAPI **HCI_Send_Raw_Command**(unsigned int BluetoothStackID,
    Byte_t Command_OGF, Word_t Command_OCF, Byte_t Command_Length,
    Byte_t Command_Data[], Byte_t *StatusResult, Byte_t *LengthResult,
    Byte_t *BufferResult, Boolean_t WaitForResponse);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Command_OGF	Opcode Group Field value – upper 6 bits of the opcode field (e.g., 0x01 for Link Control commands).
Command_OCF	Opcode Command Field value – lower 10 bits of opcode.
Command_Length	Length of the valid data in Command_Data.
Command_Data	Array of bytes that make up the command
StatusResult	Pointer to a byte to receive a returned status.
LengthResult	This parameter is both an input and output parameter. On input this parameter should contain the total length (in bytes) of the buffer that is pointed to by the BufferResult parameter. On successful return from this function this will contain the length of the valid data returned in the BufferResult.
BufferResult	Pointer to an array of bytes for the command result.
WaitForResponse	TRUE if the function should wait for the result, FALSE otherwise.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Send_ACL_Data

Send HCI ACL data packets to a Bluetooth device. Caller is not responsible for formatting an HCI ACL data packet, this is handled by the API.

Prototype:

int BTPSAPI **HCI_Send_ACL_Data**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Word_t Flags, Word_t ACLDataLength, Byte_t *ACLData)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Flags	Used along with the connection_Handle to define the header of the HCI ACL Data Packet. Possible values are: **Bluetooth Version 1.1** HCI_ACL_FLAGS_PACKET_BOUNDARY_CONTINUE_PACKET HCI_ACL_FLAGS_PACKET_BOUNDARY_FIRST_PACKET HCI_ACL_FLAGS_PACKET_BOUNDARY_FIRST_PACKET_AUTO_FLUSHABLE **Bluetooth Version 2.1** HCI_ACL_FLAGS_PACKET_BOUNDARY_FIRST_PACKET_NON_FLUSHABLE HCI_ACL_FLAGS_PACKET_BOUNDARY_COMPLETE_L2CAP_PDU_AUTO_FLUSHABLE These definitions are for Packets from Host to Host Controller. **Bluetooth Version 1.1** HCI_ACL_FLAGS_PACKET_BROADCAST_NO_BROADCAST HCI_ACL_FLAGS_PACKET_BROADCAST_ACTIVE_BROADCAST HCI_ACL_FLAGS_PACKET_BROADCAST_PICONET_BROADCAST **Bluetooth Version 1.2** HCI_ACL_FLAGS_PACKET_BROADCAST_ACTIVE_SLAVE_BROADCAST HCI_ACL_FLAGS_PACKET_BROADCAST_PARKED_SLAVE_BROADCAST

	These definitions are for Packets from Host Controller to Host. HCI_ACL_FLAGS_PACKET_BROADCAST_POINT_TO_POINT HCI_ACL_FLAGS_PACKET_BROADCAST_ACTIVE_SLAVE HCI_ACL_FLAGS_PACKET_BROADCAST_PARKED_SLAVE
ACLDataLength	Length of the data pointed to by ACLData
ACLData	Pointer to the data to be sent.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_MEMORY_ALLOCATION_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Send_SCO_Data

Send HCI SCO data packets to a Bluetooth device. Caller is not responsible for formatting an HCI SCO/eSCO data packet, this is handled by the API.

Prototype:

int BTPSAPI **HCI_Send_SCO_Data**(unsigned int BluetoothStackID,
Word_t Connection_Handle, Word_t Flags, Word_t SCODataLength, Byte_t *SCOData)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Flags	Currently not used. Set to zero.
SCODataLength	Length of the data pointed to by SCOData
SCOData	Pointer to the data to be sent.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_HCI_DRIVER_ERROR
BTPS_ERROR_MEMORY_ALLOCATION_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Change_SCO_Configuration

This function issues the appropriate call to an HCI driver to set SCO data delivery via the HCI channel to be enabled or disabled.

Prototype:

int BTPSAPI **HCI_Change_SCO_Configuration**(unsigned int BluetoothStackID,
    HCI_SCOConfiguration_t SCOConfiguration)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOConfiguration	HCI SCO Configuration to set the device to. This valus is one of: hscNoChannels hscOneChannel8BitVoice hscOneChannel16BitVoice hscTwoChannel8BitVoice hscTwoChannel16BitVoice hscThreeChannel8BitVoice hscThreeChannel16BitVoice

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Reconfigure_Driver

This function issues the appropriate call to an HCI driver to request the HCI Driver to reconfigure itself with the corresponding configuration information.

Prototype:

```
int BTPSAPI HCI_Reconfigure_Driver(unsigned int BluetoothStackID,
    Boolean_t ResetStateMachines,
    HCI_Driver_Reconfigure_Data_t *DriverReconfigureData)
```

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
ResetStatemachines	Flag which is passed to the drivers that specifies whether the HCI driver internal state machines (for example, BCSP and/or packet building state machines) should be reset (TRUE) or not (FALSE).
DriverReconfigureData	HCI Driver Reconfiguration information. This structure has the following format:

```
typedef struct
{
  DWord_t    ReconfigureCommand;
  void         *ReconfigureData;
} HCI_Driver_Reconfigure_Data_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Set_Host_Flow_Control

This function issues the appropriate call to HCI Commands to configure the Controller to Host Flow Control configuration. Controller to Host Flow Control used to limit the number of ACL or SCO Data Packets that the Controller can send to the Host without using credits back.

Notes:

Once this function is called to enable Controller to Host Flow Control, all the handling of this mechanism will be handled internally.

If the NumberOfACLPackets and NumberOfSCOPackets are both set to ZERO then Controller to Host Flow Control will be disabled.

Prototype:

int BTPSAPI **HCI_Set_Host_Flow_Control** (unsigned int BluetoothStackID,
Word_t NumberOfACLPackets,
Word_t NumberOfSCOPackets)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
NumberOfACLPackets	The number of ACL packets the Controller can send to the Host without receiving credits back.
NumberOfSCOPackets	The number of SCO packets the Controller can send to the Host without receiving credits back.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Query_Host_Flow_Control

This function queries the Controller to Host Flow Control configuration. Controller to Host Flow Control used to limit the number of ACL or SCO Data Packets that the Controller can send to the Host without using credits back.

Prototype:

int BTPSAPI **HCI_Query_Host_Flow_Control** (unsigned int BluetoothStackID,
Word_t *NumberOfACLPackets,
Word_t *NumberOfSCOPackets)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

NumberOfACLPackets	Pointer to return the number of ACL packets the Controller can send to the Host without receiving credits back.
NumberOfSCOPackets	Pointer to return the number of SCO packets the Controller can send to the Host without receiving credits back.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.10 HCI Event/Data Callbacks and Registration

In order to receive HCI Events or ACL Data, one must register callback functions with the HCI portion of the stack. The HCI Event callbacks are called whenever the appropriate HCI event trigger occurs, such as at the completion of an inquiry or when a connection is made. The ACL Data callbacks are called whenever a complete ACL packet arrives. Below are the descriptions of the Prototypes for these two callbacks, followed by the functions used to register these callbacks with the HCI portion of the stack.

HCI_Event_Callback_t

The following declared type represents thePrototype Function for an HCI Event receive callback. This function will be called whenever a complete HCI Event Packet has been received by the HCI Layer that is associated with the specified Bluetooth stack. The caller is free to use the contents of the HCI Event 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 Needs to be noted however, that if the same Callback is installed more than once, then the callbacks will be called serially. Because of this, the processing in this function should be as efficient as possible. It should also 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 another HCI Event Packet will not be processed while this function call is outstanding). ***NOTE***: This function MUST NOT Block and wait for events that can only be satisfied by receiving HCI Event Packets. A deadlock WILL occur because NO HCI Event receive callbacks will be issued while this function is currently outstanding.

Prototype:

void (BTPSAPI ***HCI_Event_Callback_t**)(unsigned int BluetoothStackID,
HCI_Event_Data_t *HCI_Event_Data, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
HCI_Event_Data	A structure which contains a union of all event data structures possible. This structure is defined as follows:

```
typedef struct
{
  HCI_Event_Type_t    Event_Data_Type;
  Word_t              Event_Data_Size;
  union
  {
    HCI_Inquiry_Complete_Event_Data_t
             *HCI_Inquiry_Complete_Event_Data;
    HCI_Inquiry_Result_Event_Data_t
             *HCI_Inquiry_Result_Event_Data;
    HCI_Connection_Complete_Event_Data_t
             *HCI_Connection_Complete_Event_Data;
    HCI_Connection_Request_Event_Data_t
             *HCI_Connection_Request_Event_Data;
    HCI_Disconnection_Complete_Event_Data_t
             *HCI_Disconnection_Complete_Event_Data;
    HCI_Authentication_Complete_Event_Data_t
             *HCI_Authentication_Complete_Event_Data;
    HCI_Remote_Name_Request_Complete_Event_Data_t
             *HCI_Remote_Name_Request_Complete_Event_Data;
    HCI_Encryption_Change_Event_Data_t
             *HCI_Encryption_Change_Event_Data;
    HCI_Change_Connection_Link_Key_Complete_Event_Data_t
             *HCI_Change_Connection_Link_Key_Complete_Event_Data;
    HCI_Master_Link_Key_Complete_Event_Data_t
             *HCI_Master_Link_Key_Complete_Event_Data;
    HCI_Read_Remote_Supported_Features_Complete_Event_Data_t
             *HCI_Read_Remote_Supported_Features_Complete_Event_Data;
    HCI_Read_Remote_Version_Information_Complete_Event_Data_t
             *HCI_Read_Remote_Version_Information_Complete_Event_Data;
    HCI_QoS_Setup_Complete_Event_Data_t
             *HCI_QoS_Setup_Complete_Event_Data;
    HCI_Hardware_Error_Event_Data_t
             *HCI_Hardware_Error_Event_Data;
    HCI_Flush_Occurred_Event_Data_t
             *HCI_Flush_Occurred_Event_Data;
    HCI_Role_Change_Event_Data_t
             *HCI_Role_Change_Event_Data;
    HCI_Number_Of_Completed_Packets_Event_Data_t
             *HCI_Number_Of_Completed_Packets_Event_Data;
```

```
HCI_Mode_Change_Event_Data_t
         *HCI_Mode_Change_Event_Data;
HCI_Return_Link_Keys_Event_Data_t
         *HCI_Return_Link_Keys_Event_Data;
HCI_PIN_Code_Request_Event_Data_t
         *HCI_PIN_Code_Request_Event_Data;
HCI_Link_Key_Request_Event_Data_t
         *HCI_Link_Key_Request_Event_Data;
HCI_Link_Key_Notification_Event_Data_t
         *HCI_Link_Key_Notification_Event_Data;
HCI_Loopback_Command_Event_Data_t
         *HCI_Loopback_Command_Event_Data;
HCI_Data_Buffer_Overflow_Event_Data_t
         *HCI_Data_Buffer_Overflow_Event_Data;
HCI_Max_Slots_Change_Event_Data_t
         *HCI_Max_Slots_Change_Event_Data;
HCI_Read_Clock_Offset_Complete_Event_Data_t
         *HCI_Read_Clock_Offset_Complete_Event_Data;
HCI_Connection_Packet_Type_Changed_Event_Data_t
         *HCI_Connection_Packet_Type_Changed_Event_Data;
HCI_QoS_Violation_Event_Data_t
         *HCI_QoS_Violation_Event_Data;
HCI_Page_Scan_Repetition_Mode_Change_Event_Data_t
         *HCI_Page_Scan_Repetition_Mode_Change_Event_Data;
HCI_Page_Scan_Mode_Change_Event_Data_t
         *HCI_Page_Scan_Mode_Change_Event_Data;
HCI_Flow_Specification_Complete_Event_Data_t
         *HCI_Flow_Specification_Complete_Event_Data;
HCI_Inquiry_Result_With_RSSI_Event_Data_t
         *HCI_Inquiry_Result_With_RSSI_Event_Data;
HCI_Read_Remote_Extended_Features_Complete_Event_Data_t
         *HCI_Read_Remote_Extended_Features_Complete_Event_Data;
HCI_Synchronous_Connection_Complete_Event_Data_t
         *HCI_Synchronous_Connection_Complete_Event_Data;
HCI_Synchronous_Connection_Changed_Event_Data_t
         *HCI_Synchronous_Connection_Changed_Event_Data;
HCI_Sniff_Subrating_Event_Data_t
         *HCI_Sniff_Subrating_Event_Data;
HCI_Extended_Inquiry_Result_Event_Data_t
         *HCI_Extended_Inquiry_Result_Event_Data;
HCI_Encryption_Key_Refresh_Complete_Event_Data_t
         *HCI_Encryption_Key_Refresh_Complete_Event_Data;
HCI_IO_Capability_Request_Event_Data_t
         *HCI_IO_Capability_Request_Event_Data;
HCI_IO_Capability_Response_Event_Data_t
         *HCI_IO_Capability_Response_Event_Data;
HCI_User_Confirmation_Request_Event_Data_t
         *HCI_User_Confirmation_Request_Event_Data;
HCI_User_Passkey_Request_Event_Data_t
         *HCI_User_Passkey_Request_Event_Data;
```

```
      HCI_Remote_OOB_Data_Request_Event_Data_t
               *HCI_Remote_OOB_Data_Request_Event_Data;
      HCI_Simple_Pairing_Complete_Event_Data_t
               *HCI_Simple_Pairing_Complete_Event_Data;
      HCI_Link_Supervision_Timeout_Changed_Event_Data_t
               *HCI_Link_Supervision_Timeout_Changed_Event_Data;
      HCI_Enhanced_Flush_Complete_Event_Data_t
               *HCI_Enhanced_Flush_Complete_Event_Data;
      HCI_User_Passkey_Notification_Event_Data_t
               *HCI_User_Passkey_Notification_Event_Data;
      HCI_Keypress_Notification_Event_Data_t
               *HCI_Keypress_Notification_Event_Data;
      HCI_Remote_Host_Supported_Features_Notification_Event_Data_t
               *HCI_Remote_Host_Supported_Features_Notification_Event_Data;
      HCI_Physical_Link_Complete_Event_Data_t
               *HCI_Physical_Link_Complete_Event_Data;
      HCI_Channel_Selected_Event_Data_t
               *HCI_Channel_Selected_Event_Data;
      HCI_Disconnection_Physical_Link_Complete_Event_Data_t
               *HCI_Disconnection_Physical_Link_Complete_Event_Data;
      HCI_Physical_Link_Loss_Early_Warning_Event_Data_t
               *HCI_Physical_Link_Loss_Early_Warning_Event_Data;
      HCI_Physical_Link_Recovery_Event_Data_t
               *HCI_Physical_Link_Recovery_Event_Data;
      HCI_Logical_Link_Complete_Event_Data_t
               *HCI_Logical_Link_Complete_Event_Data;
      HCI_Disconnection_Logical_Link_Complete_Event_Data_t
               *HCI_Disconnection_Logical_Link_Complete_Event_Data;
      HCI_Flow_Spec_Modify_Complete_Event_Data_t
               *HCI_Flow_Spec_Modify_Complete_Event_Data;
      HCI_Number_Of_Completed_Data_Blocks_Event_Data_t
               *HCI_Number_Of_Completed_Data_Blocks_Event_Data;
      HCI_Short_Range_Mode_Change_Complete_Event_Data_t
               *HCI_Short_Range_Mode_Change_Complete_Event_Data;
      HCI_AMP_Status_Change_Event_Data_t
               *HCI_AMP_Status_Change_Event_Data;
      HCI_AMP_Start_Test_Event_Data_t
               *HCI_AMP_Start_Test_Event_Data;
      HCI_AMP_Test_End_Event_Data_t
               *HCI_AMP_Test_End_Event_Data;
      HCI_AMP_Receiver_Report_Event_Data_t
               *HCI_AMP_Receiver_Report_Event_Data;
      HCI_LE_Meta_Event_Data_t
               *HCI_LE_Meta_Event_Data;
      HCI_Platform_Specific_Event_Data_t
               *HCI_Platform_Specific_Event_Data;
      void
               *HCI_Unknown_Event_Data;
   } Event_Data;
} HCI_Event_Data_t;
```

	where, HCI_Event_Type_t is an enumeration of the event types listed in the table in section 2.2.11, and each data structure in the union is described with its event in that section as well.
CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

HCI_ACL_Data_Callback_t

The following declared type represents thePrototype Function for an ACL Data Receive Data Callback. This function will be called whenever a complete ACL Data Packet has been received by the HCI Layer that is associated with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the ACL Data that was received and the HCI ACL Data Callback Parameter that was specified when this Callback was installed. The caller is free to use the ACL Data Contents **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 needs to be noted however, that if the same Callback is installed more that once, then the callbacks will be called serially. Because of this, the processing in this function should be as efficient as possible. It should also 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 another ACL Data Packet will not be processed while this function call is outstanding).

Prototype:

void (BTPSAPI ***HCI_ACL_Data_Callback_t**)(unsigned int BluetoothStackID, Word_t Connection_Handle, Word_t Flags, Word_t ACLDataLength, Byte_t *ACLData, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Flags	ACL Packet Flags.
ACLDataLength	Number of bytes returned in the array pointed to by ACLData.
ACLData	Pointer to the ACL data.
CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

HCI_SCO_Data_Callback_t

The following declared type represents thePrototype Function for an SCO Data Receive Data Callback. This function will be called whenever a complete SCO Data Packet has been received by the HCI Layer that is associated with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the SCO Data that was received and the HCI SCO Data Callback Parameter that was specified when this Callback was installed. The caller is free to use the SCO Data Contents **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 needs to be noted however, that if the same Callback is installed more that once, then the callbacks will be called serially. Because of this, the processing in this function should be as efficient as possible. It should also 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 another SCO Data Packet will not be processed while this function call is outstanding).

Prototype:

void (BTPSAPI ***HCI_SCO_Data_Callback_t**)(unsigned int BluetoothStackID,
  Word_t Connection_Handle, Word_t Flags, Byte_t SCODataLength, Byte_t *SCOData,
  unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Flags	For future use.
SCODataLength	Number of bytes returned in the array pointed to by SCOData.
SCOData	Pointer to the SCO data.
CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

HCI_Register_Event_Callback

This function registers a user-supplied callback function (as defined above) to handle HCI Events.

Prototype:

int BTPSAPI **HCI_Register_Event_Callback**(unsigned int BluetoothStackID,
  HCI_Event_Callback_t HCI_EventCallback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
HCI_EventCallback	User-supplied callback function.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive non-zero value if successful. This is the CallbackID which is used to unregister the callback.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_UNABLE_TO_REGISTER_EVENT_CALLBACK

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Register_ACL_Data_Callback

This function registers a user-supplied callback function (as defined above) for receiving ACL Data packets.

Prototype:

int BTPSAPI **HCI_Register_ACL_Data_Callback**(unsigned int BluetoothStackID,
    HCI_ACL_Data_Callback_t HCI_ACLDataCallback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
HCI_ACLDataCallback	User-supplied callback function (see definition early in this section).
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive non-zero value if successful. This is the CallbackID which is used to unregister the callback.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

BTPS_ERROR_UNABLE_TO_REGISTER_ACL_CALLBACK

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Register_SCO_Data_Callback

This function registers a user-supplied callback function (as defined above) for receiving SCO Data packets.

Prototype:

int BTPSAPI **HCI_Register_SCO_Data_Callback**(unsigned int BluetoothStackID,
    HCI_SCO_Data_Callback_t HCI_SCODataCallback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
HCI_ SCODataCallback	User-supplied callback function (see definition early in this section).
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive non-zero value if successful. This is the CallbackID which is used to unregister the callback.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_UNABLE_TO_REGISTER_SCO_CALLBACK

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

HCI_Un_Register_Callback

Remove a previously installed callback of either type: HCI Event, HCI ACL Data or HCI SCO Data.

Prototype:

int BTPSAPI **HCI_Un_Register_Callback**(unsigned int BluetoothStackID,
    unsigned int CallbackID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
CallbackID	Identifier assigned via one of the callback registrations: HCI_Register_Event_Callback HCI_Register_ACL_Data_Callback HCI_Register_SCO_Data_Callback

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.2.11 HCI Events

The table below lists the HCI events supported by the current version of the Bluetooth Stack Protocol API. Each event's parameters are further described in text below. The events are an enumeration instance of the enumeration type: HCI_Event_Type_t. The Bluetooth specification includes references to two events not included in this list: Command Complete event and Command Status event. They are omitted from this list because these events are not visible to the application programmer, but are trapped by the Bluetooth Stack and used to set the function return values.

Event	Description
etInquiry_Complete_Event	Indicates that the Inquiry is finished.
etInquiry_Result_Event	Indicates that one or more Bluetooth devices have responded so far during the current Inquiry process.
etConnection_Complete_Event	Indicates to both of the Hosts forming the connection that a new connection has been established.
etConnection_Request_Event	Indicates that a new incoming connection is trying to be established.
etDisconnection_Complete_Event	Indicates that a connection has been terminated.
etAuthentication_Complete_Event	Indicates that the authentication has been completed for the specified connection.
etRemote_Name_Request_Complete_E	Indicates that a remote name request has been

Event	Description
vent	completed.
etEncryption_Change_Event	Indicates that the change in the encryption has been completed for a connection.
etChange_Connection_Link_Key_ Complete_Event	Indicates that the change in the Link Key for the connection has been completed.
etMaster_Link_Key_Complete_ Event	Indicates that the change in the temporary Link Key or in the semi-permanent link keys on the Bluetooth master side has been completed.
etRead_Remote_Supported_Features_ Complete_Event	Indicates the completion of the process of obtaining the supported features of the remote Bluetooth device.
etRead_Remote_Version_Information_ Complete_Event	Indicates the completion of the process of obtaining the version information of the remote Bluetooth device.
etQoS_Setup_Complete_Event	Indicates the completion of the process of setting up QoS with the remote Bluetooth device.
etHardware_Error_Event	Indicates some type of hardware failure for the Bluetooth device.
etFlush_Occurred_Event	Indicates that, for the specified connection, the current user data to be transmitted has been removed.
etRole_Change_Event	Indicates that the current Bluetooth role related to the particular connection has been changed.
etNumber_Of_Completed_Packets_ Event	Indicates to the Host how many HCI Data Packets have been completed for each Connection Handle since the previous Number Of Completed Packets Event was sent. (part of flow control)
etMode_Change_Event	Indicates when the device associated with a connection changes between Active, Hold, Sniff and Park modes.
etReturn_Link_Keys_Event	Returns stored link keys after a Read_Stored_Link_Key command is used.
etPIN_Code_Request_Event	Indicates that a PIN code is required to create a new link key for a connection.
etLink_Key_Request_Event	Indicates that a Link Key is required for the connection with the device specified.
etLink_Key_Notification_Event	Indicates to the Host that a new Link Key has been created for the connection with a device.
etLoopback_Command_Event	Returns most commands that the Host sends to the

Event	Description
	Host Controller while in loopback testing mode.
etData_Buffer_Overflow_Event	Indicates that the Host Controller's data buffers have overflowed, because the Host has sent more packets than allowed.
etMax_Slots_Change_Event	Notifies the Host about the LMP_Max_Slots parameter when the value of this parameter changes.
etRead_Clock_Offset_Complete_Event	Indicates the completion of the process of obtaining the Clock offset information.
etConnection_Packet_Type_Changed_ Event	Indicates the completion of the process of changing the Packet Types used for the specified connection.
etQoS_Violation_Event	Indicates that the Link Manager is unable to provide the current QoS requirement for the connection.
etPage_Scan_Mode_Change_Event	Indicates that the connected remote Bluetooth device has successfully changed the Page_Scan_Mode.
etPage_Scan_Repetition_Mode_ Change_Event	Indicates that the connected remote Bluetooth device has successfully changed the Page_Scan_Repetition_Mode (SR).
etBluetooth_Logo_Testing_Event*	Reserved for Bluetooth Logo Testing Events.
etVendor_Specific_Debug_Event*	Reserved for Vendor Specific Debug Events.
etDevice_Reset_Event*	Indicates that the local Bluetooth device has been reset.
etFlow_Specification_Complete_Event	Indicates the Quality of Service for the ACL Connection the Controller is able to support.
etInquiry_Result_With_RSSI_Event	Indicates that one or more Bluetooth devices have responded so far during the current Inquiry process.
etRead_Remote_Extended_Features_ Complete_Event	Indicates the completion of the process of the Link Manager obtaining the remote extended LMP features of the remote device.
etSynchronous_Connection_Complete_ Event	Indicates to both the Hosts that a new Synchronous connection has been established.
etSynchronous_Connection_Changed_ Event	Indicates to the Host that an existing Synchronous connection has been reconfigured.
etSniff_Subrating_Event	Indicates that specified device has had a sniff subrating enabled or the parameters have been changed.
etExtended_Inquiry_Result_Event	Indicates that controller has responded during inquiry process with extended inquiry response data.

Event	Description
etEncryption_Key_Refresh_Complete_Event	Indicates that encryption key was refreshed on a given connection handle.
etIO_Capability_Request_Event	Indicates that IO capabilities of the host are required for simple pairing process.
etIO_Capability_Response_Event	Indicates that IO capabilities of remote host have been received.
etUser_Confirmation_Request_Event	Indicates that user confirmation of a numeric value is needed.
etUser_Passkey_Request_Event	Indicates that passkey is required as part of Simple Pairing process.
etRemote_OOB_Data_Request_Event	Indicates that Simple Pairing Hash C and Simple Pairing Randomizer R is required for the Secure Simple Pairing process.
etSimple_Pairing_Complete_Event	Indicates that Simple Pairing process has completed.
etLink_Supervision_Timeout_Changed_Event	Indicates to slave's host that Link Supervision Timeout parameter has changed in the slave controller.
etEnhanced_Flush_Complete_Event	Indicates that an Enhanced Flush is complete for specified handle.
etUser_Passkey_Notification_Event	Used to provide a passkey to display to user as required by Simple Pairing process.
etKeypress_Notification_Event	Sent to the host after a passkey notification has been received by Link Manager on specified device.
etRemote_Host_Supported_Features_Notification_Event	Used to return LMP extended features page which contains Host features.
etPhysical_Link_Complete_Event	Indicates that a new physical link has been established.
etChannel_Selected_Event	Indicates that link information data is available to be read.
etDisconnection_Physical_Link_Complete_Event	Indicates a physical link was terminated.
etPhysical_Link_Loss_Early_Warning_Event	Occurs when physical link has indications that it may be disrupted.
etPhysical_Link_Recovery_Event	Indicates that whatever caused etPhysical_Link_Loss_Early_Warning_Event has been cleared.
etLogical_Link_Complete_Event	Indicates to host that a new logical link has been successfully established.

Event	Description
etDisconnection_Logical_Link_Complete_Event	Occurs when logical link is terminated on local controller.
etFlow_Spec_Modify_Complete_Event	Indicates that Flow Spec Modify command has completed.
etNumber_Of_Completed_Data_Blocks_Event	Indicates how many ACL data packets have been completed and how many data block buffers freed.
etShort_Range_Mode_Change_Complete_Event	Indicates that a controller was asked to enable or disable the Short Range Mode for a specified physical link.
etAMP_Status_Change_Event	Indicates that a change has occurred to AMP status.
etAMP_Start_Test_Event	Indicates that HCI_AMP_Test_Command has completed.
etAMP_Test_End_Event	Indicates AMP has transmitted or received number of frames/bursts configured for a test.
etAMP_Receiver_Report_Event	Indicates number of frames received for a test.
etLE_Meta_Event	Indicates Bluetooth Low Energy event has occurred.
etPlatform_Specific_Event*	Indicates a platform specific event has occurred.

* The returned data for these events is NOT defined in the Bluetooth Core Specification.

LE specific events are contained with a LE Meta Event. Each LE event is represented as a subevent code within this Meta Event. Each one of these subevents is an enumeration of the enumeration type HCI_LE_Meta_Event_Type_t. The table below lists each of these. See section 2.2.12 for a description of these events.

Subevent	Description
meConnection_Complete_Event	Indicates that a new connection has been created.
meAdvertising_Report_Event	Indicates that a Bluetooth device or multiple devices have responded to an active scan or received some information during a passive scan.
meConnection_Update_Complete_Event	Indicates that the controller has updated the connection parameters.
meRead_Remote_Used_Features_Complete_Event	Indicates the result of a Remote used feature request to a remote Bluetooth device.
meLong_Term_Key_Request	Indicates master device is trying to encrypt or re-encrypt the link and is requesting the long term key from the host.

etInquiry_Complete_Event

This event indicates that the Inquiry operation is finished.

Return Structure:

```
typedef struct
{
  Byte_t     Status;
  Byte_t     Num_Responses;
} HCI_Inquiry_Complete_Event_Data_t
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Num_Responses	Number of responses from the inquiry. *Note, this field is only valid if the Bluetooth device is using Ver 1.0B of the Bluetooth specification. This field is not valid if using Ver 1.1 (or greater). The version can be obtained via a call to the utility function HCI_Version_Supported*

etInquiry_Result_Event

This event indicates that a Bluetooth device or multiple Bluetooth devices have responded so far during the current Inquiry process. This event will be sent as soon as an Inquiry Response from a remote device is received if the remote device supports only mandatory paging scheme. The Host Controller may queue these Inquiry Responses and send multiple Bluetooth devices information in one Inquiry Result event.

Return Structure:

The following structure represents the data returned for one inquiry result. The event result will contain an array of these structures, preceded by a one-byte quantity Num_Responses.

```
typedef struct
{
  BD_ADDR_t            BD_ADDR;
  Byte_t               Page_Scan_Repetition_Mode;
  Byte_t               Page_Scan_Period_Mode;
  Byte_t               Page_Scan_Mode;
  Class_of_Device_t    Class_of_Device;
  Word_t               Clock_Offset;
} HCI_Inquiry_Result_Data_t;
```

Event Parameters:

Num_Responses	Number of responses, i.e., instances of response structures to follow.
BD_ADDR	Address of the Bluetooth device.
Page_Scan_Repetition_Mode	Part of the supported Page Scan Modes that the remote device supports. The currently defined values are:

	HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2
Page_Scan_Period_Mode	Current setting of this parameter. Possible values are: HCI_PAGE_SCAN_PERIOD_MODE_P0 HCI_PAGE_SCAN_PERIOD_MODE_P1 HCI_PAGE_SCAN_PERIOD_MODE_P2
Page_Scan_Mode	The other part of the supported Page Scan Modes that the remote device supports. The currently defined values are: **Bluetooth Version 1.1** HCI_PAGE_SCAN_MODE_MANDATORY HCI_PAGE_SCAN_MODE_OPTIONAL_I HCI_PAGE_SCAN_MODE_OPTIONAL_II HCI_PAGE_SCAN_MODE_OPTIONAL_III **Bluetooth Version 1.2** HCI_PAGE_SCAN_MODE_MANDATORY_STANDARD_ SCAN HCI_PAGE_SCAN_MODE_OPTIONAL_INTERLACED_ SCAN
Clock_Offset	Bits 16 to 2 of the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid.
Class_of_Device	Bit mask list of features that determine the class of device for this Bluetooth device. See the HCI_Read_Class_of_Device command for a complete listing of feature bits.

etConnection_Complete_Event

This event indicates to both of the Hosts forming the connection that a new connection has been established. This event also indicates to the Host, which initiated the connection if the issued command failed or was successful.

Return Structure:

This event returns the following data, which may have zero or more responses.

```
typedef struct
{
  Byte_t                     Num_Responses;
  HCI_Inquiry_Result_Data_t  HCI_Inquiry_Result_Data[1];
} HCI_Inquiry_Result_Event_Data_t;
```

The following is used to interpret each event entry in HCI_Inquiry_Result_Data[].

```
typedef struct
{
  Byte_t          Status;
  Word_t          Connection_Handle;
  BD_ADDR_t       BD_ADDR;
  Byte_t          Link_Type;
  Byte_t          Encryption_Mode;
} HCI_Connection_Complete_Event_Data_t;
```

Event Parameters:

Num_Responses	Number of Inquiry results in this event response.
Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
BD_ADDR	Address of the other Bluetooth device.
Link_Type	Type of link established. Possible values are: HCI_LINK_TYPE_SCO_CONNECTION HCI_LINK_TYPE_ACL_CONNECTION
Encryption_Mode	Currently enabled encryption option. Possible values are: HCI_ENCRYPTION_MODE_ENCRYPTION_DISABLED HCI_ENCRYPTION_MODE_ENCRYPTION_POINT_TO_POINT_PACKETS HCI_ENCRYPTION_MODE_ENCRYPTION_POINT_TO_POINT_BROADCAST_PACKETS

etConnection_Request_Event

This event indicates that a new incoming connection is trying to be established. The connection may either be accepted or rejected. If this event is masked away and there is an incoming connection attempt and the Host Controller is not set to auto-accept this connection attempt, the Host Controller will automatically refuse the connection attempt. When the Host receives this event, it should respond with either an Accept_Connection_Request or Reject_Connection_Request command before the timer Conn_Accept_Timeout expires.

Return Structure:

```
typedef struct
{
  BD_ADDR_t            BD_ADDR;
  Class_of_Device_t    Class_of_Device;
  Byte_t               Link_Type;
} HCI_Connection_Request_Event_Data_t;
```

Event Parameters:

BD_ADDR	Address of the Bluetooth device requesting the connection.
Class_of_Device	Bit mask list of features that determine the class of device for this Bluetooth device. See the HCI_Read_Class_of_Device command for a complete listing of feature bits.
Link_Type	Type of link requested. Possible values are: **Bluetooth Version 1.1** HCI_LINK_TYPE_SCO_CONNECTION HCI_LINK_TYPE_ACL_CONNECTION **Bluetooth Version 1.2** HCI_LINK_TYPE_ESCO_CONNECTION

etDisconnection_Complete_Event

This event occurs when a connection is terminated, with the status parameter indicating if the disconnection was successful or not. The reason parameter indicates the reason for the disconnection if the disconnection was successful. Note: When a physical link fails, one Disconnection Complete event will be returned for each logical channel on the physical link with the corresponding Connection handle as a parameter.

Return Structure:

```
typedef struct
{
  Byte_t    Status;
  Word_t    Connection_Handle;
  Byte_t    Reason;
} HCI_Disconnection_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.

Reason	The reason the connection was terminated. These codes also appear in the HCI status codes (see table in the HCI introduction). The expected subset of these codes is: HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_USER_ENDED HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_LOW_RESOURCES HCI_ERROR_CODE_OTHER_END_TERMINATED_CONNECTION_ABOUT_TO_PWR_OFF HCI_ERROR_CODE_UNSUPPORTED_REMOTE_FEATURE

etAuthentication_Complete_Event

This event occurs when authentication has been completed for the specified ACL connection.

Return Structure:

```
typedef struct
{
  Byte_t    Status;
  Word_t    Connection_Handle;
} HCI_Authentication_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.

etRemote_Name_Request_Complete_Event

This event indicates that a remote name request has been completed, and if successful, returns the name in a null-terminated (0x00) string of length up to 249 bytes.

Return Structure:

```
typedef struct
{
  Byte_t          Status;
  BD_ADDR_t       BD_ADDR;
  char            Remote_Name[1];
} HCI_Remote_Name_Request_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
BD_ADDR	Address of the Bluetooth device that the name goes with.
Remote_Name	Returned name string for the remote device.

etEncryption_Change_Event

This event indicates that the change in the encryption has been completed for the ACL connection specified. This event will occur on both devices to notify both Hosts when encryption has changed for the specified connection between the two devices.

Return Structure:

```
typedef struct
{
  Byte_t    Status;
  Word_t    Connection_Handle;
  Byte_t    Encryption_Enable;
} HCI_Encryption_Change_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Encryption_Enable	Flag indicating whether the encryption should be turned on or off. Possible values are: HCI_ENCRYPTION_ENABLE_LINK_LEVEL_OFF HCI_ENCRYPTION_ENABLE_LINK_LEVEL_ON

etChange_Connection_Link_Key_Complete_Event

This event indicates that the change in the Link Key for the specified ACL connection has been completed. This event is sent only to the Host which issued the Change_Connection_Link_Key command.

Return Structure:

```
typedef struct
{
  Byte_t    Status;
  Word_t    Connection_Handle;
} HCI_Change_Connection_Link_Key_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.

etMaster_Link_Key_Complete_Event

This event indicates that the Link Key managed by the master of the piconet has been changed. The link key used for the connection will be the temporary link key of the master device or the semi-permanent link key indicated by the Key_Flag, which is also the Link Key now being used in the piconet. Note: for a master, the change from a semi-permanent Link Key to temporary Link Key will affect all connections related to the piconet. For a slave, this change affects only this particular connection.

Return Structure:

```
typedef struct
{
  Byte_t     Status;
  Word_t     Connection_Handle;
  Byte_t     Key_Flag;
} HCI_Master_Link_Key_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Key_Flag	Indicator of which link key was changed to. Possible values are: HCI_MASTER_LINK_KEY_USE_SEMI_PERMANENT_LINK_KEYS HCI_MASTER_LINK_KEY_USE_TEMPORARY_LINK_KEYS

etRead_Remote_Supported_Features_Complete_Event

This event indicates the completion of the process of obtaining the supported features of the remote Bluetooth device for the specified ACL connection, and returns the information if successful.

Return Structure:

```
typedef struct
{
  Byte_t            Status;
  Word_t            Connection_Handle;
  LMP_Features_t    LMP_Features;
} HCI_Read_Remote_Supported_Features_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes

Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
LMP_Features	Bit mask list of supported features. See the description of the HCI_Read_Local_Supported_Features command for an explanation of these bits and macros to manipulate them.

etRead_Remote_Version_Information_Complete_Event

This event indicates the completion of the process of obtaining the version information of the remote Bluetooth device for a specified ACL connection, and returns the information if successful.

Return Structure:

```
typedef struct
{
   Byte_t      Status;
   Word_t      Connection_Handle;
   Byte_t      LMP_Version;
   Word_t      Manufacturer_Name;
   Word_t      LMP_Subversion;
} HCI_Read_Remote_Version_Information_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
LMP_Version	The Link Manager Protocol version number. Possible values are: HCI_LMP_VERSION_BLUETOOTH_1_0 HCI_LMP_VERSION_BLUETOOTH_1_1 HCI_LMP_VERSION_BLUETOOTH_1_2 HCI_LMP_VERSION_BLUETOOTH_2_0 HCI_LMP_VERSION_BLUETOOTH_2_1 HCI_ LMP_VERSION_BLUETOOTH_3_0 HCI_ LMP_VERSION_BLUETOOTH_4_0
Manufacturer_Name	Manufacturer code. Possible values are: HCI_LMP_COMPID_MANUFACTURER_NAME_ ERICSSON_MOBILE_COMMUNICATIONS HCI_LMP_COMPID_MANUFACTURER_NAME_ NOKIA_MOBILE_PHONES HCI_LMP_COMPID_MANUFACTURER_NAME_ INTEL_CORPORATION HCI_LMP_COMPID_MANUFACTURER_NAME_ IBM_CORPORATION

HCI_LMP_COMPID_MANUFACTURER_NAME_TOSHIBA_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_3COM
HCI_LMP_COMPID_MANUFACTURER_NAME_MICROSOFT
HCI_LMP_COMPID_MANUFACTURER_NAME_LUCENT
HCI_LMP_COMPID_MANUFACTURER_NAME_MOTOROLA
HCI_LMP_COMPID_MANUFACTURER_NAME_INFINEON_TECHNOLOGIES_AG
HCI_LMP_COMPID_MANUFACTURER_NAME_CAMBRIDGE_SILICON_RADIO
HCI_LMP_COMPID_MANUFACTURER_NAME_SILICON_WAVE
HCI_LMP_COMPID_MANUFACTURER_NAME_DIGIANSWER
HCI_LMP_COMPID_MANUFACTURER_NAME_TEXAS_INSTRUMENTS
HCI_LMP_COMPID_MANUFACTURER_NAME_PARTHUS_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_BROADCOM
HCI_LMP_COMPID_MANUFACTURER_NAME_MITEL_SEMICONDUCTOR
HCI_LMP_COMPID_MANUFACTURER_NAME_WIDCOMM
HCI_LMP_COMPID_MANUFACTURER_NAME_TELENCOMM
HCI_LMP_COMPID_MANUFACTURER_NAME_ATMEL
HCI_LMP_COMPID_MANUFACTURER_NAME_MITSUBISHI
HCI_LMP_COMPID_MANUFACTURER_NAME_RTX_TELECOM
HCI_LMP_COMPID_MANUFACTURER_NAME_KC_TECHNOLOGY
HCI_LMP_COMPID_MANUFACTURER_NAME_NEWLOGIC
HCI_LMP_COMPID_MANUFACTURER_NAME_TRANSILICA
HCI_LMP_COMPID_MANUFACTURER_NAME_ROHDE_AND_SCHWARTZ
HCI_LMP_COMPID_MANUFACTURER_NAME_TTPCOM
HCI_LMP_COMPID_MANUFACTURER_NAME_SIGNIA_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_CONEXANT_SYSTEMS

HCI_LMP_COMPID_MANUFACTURER_NAME_
QUALCOMM
HCI_LMP_COMPID_MANUFACTURER_NAME_
INVENTEL
HCI_LMP_COMPID_MANUFACTURER_NAME_
AVM_BERLIN
HCI_LMP_COMPID_MANUFACTURER_NAME_
BANDSPEED
HCI_LMP_COMPID_MANUFACTURER_NAME_
MANSELLA
HCI_LMP_COMPID_MANUFACTURER_NAME_
NEC
HCI_LMP_COMPID_MANUFACTURER_NAME_
WAVEPLUS_TECHNOLOGY
HCI_LMP_COMPID_MANUFACTURER_NAME_
ALCATEL
HCI_LMP_COMPID_MANUFACTURER_NAME_
PHILIPS_SEMICONDUCTORS
HCI_LMP_COMPID_MANUFACTURER_NAME_
C_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_
OPEN_INTERFACE
HCI_LMP_COMPID_MANUFACTURER_NAME_
RF_MICRO_DEVICES
HCI_LMP_COMPID_MANUFACTURER_NAME_
HITACHI
HCI_LMP_COMPID_MANUFACTURER_NAME_
SYMBOL_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_
TENOVIS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MACRONIX_INTERNATIONAL
HCI_LMP_COMPID_MANUFACTURER_NAME_
GCT_SEMICONDUCTOR
HCI_LMP_COMPID_MANUFACTURER_NAME_
NORWOOD_SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MEWTEL_TECHNOLOGY
HCI_LMP_COMPID_MANUFACTURER_NAME_
ST_MICROELECTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_
SYNOPSYS
HCI_LMP_COMPID_MANUFACTURER_NAME_
RED_M_COMMUNICATIONS
HCI_LMP_COMPID_MANUFACTURER_NAME_
COMMIL_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
CATC
HCI_LMP_COMPID_MANUFACTURER_NAME_
ECLIPSE_SL

HCI_LMP_COMPID_MANUFACTURER_NAME_
RENESAS_TECHNOLOGY_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_
MOBILIAN_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
TERAX
HCI_LMP_COMPID_MANUFACTURER_NAME_
INTEGRATED_SYSTEM_SOLUTION
HCI_LMP_COMPID_MANUFACTURER_NAME_
MATSUSHITA
HCI_LMP_COMPID_MANUFACTURER_NAME_
GENNUM_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
RESEARCH_IN_MOTION
HCI_LMP_COMPID_MANUFACTURER_NAME_
IPEXTREME
HCI_LMP_COMPID_MANUFACTURER_NAME_
SYSTEMS_AND_CHIPS
HCI_LMP_COMPID_MANUFACTURER_NAME_
BLUETOOTH_SIG
HCI_LMP_COMPID_MANUFACTURER_NAME_
SEIKO_EPSON_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
INTEGRATED_SILICON_SOLUTION
HCI_LMP_COMPID_MANUFACTURER_NAME_
CONWISE_TECHNOLOGY_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
PARROT_SA
HCI_LMP_COMPID_MANUFACTURER_NAME_
SOCKET_MOBILE
HCI_LMP_COMPID_MANUFACTURER_NAME_
ATHEROS_COMMUNICATIONS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MEDIATEK_INCORPORATED
HCI_LMP_COMPID_MANUFACTURER_NAME_
BLUEGIGA
HCI_LMP_COMPID_MANUFACTURER_NAME_
MARVELL_TECHNOLOGY_GROUP
HCI_LMP_COMPID_MANUFACTURER_NAME_
3DSP_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
ACCEL_SEMICONDUCTOR
HCI_LMP_COMPID_MANUFACTURER_NAME_
CONTINENTAL_AUTOMOTIVE_SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_
APPLE_INCORPORATED
HCI_LMP_COMPID_MANUFACTURER_NAME_
STACCATO_COMMUNICATIONS
HCI_LMP_COMPID_MANUFACTURER_NAME_
AVAGO_TECHONOLOGIES

HCI_LMP_COMPID_MANUFACTURER_NAME_APT_
LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_SIRF_
TECHONOLIGY
HCI_LMP_COMPID_MANUFACTURER_NAME_
TZERO_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_J_
AND_M_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
FREE2MOVE_AB
HCI_LMP_COMPID_MANUFACTURER_NAME_3DIJOY_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
PLANTRONICS_INCORPORATED
HCI_LMP_COMPID_MANUFACTURER_NAME_SONY_
ERICSSON_MOBILE_COMM
HCI_LMP_COMPID_MANUFACTURER_NAME_
HARMAN_INTERNATIONAL_IND
HCI_LMP_COMPID_MANUFACTURER_NAME_
VIZIO_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_NORDIC_S
EMICONDUCTOR_ASA
HCI_LMP_COMPID_MANUFACTURER_NAME_EM_
MICROELECTRONIC_MARIN_SA
HCI_LMP_COMPID_MANUFACTURER_NAME_RALINK_T
ECHNOLOGY_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_BELKIN_
INTERNATIONAL_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
REALTEK_SEMICONDUCTOR_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_
STONESTREET_ONE_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
WICENTRIC_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_RIVIERA_
WAVES_SAS
HCI_LMP_COMPID_MANUFACTURER_NAME_RDA_
MICROELECTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_GIBSON_G
UITARS
HCI_LMP_COMPID_MANUFACTURER_NAME_
MICOMMAND_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_BAND_
XI_INTERNATIONAL_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
HEWLETT_PACKARD_COMPANY
HCI_LMP_COMPID_MANUFACTURER_NAME_
9SOLUTIONS_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_GN_
NETCOM_AS

HCI_LMP_COMPID_MANUFACTURER_NAME_
GENERAL_MOTORS
HCI_LMP_COMPID_MANUFACTURER_NAME_A_
AND_D_ENGINEERING_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
MINDTREE_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_POLAR_
ELECTRO_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_
BEAUTIFUL_ENTERPRISE_COMPANY
HCI_LMP_COMPID_MANUFACTURER_NAME_
BRIARTEK_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_SUMMIT_
DATA_COMMUNICATIONS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_SOUND_ID
HCI_LMP_COMPID_MANUFACTURER_NAME_
MONSTER_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
CONNECT_BLUE_AB
HCI_LMP_COMPID_MANUFACTURER_NAME_
SHANGHAI_SUPER_SMART_ELECTRON
HCI_LMP_COMPID_MANUFACTURER_NAME_GROUP_
SENSE_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_ZOMM_
LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
SAMSUNG_ELECTRONICS_CO_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
CREATIVE_TECHNOLOGY_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_LAIRD_
TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_NIKE_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
LESSWIRE_AG
HCI_LMP_COMPID_MANUFACTURER_NAME_MSTAR_
SEMICONDUCTOR_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
HANLYNN_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_A_AND_R_
CAMBRIDGE
HCI_LMP_COMPID_MANUFACTURER_NAME_SEERS_
TECHNOLOGY_CO_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_SPORTS_
TRACKING_TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_
AUTONET_MOBILE
HCI_LMP_COMPID_MANUFACTURER_NAME_
DELORME_PUBLISHING_COMPANY_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_WUXI_
VIMICRO

HCI_LMP_COMPID_MANUFACTURER_NAME_
SENNHEISER_COMMUNICATIONS_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_
TIMEKEEPING_SYSTEMS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_LUDUS_
HELSINKI_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
BLUERADIOS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
EQUINUX_AG
HCI_LMP_COMPID_MANUFACTURER_NAME_GARMIN_
INTERNATIONAL_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ECOTEST
HCI_LMP_COMPID_MANUFACTURER_NAME_GN_
RESOUND_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_JAWBONE
HCI_LMP_COMPID_MANUFACTURER_NAME_TOPCON_
POSITIONING_SYSTEMS_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUALCOMM_LABS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ZSCAN_
SOFTWARE
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUINTIC_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_
STOLLMANN_E_V_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_FUNAI
_ELECTRIC_COMPANY_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
ADVANCED_PANMOBIL_SYSTEMS_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
THINKOPTICS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
UNIVERSAL_ELECTRONICS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_AIROHA_
TECHNOLOGY_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_NEC_
LIGHTING_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_ODM_
TECHNOLOGY_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
BLUETREK_TECHNOLOGIES_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_ZERO_1_
TV_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_I_TECH_
DYNAMIC_GLOBAL_DIST_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_ALPWISE
HCI_LMP_COMPID_MANUFACTURER_NAME_JIANGSU_
TOPPOWER_AUTOMOTIVE
HCI_LMP_COMPID_MANUFACTURER_NAME_COLORFY_
INC

HCI_LMP_COMPID_MANUFACTURER_NAME_
GEOFORCE_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_BOSE_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_SUUNTO_
OY
HCI_LMP_COMPID_MANUFACTURER_NAME_
KENSINGTON_COMPUTER_PROD_GROUP
HCI_LMP_COMPID_MANUFACTURER_NAME_SR_
MEDIZINELEKTRONIK
HCI_LMP_COMPID_MANUFACTURER_NAME_VERTU_
CORPORATION_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_META_
WATCH_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_LINAK_
A_S
HCI_LMP_COMPID_MANUFACTURER_NAME_OTL_
DYNAMICS_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_PANDA_
OCEAN_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_VISTEON_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_ARP_
DEVICES_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_MAGNETI_
MARELLI_S_P_A
HCI_LMP_COMPID_MANUFACTURER_NAME_CAEN_
RFID_SRL
HCI_LMP_COMPID_MANUFACTURER_NAME_
INGENIEUR_SYSTEMGRUPPE_ZAHN
HCI_LMP_COMPID_MANUFACTURER_NAME_GREEN_
THROTTLE_GAMES
HCI_LMP_COMPID_MANUFACTURER_NAME_
PETER SYSTEMTECHNIK_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
OMEGAWAVE_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_CINETIX
HCI_LMP_COMPID_MANUFACTURER_NAME_PASSIF_
SEMICONDUCTOR_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_SARIS_
CYCLING_GROUP_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_BEKEY_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_
CLARINOX_TECHNOLOGIES_PTY_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_BDE_
TECHNOLOGY_CO_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_SWIRL_
NETWORKS
HCI_LMP_COMPID_MANUFACTURER_NAME_MESO_
INTERNATIONAL

HCI_LMP_COMPID_MANUFACTURER_NAME_TRELAB_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_QUALCOMM_INNOVATION_CENTER_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_JOHNSON_CONTROLS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_STARKEY_LABORATORIES_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_S_POWER_ELECTRONICS_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_ACE_SENSOR_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_APLIX_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_AAMP_OF_AMERICA
HCI_LMP_COMPID_MANUFACTURER_NAME_STALMART_TECHNOLOGY_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_AMICCOM_ELECTRONICS_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_SHENZHEN_EXCELSECU_DATA_TECH
HCI_LMP_COMPID_MANUFACTURER_NAME_GENEQ_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ADIDAS_AG
HCI_LMP_COMPID_MANUFACTURER_NAME_LG_ELECTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_ONSET_COMPUTER_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_SELFLY_BV
HCI_LMP_COMPID_MANUFACTURER_NAME_QUUPPA_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_GELO_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_EVLUMA
HCI_LMP_COMPID_MANUFACTURER_NAME_MC10
HCI_LMP_COMPID_MANUFACTURER_NAME_BINAURIC_SE
HCI_LMP_COMPID_MANUFACTURER_NAME_BEATS_ELECTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_MICROCHIP_TECHNOLOGY_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ELGATO_SYSTEMS_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_ARCHOS_SA
HCI_LMP_COMPID_MANUFACTURER_NAME_DEXCOM_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_POLAR_ELECTRO_EUROPE_BV

HCI_LMP_COMPID_MANUFACTURER_NAME_DIALOG_
SEMICONDUCTOR_BV
HCI_LMP_COMPID_MANUFACTURER_NAME_
TAIXINGBANG_TECHNOLOGY_HK_CO
HCI_LMP_COMPID_MANUFACTURER_NAME_
KAWANTECH
HCI_LMP_COMPID_MANUFACTURER_NAME_AUSTCO_
COMMUNICATION_SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_TIMEX_
GROUP_USA_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUALCOMM_TECHNOLOGIES_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
QUALCOMM_CONNECTED_EXPERIENCES
HCI_LMP_COMPID_MANUFACTURER_NAME_
VOYETRA_TURTLE_BEACH
HCI_LMP_COMPID_MANUFACTURER_NAME_
TXTR_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
BIOSENTRONICS
HCI_LMP_COMPID_MANUFACTURER_NAME_PROCTER_
AND_GAMBLE
HCI_LMP_COMPID_MANUFACTURER_NAME_HOSIDEN_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
MUZIK_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_MISFIT_
WEARABLES_CORP
HCI_LMP_COMPID_MANUFACTURER_NAME_GOOGLE
HCI_LMP_COMPID_MANUFACTURER_NAME_
DANLERS_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
SEMILINK_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_INMUSIC_
BRANDS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_LS_
RESEARCH_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_EDEN_
SOFTWARE_CONSULTANTS_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
FRESHTEMP
HCI_LMP_COMPID_MANUFACTURER_NAME_KS_
TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_ACTS_
TECHNOLOGIES
HCI_LMP_COMPID_MANUFACTURER_NAME_VTRACK_
SYSTEMS
HCI_LMP_COMPID_MANUFACTURER_NAME_NIELSEN_
KELLERMAN_COMPANY
HCI_LMP_COMPID_MANUFACTURER_NAME_SERVER_
TECHNOLOGY_INC

HCI_LMP_COMPID_MANUFACTURER_NAME_
BIORESEARCH_ASSOCIATES
HCI_LMP_COMPID_MANUFACTURER_NAME_JOLLY_
LOGIC_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_ABOVE_
AVERAGE_OUTCOMES_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
BITSPLITTERS_GMBH
HCI_LMP_COMPID_MANUFACTURER_NAME_
PAYPAL_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_WITRON_
TECHNOLOGY_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_MORSE_
PROJECT_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_KENT_
DISPLAYS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
NAUTILUS_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_
SMARTIFIER_OY
HCI_LMP_COMPID_MANUFACTURER_NAME_
ELCOMETER_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_VSN_
TECHNOLOGIES_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_ACEUNI_
CORP_LTD
HCI_LMP_COMPID_MANUFACTURER_NAME_
STICKNFIND
HCI_LMP_COMPID_MANUFACTURER_NAME_CRYSTAL_
CODE_AB
HCI_LMP_COMPID_MANUFACTURER_NAME_
KOUKAAM_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_
DELPHI_CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_
VALENCETECH_LIMITED
HCI_LMP_COMPID_MANUFACTURER_NAME_
RESERVED
HCI_LMP_COMPID_MANUFACTURER_NAME_TYPO_
PRODUCTS_LLC
HCI_LMP_COMPID_MANUFACTURER_NAME_TOMTOM_
INTERNATIONAL_BV
HCI_LMP_COMPID_MANUFACTURER_NAME_
FUGOO_INC
HCI_LMP_COMPID_MANUFACTURER_NAME_KEISER_
CORPORATION
HCI_LMP_COMPID_MANUFACTURER_NAME_BANG_
AND_OLUFSEN_AS
HCI_LMP_COMPID_MANUFACTURER_NAME_PLUS_
LOCATIONS_SYSTEMS_PTY_LTD

	HCI_LMP_COMPID_MANUFACTURER_NAME_UBIQUITOUS_COMPUTING_TECH_CORP HCI_LMP_COMPID_MANUFACTURER_NAME_INNOVATIVE_YACHTTER_SOLUTIONS
LMP_Subversion	The LMP sub-version number. These are defined by each manufacturer.

etQoS_Setup_Complete_Event

This event indicates the completion of the process of setting up QoS with the remote Bluetooth device for the specified ACL connection, and returns the parameters for this setup, if successful.

Return Structure:

```
typedef struct
{
  Byte_t      Status;
  Word_t      Connection_Handle;
  Byte_t      Flags;
  Byte_t      Service_Type;
  DWord_t     Token_Rate;
  DWord_t     Peak_Bandwidth;
  DWord_t     Latency;
  DWord_t     Delay_Variation;
} HCI_QoS_Setup_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Flags	(reserved for future use)
Service_Type	The type of service to establish. Possible values are: HCI_QOS_SERVICE_TYPE_NO_TRAFFIC HCI_QOS_SERVICE_TYPE_BEST_EFFORT HCI_QOS_SERVICE_TYPE_GUARANTEED
Token_Rate	Token Rate in bytes per second.
Peak_Bandwidth	Peak Bandwidth in bytes per second.
Latency	Latency in microseconds.
Delay_Variation	Delay Variation in microseconds.

etHardware_Error_Event

This event indicates that some type of Bluetooth device hardware failure has occurred.

Return Structure:

```
typedef struct
{
  Byte_t      Hardware_Code;
} HCI_Hardware_Error_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Hardware_Code	An implementation-specific code. See documentation accompanying the particular hardware.

etFlush_Occurred_Event

This event indicates that, for the specified ACL connection, the current user data to be transmitted has been dropped. This could result from the flush command, or be due to the automatic flush.

Return Structure:

```
typedef struct
{
  Word_t    Connection_Handle;
} HCI_Flush_Occurred_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	The connection that was flushed.

etRole_Change_Event

This event indicates that the current Bluetooth role related to the particular connection has changed. This event only occurs when both the remote and local Bluetooth devices have completed their role changes.

Return Structure:

```
typedef struct
{
  Byte_t           Status;
  BD_ADDR_t        BD_ADDR;
  Byte_t           New_Role;
} HCI_Role_Change_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
BD_ADDR	Address of the Bluetooth device.

New_Role	New Role for this device. Possible values are: HCI_CURRENT_ROLE_MASTER HCI_CURRENT_ROLE_SLAVE

etNumber_Of_Completed_Packets_Event

This event is used by the Host Controller to indicate to the Host how many HCI Data Packets have been completed (transmitted or flushed) for each Connection Handle since the previous Number Of Completed Packets event was sent to the Host. This means that the corresponding buffer space has been freed in the Host Controller.

Return Structure:

This event can return multiple pieces of connection information. The overall return is described by the following structure.

```
typedef struct
{
  Byte_t                                    Number_of_Handles;
  HCI_Number_Of_Completed_Packets_Data_t    HCI_Number_Of_Completed_Packets_Data[1];
} HCI_Number_Of_Completed_Packets_Event_Data_t;
```

The array HCI_Number_Of_Completed_Packets_Data[] is an array of the following structures, one for each connection.

```
typedef struct
{
  Word_t   Connection_Handle;
  Word_t   HC_Num_Of_Completed_Packets;
} HCI_Number_Of_Completed_Packets_Data_t;
```

Event Parameters:

Number_of_Handles	Number of entries in the array.
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
HC_Num_Of_Completed_Packets	Number of packets which have been processed for this connection.

etMode_Change_Event

This event indicates when the device associated with an ACL connection changes between Active, Hold, Sniff and Park mode.

Return Structure:

```
typedef struct
{
  Byte_t     Status;
  Word_t     Connection_Handle;
  Byte_t     Current_Mode;
  Word_t     Interval;
} HCI_Mode_Change_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Current_Mode	The current mode of the device associated with Connection_Handle. Possible values are: HCI_CURRENT_MODE_ACTIVE_MODE HCI_CURRENT_MODE_HOLD_MODE HCI_CURRENT_MODE_SNIFF_MODE HCI_CURRENT_MODE_PARK_MODE
Interval	Length of time to wait in the indicated mode. Values are number of baseband slots (0.625 msec), with a range of 0 (0x0000) to 40.9 sec (0xFFFF).

etReturn_Link_Keys_Event

This event is used by the Host Controller to send the Host one or more stored Link Keys. Zero or more instances of this event will occur after the Read_Stored_Link_Key command. When there are no link keys stored, no Return Link Keys events will be returned. When there are link keys stored, the number of link keys returned in each Return Link Keys event is implementation specific.

Return Structure:

The top-level return structure is as follows:

```
typedef struct
{
  Byte_t                          Num_Keys;
  HCI_Return_Link_Keys_Data_t     HCI_Return_Link_Keys_Data[1];
} HCI_Return_Link_Keys_Event_Data_t;
```

Each item in the array HCI_Return_Link_Keys_Data[ ] is a BD_ADDR – Link Key pair structure defined as follows:

```
typedef struct
{
  BD_ADDR_t       BD_ADDR;
  Link_Key_t      Link_Key;
} HCI_Return_Link_Keys_Data_t;
```

Event Parameters:

Num_Keys	Number of items in the array (at least one).
BD_ADDR	Address of the Bluetooth device.
Link_Key	Associated Link Key.

etPIN_Code_Request_Event

This event indicates that a PIN code is required to create a new link key. The Host must respond using either the PIN Code Request Reply or the PIN Code Request Negative Reply command, depending on whether the Host can provide the Host Controller with a PIN code or not. Note: If the PIN Code Request event is masked away, then the Host Controller will assume that the Host has no PIN Code.

Return Structure:

```
typedef struct
{
  BD_ADDR_t       BD_ADDR;
} HCI_PIN_Code_Request_Event_Data_t;
```

Event Parameters:

BD_ADDR	Address of the device that a new link key is being created for.

etLink_Key_Request_Event

This event indicates that a Link Key is required for the connection with the device specified in BD_ADDR. If the Host has the requested stored Link Key, then the Host will pass the requested Key to the Host Controller using the Link_Key_Request_Reply Command. If the Host does not have the requested stored Link Key, then the Host will use the Link_Key_Request_Negative_Reply Command to indicate to the Host Control-ler that the Host does not have the requested key. Note: If the Link Key Request event is masked away, then the Host Controller will assume that the Host has no additional link keys.

Return Structure:

```
typedef struct
{
  BD_ADDR_t       BD_ADDR;
} HCI_Link_Key_Request_Event_Data_t;
```

Event Parameters:

BD_ADDR	Address of the device that is requesting a new link key.

etLink_Key_Notification_Event

This event indicates to the Host that a new Link Key has been created for the connection with the device specified in BD_ADDR. The Host can save this new Link Key in its own storage for future use. Also, the Host can decided to store the Link Key in the Host Controller's Link Key Storage by using the Write_Stored_Link_Key command.

Return Structure:

```
typedef struct
{
  BD_ADDR_t      BD_ADDR;
  Link_Key_t     Link_Key;
  Byte_t         Key_Type;
} HCI_Link_Key_Notification_Event_Data_t;
```

Event Parameters:

BD_ADDR	Address of the device for which the new link key has been created.
Link_Key	The new link key.
Key_Type	This field is only valid if the Bluetooth device is using HCI specification 1.1 or later, rather than 1.0B.

etLoopback_Command_Event

This event is used to send back all HCI command packets when the device is in loopback mode.

Return Structure:

```
typedef struct
{
  Word_t      HCI_Command_Packet_Length;
  Byte_t      HCI_Command_Packet_Data[1];
} HCI_Loopback_Command_Event_Data_t;
```

Event Parameters:

HCI_Command_Packet_Length	Number of bytes in the packet data.
HCI_Command_Packet_Data	Actual command packet data.

etData_Buffer_Overflow_Event

This event indicates that the Host Controller's data buffers have been overflowed. This can occur if the Host has sent more packets than allowed.

Return Structure:

```
typedef struct
{
  Byte_t      Link_Type;
} HCI_Data_Buffer_Overflow_Event_Data_t;
```

Event Parameters:

Link_Type	Whether the overflow was on an ACL or SCO link. Possible values are: HCI_LINK_TYPE_SCO_CONNECTION HCI_LINK_TYPE_ACL_CONNECTION

etMax_Slots_Change_Event

This event notifies the Host about the LMP_Max_Slots parameter when the value of this parameter changes. It will be sent each time the value of the LMP_Max_Slots parameter changes, as long as there is at least one connection to another device.

Return Structure:

```
typedef struct
{
  Word_t          Connection_Handle;
  Byte_t          LMP_Max_Slots;
} HCI_Max_Slots_Change_Event_Data_t;
```

Event Parameters:

Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
LMP_Max_Slots	Maximum number of slots allowed for baseband packets.

etRead_Clock_Offset_Complete_Event

This event indicates the completion of the process of obtaining the Clock Offset information of the remote Bluetooth device for an ACL connection, and if successful, returns the value.

Return Structure:

```
typedef struct
{
  Byte_t          Status;
  Word_t          Connection_Handle;
  Word_t          Clock_Offset;
} HCI_Read_Clock_Offset_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes

Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Clock_Offset	Bits 16 to 2 of the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid.

etConnection_Packet_Type_Changed_Event

This event is used to indicate that the process has completed of changing which packet types can be used for the connection. This allows current connections to be dynamically modified to support different types of user data.

Return Structure:

```
typedef struct
{
  Byte_t      Status;
  Word_t      Connection_Handle;
  Word_t      Packet_Type;
} HCI_Connection_Packet_Type_Changed_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Unique identifier for the connection returned in the Connection Complete event associated with the HCI_Create_Connection command.
Packet_Type	Which packet types the Link Manager shall use for the ACL link. This can be an ORing of multiple packet types. The currently defined packet types are – For ACL Links: HCI_PACKET_ACL_TYPE_DM1 HCI_PACKET_ACL_TYPE_DH1 HCI_PACKET_ACL_TYPE_DM3 HCI_PACKET_ACL_TYPE_DH3 HCI_PACKET_ACL_TYPE_DM5 HCI_PACKET_ACL_TYPE_DH5 **Bluetooth Version 2.0** HCI_PACKET_ACL_TYPE_2_DH1_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_3_DH1_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_2_DH3_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_3_DH3_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_2_DH5_MAY_NOT_BE_USED HCI_PACKET_ACL_TYPE_3_DH5_MAY_NOT_BE_USED

For SCO Links:

HCI_PACKET_SCO_TYPE_HV1
HCI_PACKET_SCO_TYPE_HV2
HCI_PACKET_SCO_TYPE_HV3

etQoS_Violation_Event

This event indicates that the Link Manager is unable to provide the current QoS requirement for the connection. The Host chooses what action should be done as a result. The Host can reissue QoS_Setup command to renegotiate the QoS setting for the connection.

Return Structure:

```
typedef struct
{
  Word_t          Connection_Handle;
} HCI_QoS_Violation_Event_Data_t;
```

Event Parameters:

Connection_Handle	The identifier for the ACL connection with the QoS violation.

etPage_Scan_Mode_Change_Event

This event indicates that a remote Bluetooth device has successfully changed the Page_Scan_Mode.

Return Structure:

```
typedef struct
{
  BD_ADDR_t       BD_ADDR;
  Byte_t          Page_Scan_Mode;
} HCI_Page_Scan_Mode_Change_Event_Data_t;
```

Event Parameters:

BD_ADDR	Address of the Bluetooth device.
Page_Scan_Mode	The new Page Scan Mode. Possible values are:

Bluetooth Version 1.1

HCI_PAGE_SCAN_MODE_MANDATORY
HCI_PAGE_SCAN_MODE_OPTIONAL_I
HCI_PAGE_SCAN_MODE_OPTIONAL_II
HCI_PAGE_SCAN_MODE_OPTIONAL_III

Bluetooth Version 1.2

HCI_PAGE_SCAN_MODE_MANDATORY_STANDARD_SCAN
HCI_PAGE_SCAN_MODE_OPTIONAL_INTERLACED_SCAN

etPage_Scan_Repetition_Mode_Change_Event

This event indicates that the remote Bluetooth device has successfully changed the Page_Scan_Repetition_Mode.

Return Structure:

```
typedef struct
{
  BD_ADDR_t      BD_ADDR;
  Byte_t         Page_Scan_Repetition_Mode;
} HCI_Page_Scan_Repetition_Mode_Change_Event_Data_t;
```

Event Parameters:

BD_ADDR	Address of the Bluetooth device.
Page_Scan_Repetition_Mode	New repetition mode. Possible values are: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2

etFlow_Specification_Complete_Event

This event informs the Host about the Quality of Service for the ACL connection the Controller is able to support.

Return Structure:

```
typedef struct
{
  Byte_t      Status;
  Word_t      Connection_Handle;
  Byte_t      Flags;
  Byte_t      Flow_Direction;
  Byte_t      Service_Type;
  DWord_t     Token_Rate;
  DWord_t     Token_Bucket_Size;
  DWord_t     Peak_Bandwidth;
  DWord_t     Access_Latency;
} HCI_Flow_Specification_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Connection_Handle	Connection Handle used to identify for which ACL connection the Flow is specified.
Flags	Reserved for future use.
Flow_Direction	Outgoing or incoming flow over the ACL connection. Possible values are: HCI_FLOW_SPECIFICATION_FLOW_DIRECTION_OUTGOING_FLOW HCI_FLOW_SPECIFICATION_FLOW_DIRECTION_INCOMING_FLOW
Service_Type	No traffic, best effort, or guaranteed. Possible values are: HCI_FLOW_SPECIFICATION_SERVICE_TYPE_NO_TRAFFIC HCI_FLOW_SPECIFICATION_SERVICE_TYPE_BEST_EFFORT HCI_FLOW_SPECIFICATION_SERVICE_TYPE_GUARANTEED
Token_Rate	The token rate in octets per second.
Token_Bucket_Size	Token bucket size in octets.
Peak_Bandwidth	Peak bandwidth in octets per second.
Access_Latency	Access latency in microseconds.

etInquiry_Result_With_RSSI_Event

This event indicates that a Bluetooth device or multiple Bluetooth devices have responded so far during the current Inquiry process with an RSSI value. The following structure represents the data returned for one inquiry result with RSSI information. The event result will contain an array of these structures, preceded by a one-byte quantity Num_Responses.

Return Structure:

```
typedef struct
{
  BD_ADDR_t          BD_ADDR;
  Byte_t             Page_Scan_Repetition_Mode;
  Byte_t             Page_Scan_Period_Mode;
  Class_of_Device_t  Class_of_Device;
  Word_t             Clock_Offset;
  Byte_t             RSSI;
} HCI_Inquiry_Result_With_RSSI_Data_t;
```

Event Parameters:

BD_ADDR	Address of the Bluetooth device.
Page_Scan_Repetition_Mode	Part of the supported Page Scan Modes that the remote device supports. The currently defined values are: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2
Page_Scan_Period_Mode	Current setting of this parameter. Possible values are: HCI_PAGE_SCAN_PERIOD_MODE_P0 HCI_PAGE_SCAN_PERIOD_MODE_P1 HCI_PAGE_SCAN_PERIOD_MODE_P2
Class_of_Device	Bit mask list of features that determine the class of device for this Bluetooth device. See the HCI_Read_Class_of_Device command for a complete listing of feature bits.
Clock_Offset	Bits 16 to 2 of the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid
RSSI	RSSI value in dBm from -127 to +20

etRead_Remote_Extended_Features_Complete_Event

This event is used to indicate the completion of the process of the Link Manager obtaining the remote extended LMP features of the remote device specified by the connection handle event parameter.

Return Structure:

```
typedef struct
{
   Byte_t          Status;
   Word_t          Connection_Handle;
   Byte_t          Page_Number;
   Byte_t          Maximum_Page_Number;
   LMP_Features_t  Extended_LMP_Features;
} HCI_Read_Remote_Extended_Features_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Connection_Handle	Connection Handle used to identify the connection between two Bluetooth devices.
Page_Number	Normal LMP features as returned by HCI_Read_Local_Supported_Features (if 0) or the corresponding page of features (non-zero).
Maximum_Page_Number	The highest features page number which contains non-zero bits for the local device.
Extended_LMP_Features	Bit map of requested page of LMP features. Defined bit numbers are:

Bluetooth Version 1.1

HCI_LMP_FEATURE_THREE_SLOT_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_FIVE_SLOT_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_ENCRYPTION_BIT_NUMBER
HCI_LMP_FEATURE_SLOT_OFFSET_BIT_NUMBER
HCI_LMP_FEATURE_TIMING_ACCURACY_BIT_NUMBER
HCI_LMP_FEATURE_SWITCH_BIT_NUMBER
HCI_LMP_FEATURE_HOLD_MODE_BIT_NUMBER
HCI_LMP_FEATURE_SNIFF_MODE_BIT_NUMBER
HCI_LMP_FEATURE_PARK_MODE_BIT_NUMBER
HCI_LMP_FEATURE_RSSI_BIT_NUMBER
HCI_LMP_FEATURE_CHANNEL_QUALITY_DRIVEN_
DATA_RATE_BIT_NUMBER
HCI_LMP_FEATURE_SCO_LINK_BIT_NUMBER
HCI_LMP_FEATURE_HV2_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_HV3_PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_U_LAW_LOG_BIT_NUMBER
HCI_LMP_FEATURE_A_LAW_LOG_BIT_NUMBER
HCI_LMP_FEATURE_CVSD_BIT_NUMBER
HCI_LMP_FEATURE_PAGING_SCHEME_BIT_NUMBER
HCI_LMP_FEATURE_POWER_CONTROL_BIT_NUMBER

Bluetooth Version 1.2

HCI_LMP_FEATURE_ROLE_SWITCH_BIT_NUMBER
HCI_LMP_FEATURE_PARK_STATE_BIT_NUMBER

HCI_LMP_FEATURE_POWER_CONTROL_REQUESTS_
BIT_NUMBER
HCI_LMP_FEATURE_PAGING_PARAMETER_
NEGOTIATION_BIT_NUMBER
HCI_LMP_FEATURE_TRANSPARENT_SYNCHRONOUS_
DATA_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_LEAST_
SIGNIFICANT_BIT_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_MIDDLE_
BIT_BIT_NUMBER
HCI_LMP_FEATURE_FLOW_CONTROL_LAG_MOST_
SIGNIFICANT_BIT_BIT_NUMBER
HCI_LMP_FEATURE_BROADCAST_ENCRYPTION_BIT_
NUMBER
HCI_LMP_FEATURE_ENHANCED_INQUIRY_SCAN_BIT_N
UMBER
HCI_LMP_FEATURE_INTERLACED_INQUIRY_SCAN_
BIT_NUMBER
HCI_LMP_FEATURE_INTERLACED_PAGE_SCAN_BIT_
NUMBER
HCI_LMP_FEATURE_RSSI_WITH_INQUIRY_RESULTS_
BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_SCO_LINKS_EV3_
PACKETS_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_EV4_PACKETS_BIT_
NUMBER
HCI_LMP_FEATURE_EXTENDED_EV5_PACKETS_BIT_
NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_CAPABLE_
SLAVE_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_
CLASSIFICATION_SLAVE_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_CAPABLE_
MASTER_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_AFH_
CLASSIFICATION_MASTER_BIT_NUMBER
HCI_LMP_FEATURE_EXTENDED_FEATURES_BIT_
NUMBER

Useful macros defined for manipulation of LMP Features are:

COMPARE_LMP_FEATURES(feats1, feats2)
SET_FEATURES_BIT(feats, bitnumb)
RESET_FEATURES_BIT(feats, bitnum)
TEST_FEATURES_BIT(feats, bitnum)

etSynchronous_Connection_Complete_Event

This event indicates to both the Hosts that a new Synchronous connection has been established.

Return Structure:

```
typedef struct
{
  Byte_t        Status;
  Word_t        Connection_Handle;
  BD_ADDR_t     BD_ADDR;
  Byte_t        Link_Type;
  Byte_t        Transmission_Interval;
  Byte_t        Retransmission_Window;
  Word_t        Rx_Packet_Length;
  Word_t        Tx_Packet_Length;
  Byte_t        Air_Mode;
} HCI_Synchronous_Connection_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes
Connection_Handle	Connection Handle used to identify the connection between two Bluetooth devices.
BD_ADDR	Address of the Bluetooth device.
Link_Type	SCO or eSCO connection. Possible values are: HCI_LINK_TYPE_SCO_CONNECTION HCI_LINK_TYPE_ESCO_CONNECTION
Transmission_Interval	Time between two consecutive eSCO instants measured in slots. Must be zero for SCO links.
Retransmission_Window	The size of the retransmission window measured in slots. Must be zero for SCO links.
Rx_Packet_Length	Length in bytes of the eSCO payload in the receive direction. Must be zero for SCO links.
Tx_Packet_Length	Length in bytes of the eSCO payload in the transmit direction. Must be zero for SCO links.
Air_Mode	Parameter describing air mode settings. Possible values are: HCI_AIR_MODE_FORMAT_U_LAW HCI_AIR_MODE_FORMAT_A_LAW HCI_AIR_MODE_FORMAT_CVSD HCI_AIR_MODE_FORMAT_TRANSPARENT_DATA

etSynchronous_Connection_Changed_Event

This event indicates to the Host that an existing Synchronous connection has been reconfigured. This event also indicates to the initiating Host (if the change was host initiated) if the issued command failed or was successful.

Return Structure:

```
typedef struct
{
  Byte_t       Status;
  Word_t       Connection_Handle;
  Byte_t       Transmission_Interval;
  Byte_t       Retransmission_Window;
  Word_t       Rx_Packet_Length;
  Word_t       Tx_Packet_Length;
} HCI_Synchronous_Connection_Changed_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Connection_Handle	Connection Handle used to identify the connection between two Bluetooth devices.
Transmission_Interval	Time between two consecutive SCO/eSCO instants measured in slots.
Retransmission_Window	The size of the retransmission window measured in slots. Must be zero for SCO links.
Rx_Packet_Length	Length in bytes of the SCO/eSCO payload in the receive direction.
Tx_Packet_Length	Length in bytes of the SCO/eSCO payload in the transmit direction.

etSniff_Subrating_Event

Indicates that the device associated with Connection_Handle has either enabled sniff subrating or sniff subrating parameters have changed.

Return Structure:

```
typedef struct
{
  Byte_t    Status;
  Word_t    Connection_Handle;
  Word_t    Maximum_Transmit_Latency;
  Word_t    Maximum_Receive_Latency;
  Word_t    Minimum_Remote_Timeout;
  Word_t     Minimum_Local_Timeout;
} HCI_Sniff_Subrating_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Connection_Handle	Connection Handle used to identify the connection between two Bluetooth devices.

Maximum_Transmit_Latency	Max latency for data transmitted from local to remote device.
Maximum_Receive_Latency	Max latency for data received by local from the remote device.
Minimum_Remote_Timeout	Base sniff subrate timeout that remote device should use. Expressed in baseband slots.
Minimum_Local_Timeout	Base sniff subrate, in baseband slots, that local device will use.

etExtended_Inquiry_Result_Event

Indicates that BR/EDR controller has responded during inquiry process with extended inquiry results. Sent from controller to host upon reception of Extended Inquiry Response from a remote device. This event is only generated when Inquiry_Mode was set to 0x02 of last Write_Inquiry_Mode command.

Return Structure:

```
typedef struct
{
  Byte_t                               Num_Responses;
  HCI_Extended_Inquiry_Result_Data_t   HCI_Inquiry_Result_Data;
} HCI_Extended_Inquiry_Result_Event_Data_t;
```

Event Parameters:

Num_Responses	Number of responses from the inquiry, Extended Inquiry Result event always has this set to 0x01.
HCI_Inquiry_Result_Data	Extended inquiry response data as defined in the Specification.

```
typedef struct
{
  BD_ADDR_t            BD_ADDR;
  Byte_t               Page_Scan_Repetition_Mode;
  Byte_t               Reserved;
  Class_of_Device_t    Class_of_Device;
  Word_t               Clock_Offset;
  Byte_t               RSSI;
  Extended_Inquiry_Response_Data_t Extended_Inquiry_Response;
} HCI_Extended_Inquiry_Result_Data_t;
```

etEncryption_Key_Refresh_Complete_Event

Indicates that encryption key was refreshed on the given connection handle.

Return Structure:

```
typedef struct
{
  Byte_t     Status;
  Word_t     Connection_Handle;
} HCI_Encryption_Key_Refresh_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Connection_Handle	Connection Handle used to identify the connection between two Bluetooth devices.

etIO_Capability_Request_Event

Indicates that the IO capabilities of the host are required for Simple Pairing.

Return Structure:

```
typedef struct
{
   BD_ADDR_t      BD_ADDR;
} HCI_IO_Capability_Request_Event_Data_t;
```

Event Parameters:

BD_ADDR	Bluetooth address of the remote device involved in the Simple Pairing.

etIO_Capability_Response_Event

Indicates that IO capabilities from remote device have been received.

Return Structure:

```
typedef struct
{
  BD_ADDR_t  BD_ADDR;
  Byte_t         IO_Capability;
  Byte_t         OOB_Data_Present;
  Byte_t         Authentication_Requirements;
} HCI_IO_Capability_Response_Event_Data_t;
```

Event Parameters:

BD_ADDR	Bluetooth device address of the remote device whose IO capabilities have been received.
IO_Capability	This value is the received IO_Capability and may be one of the following (all others reserved): 0x00 : DisplayOnly 0x01 : DisplayYesNo 0x02 : KeyboardOnly

	0x03 : NoInputNoOutput
OOB_Data_Present	Value indicating the OOB Data present and may be one of the following values (all others reserved): 0x00 : OOB authentication data not present 0x01 : OOB authentication data from remote device present
Authentication_Requirements	Contains the authentication requirements and may be one of the following (all others reserved): 0x00 : MITM Protection Not Required –No Bonding 0x01 : MITM Protection Required – No Bonding 0x02:MITM Protection Not Required – Dedicated Bonding 0x03 : MITM Protection Required – Dedicated Bonding 0x04 : MITM Protection Not Required – General Bonding 0x05 : MITM Protection Required – General Bonding

etUser_Confirmation_Request_Event

This event occurs when user confirmation the number value in the event parameter Numeric_Value is required.

Return Structure:

```
typedef struct
{
  BD_ADDR_t       BD_ADDR;
  DWord_t         Numeric_Value;
} HCI_User_Confirmation_Request_Event_Data_t;
```

Event Parameters:

BD_ADDR	Bluetooth device address of the remote device involved in Simple Pairing.
Numeric_Value	The numeric value in the range 0 – 999999 (decimal) that needs confirmation.

etUser_Passkey_Request_Event

Indicates that a passkey is required as part of Simple Pairing.

Return Structure:

```
typedef struct
{
  BD_ADDR_t       BD_ADDR;
} HCI_User_Passkey_Request_Event_Data_t;
```

Event Parameters:

BD_ADDR	Bluetooth device address of the remote device involved in Simple Pairing.

etRemote_OOB_Data_Request_Event

Indicates that Simple Pairing Hash C and the Simple Pairing Randomizer R is required for Secure Simple Pairing.

Return Structure:

```
typedef struct
{
  BD_ADDR_t       BD_ADDR;
} HCI_Remote_OOB_Data_Request_Event_Data_t;
```

Event Parameters:

BD_ADDR	Bluetooth device address of the remote device involved in Simple Pairing.

etSimple_Pairing_Complete_Event

Indicates that Simple Pairing has completed with the status returned in Status event parameter.

Return Structure:

```
typedef struct
{
  Byte_t          Status;
  BD_ADDR_t       BD_ADDR;
} HCI_Simple_Pairing_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
BD_ADDR	Bluetooth device address of the remote device involved in Simple Pairing.

etLink_Supervision_Timeout_Changed_Event

This event notifies a slave's host that the slave's controller has had it Link Supervision Timeout parameter changed.

Return Structure:

```
typedef struct
{
  Word_t   Connection_Handle;
  Word_t   Link_Supervision_Timeout;
} HCI_Link_Supervision_Timeout_Changed_Event_Data_t;
```

Event Parameters:

Connection_Handle	Connection handle whose Link Supervision Timeout parameter has changed.

Link_Supervision_Timeout	The new Link Supervision Timeout parameter value in number of baseband slots.

etEnhanced_Flush_Complete_Event

Indicates that for the specified handle an Enhanced Flush has completed.

Return Structure:

```
typedef struct
{
  Word_t    Connection_Handle;
} HCI_Enhanced_Flush_Complete_Event_Data_t;
```

Event Parameters:

Connection_Handle	Connection Handle used to identify the connection between two Bluetooth devices.

etUser_Passkey_Notification_Event

Used to provide a passkey for display to user as part of Simple Pairing.

Return Structure:

```
typedef struct
{
  BD_ADDR_t      BD_ADDR;
  DWord_t        Passkey;
} HCI_User_Passkey_Notification_Event_Data_t;
```

Event Parameters:

BD_ADDR	Bluetooth device address of the remote device involved in Simple Pairing.
Passkey	The passkey to be displayed, in range 0 – 999999 (decimal).

etKeypress_Notification_Event

Sent after passkey notification has been received by remote device whose Bluetooth device address is BD_ADDR.

Return Structure:

```
typedef struct
{
  BD_ADDR_t      BD_ADDR;
  Byte_t         Notification_Type;
} HCI_Keypress_Notification_Event_Data_t;
```

Event Parameters:

BD_ADDR	Bluetooth device address of the remote device involved in Simple Pairing.

Notification_Type	Type of notification which may be one of the following (all others reserverd): 0x00 : Passkey entry started 0x01 : Passkey digit entered 0x02 : Passkey digit erased 0x03 : Passkey cleared 0x04 : Passkey entry completed

etRemote_Host_Supported_Features_Notification_Event

Returns the LMP extended features page which contains host features.

Return Structure:

```
typedef struct
{
  BD_ADDR_t       BD_ADDR;
  LMP_Features_t  Host_Supported_Features;
} HCI_Remote_Host_Supported_Features_Notification_Event_Data_t;
```

Event Parameters:

BD_ADDR	Address of the remote device.
Host_Supported_Features	Bitmap of host supported features page of LMP extended features.

etPhysical_Link_Complete_Event

Indicates to the host that a new physical link has been established.

Return Structure:

```
typedef struct
{
  Byte_t Status;
  Byte_t Physical_Link_Handle;
} HCI_Physical_Link_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Physical_Link_Handle	Handle identifying the physical link that has been established.

etChannel_Selected_Event

Indicates that link information data is available to be read using Read Local Amp ASSOC command.

Return Structure:

```
typedef struct
{
  Byte_t Physical_Link_Handle;
} HCI_Channel_Selected_Event_Data_t;
```

Event Parameters:

Physical_Link_Handle	Handle of the physical link.

etDisconnection_Physical_Link_Complete_Event

Occurs when the physical link identified by Physical_Link_Handle is terminated.

Return Structure:

```
typedef struct
{
  Byte_t Status;
  Byte_t Physical_Link_Handle;
  Byte_t Reason;
} HCI_Disconnection_Physical_Link_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Physical_Link_Handle	Handle of the physical link that was terminated.
Reason	Reason that the physical link was terminated, specified in Error Code section of the Bluetooth Specification.

etPhysical_Link_Loss_Early_Warning_Event

Occurs when there is indication that the physical link indentified by Physical_Link_Handle may be disrupted.

Return Structure:

```
typedef struct
{
  Byte_t Physical_Link_Handle;
  Byte_t Link_Loss_Reason;
} HCI_Physical_Link_Loss_Early_Warning_Event_Data_t;
```

Event Parameters:

Physical_Link_Handle	Handle of the physical link that may be disrupted.
Reason	Value indicating the reason for this event. May be one of the following (all others reserved): 0x00 : Unknown 0x01 : Range related 0x02 : Bandwidth related

0x03 : Resolving conflict
0x04 : Interference

etPhysical_Link_Recovery_Event

Indicates that whatever caused a previous etPhysical_Link_Loss_Early_Warning_Event has now been cleared.

Return Structure:

```
typedef struct
{
   Byte_t     Physical_Link_Handle
} HCI_Physical_Link_Recovery_Event_Data_t;
```

Event Parameters:

Physical_Link_Handle	Handle of the physical link to which this pertains.

etLogical_Link_Complete_Event

Indicates to both end whether a Logical Link was successfully established or not.

Return Structure:

```
typedef struct
{
  Byte_t      Status;
  Word_t     Logical_Link_Handle;
  Byte_t      Physical_Link_Handle;
  Byte_t      Tx_Flow_Spec_ID;
} HCI_Logical_Link_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Logical_Link_Handle	Handle of Logical Link to be used to identify a connection between two controllers.
Physical_Link_Handle	Handle of the physical link over which the logical link has been established.
Tx_Flow_Spec_ID	Flow Spec ID of the newly created Logical Link.

etDisconnection_Logical_Link_Complete_Event

Occurs when a Logical Link on the local controller is terminated.

Return Structure:

```
typedef struct
{
  Byte_t Status;
  Word_t Logical_Link_Handle;
  Byte_t Reason;
} HCI_Disconnection_Logical_Link_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Logical_Link_Handle	Handle of the Logical Link that was terminated.
Reason	Reason, defined in Bluetooth Specification Error Codes, for the termination.

etFlow_Spec_Modify_Complete_Event

Indicates that a Flow Spec Modify command has completed.

Return Structure:

```
typedef struct
{
  Byte_t Status;
  Word_t Handle;
} HCI_Flow_Spec_Modify_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Handle	Connection handle if receiving controller is a BR/EDR Controller, Logical Link Handle if receiver is AMP Controller or if it is a connection between BR/EDR controllers with communicating AMPS.

etNumber_Of_Completed_Data_Blocks_Event

Indicates to the host HCI ACL Data Packets completed and data block buffers freed for each handle since previous etNumber_Of_Completed_Data_Blocks_Event

Return Structure:

```
typedef struct
{
  Word_t                  Total_Num_Data_Blocks;
  Byte_t                  Number_of_Handles;
  HCI_Number_Of_Completed_Data_Blocks_Data_t
    HCI_Number_Of_Completed_Data_Blocks_Data[1];
} HCI_Number_Of_Completed_Data_Blocks_Event_Data_t;
```

Event Parameters:

Total_Num_Data_Blocks	If 0 indicates the size of the buffer pool may have changed. If non-zero indicates the number of free data block buffers in the Controller.
Number_of_Handles	Number of handles included in this event.

HCI_Number_Of_Completed_Data_Blocks_Data[1]	Contains for each handle the number of completed packets and freed blocks since the previous etNumber_Of_Completed_Data_Blocks_Event
	
	typedef struct
	{
	  Word_t Handle;
	  Word_t Num_Of_Completed_Packets;
	  Word_t Num_Of_Completed_Blocks;
	} **HCI_Number_Of_Completed_Data_Blocks_Data_t**;

etShort_Range_Mode_Change_Complete_Event

Occurs after a notification has been made to the Controller to change the Short Range Mode.

Return Structure:

```
typedef struct
{
  Byte_t Status;
  Byte_t Physical_Link_Handle;
  Byte_t Short_Range_Mode_State;
} HCI_Short_Range_Mode_Change_Complete_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Physical_Link_Handle	Handle of physical link to which change occurred.
Short_Range_Mode_State	The state of the Short Range Mode (0 – Disabled, 1 – Enable).

etAMP_Status_Change_Event

Indicates that the AMP status has changed.

Return Structure:

```
typedef struct
{
  Byte_t Status;
  Byte_t AMP_Status;
} HCI_AMP_Status_Change_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
AMP_Status	The new AMP status. See HCI_Read_Local_AMP_Info parameter listing for the possible values.

etAMP_Start_Test_Event

Occurs when HCI_AMP_Test_Command has completed and data is ready to be sent or received.

Return Structure:

```
typedef struct
{
  Byte_t Status;
  Byte_t Test_Scenario;
} HCI_AMP_Start_Test_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.
Test_Scenario	The scenario of the currently running test as defined in the Test Commands section of the PAL Specification. May be one of the following (all others reserved): 0x01 : Transmit Single Frames 0x02 : Receive frames

etAMP_Test_End_Event

Indicates that AMP controller has sent/received number of frames/burst configured

Return Structure:

```
typedef struct
{
  Byte_t Status;
  Byte_t Test_Scenario;
} HCI_AMP_Test_End_Event_Data_t;
```

Event Parameters:

Status	Status of this event. Zero (0) indicates event completed OK. Values from 0x01 to 0xFF are HCI status codes.

Test_Scenario	The scenario of the running test. May be one of the following (all others reserved): 0x01 : Transmit Single Frames 0x02 : Receive frames

etAMP_Receiver_Report_Event

The receiver report received by the tester from the AMP at interval configured by HCI_Enable_AMP_Receiver_Reports command.

Return Structure:

```
typedef struct
{
  Byte_t  Controller_Type;
  Byte_t  Reason;
  DWord_t Event_Type;
  Word_t  Number_Of_Frames;
  Word_t  Number_Of_Error_Frames;
  DWord_t Number_Of_Bits;
  DWord_t Number_Of_Error_Bits;
} HCI_AMP_Receiver_Report_Event_Data_t;
```

Event Parameters:

Controller_Type	The number for the controller. See Bluetooth Assigned Numbers.
Reason	Reasons for the report. Must be one of the following (all others reserved): 0x00 : Configured Interval Report 0x01 : Test Ended Report
Event_Type	The type of the event. Must be one of the following (all others reserved): 0x00 : Frames Received Report 0x01 : Frames Received and bits in error report (optional)
Number_Of_Frames	The number of frames received so far.
Number_Of_Error_Frames	The number of frames with bit errors received so far.
Number_Of_Bits	Number of bits received so far. Set to 0x00000000 if Event_Type is not 0x01.
Number_Of_Error_Bits	Number of error bits received so far. Set to 0x00000000 if Event_Type is not 0x01.

etPlatform_Specific_Event

Event type for platform specific events.

Return Structure:

```
typedef struct
{
  DWord_t      Platform_Event_Type;
  void         *Platform_Event_Data;
} HCI_Platform_Specific_Event_Data_t;
```

Event Parameters:

Platform_Event_Type	The type of the platform specific event
Platform_Event_Data	Void pointer for the platform specific event data.

2.2.12 HCI LE Meta Event Sub-events

The table below lists the HCI LE Meta sub-events supported by the current version of the Bluetooth Stack Protocol API. Each event's parameters are further described in text below. The events are an enumeration instance of the enumeration type: HCI_LE_Meta_Event_Type_t.

Subevent	Description
meConnection_Complete_Event	Indicates that a new connection has been created.
meAdvertising_Report_Event	Indicates that a Bluetooth device or multiple devices have responded to an active scan or received some information during a passive scan.
meConnection_Update_Complete_Event	Indicates that the controller has updated the connection parameters.
meRead_Remote_Used_Features_Complete_Event	Indicates the result of a Remote used feature request to a remote Bluetooth device.
meLong_Term_Key_Request	Indicates master device is trying to encrypt or re-encrypt the link and is requesting the long term key from the host.

meConnection_Complete_Event

This event indicates that a connection has been completed.

Return Structure:

```
typedef struct
{
  Byte_t       Status;
  Word_t       Connection_Handle;
  Byte_t       Role;
  Byte_t       Peer_Address_Type;
  BD_ADDR_t Peer_Address;
  Word_t       Conn_Interval;
  Word_t       Conn_Latency;
  Word_t       Supervision_Timeout;
  Byte_t       Master_Clock_Accuracy;
} HCI_LE_Connection_Complete_Event_Data_t;
```

Event Parameters:

Status	Contains the result connection attempt (success or fail)
Connection_Handle	Handle that identifies the connection created (success)
Role	Determines role of device in connection. Possible values are: HCI_LE_ROLE_IS_MASTER HCI_LE_ROLE_IS_SLAVE
Peer_Address_Type	Indicates type of address of peer. Possible values are: HCI_LE_ADDRESS_TYPE_PUBLIC HCI_LE_ADDRESS_TYPE_RANDOM
Peer_Address	Contains the device address of the peer device.
Conn_Interval	Contains the interval of the connection.
Conn_Latency	Contains the latency for this connection.
Supervision_Timeout	Contains the supervision timeout.
Master_Clock_Accuracy	Contains the accuracy of the master clock. Possible values are: HCI_LE_MASTER_CLOCK_ACCURACY_500_PPM HCI_LE_MASTER_CLOCK_ACCURACY_250_PPM HCI_LE_MASTER_CLOCK_ACCURACY_150_PPM HCI_LE_MASTER_CLOCK_ACCURACY_100_PPM HCI_LE_MASTER_CLOCK_ACCURACY_75_PPM HCI_LE_MASTER_CLOCK_ACCURACY_50_PPM HCI_LE_MASTER_CLOCK_ACCURACY_30_PPM HCI_LE_MASTER_CLOCK_ACCURACY_20_PPM

meAdvertising_Report_Event

This event indicates that a response to a scan has been received.

Return Structure:

```
typedef struct
{
  Byte_t                           Num_Responses;
  HCI_LE_Advertising_Report_Data_t HCI_LE_Advertising_Report_Data[1];
} HCI_LE_Advertising_Report_Event_Data_t;
```

Event Parameters:

Num_Responses	Number of devices responding to the scan
HCI_LE_Advertising_Report_Data	An array of Num_Responses size that contains the reporting data from the devices. This array will contain zero (or more) entries. The total number of entries is given by the Num_Reponses member. Each entry is of the following structure:

```
typedef struct
{
  Byte_t               Event_Type;
  Byte_t               Address_Type;
  BD_ADDR_t            Address;
  Byte_t               Data_Length;
  Advertising_Data_t   Data;
  Byte_t               RSSI;
} HCI_LE_Advertising_Report_Data_t;
```

Where,

Event_Type has the following possible values:

```
HCI_LE_ADVERTISING_REPORT_EVENT_
      TYPE_CONNECTABLE_
      UNDIRECTED
HCI_LE_ADVERTISING_REPORT_EVENT_
      TYPE_CONNECTABLE_DIRECTED
HCI_LE_ADVERTISING_REPORT_EVENT_
      TYPE_SCANNABLE_UNDIRECTED
HCI_LE_ADVERTISING_REPORT_EVENT_
      TYPE_NONCONNECTABLE_
      UNDIRECTED
HCI_LE_ADVERTISING_REPORT_EVENT_
      TYPE_SCAN_RESPONSE
```

Address_Type has the following possible values:

```
HCI_LE_ADDRESS_TYPE_PUBLIC
HCI_LE_ADDRESS_TYPE_RANDOM
```

Data_Length specifies the total number of advertising data bytes contained in the Data member.

Data contains the advertising data returned from the peer device.
RSSI contains the peer devices RSSI value.

meConnection_Update_Complete_Event

This event indicates the completion of the updating of the connection parameters.

Return Structure:

```
typedef struct
{
  Byte_t    Status;
  Word_t    Connection_Handle;
  Word_t    Conn_Interval;
  Word_t    Conn_Latency;
  Word_t    Supervision_Timeout;
} HCI_LE_Connection_Update_Complete_Event_Data_t;
```

Event Parameters:

Status	Determines whether the command was completed successfully.
Connection_Handle	Handle to identify the connection that was updated.
Conn_Interval	Contains the current connection's interval.
Conn_Latency	Contains the current connection's latency.
Surpervision_Timeout	Contains the current connection's supervision timeout.

meRead_Remote_Used_Features_Complete_Event

This event indicates the completion of the reading of features supported by a remote device.

Return Structure:

```
typedef struct
{
  Byte_t          Status;
  Word_t          Connection_Handle;
  LE_Features_t   LE_Features;
} HCI_LE_Read_Remote_Used_Features_Complete_Event_Data_t;
```

Event Parameters:

Status	Determines whether the command was completed successfully.
Connection_Handle	Handle to identify the connection created.
LE_Features	Bit Mask List of used LE features.

meLong_Term_Key_Request_Event

This event indicates the request of a long term key from the host for a specific peer device.

Return Structure:

```
typedef struct
{
  Word_t            Connection_Handle;
  Random_Number_t   Random_Number;
  Word_t            Encrypted_Diversifier;
} HCI_LE_Long_Term_Key_Request_Event_Data_t;
```

Event Parameters:

Connection_Handle	Handle to identify the connection.
Random_Number	A 64 bit random number.
Encrypted_Diversifier	16 bit diversifier.

2.3 L2CAP API

L2CAP provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability, segmentation and reassembly operation, and group abstractions. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 kilobytes in length. This section is divided into three subsections: 2.3.1 covers the L2CAP service primitives, 2.3.2 covers the L2CAP event functions andPrototype and 2.3.3 covers the L2CAP events. The actual prototypes and constants outlined in this section can be found in the **L2CAPAPI.H** header file in the Bluetopia distribution.

2.3.1 L2CAP Service Primitives

The available service primitives are accessed via the functions listed in the table below, and are described in the text that follows.

Function	Description
L2CA_Set_Timer_Values	Set timers used to control operation of the stack.
L2CA_Get_Timer_Values	Retrieve timers that control stack operation.
L2CA_Connect_Request	Create a logical L2CAP connection.
L2CA_Connect_Response	Respond to an L2CAP connection indication.
L2CA_Config_Request	Configure a channel prior to sending any data.
L2CA_Config_Response	Respond to an L2CAP configuration indication.
L2CA_Disconnect_Request	Break a logical L2CAP connection.
L2CA_Disconnect_Response	Respond to a L2CAP disconnection indication.
L2CA_Data_Write	Send data over a connection.
L2CA_Enhanced_Data_Write	Send data over a connection (optionally specifying queuing parameters).
L2CA_Fixed_Channel_Data_Write	Send data to a connected device over a fixed

	channel.
L2CA_Enhanced_Fixed_Channel_Data_Write	Send data to a connected device over a fixed channel (optionally specifying queuing parameters).
L2CA_Group_Data_Write	Send data to a group.
L2CA_Ping	Send an L2CA echo request.
L2CA_Get_Info	Request the value of a Bluetooth device parameter.
L2CA_Connection_Parameter_Update_Request	Request connection parameter update.
L2CA_Connection_Parameter_Update_Response	Respond to a connection parameter update.
L2CA_Group_Create	Create a group in order to send and receive connectionless data from other devices.
L2CA_Group_Close	Close out a group.
L2CA_Group_Add_Member	Add a Bluetooth device to a group.
L2CA_Group_Remove_Member	Remove a Bluetooth device from a group.
L2CA_Get_Group_Membership	Obtain a list of members to a group.
L2CA_Enable_CLT	Enable reception of group messages.
L2CA_Disable_CLT	Disable reception of group messages.
L2CA_Flush_Channel_Data	Flush queued L2CAP data.
L2CA_Get_Current_Channel_Configuration	Retrieve configuration information on a channel.
L2CA_Get_Link_Connection_Configuration	Queries the current Link Connection Request/Response Configuration.
L2CA_Set_Link_Connection_Configuration	Changes the current L2CA_Set_Link_Connection_Configuration.
L2CA_Get_Channel_Queue_Threshold	Retrieves the L2CAP Channel Queing Threshold information for the Bluetooth Stack L2CAP Module.
L2CA_Set_Channel_Queue_Threshold	Changes the L2CAP Channel Queing Threshold information for the Bluetooth Stack L2CAP Module.
L2CA_Register_PSM	Registers an L2CAP callback function (for a PSM).
L2CA_Un_Register_PSM	Un-registers a previously register L2CAP callback function (for a PSM).

L2CA_Register_Fixed_Channel	Registers an L2CAP callback function (for a fixed channel).
L2CA_Un_Register_Fixed_Channel	Un-registers a previously register L2CAP callback function (for a fixed channel).

L2CA_Set_Timer_Values

Set timer values that are used to control operation of the stack.

Prototype:

int BTPSAPI **L2CA_Set_Timer_Values**(unsigned int BluetoothStackID,
    L2CA_Timer_Values_t *L2CA_Timer_Values)

Parameters:

BluetoothStackID[1] Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

L2CA_Timer_Values Stack control timer values. This is the structure defined as:

```
typedef struct
{
   unsigned int        RTXTimerVal;
   unsigned int        ERTXTimerVal;
   unsigned int        IdleTimerVal;
   unsigned int        ConfigStateTimerVal;
   unsigned int        ReceiveSegmentTimerVal;
} L2CA_Timer_Values_t;
```

The timers that are provided in this structure can be adjusted to provide appropriate timing for the profile being implemented. The timer values are specified in seconds. Timers RTXTimerVal and ERTXTimerVal are defined in the L2CAP specifications. Refer to the specification for information on these timers. The IdleTimerVal is added to support the idea of Client and Server L2CAP connections. L2CAP connections are established by Clients to Servers. At the time that the ACL connection is to be terminated, the Client should be the one to initiate the disconnection of the ACL link. When an L2CAP server denotes that no CIDs are open on an ACL link, a timer of value IdleTimerVal is started to allow the Client time to disconnect the ACL link. If the Client fails to disconnect the ACL link that the expiration of this timer, the server will then perform the disconnection. If this timer is set to a value of Zero, then the Server will attempt to disconnect the ACL when the last L2CAP channel is released. The ConfigStateTimerVal is used to control the amount of time that the stack is allowed to be in the Config State. If the configuration process is not complete at the expiration of this timer, the connection will be terminated. The ReceiveSegmentTimerVal is used to control the time that the stack will wait for the next segmented data packet to arrive. If the stack is waiting on a continuation information during the recombination of packets at the time this timer expires, the collected data will be discarded and an Error Event will be issued. The following constants for each timer define the range of values that each timer may be set:

L2CAP_RTX_TIMER_MINIMUM_VALUE
L2CAP_RTX_TIMER_MAXIMUM_VALUE
L2CAP_RTX_TIMER_DEFAULT_VALUE

L2CAP_ERTX_TIMER_MINIMUM_VALUE
L2CAP_ERTX_TIMER_MAXIMUM_VALUE
L2CAP_ERTX_TIMER_DEFAULT_VALUE

L2CAP_IDLE_TIMER_MINIMUM_VALUE
L2CAP_IDLE_TIMER_MAXIMUM_VALUE
L2CAP_IDLE_TIMER_DEFAULT_VALUE

L2CAP_CONFIG_TIMER_MINIMUM_VALUE
L2CAP_CONFIG_TIMER_MAXIMUM_VALUE
L2CAP_CONFIG_TIMER_DEFAULT_VALUE

L2CAP_RECEIVE_TIMER_MINIMUM_VALUE
L2CAP_RECEIVE_TIMER_MAXIMUM_VALUE
L2CAP_RECEIVE_TIMER_DEFAULT_VALUE

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Get_Timer_Values

Retrieve the timers which control the operation of the stack.

Prototype:

int BTPSAPI **L2CA_Get_Timer_Values**(unsigned int BluetoothStackID,
L2CA_Timer_Values_t *L2CA_Timer_Values)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
L2CA_Timer_Values	Stack control timer values. This is the structure defined as:

```
typedef struct
{
   unsigned int       RTXTimerVal;
   unsigned int       ERTXTimerVal;
   unsigned int       IdleTimerVal;
   unsigned int       ConfigStateTimerVal;
   unsigned int       ReceiveSegmentTimerVal;
} L2CA_Timer_Values_t;
```

See description of these timers in the Set function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Connect_Request

This function is responsible for requesting the creation of a Logical L2CAP Connection with the specified Bluetooth device address. This function returns a positive, non-zero Local Channel Identifier (LCID) if the L2CAP Connection Request was issued successfully, or a negative, return error code indicating an error.

Prototype:

int BTPSAPI **L2CA_Connect_Request**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Word_t PSM, L2CA_Event_Callback_t L2CA_Event_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device to which an L2CAP logical channel is to be established.

PSM	Protocol/Service Multiplexer identifier of the remote device to which the logical channel connection is to be made.
L2CA_Event_Callback	Pointer to a callback function to be used by the L2CAP layer to dispatch L2CAP Event information for this connection.
CallbackParameter	User defined value to be used by the L2CAP layer as an input parameter for all callbacks.

Return:

Positive non-zero value if function was successful. The values represent the Connection Identifier (CID) that identifies the channel created.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_CONNECTION_STATE
BTPS_ERROR_ATTEMPTING_CONNECTION_TO_DEVICE
BTPS_ERROR_ADDING_CID_INFORMATION
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etConnect_Confirmation

etTimeout_Indication

etDisconnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Connect_Response

This function is used when responding to an L2CA_Connect_Indication Event.

Prototype:

int BTPSAPI **L2CA_Connect_Response**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t Identifier, Word_t LCID, Word_t Response,
Word_t Status);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device to which an L2CAP logical channel is to be established. The BD_ADDR is obtained from the L2CA_Connect_Indication event.

Identifier	L2CAP assigned number used to match requests with responses. The Identifier value is obtained from the L2CA_Connect_Indication event.
LCID	Local CID value used by the L2CAP layer to reference the logical channel being requested. The LCID value is obtained from the L2CA_Connect_Indication event.
Response	User supplied response to the connection request. The connection is accepted, rejected or pended by the value of this parameter. The currently defines response values are:

L2CAP_CONNECT_RESPONSE_RESPONSE_SUCCESSFUL
L2CAP_CONNECT_RESPONSE_RESPONSE_PENDING
L2CAP_CONNECT_RESPONSE_RESPONSE_REFUSED_PSM_NOT_REGISTERED
L2CAP_CONNECT_RESPONSE_RESPONSE_REFUSED_SECURITY_BLOCK
L2CAP_CONNECT_RESPONSE_RESPONSE_REFUSED_NO_RESOURCES

Status	The Status parameter only has significance when the Connection Pending response is provided and is used to provide extra information about the status of the connection. The currently defined status values are:

L2CAP_CONNECT_RESPONSE_STATUS_NO_FURTHER_INFORMATION
L2CAP_CONNECT_RESPONSE_STATUS_AUTHENTICATION_PENDING
L2CAP_CONNECT_RESPONSE_STATUS_AUTHORIZATION_PENDING

Return:

Zero (0) if successful submitting the Connect Response. This does not mean that the connect response has been delivered, but that the response was successfully submitted for delivery.

Negative if an Error occurred and the Response was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etTimeout_Indication

etDisconnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Config_Request

This function is used to issue a request to configure a channel. Channel configuration must be performed and successfully completed prior to the transfer of any user data over the channel. The configuration options to be negotiated for the channel are specified in the L2CA_Config_Request structure. Options that are not specified will be interpreted as the default value. The LinkTO value specifies the suggested Link Timeout value to be used for the CONNECTION. This value will only be used if it is less than the current Link Timeout setting.

Prototype:

int BTPSAPI **L2CA_Config_Request**(unsigned int BluetoothStackID, Word_t LCID, Word_t LinkTO, L2CA_Config_Request_t *ConfigRequest);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
LCID	Local CID value referencing the logical channel being configures.
LinkTO	Suggested Baseband Link Timeout value to be used for the connection.
ConfigRequest	Structure containing the configuration parameters to be negotiated.

```
typedef struct
{
  Word_t                        Option_Flags;
  Word_t                        InMTU;
  Word_t                        OutFlushTO;
  L2CA_Flow_Spec_t              OutFlow;
  L2CA_Mode_Info_t              ModeInfo;
  Byte_t                        FCS_Option;
  L2CA_Extended_Flow_Spec_t     ExtendedFlowSpec;
  Word_t                        ExtendedWindowSize;
} L2CA_Config_Request_t;
```

where, Option_Flags is a bit list. Possible bit values are:

```
L2CA_CONFIG_OPTION_FLAG_MTU
L2CA_CONFIG_OPTION_FLAG_FLUSH_TIMEOUT
L2CA_CONFIG_OPTION_FLAG_QOS
L2CA_CONFIG_OPTION_FLAG_MODE_INFO
L2CA_CONFIG_OPTION_FLAG_FCS_OPTION
L2CA_CONFIG_OPTION_FLAG_EXTENDED_FLOW_SPEC
L2CA_CONFIG_OPTION_FLAG_EXTENDED_WINDOW
              _SIZE
L2CA_CONFIG_OPTION_FLAG_CONTINUATION
```

and, the L2CA_Flow_Spec_t structure is defined as follows:

```
typedef struct
```

```
{
   Byte_t       Flags;
   Byte_t       ServiceType;
   DWord_t      TokenRate;
   DWord_t      TokenBucketSize;
   DWord_t      PeakBandwidth;
   DWord_t      Latency;
   DWord_t      DelayVariation;
} L2CA_Flow_Spec_t;
```

and, the L2CA_Extended_Flow_Spec_t structure is defined as follows:

```
typedef
{
   Byte_t    Identifier;
   Byte_t    ServiceType;
   Word_t    MaxSDU;
   DWord_t   SDUInterArrivalTime;
   DWord_t   AccessLatency;
   DWord_t   FlushTimeout;
} L2CA_Extended_Flow_Spec_t;
```

Response — User supplied response to the connection request. The connection is accepted, rejected or pended by the value of this parameter. The currently defined response types are:

L2CAP_CONNECT_RESPONSE_RESPONSE_SUCCESSFUL
L2CAP_CONNECT_RESPONSE_RESPONSE_PENDING
L2CAP_CONNECT_RESPONSE_RESPONSE_REFUSED_PSM_NOT_REGISTERED
L2CAP_CONNECT_RESPONSE_RESPONSE_REFUSED_SECURITY_BLOCK
L2CAP_CONNECT_RESPONSE_RESPONSE_REFUSED_NO_RESOURCES

Status — The Status parameter only has significance when the Connection Pending response is provided and is used to provide extra information about the status of the connection. The currently defined response types are:

L2CAP_CONNECT_STATUS_NO_FURTHER_INFORMATION
L2CAP_CONNECT_STATUS_AUTHENTICATION_PENDING
L2CAP_CONNECT_STATUS_AUTHORIZATION_PENDING

Return:

Zero (0) if successful submitting the Connect Response. This does not mean that the connect response has been delivered, but that the response was successfully submitted for delivery.

Negative if an Error occurred and the Response was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_ADDING_IDENTIFIER_INFORMATION
BTPS_ERROR_INVALID_FLUSH_TIMEOUT_VALUE
BTPS_ERROR_INVALID_STATE_FOR_CONFIG

BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etConfig_Confirmation

etTimeout_Indication

etDisconnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Config_Response

This function is used when responding to an L2CA_Config_Indication Event.

Prototype:

int BTPSAPI **L2CA_Config_Response**(unsigned int BluetoothStackID, Word_t LCID, Word_t Result, L2CA_Config_Response_t *ConfigResponse);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
LCID	Local CID value referencing the logical channel being configured.
Result	Parameter that indicates the result of the Configuration Request. The currently defined Result values are:

L2CAP_CONFIGURE_RESPONSE_RESULT_SUCCESS
L2CAP_CONFIGURE_RESPONSE_RESULT_FAILURE_UNACCEPTABLE_PARAMETERS
L2CAP_CONFIGURE_RESPONSE_RESULT_FAILURE_REJECTED_NO_REASON
L2CAP_CONFIGURE_RESPONSE_RESULT_FAILURE_UNKNOWN_OPTIONS
L2CAP_CONFIGURE_RESPONSE_RESULT_TIMEOUT

ConfigResponse	Structure containing the configuration parameter being negotiated.

```
typedef struct
{
   Word_t                  Option_Flags;
   Word_t                  OutMTU;
   Word_t                  InFlushTO;
   L2CA_Flow_Spec_t        InFlow;
   L2CA_Mode_Info_t              ModeInfo;
   Byte_t                        FCS_Option;
   L2CA_Extended_Flow_Spec_t     ExtendedFlowSpec;
   Word_t                        ExtendedWindowSize;
} L2CA_Config_Response_t;
```

where, Option_Flags is a bit list. Possible bit values are:

```
L2CA_CONFIG_OPTION_FLAG_MTU
L2CA_CONFIG_OPTION_FLAG_FLUSH_TIMEOUT
L2CA_CONFIG_OPTION_FLAG_QOS
L2CA_CONFIG_OPTION_FLAG_MODE_INFO
L2CA_CONFIG_OPTION_FLAG_FCS_OPTION
L2CA_CONFIG_OPTION_FLAG_EXTENDED_FLOW_SPEC
L2CA_CONFIG_OPTION_FLAG_EXTENDED_WINDOW
              _SIZE
L2CA_CONFIG_OPTION_FLAG_CONTINUATION
```

and, the L2CA_Flow_Spec_t structure is defined as follows:

```
typedef struct
{
   Byte_t      Flags;
   Byte_t      ServiceType;
   DWord_t     TokenRate;
   DWord_t     TokenBucketSize;
   DWord_t     PeakBandwidth;
   DWord_t     Latency;
   DWord_t     DelayVariation;
} L2CA_Flow_Spec_t;
```

and, the L2CA_Extended_Flow_Spec_t structure is defined as follows:

```
typedef
{
   Byte_t    Identifier;
   Byte_t    ServiceType;
   Word_t    MaxSDU;
   DWord_t   SDUInterArrivalTime;
   DWord_t   AccessLatency;
   DWord_t   FlushTimeout;
} L2CA_Extended_Flow_Spec_t;
```

Return:

Zero (0) if successful submitting the Configuration Response. This does not mean that the configuration response has been delivered, but that the response was successfully submitted for delivery.

Negative if an Error occurred and the Response was not submitted. Possible values are:

```
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER
```

Possible Events:

etTimeout_Indication

etDisconnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Disconnect_Request

This function is responsible for requesting a Disconnect of a Logical L2CAP Connection with the specified Bluetooth device address. This function returns a Zero if the L2CAP Disconnection Request was successfully submitted, or a negative return error code indicating an error. When the Disconnect of the channel is complete, an L2CA_Disconnect_Confirmation event will be issued.

Prototype:

int BTPSAPI **L2CA_Disconnect_Request**(unsigned int BluetoothStackID, Word_t LCID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
LCID	Local CID value referencing the logical channel to be disconnected.

Return:

Zero (0) if the disconnect request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_ADDING_IDENTIFIER_INFORMATION
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etTimeout_Indication

etDisconnect_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Disconnect_Response

This function is used when responding to an L2CA_Disconnect_Indication Event. This function must be called from within the callback for the L2CA_Disconnect_Indication. If this function is not called from within the L2CA_Disconnect_Indication event callback, the L2CAP layer will provide a response automatically.

Prototype:

int BTPSAPI **L2CA_Disconnect_Response**(unsigned int BluetoothStackID, Word_t LCID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
LCID	Local CID value used by the L2CAP layer to reference the logical channel to disconnect.

Return:

Zero (0) if successful submitting the Disconnect Response. This does not mean that the Disconnect Response has been delivered, but that the Response was successfully submitted for delivery.

Negative if an Error occurred and the Response was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etTimeout_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Data_Write

This function is used to send data over a specified channel.

Prototype:

int BTPSAPI **L2CA_Data_Write**(unsigned int BluetoothStackID, Word_t LCID,
    Word_t Data_Length, Byte_t *Data);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

LCID	Local CID value used by the L2CAP layer to reference the logical channel on which to send the data.
Data_Length	Number of characters to be sent over the channel.
Data	Pointer to a buffer of data to be sent over the channel.

Return:

Zero (0) if successful submitting the Data for transmission.

Negative if an Error occurred and the Data was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_WRITING_DATA_TO_DEVICE
BTPS_ERROR_MEMORY_ALLOCATION_ERROR
BTPS_ERROR_NEGOTIATED_MTU_EXCEEDED
BTPS_ERROR_CHANNEL_NOT_IN_OPEN_STATE
BTPS_ERROR_INVALID_CID_TYPE
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Note that if this function returns the error code:

BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

then this is a signal to the caller that the requested data could NOT be sent because the requested data could not be queued in the outgoing L2CAP Queue (i.e. queuing criteria was not met). The caller then, must wait for the etChannel_Buffer_Empty_Indication Event before trying to send any more data. When this event is signaled, another attempt can be made to send the data to the remote device.

Possible Events:

etData_Error_Indication

etDisconnect_Indication

etChannel_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Enhanced_Data_Write

This function is used to send data over a specified channel while optionally specifying queuing parameters. This function is similar to the L2CA_Data_Write() function except that this function allows the ability to specify optional queuing parameters. These queuing parameters can specify the following:

- How deep the queue should be (by number of queued packets)

This function provides two mechanisms when the (optional) queue thresholds are reached:

- Discard the oldest packet in the queue (and queue the specified packet)
- Do not queue the packet and inform the caller via a specific return value

Notes:

If this function is called with the QueueingParametes parameter set to NULL then this function behaves EXACTLY like calling the L2CA_Data_Write() function (i.e. packet is queued regardless).

If the L2CA_QUEUEING_FLAG_DISCARD_OLDEST is specified then this function will discard the oldest packet in the queue if the queue threshold criteria is satisfied. This allows a streaming-like mechanism to be implemented (i.e. the data will not back up, it will just be discarded).

Prototype:

int BTPSAPI **L2CA_Enhanced_Data_Write**(unsigned int BluetoothStackID,
    Word_t LCID, L2CA_Queueing_Parameters_t *QueueingParameters,
    Word_t Data_Length, Byte_t *Data);

Parameters:

BluetoothStackID[1]
Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

LCID
Local CID value used by the L2CAP layer to reference the logical channel on which to send the data.

QueingParameters
Optional pointer to a structure which describes the parameters that dictate how the packet is queued. This structure is defined as follows:

```
typedef struct
{
  DWord_t Flags;
  DWord_t QueueLimit;
  DWord_t LowThreshold;
} L2CA_Queueing_Parameters_t;
```

where, Flags is defined to be one of the following values:

```
L2CA_QUEUEING_FLAG_LIMIT_BY_PACKETS
L2CA_QUEUEING_FLAG_DISCARD_OLDEST
```

where, QueueLimit defines the maximum queue limit specified in either number of packets or size (in bytes) depending on the Flags member value.

where, LowThreshold defines the lower threshold limit that must be reached before the

etChannel_Buffer_Empty_Indication

event is dispatched when the queue drains to the threshold limit

Data_Length	Number of characters to be sent over the channel.
Data	Pointer to a buffer of data to be sent over the channel.

Return:

Zero (0) if successful submitting the Data for transmission.

Negative if an Error occurred and the Data was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_WRITING_DATA_TO_DEVICE
BTPS_ERROR_MEMORY_ALLOCATION_ERROR
BTPS_ERROR_NEGOTIATED_MTU_EXCEEDED
BTPS_ERROR_CHANNEL_NOT_IN_OPEN_STATE
BTPS_ERROR_INVALID_CID_TYPE
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Note that if this function returns the error code:

BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

then this is a signal to the caller that the requested data could NOT be sent because the requested data could not be queued in the outgoing L2CAP Queue (i.e. queuing criteria was not met). The caller then must wait for the etChannel_Buffer_Empty_Indication Event before trying to send any more data. When this event is signaled, another attempt can be made to send the data to the remote device.

Possible Events:

etData_Error_Indication

etDisconnect_Indication

etChannel_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Fixed_Channel_Data_Write

This function is used to send data over a specific fixed channel.

Prototype:

int BTPSAPI **L2CA_Fixed_Channel_Data_Write**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, Word_t FCID, Word_t Data_Length, Byte_t *Data);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth protocol stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the device to send the specified fixed channel data.
FCID	Fixed channel ID that represents the fixed channel to send the data. This value is not the actual fixed channel itself, rather this a value that was returned from a successful call to the L2CA_Register_Fixed_Channel function.
Data_Length	Number of characters to be sent over the fixed channel.
Data	Pointer to a buffer of data to be sent over the fixed channel.

Return:

Zero (0) if successful submitting the Data for transmission.

Negative if an Error occurred and the Data was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_CID
BTPS_ERROR_DEVICE_NOT_CONNECTED
BTPS_ERROR_WRITING_DATA_TO_DEVICE
BTPS_ERROR_MEMORY_ALLOCATION_ERROR
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

L2CA_Enhanced_Fixed_Channel_Data_Write

This function is used to send data over a specified fixed channel while optionally specifying queuing parameters. This function is similar to the L2CA_Fixed_Channel_Data_Write() function except that this function allows the ability to specify optional queuing parameters. These queuing parameters can specify the following:

- How deep the queue should be (by number of queued packets)

This function provides two mechanisms when the (optional) queue thresholds are reached:

- Discard the oldest packet in the queue (and queue the specified packet)
- Do not queue the packet and inform the caller via a specific return value

Notes:

If this function is called with the QueueingParameters parameter set to NULL then this function behaves EXACTLY like calling the L2CA_Fixed_Channel_Write() function (i.e. packet is queued regardless).

If the L2CA_QUEUEING_FLAG_DISCARD_OLDEST is specified then this function will discard the oldest packet in the queue if the queue threshold criteria is satisfied. This allows a streaming-like mechanism to be implemented (i.e. the data will not back up, it will just be discarded).

Prototype:

```
int BTPSAPI L2CA_Fixed_Channel_Data_Write(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, Word_t FCID,
    L2CA_Queueing_Parameters_t *QueueingParameters,
    Word_t Data_Length, Byte_t *Data);
```

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the device to send the specified fixed channel data.
FCID	Fixed channel ID that represents the fixed channel to send the data. This value is not the actual fixed channel itself, rather this a value that was returned from a successful call to the L2CA_Register_Fixed_Channel function.
QueueingParameters	Optional pointer to a structure which describes the parameters that dictate how the packet is queued. This structure is defined as follows: `typedef struct { DWord_t Flags; DWord_t QueueLimit; DWord_t LowThreshold; } L2CA_Queueing_Parameters_t;` where, Flags is defined to be one of the following values: L2CA_QUEUEING_FLAG_LIMIT_BY_PACKETS L2CA_QUEUEING_FLAG_DISCARD_OLDEST where, QueueLimit defines the maximum queue limit specified in either number of packets or size (in bytes) depending on the Flags member value. where, LowThreshold defines the lower threshold limit that must be reached before the etFixed_Channel_Buffer_Empty_Indication event is dispatched when the queue drains to the threshold limit
Data_Length	Number of characters to be sent over the fixed channel.
Data	Pointer to a buffer of data to be sent over the fixed channel.

Return:

Zero (0) if successful submitting the Data for transmission.

Negative if an Error occurred and the Data was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_WRITING_DATA_TO_DEVICE
BTPS_ERROR_MEMORY_ALLOCATION_ERROR
BTPS_ERROR_NEGOTIATED_MTU_EXCEEDED
BTPS_ERROR_CHANNEL_NOT_IN_OPEN_STATE
BTPS_ERROR_INVALID_CID_TYPE
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Note that if this function returns the error code:

BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

then this is a signal to the caller that the requested data could NOT be sent because the requested data could not be queued in the outgoing L2CAP Queue (i.e. queuing criteria was not met). The caller then must wait for the etFixed_Channel_Buffer_Empty_Indication Event before trying to send any more data. When this event is signaled, another attempt can be made to send the data to the remote device.

Possible Events:

etFixed_Channel_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Group_Data_Write

This function is used to send data over a connectionless channel. This function makes a 'best effort' attempt to deliver the data to all members of the group.

Prototype:

int BTPSAPI **L2CA_Group_Data_Write**(unsigned int BluetoothStackID, Word_t LCID, Word_t Data_Length, Byte_t *Data);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
LCID	Local CID value used by the L2CAP layer to reference the Group to which to send the data. This values is obtained from a successful call to L2CA_Group_Create.

Data_Length	Number of characters to be sent to the group.
Data	Pointer to a buffer of data to be sent to the group.

Return:

Zero (0) if successful submitting the Data for transmission.

Negative if an Error occurred and the Data was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_WRITING_DATA_TO_DEVICE
BTPS_ERROR_MEMORY_ALLOCATION_ERROR
BTPS_ERROR_CONECTIONLESS_MTU_EXCEEDED
BTPS_ERROR_INVALID_CID_TYPE
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

NOTE - If this function returns the Error Code: BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE then this is a signal to the caller that the requested data could NOT be sent because the requested data could not be queued in the Outgoing L2CAP Queue. The caller then, must wait for the etChannel_Buffer_Empty_Indication Event before trying to send any more data. When this event is signaled, another attempt can be made to send the data to the remote device.

Possible Events:

etData_Error_Indication

etDisconnect_Indication

etChannel_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Ping

This function is used to send a Echo Request to a specified Bluetooth device. This function allows a message to be sent with the Ping, to which the receiver will echo back to the caller if the request is successful. If no message is to be sent with the request, the Data_Length parameter must be 0.

Prototype:

int BTPSAPI **L2CA_Ping**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Word_t Data_Length, Byte_t *Data, L2CA_Event_Callback_t L2CA_Event_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device to which an L2CAP logical channel is to be established.
Data_Length	Number of characters to be sent with the Ping.
Data	Pointer to a buffer of data to be sent with the Ping.
L2CA_Event_Callback	Pointer to a callback function to be used by the L2CAP layer to dispatch a reply to the Ping.
CallbackParameter	User defined value to be used by the L2CAP layer as an input parameter for the callbacks.

Return:

Positive, non-zero value if successful submitting the Ping Request.

Negative if an Error occurred and the Ping was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_IDENTIFIER_INFORMATION
BTPS_ERROR_MEMORY_ALLOCATION_ERROR
BTPS_ERROR_ADDING_CID_INFORMATION

Possible Events:

etEcho_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Get_Info

This function is used to retrieve specific information from a specified Bluetooth device.

Prototype:

int BTPSAPI **L2CA_Get_Info**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR,
    Word_t Info_Type, L2CA_Event_Callback_t L2CA_Event_Callback,
    unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

BD_ADDR	Device address of the Bluetooth device to which an L2CAP logical channel is to be established.
InfoType	Identifier of the information element to be retrieved. The currently definesdInfotypes are: L2CAP_INFORMATION_REQUEST_INFOTYPE_ CONNECTIONLESS_MTU L2CAP_INFORMATION_REQUEST_INFOTYPE_ EXTENDED_FEATURE_MASK
L2CA_Event_Callback	Pointer to a callback function to be used by the L2CAP layer to dispatch a reply to the Info Request.
CallbackParameter	User defined value to be used by the L2CAP layer as an input parameter for the callbacks.

Return:

Positive, non zero value if successful submitting the Info Request Request.

Negative if an Error occurred and the Info Request was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_IDENTIFIER_INFORMATION
BTPS_ERROR_ADDING_CID_INFORMATION

Possible Events:

etInformation_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Connection_Parameter_Update_Request

This function is used to request the remote device (LE master) update the connection parameters. This function can only be issued by an LE slave and the local host must have registered for the following fixed channel:

L2CAP_CHANNEL_IDENTIFIER_LE_SIGNALLING_CHANNEL

Prototype:

int BTPSAPI **L2CA_Connection_Parameter_Update_Request**(
unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Word_t IntervalMin,
Word_t IntervalMax, Word_t SlaveLatency, Word_t TimeoutMultiplier);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device to which an L2CAP logical channel is already established.
IntervalMin	Minimum value for the the connection interval. This should fall within the range: HCI_LE_CONNECTION_INTERVAL_MINIMUM HCI_LE_CONNECTION_INTERVAL_MAXIMUM
IntervalMax	This should be greater than or equal to Conn_Interval_Min and shall fall within the range: HCI_LE_CONNECTION_INTERVAL_MINIMUM HCI_LE_CONNECTION_INTERVAL_MAXIMUM Both intervals follow the rule: Time = N * 1.25 msec
SlaveLatency	Slave latency for connection. This should be in range: HCI_LE_CONNECTION_LATENCY_MINIMUM HCI_LE_CONNECTION_LATENCY_MAXIMUM
TimeoutMultiplier	Supervision timeout multiplier for LE link. This should be in range: HCI_LE_SUPERVISION_TIMEOUT_MINIMUM HCI_LE_SUPERVISION_TIMEOUT_MAXIMUM The Supervision_Timeout follows the rule: Time = N * 10 msec

Return:

Positive, non zero value if successful submitting the Connection Parameter Update Request.

Negative if an Error occurred and the Connection Parameter Update Request was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_ACTION_NOT_ALLOWED
BTPS_ERROR_NO_CALLBACK_REGISTERED
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_IDENTIFIER_INFORMATION
BTPS_ERROR_ADDING_CID_INFORMATION

Possible Events:

etConnection_Parameter_Update_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Connection_Parameter_Update_Response

This function is used to respond to connection parameter upate request received from the remote device (LE slave) to update the connection parameters. This function can only be issued by an LE master and the local host must have registered for the following fixed channel:

L2CAP_CHANNEL_IDENTIFIER_LE_SIGNALLING_CHANNEL

Prototype:

int BTPSAPI **L2CA_Connection_Parameter_Update_Response**(
unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Word_t Result);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device to which an L2CAP logical channel is already established.
Result	Result of the connection parameter update request. This will be one of the following values: L2CAP_CONNECTION_PARAMETER_UPDATE_RESPONSE_RESULT_ACCEPTED L2CAP_CONNECTION_PARAMETER_UPDATE_RESPONSE_RESULT_REJECTED

Return:

Positive, non zero value if successful submitting the Connection Parameter Update Response.

Negative if an Error occurred and the Connection Parameter Update Response was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_ACTION_NOT_ALLOWED
BTPS_ERROR_NO_CALLBACK_REGISTERED
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_IDENTIFIER_INFORMATION
BTPS_ERROR_ADDING_CID_INFORMATION

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Group_Create

This function is used to create a Group for the purpose of receiving Group Messages. The PSM value is used to filter the group messages. All group messages received having a matching PSM will be dispatched to the user if reception is enabled at the time the message is received. The RxEnable flag is used to specify the initial state of the receiver.

Prototype:

int BTPSAPI **L2CA_Group_Create**(unsigned int BluetoothStackID, Word_t PSM,
Boolean_t RxEnabled, L2CA_Event_Callback_t L2CA_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
PSM	Protocol/Service Multiplexer identifier of the Group messages to be received.
RxEnabled	Flag to controls the state of the receiver a creation. If this is TRUE, reception of the group messages is enabled. If FALSE, group messages are disabled.
L2CA_Event_Callback	Pointer to a callback function to be used by the L2CAP layer to dispatch group messages.
CallbackParameter	User defined value to be used by the L2CAP layer as an input parameter for the callbacks.

Return:

A positive, non-Zero value is returned after successfully creating the group. This value is the Group CID and is used to identify the group when future modifications to the group are made.

Negative if an Error occurred and the group was not created. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_ADDING_CID_INFORMATION
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etDisconnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Group_Close

This function is used to remove a Group and its members.

Prototype:

int BTPSAPI **L2CA_Group_Close**(unsigned int BluetoothStackID, Word_t CID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
CID	Connection Identifier that uniquely identifies the Group.

Return:

Zero (0) if successful removing the group.

Negative if an Error occurred and the group was not removed. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_CID

Possible Events:

etDisconnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Group_Add_Member

This function is used to add a member to a Group. If a connection to the specified device does not exist when the function is called, an attempt to establish a connection will be performed. The member is not added until a successful connection establishment has been made. Notification of the addition of the member will be made via the Group Callback function.

Prototype:

int BTPSAPI **L2CA_Group_Add_Member**(unsigned int BluetoothStackID, Word_t CID, BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
CID	Connection Identifier that uniquely identifies the Group.
BD_ADDR	Device address of the Bluetooth device to be added to the group.

Return:

Zero (0) if the add member request was successfully submitted. Notification of the result of the addition of the member will be received via the Group Callback function.

Negative if an Error occurred and the member was not added. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ATTEMPTING_CONNECTION_TO_DEVICE
BTPS_ERROR_ADDING_CID_INFORMATION
BTPS_ERROR_GROUP_MEMBER_ALREADY_EXISTS
BTPS_ERROR_CID_NOT_GROUP_CID
BTPS_ERROR_INVALID_CID

Possible Events:

etGroup_Member_Status

etDisconnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Group_Remove_Member

This function is used to remove a member to a Group.

Prototype:

int BTPSAPI **L2CA_Group_Remove_Member**(unsigned int BluetoothStackID,
  Word_t CID, BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
CID	Connection Identifier that uniquely identifies the Group.
BD_ADDR	Device address of the Bluetooth device to be removed from the group.

Return:

Zero (0) if the member was successfully removed.

Negative if an Error occurred and the member was not added. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GROUP_MEMBER_NOT_FOUND
BTPS_ERROR_CID_NOT_GROUP_CID
BTPS_ERROR_INVALID_CID

Possible Events:

etDisconnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Get_Group_Membership

This function is used to retrieve a list of members of a specified Group.

Prototype:

int BTPSAPI **L2CA_Get_Group_Membership**(unsigned int BluetoothStackID, Word_t CID, unsigned int *Result, unsigned int *MemberCount, unsigned int BufferSize, BD_ADDR_t *BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
CID	Connection Identifier that uniquely identifies the Group.
Result	Pointer to an integer to receive status information for the request. The currently defined result values are: L2CAP_GROUP_MEMBERSHIP_RESPONSE_RESULT_SUCCESS L2CAP_GROUP_MEMBERSHIP_RESPONSE_RESULT_FAILURE
MemberCount	Pointer to an integer to receive a count of the number of member entries that were moved to the BD_ADDR array.
BufferSize	Size in Bytes of the BD_ADDR buffer that will receive the array of member addresses.
BD_ADDR	Pointer to an array of type BD_ADDR_t. The function will fill the array with the device address of each member of the group.

Return:

Zero (0) if the member list was successfully created.

Negative if an Error occurred and the member was not added. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE
BTPS_ERROR_CID_NOT_GROUP_CID

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Enable_CLT

This function is used to enable the reception of Connectionless (Group) traffic.

Prototype:

int BTPSAPI **L2CA_Enable_CLT**(unsigned int BluetoothStackID, Word_t PSM);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
PSM	Protocol/Service Multiplexer identifier of the Group PSM message to be enabled.

Return:

Zero (0) if the traffic was successfully enabled.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Disable_CLT

This function is used to disable the reception of Connectionless (Group) traffic.

Prototype:

int BTPSAPI **L2CA_Disable_CLT**(unsigned int BluetoothStackID, Word_t PSM);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
PSM	Protocol/Service Multiplexer identifier of the Group PSM message to be disabled.

Return:

Zero (0) if the traffic was successfully disabled.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Flush_Channel_Data

This function is responsible for requesting that all queued L2CAP data for the specified Channel be flushed. This function should only be called under extreme circumstances, and normally need not be called. This function should be called when the caller has determined (by some means) that L2CAP Data has been sent (locally) and NOT received on the remote side AND the user wants to clear out any (potentially) buffered L2CAP Data for the channel (such that it will not be sent when next allowable). This condition can occur due to HCI Transport issues (infinite retransmits for example). This function returns a Zero if the L2CAP Channel data for the specified Channel was deleted successfully.

Prototype:

int BTPSAPI **L2CA_Flush_Channel_Data**(unsigned int BluetoothStackID, Word_t CID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
CID	Local CID value referencing the logical channel to be flushed.

Return:

Zero (0) if the channel flush was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_CID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Get_Current_Channel_Configuration

This function is used retrieve configuration information for a specified channel.

Prototype:

int BTPSAPI **L2CA_Get_Current_Channel_Configuration**(unsigned int BluetoothStackID, Word_t CID, L2CA_Config_Params_t *Channel_Config_Params);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
CID	Channel Identifier.
Channel_Config_Params	Pointer to a structure to receive the configuration information.

```
typedef struct
{
   Word_t               OutMTU;
   Word_t               InFlushTO;
   Word_t               OutFlushTO;
   L2CA_Flow_Spec_t     InFlow;
} L2CA_Config_Params_t;
```

where, the L2CA_Flow_Spec_t structure is defined as follows:

```
typedef struct
{
   Byte_t     Flags;
   Byte_t     ServiceType;
   DWord_t    TokenRate;
   DWord_t    TokenBucketSize;
   DWord_t    PeakBandwidth;
   DWord_t    Latency;
   DWord_t    DelayVariation;
```

} **L2CA_Flow_Spec_t**;

Return:

Zero (0) if the information was successfully transferred.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_CONNECTION_TO_DEVICE_LOST
BTPS_ERROR_INVALID_CID_TYPE
BTPS_ERROR_INVALID_CID

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Get_Link_Connection_Configuration

Get Lower Link Connection request/response configuration. This function exists to allow the programmer a method of determining how L2CAP is currently handling HCI ACL connection requests (both when L2CAP originates the HCI ACL connection and when L2CAP responds to remote HCI ACL requests). This functionality is provided to allow programmers a means to control L2CAP in a Point to Multi-Point environment. The default handling is that L2CAP doesn't allow a Role Switch at connection setup. This function allows the programmer to query/change this functionality if desired.

Prototype:

int BTPSAPI **L2CA_Get_Link_Connection_Configuration**(
  unsigned int BluetoothStackID,
  L2CA_Link_Connect_Params_t *L2CA_Link_Connect_Params)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
L2CA_Link_Connect_Params	Stack connection configuration values. This is the structure defined as:

```
typedef struct
{
    L2CA_Link_Connect_Request_Config_t
            L2CA_Link_Connect_Request_Config;
    L2CA_Link_Connect_Response_Config_t
            L2CA_Link_Connect_Response_Config;
} L2CA_Link_Connect_Params_t;
```

The values that are provided in this structure can be adjusted to change the way that L2CAP handles the requesting of lower link connections (HCI ACL) and how L2CAP handles the acceptance of lower link connections (HCI ACL). Changing these values allows L2CAP to function in a Point to Multi-Point environment. The possible values for the L2CA_Link_Connect_Request_Config parameter are as follows:

cqNoRoleSwitch
cqAllowRoleSwitch

The default value is cqNoRoleSwitch which instructs L2CAP to NOT allow a Role Switch to happen during an HCI ACL connection (when L2CAP originates the connection). The cqAllowRoleSwitch value would signal L2CAP to allow Role Switching when a HCI Connection is established (again, only when L2CAP originates the connection).

The possible values for the L2CA_Link_Connect_Response_Config parameter are as follows:

csMaintainCurrentRole
csRequestRoleSwitch
csIgnoreConnectionRequest

The default value is csMaintainCurrentRole which instructs L2CAP to NOT try to change the current Role when accepting a HCI ACL Connection. The csRequestRoleSwitch value instructs L2CAP to attempt to switch Roles whenever L2CAP accepts an HCI ACL Connection. The csIgnoreConnectionRequest value instructs L2CAP to NEVER accept ANY HCI Connections (or reject them). This functionality would be used if there was another entity handling the physical setup up HCI ACL Connections (i.e. not L2CAP). It is envisioned that the csIgnoreConnectionRequest value will rarely be used, however it exists for applications that do not want L2CAP to handle incoming HCI ACL Connection Requests.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Set_Link_Connection_Configuration

Set Lower Link Connection request/response configuration. This function exists to allow the programmer a method of controlling how L2CAP handles HCI ACL connection requests (both when L2CAP originates the HCI ACL connection and when L2CAP responds to remote HCI ACL requests). This functionality is provided to allow programmers a means to control L2CAP in a Point to Multi-Point environment. The default handling is that L2CAP doesn't allow a Role Switch at connection setup. This function allows the programmer to change this functionality if desired.

Prototype:

int BTPSAPI **L2CA_Set_Link_Connection_Configuration**(unsigned int BluetoothStackID, L2CA_Link_Connect_Params_t *L2CA_Link_Connect_Params)

Parameters:

BluetoothStackID[1] Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

L2CA_Link_Connect_Params Stack connection configuration values. This is the structure defined as:

```
typedef struct
{
    L2CA_Link_Connect_Request_Config_t
                L2CA_Link_Connect_Request_Config;
    L2CA_Link_Connect_Response_Config_t
                L2CA_Link_Connect_Response_Config;
} L2CA_Link_Connect_Params_t;
```

The values that are provided in this structure can be adjusted to change the way that L2CAP handles the requesting of lower link connections (HCI ACL) and how L2CAP handles the acceptance of lower link connections (HCI ACL). Changing these values allows L2CAP to function in a Point to Multi-Point environment. The possible values for the L2CA_Link_Connect_Request_Config parameter are as follows:

cqNoRoleSwitch
cqAllowRoleSwitch

The default value is cqNoRoleSwitch which instructs L2CAP to NOT allow a Role Switch to happen during an HCI ACL connection (when L2CAP originates the connection). The cqAllowRoleSwitch value would signal L2CAP to allow Role Switching when a HCI Connection is established (again, only when L2CAP originates the connection).

The possible values for the L2CA_Link_Connect_Response_Config parameter are as follows:

csMaintainCurrentRole
csRequestRoleSwitch
csIgnoreConnectionRequest

The default value is csMaintainCurrentRole which instructs L2CAP to NOT try to change the current Role when accepting a HCI ACL Connection. The csRequestRoleSwitch value instructs L2CAP to attempt to switch Roles whenever L2CAP accepts an HCI ACL Connection. The csIgnoreConnectionRequest value instructs L2CAP to NEVER accept ANY HCI Connections (or reject them). This functionality would be used if there was another entity handling the physical setup up HCI ACL Connections (i.e. not L2CAP). It is envisioned that the csIgnoreConnectionRequest value will rarely be used, however it exists for applications that do not want L2CAP to handle incoming HCI ACL Connection Requests.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Get_Link_Connection_State

The following function allows the caller to ascertain the current connection link state as it pertains to a specific Bluetooth device. This information is useful to know if L2CAP is currently utilizing the current link (either an incoming or outgoing connection).

Prototype:

int BTPSAPI **L2CA_Get_Link_Connection_State**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR,
    L2CA_Link_Connection_State_t *L2CA_Link_Connection_State)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address of the Bluetooth link to query the current state.
L2CA_Link_Connection_State	Link connection state. This is one of the following values:

lcsDisconnected

lcsConnecting
lcsConnected
lcsDisconnecting

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Get_Channel_Queue_Threshold

This function retrieves the current L2CAP Channel Queing Threshold information for the Bluetooth Stack L2CAP Module. This is used by the L2CAP module to limit the amount of data that the L2CAP Module will buffer, per L2CAP channel, internally. This will help alleviate the case where L2CAP always accepts data to be written when memory is available, which can lead to complete memory allocation usage (in the future). Note, only packets larger than SizeThreshold will be used to count towards DepthThreshold.

Prototype:

int BTPSAPI **L2CA_Get_Channel_Queue_Threshold**(unsigned int BluetoothStackID,
L2CA_Channel_Queue_Threshold_t *L2CA_Channel_Queue_Threshold)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
L2CA_Channel_Queue_Threshold	The retrieved Channel Queue Threshold. The SizeThreshold is the minimum size in bytes of an individual L2CAP ACL Segment. The DepthThreshold is the number of packets of SizeThreshold that are allowed. A DepthThreshold of zero means that this functionality is disabled. The LowQueueThreshold parameter specifies the lower threshold of the number of packets in the queue that must be met before a Channel empty indication event is dispatched.

```
typedef struct
{
   unsigned int SizeThreshold;
   unsigned int DepthThreshold;
   unsigned int LowQueueThreshold;
} L2CA_Channel_Queue_Threshold_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Set_Channel_Queue_Threshold

This function changes the current L2CAP Channel Queing Threshold information for the Bluetooth Stack L2CAP Module. This is used by the L2CAP module to limit the amount of data that the L2CAP Module will buffer, per L2CAP channel, internally. This will help alleviate the case where L2CAP always accepts data to be written when memory is available, which can lead to complete memory allocation usage (in the future). Note, only packets larger than SizeThreshold will be used to count towards DepthThreshold.

Prototype:

int BTPSAPI **L2CA_Set_Channel_Queue_Threshold**(unsigned int BluetoothStackID,
    L2CA_Channel_Queue_Threshold_t *L2CA_Channel_Queue_Threshold)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
L2CA_Channel_Queue_Threshold	The L2CAP Channel Queing Threshold to change to. The SizeThreshold is the minimum size in bytes of an individual L2CAP ACL Segment. The DepthThreshold is the number of packets of SizeThreshold that are allowed. A DepthThreshold of zero means that this functionality is disabled. The LowQueueThreshold parameter specifies the lower threshold of the number of packets in the queue that must be met before a Channel empty indication event is dispatched.

```
typedef struct
{
   unsigned int SizeThreshold;
   unsigned int DepthThreshold;
   unsigned int LowQueueThreshold;
} L2CA_Channel_Queue_Threshold_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.3.2 L2CAP Event Functions/Prototype

The first four functions are used to register and unregister event callbacks. The fifth function is a Prototype for an event callback function.

L2CA_Register_PSM

This function is used to register an L2CAP callback function with the L2CAP layer associated with the specified Bluetooth stack ID. The callback is used to handle incoming L2CAP events destined for the specified PSM Number. This function returns a non-zero, positive return value, which represents the L2CAP PSM callback ID, if successful. A negative return value is returned if the function is unsuccessful. The caller can use the return value from this function as the L2CAP_PSMID parameter for the L2CA_Un_Register_PSM function, when the caller wants to Unregister the callback.

Prototype:

int BTPSAPI **L2CA_Register_PSM**(unsigned int BluetoothStackID, Word_t PSM,
    L2CA_Event_Callback_t L2CA_Event_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth protocol stack via a call to BSC_Initialize.
PSM	Protocol/Service Multiplexer value to which this callback is to be registered.
L2CA_EventCallback	Function pointer to be used by the L2CAP layer to notify higher layers of L2CAP events.

CallbackParameter	User defined value to be supplied as an input parameter for all event callbacks.

Return:

Positive if function was successful. A positive return value represents a L2CAP_PSMID that uniquely identifies the callback. This value is used in the L2CA_Un_Register_PSM function.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_UNABLE_TO_REGISTER_PSM
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etConnect_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Un_Register_PSM

This function is used to un-register an L2CAP callback function with the L2CAP layer associated with the specified Bluetooth stack ID. This function returns a value of zero if successful. A negative return value is indicates the function was unsuccessful.

Prototype:

int BTPSAPI **L2CA_Un_Register_PSM**(unsigned int BluetoothStackID,
    unsigned int L2CAP_PSMID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth protocol stack via a call to BSC_Initialize.
L2CAP_PSMID	PSMID value that uniquely identifies the callback function for a PSM value. The L2CAP_PSMID supplied is the return value of a successful call to the L2CA_Register_PSM function.

Return:

Zero (0) if function was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_UNABLE_TO_UNREGISTER_PSM
BTPS_ERROR_PSM_NOT_REGISTERED

BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Register_Fixed_Channel

This function is used to register an L2CAP callback function with the L2CAP layer associated with the specified Bluetooth stack ID. This callback is used to handle incoming L2CAP events destined for the specified fixed channel. This function returns a non-zero, positive return value, which represents the L2CAP fixed channel ID, if successful. A negative return value is returned if the function is unsuccessful. The caller can use the return value from this function as the FCID parameter for the L2CA_Un_Register_Fixed_Channel and the L2CA_Fixed_Channel_Data_Write functions, when the caller wants to un-register the callback or send fixed channel data (respectively).

Prototype:

int BTPSAPI **L2CA_Register_Fixed_Channel**(unsigned int BluetoothStackID,
Word_t FixedChannel, void *ChannelParameters,
L2CA_Event_Callback_t L2CA_Event_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth protocol stack via a call to BSC_Initialize.
FixedChannel	L2CAP fixed channel number to register. This value must be greater than: L2CAP_CHANNEL_IDENTIFIER_CONNECTIONLESS_CHANNEL and less than: L2CAP_CHANNEL_IDENTIFIER_MINIMUM_CHANNEL_IDENTIFIER
ChannelParameters	Pointer to channel specific parameter information. Currently this value is not used and should be passed as NULL.
L2CA_EventCallback	Function pointer to be used by the L2CAP layer to notify higher layers of L2CAP events.
CallbackParameter	User defined value to be supplied as an input parameter for all event callbacks.

Return:

Positive if function was successful. A positive return value represents a FCID that uniquely identifies the callback. This value is used in the L2CA_Un_Register_Fixed_Channel function.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_UNABLE_TO_REGISTER_EVENT_CALLBACK
BTPS_ERROR_ADDING_CALLBACK_INFORMATION
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_FEATURE_NOT_AVAILABLE

Possible Events:

etFixed_Channel_Connect_Indication
etConnection_Parameter_Update_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Un_Register_Fixed_Channel

This function is used to un-register an L2CAP callback function with the L2CAP Layer associated with the specified Bluetooth stack ID. This function returns a value of zero if successful. A negative return value is indicates the function was unsuccessful.

Prototype:

int BTPSAPI **L2CA_Un_Register_Fixed_Channel**(unsigned int BluetoothStackID,
    Word_t FCID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FCID	Fixed channel ID value that uniquely identifies the callback function for a fixed channel. The FCID supplied is the return value of a successful call to the L2CA_Register_Fixed_Channel function.

Return:

Zero (0) if function was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_L2CAP_NOT_INITIALIZED
BTPS_ERROR_NO_CALLBACK_REGISTERED

BTPS_ERROR_PSM_NOT_REGISTERED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

L2CA_Event_Callback_t

The callback function provides the L2CAP layer a means to inform the user about L2CAP related events that occur. The event information is passed to the user in an L2CA_Event_Data_t structure. This structure contains all the information about the event that occurred.

Prototype:

void (BTPSAPI ***L2CA_Event_Callback_t**)(unsigned int BluetoothStackID,
    L2CA_Event_Data_t *L2CA_Event_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack on which the event occurred.
L2CA_Event_Data	Pointer to a structure that contains information about the event that has occurred. This structure is of the form:

```
typedef struct
{
  L2CA_Event_Type_t     L2CA_Event_Type;
  Word_t                Event_Data_Length;
  union
  {
    L2CA_Connect_Indication_t
              *L2CA_Connect_Indication;
    L2CA_Connect_Confirmation_t
              *L2CA_Connect_Confirmation;
    L2CA_Config_Indication_t
              *L2CA_Config_Indication;
    L2CA_Config_Confirmation_t
              *L2CA_Config_Confirmation;
    L2CA_Disconnect_Indication_t
              *L2CA_Disconnect_Indication;
    L2CA_Disconnect_Confirmation_t
              *L2CA_Disconnect_Confirmation;
    L2CA_Echo_Confirmation_t
              *L2CA_Echo_Confirmation;
    L2CA_Information_Confirmation_t
              *L2CA_Information_Confirmation;
```

```
      L2CA_Timeout_Indication_t
            *L2CA_Timeout_Indication;
      L2CA_Data_Indication_t
            *L2CA_Data_Indication;
      L2CA_Data_Error_Indication_t
            *L2CA_Data_Error_Indication;
      L2CA_Group_Data_Indication_t
            *L2CA_Group_Data_Indication;
      L2CA_Group_Member_Status_t
            *L2CA_Group_Member_Status;
      L2CA_Channel_Buffer_Empty_Indication_t
            *L2CA_Channel_Buffer_Empty_Indication;
      L2CA_Connection_Parameter_Update_Indication_t
            *L2CA_Connection_Parameter_Update_Indication;
      L2CA_Connection_Parameter_Update_Confirmation_t
            *L2CA_Connection_Parameter_Update_Confirmation;
      L2CA_Fixed_Channel_Connect_Indication_t
            *L2CA_Fixed_Channel_Connect_Indication;
      L2CA_Fixed_Channel_Disconnect_Indication_t
            *L2CA_Fixed_Channel_Disconnect_Indication;
      L2CA_Fixed_Channel_Data_Indication_t
            *L2CA_Fixed_Channel_Data_Indication;
   } Event_Data;
} L2CA_Event_Data_t;
```

where, L2CA_Event_Type_t is an enumerated type with the values listed in the table in section 2.3.3.

CallbackParameter	User defined value to was supplied as an input parameter from a prior L2CAP request.

2.3.3 L2CAP Events

The events that can be generated by the L2CAP portion of the Bluetooth Stack are listed in the table below and are described in the text that follows.

Event	Description
etConnect_Indication	Notify the host of a connection request from a remote device.
etConnect_Confirmation	Notify the host that a connection request has completed or is pending.
etConfig_Indication	Notify the host of a configuration request from a remote device.
etConfig_Confirmation	Notify the host that the configuration request has completed.
etDisconnect_Indication	Notify the host of a disconnection request from a remote device.
etDisconnect_Confirmation	Notify the host that the disconnection request has completed.
etEcho_Confirmation	Notify the host that an L2CA Ping request has completed.
etInformation_Confirmation	Return the requested device information to the Host.

etTimeout_Indication	Notify the host that a response from a remote device has timed out..
etData_Indication	Notify the host of incoming L2CAP data.
etData_Error_Indication	Notify the host of incoming L2CAP data error.
etGroup_Data_Indication	Notify the host of incoming connectionless data.
etGroup_Member_Status	Notify the host that a member has been added to a group.
etChannel_Buffer_Empty_ Indication	Notify the host that all buffered data has been sent to the device.
etConnection_Parameter_ Update_Indication	Notify the host of a received connection parameter update request (fixed channel LE only).
etConnection_Parameter_ Update_Confirmation	Notify the host of a received connection parameter update confirmation (fixed channel LE only).
etFixed_Channel_Connect_ Indication	Notify the host that a fixed channel is now connected to a specific device (fixed channel only).
etFixed_Channel_ Disconnect_Indication	Notify the host that a fixed channel is now disconnected from a specific device (fixed channel only).
etFixed_Channel_Data_ Indication	Notify the host that data has been received on a fixed channel (fixed channel only).
etFixed_Channel_Buffer_E mpty_Indication	Notify the host that all buffered data has been sent to the device for the specified fixed channel.

etConnect_Indication

Notify the host of a connection request from a remote device.

Return Structure:

```
typedef struct
{
  Word_t                          PSM;
  Word_t                          LCID;
  Byte_t                          Identifier;
  BD_ADDR_t                       BD_ADDR;
  L2CAP_Extended_Feature_Mask_t   ExtendedFeatures;
} L2CA_Connect_Indication_t;
```

Event Parameters:

PSM	Protocol/Service Multiplexer value to which this callback is to be registered.
LCID	Local channel identifier.
Identifier	Requestor’s identifier used to match up responses
BD_ADDR	Address of the Bluetooth device requesting the connection.

ExtendedFeatures	The extended features of the device that is attempting to connect. Access should be made using the following bit masks: L2CAP_EXTENDED_FEATURE_FLOW_CONTROL_BIT_NUMBER L2CAP_EXTENDED_FEATURE_RETRANSMIT_MODE_BIT_NUMBER L2CAP_EXTENDED_FEATURE_BI_DIRECTIONAL_QOS_BIT_NUMBER L2CAP_EXTENDED_FEATURE_ENHANCED_RETRANSMISSION_MODE_BIT_NUMBER L2CAP_EXTENDED_FEATURE_STREAMING_MODE_BIT_NUMBER L2CAP_EXTENDED_FEATURE_FCS_OPTION_BIT_NUMBER L2CAP_EXTENDED_FEATURE_ENHANCED_FLOW_SPEC_BIT_NUMBER L2CAP_EXTENDED_FEATURE_FIXED_CHANNELS_BIT_NUMBER L2CAP_EXTENDED_FEATURE_EXTENDED_WINDOW_SIZE_BIT_NUMBER L2CAP_EXTENDED_FEATURE_UNICAST_DATA_RECEPTION_BIT_NUMBER L2CAP_EXTENDED_FEATURE_MASK_EXTENSION_BIT_NUMBER The structure definition: typedef struct { Byte_t Extended_Feature_Mask0; Byte_t Extended_Feature_Mask1; Byte_t Extended_Feature_Mask2; Byte_t Extended_Feature_Mask3; } **L2CAP_Extended_Feature_Mask_t**;

etConnect_Confirmation

Notify the host that a connection request has completed or is pending.

Return Structure:

```
typedef struct
{
  Word_t                          LCID;
  Word_t                          Result;
  Word_t                          Status;
   L2CAP_Extended_Feature_Mask_t  ExtendedFeatures;
} L2CA_Connect_Confirmation_t;
```

Event Parameters:

LCID	Local channel identifier.
Result	Result of the connection attempt. Possible values are:

	L2CAP_CONNECT_RESULT_CONNECTION_SUCCESSFUL L2CAP_CONNECT_RESULT_CONNECTION_PENDING L2CAP_CONNECT_RESULT_CONNECTION_REFUSED_PSM_NOT_REGISTERED L2CAP_CONNECT_RESULT_CONNECTION_REFUSED_SECURITY_RELATED L2CAP_CONNECT_RESULT_CONNECTION_TIMEOUT
Status	If the Result indicates connection Pending, then this field contains the reason for the hold up. Possible values are: L2CAP_CONNECT_STATUS_NO_FURTHER_INFORMATION L2CAP_CONNECT_STATUS_AUTHENTICATION_PENDING L2CAP_CONNECT_STATUS_AUTHORIZATION_PENDING
ExtendedFeatures	The extended features of the device whose connection is pending. Access should be made using the following bit masks: L2CAP_EXTENDED_FEATURE_FLOW_CONTROL_BIT_NUMBER L2CAP_EXTENDED_FEATURE_RETRANSMIT_MODE_BIT_NUMBER L2CAP_EXTENDED_FEATURE_BI_DIRECTIONAL_QOS_BIT_NUMBER L2CAP_EXTENDED_FEATURE_ENHANCED_RETRANSMISSION_MODE_BIT_NUMBER L2CAP_EXTENDED_FEATURE_STREAMING_MODE_BIT_NUMBER L2CAP_EXTENDED_FEATURE_FCS_OPTION_BIT_NUMBER L2CAP_EXTENDED_FEATURE_ENHANCED_FLOW_SPEC_BIT_NUMBER L2CAP_EXTENDED_FEATURE_FIXED_CHANNELS_BIT_NUMBER L2CAP_EXTENDED_FEATURE_EXTENDED_WINDOW_SIZE_BIT_NUMBER L2CAP_EXTENDED_FEATURE_UNICAST_DATA_RECEPTION_BIT_NUMBER L2CAP_EXTENDED_FEATURE_MASK_EXTENSION_BIT_NUMBER

The structure definition is:

```
typedef struct
{
   Byte_t Extended_Feature_Mask0;
   Byte_t Extended_Feature_Mask1;
   Byte_t Extended_Feature_Mask2;
   Byte_t Extended_Feature_Mask3;
} L2CAP_Extended_Feature_Mask_t;
```

etConfig_Indication

Notify the host of a configuration request from a remote device.

Return Structure:

```
typedef struct
{
  Word_t                        LCID;
  Word_t                        Option_Flags;
  Word_t                        OutMTU;
  Word_t                        InFlushTO;
  L2CA_Flow_Spec_t              InFlow;
  L2CA_Mode_Info_t              ModeInfo;
  Byte_t                        FCS_Option;
  L2CA_Extended_Flow_Spec_t     ExtendedFlowSpec;
  Word_t                        ExtendedWindowSize;
} L2CA_Config_Indication_t;
```

Event Parameters:

LCID	Local channel identifier.
Option_Flags	A bit list. Possible bit values are: L2CA_CONFIG_OPTION_FLAG_MTU L2CA_CONFIG_OPTION_FLAG_FLUSH_TIMEOUT L2CA_CONFIG_OPTION_FLAG_QOS L2CA_CONFIG_OPTION_FLAG_MODE_INFO L2CA_CONFIG_OPTION_FLAG_FCS_OPTION L2CA_CONFIG_OPTION_FLAG_EXTENDED_FLOW_SPEC L2CA_CONFIG_OPTION_FLAG_EXTENDED_WINDOW_ SIZE L2CA_CONFIG_OPTION_FLAG_CONTINUATION
OutMTU	Maximum transmission unit that the remote unit will send across this channel (maybe less or equal to the InMTU input parameter).
InFlushTO	Number of milliseconds before an L2CAP packet that cannot be acknowl-edged at the physical layer is dropped. This value is indicates the actual value that will be used for outgoing packets and may be less than or equal to the OutFlushTO parameter given as input.

InFlow	Quality of service parameters dealing with the traffic characteristics of the agreed-upon outgoing data flow. This structure is defined as follows:

```
typedef struct
{
  Byte_t       Flags;
  Byte_t       ServiceType;
  DWord_t      TokenRate;
  DWord_t      TokenBucketSize;
  DWord_t      PeakBandwidth;
  DWord_t      Latency;
  DWord_t      DelayVariation;
} L2CA_Flow_Spec_t;
```

ModeInfo	Specifies the requested operating mode of the L2CAP channel.
FCSOption	Specifies the requested operating FCS mode of the L2CAP channel.
ExtendedFlowSpec	Specifies the requested extended Flow Specification. This structure is defined as follows:

```
typedef
{
  Byte_t    Identifier;
  Byte_t    ServiceType;
  Word_t    MaxSDU;
  DWord_t SDUInterArrivalTime;
  DWord_t AccessLatency;
  DWord_t FlushTimeout;
} L2CA_Extended_Flow_Spec_t;
```

ExtendedWindowSize	Specifies the requested extended window size (ERTM modes).

etConfig_Confirmation

Notify the host that the configuration request has completed.

Return Structure:

```
typedef struct
{
 Word_t                       LCID;
 Word_t                       Result;
 Word_t                       Option_Flags;
 Word_t                       InMTU;
 Word_t                       OutFlushTO;
 L2CA_Flow_Spec_t             OutFlow;
 L2CA_Mode_Info_t             ModeInfo;
 Byte_t                       FCS_Option;
 L2CA_Extended_Flow_Spec_t    ExtendedFlowSpec;
 Word_t                       ExtendedWindowSize;
} L2CA_Config_Confirmation_t;
```

Event Parameters:

LCID	Local channel identifier.
Result	Outcome of the configuration operation. Possible values are: L2CAP_CONFIGURE_RESPONSE_RESULT_SUCCESS L2CAP_CONFIGURE_RESPONSE_RESULT_FAILURE_UNACCEPTABLE_PARAMETERS L2CAP_CONFIGURE_RESPONSE_RESULT_FAILURE_REJECTED_NO_REASON L2CAP_CONFIGURE_RESPONSE_RESULT_FAILURE_UNKNOWN_OPTIONS L2CAP_CONFIGURE_RESPONSE_RESULT_TIMEOUT
Option_Flags	A bit list. Possible bit values are: L2CA_CONFIG_OPTION_FLAG_MTU L2CA_CONFIG_OPTION_FLAG_FLUSH_TIMEOUT L2CA_CONFIG_OPTION_FLAG_QOS L2CA_CONFIG_OPTION_FLAG_MODE_INFO L2CA_CONFIG_OPTION_FLAG_FCS_OPTION L2CA_CONFIG_OPTION_FLAG_EXTENDED_FLOW_SPEC L2CA_CONFIG_OPTION_FLAG_EXTENDED_WINDOW_SIZE L2CA_CONFIG_OPTION_FLAG_CONTINUATION
InMTU	Maximum transmission unit that the remote unit will send across this channel (maybe less or equal to the InMTU input parameter).
OutFlushTO	Number of milliseconds before an L2CAP packet that cannot be acknowl-edged at the physical layer is dropped. This value is indicates the actual value that will be used for outgoing packets and may be less than or equal to the OutFlushTO parameter given as input.

OutFlow	Quality of service parameters dealing with the traffic characteristics of the agreed-upon outgoing data flow. This structure is defined as follows:

```
typedef struct
{
  Byte_t      Flags;
  Byte_t      ServiceType;
  DWord_t     TokenRate;
  DWord_t     TokenBucketSize;
  DWord_t     PeakBandwidth;
  DWord_t     Latency;
  DWord_t     DelayVariation;
} L2CA_Flow_Spec_t;
```

ModeInfo	Specifies the requested operating mode of the L2CAP channel.
FCSOption	Specifies the requested operating FCS mode of the L2CAP channel.
ExtendedFlowSpec	Specifies the requested extended Flow Specification. This structure is defined as follows:

```
typedef
{
  Byte_t    Identifier;
  Byte_t    ServiceType;
  Word_t    MaxSDU;
  DWord_t SDUInterArrivalTime;
  DWord_t AccessLatency;
  DWord_t FlushTimeout;
} L2CA_Extended_Flow_Spec_t;
```

ExtendedWindowSize	Specifies the requested extended window size (ERTM modes).

etDisconnect_Indication

Notify the host of a disconnection request from a remote device.

Return Structure:

```
typedef struct
{
  Word_t LCID;
} L2CA_Disconnect_Indication_t;
```

Event Parameters:

LCID	Local channel identifier.

etDisconnect_Confirmation

Notify the host that the disconnection request has completed.

Return Structure:

```
typedef struct
{
  Word_t     Result;
  Word_t     LCID;
} L2CA_Disconnect_Confirmation_t;
```

Event Parameters:

Result	Disconnection action result. Possible values are: L2CAP_DISCONNECT_RESPONSE_RESULT_SUCCESS L2CAP_DISCONNECT_RESPONSE_RESULT_TIMEOUT
LCID	Local channel identifier.

etTimeout_Indication

Notify the host that a response from a remote device has timed out. The handshake may be retried as determined by the Bluetooh implemenation.

Return Structure:

```
typedef struct
{
  Word_t     LCID;
} L2CA_Timeout_Indication_t;
```

Event Parameters:

LCID	Local channel identifier.

etEcho_Confirmation

Notify the host that an L2CA Ping request has completed.

Return Structure:

```
typedef struct
{
   BD_ADDR_t       BD_ADDR;
   Word_t          Result;
   Word_t          Echo_Data_Length;
   Byte_t          Variable_Data[1];
} L2CA_Echo_Confirmation_t;
```

Event Parameters:

BD_ADDR	Bluetooth address of the remote device that participated in the L2CAP Ping request.
Result	Outcome of the Ping operation. Possible values are: L2CAP_ECHO_REQUEST_RESULT_RESPONSE_RECEIVED L2CAP_ECHO_REQUEST_RESULT_RESPONSE_TIMEOUT
Echo_Data_Length	Number of bytes in the response, Variable_Data, array

Variable_Data	Echo response data.

etInformation_Confirmation

Return the requested device information to the Host.

Return Structure:

```
typedef struct
{
  BD_ADDR_t      BD_ADDR;
  Word_t         InfoType;
  Word_t         Result;
  Byte_t         Variable_Data[1];
} L2CA_Information_Confirmation_t;
```

Event Parameters:

BD_ADDR	Bluetooth device address whose device information if being returned.
InfoType	Type of information returned. Possible values are: L2CAP_INFORMATION_REQUEST_INFOTYPE_ CONNECTIONLESS_MTU L2CAP_INFORMATION_REQUEST_INFOTYPE_ EXTENDED_FEATURE_MASK
Result	Outcome of this operation. Possible values are: L2CAP_INFORMATION_RESPONSE_RESULT_SUCCESS L2CAP_INFORMATION_RESPONSE_RESULT_ NOT_SUPPORTED L2CAP_INFORMATION_RESPONSE_RESULT_ PDU_REJECTED L2CAP_INFORMATION_RESPONSE_RESULT_TIMEOUT
Variable_Data	Returned device information.

etData_Indication

Notify the host of incoming L2CAP data.

Return Structure:

```
typedef struct
{
  Word_t     CID;
  Word_t     Data_Length;
  Byte_t     Variable_Data[1];
} L2CA_Data_Indication_t;
```

Event Parameters:

Data_Length	Number of bytes read in, i.e., in Variable_Data.
CID	Channel identifier.

Variable_Data	Data read in.

etData_Error_Indication

Notify the host of incoming L2CAP data errors. The Data Error Event is issued when an inconsistency is detected in the reception of data on a channel that is configured for reliable operation.

Return Structure:

```
typedef struct
{
  Word_t       Result;
  Word_t       Status;
  Word_t       CID;
} L2CA_Data_Error_Indication_t;
```

Event Parameters:

Result	Outcome of this operation. Possible values are: L2CAP_DATA_READ_RESULT_SUCCESS L2CAP_DATA_READ_RESULT_ERROR
Status	If Result was an error, what the cause of the error was. Possible values are: L2CAP_DATA_READ_STATUS_MTU_EXCEEDED L2CAP_DATA_READ_STATUS_RECEIVE_TIMEOUT L2CAP_DATA_READ_STATUS_SIZE_ERROR
CID	Channel identifier.

etGroup_Data_Indication

Notify the host of incoming connectionless data.

Return Structure:

```
typedef struct
{
  Word_t       PSM;
  Word_t       Data_Length;
  Byte_t       Variable_Data[1];
} L2CA_Group_Data_Indication_t;
```

Event Parameters:

Data_Length	Number of bytes read in, i.e., in Variable_Data.
PSM	Protocol/Service Multiplexer value to which this callback is to be registered.
Variable_Data	Data read in.

etGroup_Member_Status

Notify the host that a member has been added to a group and notify the host of the connection status.

Return Structure:

```
typedef struct
{
  Word_t       PSM;
  Word_t       GroupCID;
  BD_ADDR_t BD_ADDR;
  Boolean_t    Connected;
} L2CA_Group_Member_Status_t;
```

Event Parameters:

PSM	Registered PSM associated with the group.
GroupCID	Channel identifier that uniquely identifies the group.
BD_ADDR	Address of the Bluetooth device.
Connected	Specifies whether or not the specified device is currently connected or not.

etChannel_Buffer_Empty_Indication

Notify the host that all buffered data has been sent to a remote device.

Return Structure:

```
typedef struct
{
 Word_t     CID;
} L2CA_Channel_Buffer_Empty_Indication_t;
```

Event Parameters:

CID	Channel identifier which has no longer had any data available for transmitting.

etConnection_Parameter_Update_Indication

Notify the host that a connection parameter update request indication has been received. This event is only dispatched to the following fixed channel:

L2CAP_CHANNEL_IDENTIFIER_LE_SIGNALLING_CHANNEL

Return Structure:

```
typedef
{
  Word_t      FCID;
  BD_ADDR_t BD_ADDR;
  Word_t      IntervalMin;
  Word_t      IntervalMax;
  Word_t      SlaveLatency;
  Word_t      TimeoutMultiplier;
} L2CA_Connection_Parameter_Update_Indication_t;
```

Event Parameters:

FCID	Fixed channel identifier which the connection parameter update indication request was received.
BD_ADDR	Bluetooth device address of the device that has requested the connection parameter update.
IntervalMin	Minimum value for the the connection interval. This should fall within the range: HCI_LE_CONNECTION_INTERVAL_MINIMUM HCI_LE_CONNECTION_INTERVAL_MAXIMUM
IntervalMax	This should be greater than or equal to Conn_Interval_Min and shall fall within the range: HCI_LE_CONNECTION_INTERVAL_MINIMUM HCI_LE_CONNECTION_INTERVAL_MAXIMUM Both intervals follow the rule: Time = N * 1.25 msec
SlaveLatency	Slave latency for connection. This should be in range: HCI_LE_CONNECTION_LATENCY_MINIMUM HCI_LE_CONNECTION_LATENCY_MAXIMUM
TimeoutMultiplier	Supervision timeout multiplier for LE link. This should be in range: HCI_LE_SUPERVISION_TIMEOUT_MINIMUM HCI_LE_SUPERVISION_TIMEOUT_MAXIMUM The Supervision_Timeout follows the rule: Time = N * 10 msec

etConnection_Parameter_Update_Confirmation

Notify the host that a connection parameter update response (confirmation) has been received. This event is only dispatched to the following fixed channel:

L2CAP_CHANNEL_IDENTIFIER_LE_SIGNALLING_CHANNEL

Return Structure:

```
typedef struct
{
  Word_t      FCID;
  BD_ADDR_t BD_ADDR;
  Word_t      Result;
} L2CA_Connection_Parameter_Update_Confirmation_t;
```

Event Parameters:

FCID	Fixed channel identifier which the connection parameter update indication response (confirmation) was received.
BD_ADDR	Bluetooth device address of the device that has responsed to the connection parameter update request.
Result	Result of the connection parameter update request. This will be one of the following values: L2CAP_CONNECTION_PARAMETER_UPDATE_ RESPONSE_RESULT_ACCEPTED L2CAP_CONNECTION_PARAMETER_UPDATE_ RESPONSE_RESULT_REJECTED

etFixed_Channel_Connect_Indication

Notify the host that a fixed channel connection from a remote device has occurred. This event is only dispatched to the callback that registered for a specific fixed channel.

Return Structure:

```
typedef
{
  Word_t                    FCID;
  BD_ADDR_t                 BD_ADDR;
  L2CA_Controller_Type_t ControllerType;
} L2CA_Fixed_Channel_Connect_Indication_t;
```

Event Parameters:

FCID	Fixed channel identifier which the connection event was received.
BD_ADDR	Bluetooth device address of the device that has connected to the local device on the corresponding fixed channel.
ControllerType	Value that specifies the controller type of the fixed channel connection. This will be one of the following: ctBR_EDR ctLE

etFixed_Channel_Disconnect_Indication

Notify the host that a fixed channel disconnect from a remote device has occurred. This event is only dispatched to the callback that registered for a specific fixed channel.

Return Structure:

```
typedef
{
  Word_t      FCID;
  BD_ADDR_t   BD_ADDR;
} L2CA_Fixed_Channel_Disconnect_Indication_t;
```

Event Parameters:

FCID	Fixed channel identifier which the connection event was received.
BD_ADDR	Bluetooth device address of the device that has disconnected from the local device on the corresponding fixed channel.

etFixed_Channel_Data_Indication

Notify the host of incoming fixed channel L2CAP data. This event is only dispatched to the callback that registered for a specific fixed channel.

Return Structure:

```
typedef struct
{
  Word_t      FCID;
  BD_ADDR_t   BD_ADDR;
  Word_t      Data_Length;
  Byte_t      Variable_Data[1];
} L2CA_Fixed_Channel_Data_Indication_t;
```

Event Parameters:

FCID	Fixed channel identifier which the data was received.
BD_ADDR	Bluetooth device address of the device that has sent the data to the local device on the corresponding fixed channel.
Data_Length	Number of bytes read in, i.e., in Variable_Data.
Variable_Data	Data read in.

etFixed_Channel_Buffer_Empty_Indication

Event that is dispatched to notify the host that all buffered data has been sent to a remote device for the specified fixed channel connection.

Return Structure:

```
typedef struct
{
  Word_t      FCID;
  BD_ADDR_t  BD_ADDR;
} L2CA_Fixed_Channel_Buffer_Empty_Indication_t;
```

Event Parameters:

FCID	Fixed channel identifier which the data was received.
BD_ADDR	Bluetooth device address of the device that has sent the data to the local device on the corresponding fixed channel.

2.4 SDP API

The Service Discovery Protocol (SDP) provides a means for finding services available from or through a Bluetooth device. Commonly used data types are listed in section 2.4.1. Section 2.4.2 describes the SDP response callback prototype. Section 2.4.3 lists the SDP function calls. The actual prototypes and constants outlined in this section can be found in the **SDPAPI.H** header file in the Bluetopia distribution.

2.4.1 Commonly Used SDP Data Types

The following data types and structures are commonly used in the SDP functions. The list of data types covered in this section are listed in the table below.

Data Type	Description
SDP_Data_Element_Type_t	Enumeration of all data types used with the SDP API.
SDP_UUID_Entry_t	Structure to hold a Universally Unique ID information.
SDP_Attribute_ID_List_Entry_t	Structure to hold the Attribute ID information.
SDP_Data_Element_t	Structure to hold an individual SDP data element (any type).
SDP_Response_Data_Type_t	Enumeration of all SDP request response data types.
SDP_Error_Response_Data_t	Structure to hold error response information returned from a remote SDP server when a invalid request occurs.

SDP_Data_Element_Type_t

Enumeration of all data types used with the SDP API.

Enumeration:

```
typedef enum
{
   deNIL,
   deNULL,
   deUnsignedInteger1Byte,
   deUnsignedInteger2Bytes,
   deUnsignedInteger4Bytes,
   deUnsignedInteger8Bytes,
   deUnsignedInteger16Bytes,
   deSignedInteger1Byte,
   deSignedInteger2Bytes,
   deSignedInteger4Bytes,
   deSignedInteger8Bytes,
   deSignedInteger16Bytes,
   deTextString,
   deBoolean,
   deURL,
   deUUID_16,
   deUUID_32,
   deUUID_128,
   deSequence,
   deAlternative
} SDP_Data_Element_Type_t;
```

SDP_UUID_Entry_t

Structure to hold a Universally Unique ID information. Since there are three possible sizes of UUID, the main structure is a union of the three optional size UUID structures

Structures:

```
typedef struct
{
   Byte_t     UUID_Byte0;
   Byte_t     UUID_Byte1;
} UUID_16_t;

typedef struct
{
   Byte_t     UUID_Byte0;
   Byte_t     UUID_Byte1;
   Byte_t     UUID_Byte2;
   Byte_t     UUID_Byte3;
} UUID_32_t;
```

```
typedef struct
{
  Byte_t     UUID_Byte0;
  Byte_t     UUID_Byte1;
  Byte_t     UUID_Byte2;
  Byte_t     UUID_Byte3;
  Byte_t     UUID_Byte4;
  Byte_t     UUID_Byte5;
  Byte_t     UUID_Byte6;
  Byte_t     UUID_Byte7;
  Byte_t     UUID_Byte8;
  Byte_t     UUID_Byte9;
  Byte_t     UUID_Byte10;
  Byte_t     UUID_Byte11;
  Byte_t     UUID_Byte12;
  Byte_t     UUID_Byte13;
  Byte_t     UUID_Byte14;
  Byte_t     UUID_Byte15;
} UUID_128_t;

typedef struct
{
  SDP_Data_Element_Type_t   SDP_Data_Element_Type;
  union
  {
    UUID_16_t      UUID_16;
    UUID_32_t      UUID_32;
    UUID_128_t     UUID_128;
  } UUID_Value;
} SDP_UUID_Entry_t;
```

SDP_Attribute_ID_List_Entry_t

Structure to hold the Attribute ID information.

Structure:

```
typedef struct
{
  Boolean_t     Attribute_Range;
  Word_t        Start_Attribute_ID;
  Word_t        End_Attribute_ID;
} SDP_Attribute_ID_List_Entry_t;
```

Fields:

Attribute_Range	Whether or not this Attribute is a range of IDs versus a single ID. If TRUE, than the range is specified by the Start_ and End_ fields. If FALSE, then only the Start_ field is valid and holds the Attribute ID.
Start_Attribute_ID	Either the only Attribute ID or the first Attribute ID, depending on the setting of the Attribute_Range field.

End_Attribute ID	The last Attribute ID, if Attribute_Range field is the FALSE.

SDP_Data_Element_t

Structure to hold an individual SDP data element (any type).

Structure:

```
typedef struct _tagSDP_Data_Element_t
{
  SDP_Data_Element_Type_t          SDP_Data_Element_Type;
  DWord_t                          SDP_Data_Element_Length;
  union
  {
    Byte_t                         UnsignedInteger1Byte;
    Word_t                         UnsignedInteger2Bytes;
    DWord_t                        UnsignedInteger4Bytes;
    Byte_t                         UnsignedInteger8Bytes[8];
    Byte_t                         UnsignedInteger16Bytes[16];
    SByte_t                        SignedInteger1Byte;
    SWord_t                        SignedInteger2Bytes;
    SDWord_t                       SignedInteger4Bytes;
    Byte_t                         SignedInteger8Bytes[8];
    Byte_t                         SignedInteger16Bytes[16];
    Byte_t                         Boolean;
    UUID_16_t                      UUID_16;
    UUID_32_t                      UUID_32;
    UUID_128_t                     UUID_128;
    Byte_t                         *TextString;
    Byte_t                         *URL;
    struct _tagSDP_Data_Element_t  *SDP_Data_Element_Sequence;
    struct _tagSDP_Data_Element_t  *SDP_Data_Element_Alternative;
  } SDP_Data_Element;
} SDP_Data_Element_t;
```

Fields:

SDP_Data_Element_Type	One of the enumerated types of data elements.
SDP_Data_Element_Length	Length in bytes of the data element.
SDP_Data_Element	The data element itself.

SDP_Response_Data_Type_t

Enumeration of all SDP request response data types.

Enumeration:

```
typedef enum
{
  rdTimeout,
  rdConnectionError,
  rdErrorResponse,
  rdServiceSearchResponse,
  rdServiceAttributeResponse,
  rdServiceSearchAttributeResponse,
  rdServiceAttributeResponse_Raw,
  rdServiceSearchAttributeResponse_Raw
} SDP_Response_Data_Type_t;
```

SDP_Error_Response_Data_t

Structure to hold error response information returned from a remote SDP server when a invalid request occurs.

Structure:

```
typedef struct
{
  Word_t    Error_Code;
  Word_t    Error_Info_Length;
  Byte_t    *Error_Info;
} SDP_Error_Response_Data_t;
```

Fields:

Error_Code	Type of error that occurred. Possible values are: SDP_ERROR_CODE_INVALID_UNSUPPORTED_SDP_VERSION SDP_ERROR_CODE_INVALID_SERVICE_RECORD_HANDLE SDP_ERROR_CODE_INVALID_REQUEST_SYNTAX SDP_ERROR_CODE_INVALID_PDU_SIZE SDP_ERROR_CODE_INVALID_CONTINUATION_STATE SDP_ERROR_CODE_INSUFFICIENT_RESOURCES
Error_Info_Length	Length in bytes of Error_Info.
Error_Info	Optional additional error information for some error codes.

2.4.2 SDP Event Callbacks

The SDP event callbacks available in the Bluetooth Protocol Stack API and the functions used to register and unregister them are listed in the table below and described in the text which follows.

Callback/Function	Description/Purpose
SDP_Service_Search_Request	Issues a Service Search request to a remote device.

SDP_Service_Attribute_Request	Issues a Service Attribute request to a remote device.
SDP_Service_Search_Attribute_Request	Issues a Service Search Attribute request to a remote device.
SDP_Service_Attribute_Request_Raw	Issues a Service Attribute request to a remote device whose response will not be parsed by the SDP layer.
SDP_Service_Search_Attribute_Request_Raw	Issues a Service Search Attribute request to a remote device whose response will not be parsed by the SDP layer.
SDP_Set_Server_Connection_Mode	Sets the SDP connection mode and allows for a callback to be registered to watch connection events.

The SDP Response Callback is not used as a permanent registered callback, but as a dynamic callback which is passed to the search functions and gets called when search results are available. The SDP Connection Callback, registered with SDP_Set_Server_Connection_Mode(), is registered until the same function is called to un-register it.

SDP_Response_Callback_t

This user-supplied function will be called whenever a SDP Request Response returns with the Bluetooth Protocol Stack that is specified with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the SDP Request ID that was assigned to the SDP Service Request, the SDP Response Data associated with the SDP Request Response that occurred, and the SDP Callback Parameter that was specified when this Callback was installed. The caller is free to use the contents of the SDP Request Response 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(s). 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).

Prototype:

void (BTPSAPI ***SDP_Response_Callback_t**)(unsigned int BluetoothStackID, unsigned int SDPRequestID, SDP_Response_Data_t *SDP_Response_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
SDPRequestID	Unique identifier associated with an outstanding Request.
SDP_Response_Data	Pointer to aSDP_Response_Data_t structure that contains the results from an SDP request. This structure is defined below.

CallbackParameter	User defined value received in the DSP request and dispatched with the SDP response. This can be used to uniquely identify a response when multiple requests are outstanding.

SDP Response Data Structures

The following structures define the SDP_Response_Data returned in the callback.

Structures:

```
typedef struct
{
  SDP_Response_Data_Type_t   SDP_Response_Data_Type;
  union
  {
    SDP_Error_Response_Data_t                      SDP_Error_Response_Data;
    SDP_Service_Search_Response_Data_t             SDP_Service_Service_Search_Response_Data;
    SDP_Service_Attribute_Response_Data_t          SDP_Service_Attribute_Response_Data;
    SDP_Service_Search_Attribute_Response_Data_t  SDP_Service_Search_Attribute_Response_Data;
    SDP_Raw_Attribute_Response_Data_t              SDP_Raw_Attribute_Response_Data;
  } SDP_Response_Data;
} SDP_Response_Data_t;
```

Where the response data types in the union are defined by the following structures:

```
typedef struct
{
  Word_t   Error_Code;
  Word_t   Error_Info_Length;
  Byte_t   *Error_Info;
} SDP_Error_Response_Data_t;

typedef struct
{
  Word_t   Total_Service_Record_Count;
  DWord_t *Service_Record_List;
} SDP_Service_Search_Response_Data_t;

typedef struct
{
  Word_t                 Attribute_ID;
  SDP_Data_Element_t     SDP_Data_Element;
} SDP_Service_Attribute_Value_Data_t;

typedef struct
{
  Word_t                                Number_Attribute_Values;
  SDP_Service_Attribute_Value_Data_t    *SDP_Service_Attribute_Value_Data;
} SDP_Service_Attribute_Response_Data_t;
```

```
typedef struct
{
  Word_t                                 Number_Service_Records;
  SDP_Service_Attribute_Value_Data_t     *SDP_Service_Attribute_Value_Data;
} SDP_Service_Search_Attribute_Response_Data_t;

typedef struct
{
  SDP_Response_Data_Type_t   SDP_Response_Data_Type;
  DWord_t                    Raw_Attribute_Data_Length;
  Byte_t                     *Raw_Attribute_Data;
} SDP_Raw_Attribute_Response_Data_t;
```

SDP_Connection_Event_Callback_t

The following declared type represents the Prototype Function for the SDP Connection event callback. This function will be called whenever a callback has been registered for SDP connection events that is associated with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the SDP Connection event data of the specified event, and the SDP Connection event callback parameter that was specified when this callback was installed. The caller is free to use the contents of the SDP Connection event 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 needs to be noted however, that if the same callback is installed more than once, then the callbacks will be called serially. Because of this, the processing in this function should be as efficient as possible. It should also 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 other SDP Connection events will not be processed while this function call is outstanding).

Note: This function MUST NOT Block and wait for events that can only be satisfied by receiving other SDP Connection events. A deadlock WILL occur because NO SDP Connection event callbacks will be issued while this function is currently outstanding.

Prototype:

void (BTPSAPI ***SDP_Connection_Event_Callback_t**)(unsigned int BluetoothStackID, SDP_Connection_Event_Data_t *SDP_Connection_Event_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SDP_Connection_Event_Data	Pointer to the passed event data. See definitions in the next section.

CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP Connection Event Structures

The following structures define the SDP_Connection_Event_Data returned in the callback.

Structures:

```
typedef struct
{
  SDP_Connection_Event_Type_t Event_Data_Type;
  Word_t                      Event_Data_Size;
  union
  {
    SDP_Connect_Request_Indication_Data_t  *SDP_Connect_Request_Indication_Data;
    SDP_Connect_Indication_Data_t          *SDP_Connect_Indication_Data;
    SDP_Disconnect_Indication_Data_t       *SDP_Disconnect_Indication_Data;
  } Event_Data;
} SDP_Connection_Event_Data_t;
```

Where the event data types in the union are defined by the following structures:

```
typedef struct
{
  BD_ADDR_t BD_ADDR;
} SDP_Connect_Request_Indication_Data_t;

typedef struct
{
  BD_ADDR_t BD_ADDR;
} SDP_Connect_Indication_Data_t;

typedef struct
{
  BD_ADDR_t BD_ADDR;
} SDP_Disconnect_Indication_Data_t;
```

2.4.3 SDP Functions

The function calls available in the SDP layer API are listed in the table below and are described in the text that follows.

Function	Description
SDP_Create_Service_Record	Add an SDP Service Record to the SDP database.
SDP_Update_Service_Record_Service_Class	Updates the Service Class UUID(s) of a Serivce Record
SDP_Delete_Service_Record	Delete an SDP Service Record from the SDP database.
SDP_Add_Attribute	Adds a Service Attribute to an SDP Service Record in the SDP database.
SDP_Add_Raw_Attribute	Adds a pre-parsed Service Attribute to an SDP Service Record in the SDP database.
SDP_Delete_Attribute	Delete a Service Attribute from an SDP Service Record in the SDP database.
SDP_Service_Search_Request	Make an SDP Service Search request.
SDP_Service_Attribute_Request	Make an SDP Service Attribute request.
SDP_Service_Attribute_Request_Raw	Makes an SDP Service Attribute request with the response being dispatched to the caller without being parsed.
SDP_Service_Search_Attribute_Request	Make a combined Service search and Attribute search request.
SDP_Service_Search_Attribute_Request_Raw	Make a combined Service search and Attribute search request with the response being dispatched to the caller without being parsed.
SDP_Cancel_Service_Request	Terminate the currently active search request.
SDP_Parse_Raw_Attribute_Response_Data	Parses the specified raw SDP attribute data into Bluetopia SDP API format.
SDP_Free_Parsed_Attribute_Response_Data	Frees parsed data that was parsed with SDP_Parse_Raw_Attribute_Response_Data().
SDP_Set_Disconnect_Mode	Instruct SDP Module on how to handle Disconnect requests.
SDP_Disconnect_Server	Instruct SDP Module to disconnect from remote SDP Server.
SDP_Get_Server_Connection_Mode	Allows a mechanism of querying the current connection mode of the SDP layer.
SDP_Set_Server_Connection_Mode	Allows a mechanism of setting the current connection mode of the SDP layer and registering a callback to watch SDP connection events.
SDP_Connect_Request_Response	Allows a mechanism of accepting a SDP

Function	Description
	connection request from a remote device.

SDP_Create_Service_Record

This function is responsible for adding an SDP Service Record to the SDP Database. The first parameter to this function is the Bluetooth Stack ID of the SDP Server to create the SDP Service Record on. The second parameter is the number of UUID Entries that are present in the third parameter array. The second parameter CANNOT be zero, and the third parameter must contain at least as many entries as specified by the second parameter. If this function is successful, this function will return a positive, non-zero, value which represents the SDP Server Record Handle of the Service Record that was created on the specified SDP Server.

Prototype:

long BTPSAPI **SDP_Create_Service_Record**(unsigned int BluetoothStackID,
unsigned int NumberServiceClassUUID, SDP_UUID_Entry_t SDP_UUID_Entry[]);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
NumberServiceClassUUID	Number of UUIDs that are present in the array of UUIDs
SDP_UUID_Entry[]	Array of UUIDs that represent the ServiceClassIDList attributes of the Service Record.

Return:

Positive non-Zero value if successful. This represents the SDP Server Record Handle of the Service Record that was created on the specified SDP Server.

Negative if an error occurred and the record was not added. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INTERNAL_ERROR
BTPS_ERROR_ADDING_SERVICE_ATTRIBUTE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Update_Service_Record_Service_Class

The following function is responsible for Updating the Service Class Attribute for the specified SDP Service Record. This function accepts as input the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Server resides on and the SDP Service Record Handle whose service class will be updated. The Service_Record_Handle parameter must have been obtained via a successful call to the SDP_Create_Service_Record() function. The third parameter specifies the number of UUIDs for the service class and the final parameter lists the UUIDs to be used. This function returns zero if the Service class was updated successfully, or a negative value if there was an error.

Prototype:

int BTPSAPI **SDP_Update_Service_Record_Service_Class** (unsigned int BluetoothStackID, DWord_t Service_Record_Handle, unsigned int NumberServiceClassUUID, SDP_UUID_Entry_t SDP_UUID_Entry[]);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Service_Record_Handle	Handle to the service record to be deleted. This value is obtained from a successful call to SDP_Create_Service_Record.
NumberServiceClassUUID	Number of UUIDs that are present in the array of UUIDs
SDP_UUID_Entry[]	Array of UUIDs that represent the ServiceClassIDList attributes of the Service Record.

Return:

Zero value if successful.

Negative if an error occurred and the record was not added. Possible values are:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DELETING_SERVICE_RECORD
BTPS_ERROR_EXPECTED_UUID_ENTRY
BTPS_ERROR_INTERNAL_ERROR
BTPS_ERROR_INSUFFICIENT_RESOURCES

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Delete_Service_Record

This function is responsible for deleting a SDP Service Record that was added with the SDP_Create_Service_Record function. This function accepts as input, the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Server resides on and the SDP Service Record Handle to delete from the specified SDP Server. The second parameter to this function is obtained via a successful call to the SDP_Create_Service_Record function. This function deletes the specified SDP Service Record and deletes ALL SDP Attributes that are associated with the specified Service Record.

Prototype:

int BTPSAPI **SDP_Delete_Service_Record**(unsigned int BluetoothStackID,
    DWord_t Service_Record_Handle);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Service_Record_Handle	Handle to the service record to be deleted. This value is obtained from a successful call to SDP_Create_Service_Record.

Return:

Zero (0) if the specified Service Record was deleted successfully

Negative return error code if the Service Record was NOT deleted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DELETING_SERVICE_RECORD

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Add_Attribute

This function is responsible for adding an SDP Service Attribute to the specified SDP Service Record. This function accepts as input the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Server resides on and the SDP Service Record Handle to Add the specified Attribute. The third parameter specifies the Attribute Value that is to be associated with the specified Attribute. This value must contain a valid entry.

Prototype:

int BTPSAPI **SDP_Add_Attribute**(unsigned int BluetoothStackID,
    DWord_t Service_Record_Handle, Word_t Attribute_ID,
    SDP_Data_Element_t *SDP_Data_Element);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Service_Record_Handle	Handle to the service record of the service to add the attribute to. This value is obtained from a successful call to SDP_Create_Service_Record.
Attribute_ID	Unique identifier that distinguishes this attribute from other service attributes.
SDP_Data_Element	Pointer to an SDP_Data_Element_t structure. This structure contains the Attribute information to be associated with the Attribute_ID.

Return:

Zero (0) if the specified Attribute was added successfully.

Negative return error code if the Attribute was NOT added. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Add_Raw_Attribute

This function is responsible for adding an SDP Service Attribute to the specified SDP Service Record. This function is identical to the **SDP_Add_Attribute**() with the exception that this function takes the Attribute Data for the attribute in pre-parsed format (that can be sent directly out over the air with no conversion).

Prototype:

int BTPSAPI **SDP_Add_Raw_Attribute**(unsigned int BluetoothStackID,
    DWord_t Service_Record_Handle,
    SDP_ConstantServiceAttributeEntry_t *AttributeEntry);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Service_Record_Handle	Handle to the service record of the service to add the attribute to. This value is obtained from a successful call to SDP_Create_Service_Record.

AttributeEntry	Pointer to a structure containing information on the attribute that is to be added to the specified service. This structure is defined as follows:

```
typedef struct
{
  Byte_t      Flags;
  Word_t      AttributeID;
  DWord_t     AttributeLength;
  DWord_t     NumberOfUUIDOffsets;
  Word_t     *UUIDOffsets;
  Byte_t     *AttributeData;
} SDP_ConstantServiceAttributeEntry_t;
```

Return:

Zero (0) if the specified Attribute was added successfully.

Negative return error code if the Attribute was NOT added. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Delete_Attribute

This function is responsible for deleting an SDP Service Attribute from the specified SDP Service Record. This function accepts as input the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Server resides on and the SDP Service Record Handle in which the specified Attribute exists. The third parameter specifies the Attribute ID to be removed.

Prototype:

int BTPSAPI **SDP_Delete_Attribute**(unsigned int BluetoothStackID,
    DWord_t Service_Record_Handle, Word_t Attribute_ID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Service_Record_Handle	Handle to the service record to be deleted. This value is obtained from a successful call to SDP_Create_Service_Record.
Attribute_ID	Unique identifier that distinguishes this attribute to be removed.

Return:

Zero (0) if the specified Attribute was deleted successfully.

Negative return error code if the Attribute was NOT deleted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DELETING_SERVICE_RECORD

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Service_Search_Request

This function is responsible for issuing an SDP Service Search Request to the specified BD_ADDR. This function will return the result of the Search Request in the SDP Response Callback that is specified in the calling of this function. This function accepts as input, the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Client resides on, the Bluetooth device address to remotely connect to (the Remote SDP Server will reside on this BD_ADDR), the Maximum Number of Service Records, the Number of Service UUID's that are to be searched for, the Service UUID's to actually search for, the SDP Response Callback Function, and the SDP Response Callback Function Callback Parameter.

Prototype:

int BTPSAPI **SDP_Service_Search_Request**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, Word_t MaximumServiceRecordCount,
    unsigned int NumberServiceUUID, SDP_UUID_Entry_t SDP_UUID_Entry[],
    SDP_Response_Callback_t SDP_Response_Callback,
    unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device where the SDP Server resides.
MaximumServiceRecordCount	Specifies the Maximum number of service records to be returned for this request.
NumberServiceUUID	Number of Service UUIDs that are contained in the array of Service UUIDs.

SDP_UUID_Entry	Pointer to an array of Service UUIDs that will serve as the Service Search Pattern. This parameter must point to an array that contains the number of entries specified by the NumberServiceUUID parameter.
SDP_Response_Callback	Callback function pointer of type SDP_Response_Callback_t to be used to dispatch the result of the Service Search.
CallbackParameter	User-defined value to be dispatched with the result of the request. This can be used to uniquely identify a response when multiple requests are outstanding.

Return:

Positive, non-Zero value if the specified Request was successfully submitted. This value is a Reference ID to the request. This value is specified in the SDP_Cancel_Service_Request function when the request is to be canceled.

Negative return error code if the Request was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_CONNECTION_INFORMATION
BTPS_ERROR_ATTEMPTING_CONNECTION_TO_DEVICE
BTPS_ERROR_EXPECTED_UUID_ENTRY

Possible Events:

rdTimeout

rdConnectionError

rdErrorResponse

rdServiceSearchResponse

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Service_Attribute_Request

This function is responsible for issuing an SDP Service Attribute Request to the specified BD_ADDR. This function will return the result of the Attribute Request in the SDP Response Callback that is specified in the calling of this function. This function accepts as input, the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Client resides on, the Bluetooth device address to remotely connect to (the Remote SDP Server will reside on this BD_ADDR), the Service Record Handle of the SDP Service Record to query, the Number of Entries in the Attribute List that are to be queried, the Attribute List to actually use in the Query, the SDP Response Callback Function, and the SDP Response Callback Function Callback Parameter.

Prototype:

```
int BTPSAPI SDP_Service_Attribute_Request(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, DWord_t ServiceRecordHandle,
    unsigned int NumberAttributeListElements,
    SDP_Attribute_ID_List_Entry_t AttributeIDList[],
    SDP_Response_Callback_t SDP_Response_Callback,
    unsigned long CallbackParameter);
```

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device where the SDP Server resides.
Service_Record_Handle	Handle of the remote service record to be searched. This value is either known in advance or is determined by looking at the SDP_Service_Search_Response data.
NumberAttributeListElements	Number of Attribute Elements that are contained in the array of Attribute Elements.
AttributeIDList	Array of Attribute Elements on which to search.
SDP_Response_Callback	Callback function pointer of type SDP_Response_Callback_t to be used to dispatch the result of the Service Search.
CallbackParameter	User-defined value to be dispatched with the result of the request. This can be used to uniquely identify a response when multiple requests are outstanding.

Return:

Positive, non-Zero value if the specified Request was successfully submitted. This value is a Reference ID to the request. This value is specified in the SDP_Cancel_Service_Request function when the request is to be canceled.

Negative return error code if the Request was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_CONNECTION_INFORMATION
BTPS_ERROR_ATTEMPTING_CONNECTION_TO_DEVICE
BTPS_ERROR_EXPECTED_UUID_ENTRY

Possible Events:

rdTimeout

rdConnectionError

rdErrorResponse

rdServiceAttributeResponse

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Service_Attribute_Request_Raw

This function is responsible for issuing an SDP Service Attribute Request to the specified BD_ADDR. This function will return the result of the Attribute Request in the SDP Response Callback that is specified in the calling of this function. This function accepts as input, the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Client resides on, the Bluetooth device address to remotely connect to (the Remote SDP Server will reside on this BD_ADDR), the Service Record Handle of the SDP Service Record to query, the Number of Entries in the Attribute List that are to be queried, the Attribute List to actually use in the Query, the SDP Response Callback Function, and the SDP Response Callback Function Callback Parameter.

Note:

This function is identical to the SDP_Service_Attribute_Request_API() function with the exception that a successful response will be dispatched in the rdServiceAttributeResponse_Raw event and the SDP response data will be un-parsed.

Prototype:

int BTPSAPI **SDP_Service_Attribute_Request**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, DWord_t ServiceRecordHandle,
    unsigned int NumberAttributeListElements,
    SDP_Attribute_ID_List_Entry_t AttributeIDList[],
    SDP_Response_Callback_t SDP_Response_Callback,
    unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device where the SDP Server resides.
Service_Record_Handle	Handle of the remote service record to be searched. This value is either known in advance or is determined by looking at the SDP_Service_Search_Response data.
NumberAttributeListElements	Number of Attribute Elements that are contained in the array of Attribute Elements.
AttributeIDList	Array of Attribute Elements on which to search.
SDP_Response_Callback	Callback function pointer of type SDP_Response_Callback_t to be used to dispatch the result of the Service Search.

CallbackParameter	User-defined value to be dispatched with the result of the request. This can be used to uniquely identify a response when multiple requests are outstanding.

Return:

Positive, non-Zero value if the specified Request was successfully submitted. This value is a Reference ID to the request. This value is specified in the SDP_Cancel_Service_Request function when the request is to be canceled.

Negative return error code if the Request was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_CONNECTION_INFORMATION
BTPS_ERROR_ATTEMPTING_CONNECTION_TO_DEVICE
BTPS_ERROR_EXPECTED_UUID_ENTRY

Possible Events:

rdTimeout

rdConnectionError

rdErrorResponse

rdServiceAttributeResponse_Raw

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Service_Search_Attribute_Request

This function is responsible for issuing an SDP Service Search Attribute Request to the specified BD_ADDR, i.e., a combined Service and Attribute search. This function will return the result of the Service Search Attribute Request in the SDP Response Callback that is specified in the calling of this function. This function accepts as input, the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Client resides on, the Bluetooth device address to remotely connect to (the Remote SDP Server will reside on this BD_ADDR), the Number of Service UUID's that are to be searched for, the Service UUID's to actually search for, the Number of Entries in the Attribute List that are to be queried, the Attribute List to actually use in the Query, the SDP Response Callback Function, and the SDP Response Callback Function Callback Parameter.

Prototype:

int BTPSAPI **SDP_Service_Search_Attribute_Request**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, unsigned int NumberServiceUUID,
SDP_UUID_Entry_t SDP_UUID_Entry[], unsigned int NumberAttributeListElements,
SDP_Attribute_ID_List_Entry AttributeIDList[],

```
SDP_Response_Callback_t SDP_Response_Callback,
unsigned long CallbackParameter);
```

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device where the SDP Server resides.
NumberServiceUUID	Number of Service UUIDs that area contained in an array of Service UUIDs.
SDP_UUID_Entry	Pointer to an array of Service UUIDs that will serve as the Service Search Pattern. This parameter must point to an array that contains the number of entries specified by the NumberServiceUUID parameter.
NumberAttributeListElements	Number of Attribute Elements that are contained in the array of Attribute Elements.
AttributeIDList	Array of Attribute Elements on which to search.
SDP_Response_Callback	Callback function pointer of type SDP_Response_Callback_t to be used to dispatch the result of the Service Search.
CallbackParameter	User-defined value to be dispatched with the result of the request. This can be used to uniquely identify a response when multiple requests are outstanding.

Return:

Positive, non-Zero value if the specified Request was successfully submitted. This value is a Reference ID to the request. This value is specified in the SDP_Cancel_Service_Request function when the request is to be canceled.

Negative return error code if the Request was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_CONNECTION_INFORMATION
BTPS_ERROR_ATTEMPTING_CONNECTION_TO_DEVICE
BTPS_ERROR_EXPECTED_UUID_ENTRY

Possible Events:

rdTimeout

rdConnectionError

rdErrorResponse

rdServiceSearchAttributeResponse

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Service_Search_Attribute_Request_Raw

This function is responsible for issuing an SDP Service Search Attribute Request to the specified BD_ADDR, i.e., a combined Service and Attribute search. This function will return the result of the Service Search Attribute Request in the SDP Response Callback that is specified in the calling of this function. This function accepts as input, the Bluetooth Stack ID of the Bluetooth Protocol Stack that the SDP Client resides on, the Bluetooth device address to remotely connect to (the Remote SDP Server will reside on this BD_ADDR), the Number of Service UUID's that are to be searched for, the Service UUID's to actually search for, the Number of Entries in the Attribute List that are to be queried, the Attribute List to actually use in the Query, the SDP Response Callback Function, and the SDP Response Callback Function Callback Parameter.

Note:

This function is identical to the SDP_Service_Search_Attribute_Request() function with the exception that a successful response will be dispatched in the rdServiceSearchAttributeResponse_Raw event and the SDP response data will be un-parsed.

Prototype:

int BTPSAPI **SDP_Service_Search_Attribute_Request**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, unsigned int NumberServiceUUID,
    SDP_UUID_Entry_t SDP_UUID_Entry[], unsigned int NumberAttributeListElements,
    SDP_Attribute_ID_List_Entry AttributeIDList[],
    SDP_Response_Callback_t SDP_Response_Callback,
    unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device where the SDP Server resides.
NumberServiceUUID	Number of Service UUIDs that area contained in an array of Service UUIDs.
SDP_UUID_Entry	Pointer to an array of Service UUIDs that will serve as the Service Search Pattern. This parameter must point to an array that contains the number of entries specified by the NumberServiceUUID parameter.
NumberAttributeListElements	Number of Attribute Elements that are contained in the array of Attribute Elements.

AttributeIDList	Array of Attribute Elements on which to search.
SDP_Response_Callback	Callback function pointer of type SDP_Response_Callback_t to be used to dispatch the result of the Service Search.
CallbackParameter	User-defined value to be dispatched with the result of the request. This can be used to uniquely identify a response when multiple requests are outstanding.

Return:

Positive, non-Zero value if the specified Request was successfully submitted. This value is a Reference ID to the request. This value is specified in the SDP_Cancel_Service_Request function when the request is to be canceled.

Negative return error code if the Request was not submitted. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_CONNECTION_INFORMATION
BTPS_ERROR_ATTEMPTING_CONNECTION_TO_DEVICE
BTPS_ERROR_EXPECTED_UUID_ENTRY

Possible Events:

rdTimeout

rdConnectionError

rdErrorResponse

rdServiceSearchAttributeResponse_Raw

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Cancel_Service_Request

This function is responsible for terminating a currently executing SDP Service Request. This function accepts as input the Bluetooth Protocol Stack ID of the Bluetooth Protocol Stack the SDP Service Request was issued on, and the SDP Service Request ID of the SDP Service Request that was issued. The SDP Service Request ID is obtained via a successful call to one of the following functions:

SDP_Service_Search_Request
SDP_Service_Attribute_Request
SDP_Service_Search_Attribute_Request

After this function is called, the callback that was installed for the specified SDP Service Request will **not** be called and the caller will **not** have access to the SDP Service Response Information for the SDP Service Request.

Prototype:

void BTPSAPI **SDP_Cancel_Service_Request**(unsigned int BluetoothStackID,
    unsigned int ServiceRequestID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
ServiceRequestID	Unique identifier associated with an outstanding Request.

Return:

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Parse_Raw_Attribute_Response_Data

This function is utility function that exists to parse the specified Raw SDP Attribute Response Data into the Bluetopia SDP API (Parsed) format.

Prototype:

int BTPSAPI **SDP_Parse_Raw_Attribute_Response_Data**(
    unsigned int BluetoothStackID,
    SDP_Raw_Attribute_Response_Data_t *RawAttributeResponseData,
    SDP_Parsed_Attribute_Response_Data_t *ParsedAttributeResponseData);

Parameters:

BluetoothStackID[1] — Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

RawAttributeResponseData — The raw SDP Attribute Response data to parse. This structure is defined as follows:

```
typedef
{
   SDP_Response_Data_Type_t  SDP_Response_Data_Type;
   DWord_t                   Raw_Attribute_Data_Length;
   Byte_t                   *Raw_Attribute_Data;
} SDP_Raw_Attribute_Response_Data_t;
```

Note that SDP_Response_Data_Type must be either of the following types:

rdServiceAttributeResponse_Raw
rdServiceSearchAttributeResponse_Raw

ParsedAttributeResponseData	Must contain a pointer to a structure that is to receive the parsed SDP Attribute Response information upon a successful return. This structure is defined as follows:

```
typedef struct
{
   SDP_Response_Data_t   SDP_Response_Data;
   void                  *RESERVED;
} SDP_Parsed_Attribute_Response_Data_t;
```

Note, this **MUST** be freed using the **SDP_Free_Parsed_Attribute_Response_Data**() API if this function returns success to prevent a memory leak.

Return:

Zero (0) if the specified Attribute was added successfully.

Negative return error code if the Attribute was NOT added. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Free_Parsed_Attribute_Response_Data

The following function is provided to allow a mechanism to free all resources that were allocated to parse Raw SDP Response Data into Bluetopia Parsed SDP Data.

Prototype:

void BTPSAPI **SDP_Free_Parsed_Attribute_Response_Data**(
    SDP_Parsed_Attribute_Response_Data_t *ParsedAttributeResponseData);

Parameters:

ParsedAttributeResponseData	Must contain a pointer to a structure that was passed to the successful call to **SDP_Parse_Raw_Attribute_Response_Data**(). This structure is defined as follows:

```
typedef struct
{
   SDP_Response_Data_t   SDP_Response_Data;
   void                  *RESERVED;
} SDP_Parsed_Attribute_Response_Data_t;
```

Return:

Possible Events:

Notes:

SDP_Set_Disconnect_Mode

This function is responsible for informing the SDP Module how it is to execute SDP Service Requests regarding the Connection Disconnection. This function accepts as input the Bluetooth Protocol Stack ID of the Bluetooth Protocol Stack for which the SDP Server resides and the SDP Connection Mode that is to be set. This function will return zero if the Connection Mode was successfully set, or a negative return error code if there was an error. Note, if the caller specifies SDP Disconnect Mode *dmManual* then the caller is responsible for disconnecting the the SDP Connection (to the remote server) by calling the SDP_Disconnect_Server( ) function. If the SDP Disconnect Mode *dmAutomatic* is chosen (default) then the Connection to the server is automatically terminated when the SDP Transaction completes. The SDP Connection Mode can **only** be changed when there are no Client SDP Transactions outstanding.

Prototype:

int BTPSAPI **SDP_Set_Disconnect_Mode**(unsigned int BluetoothStackID,
SDP_Disconnect_Mode_t SDPDisconnectMode)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SDPDisconnectMode	What type of mode should be set. The possible values are: dmAutomatic *{default mode}* dmManual

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Disconnect_Server

This function is responsible for disconnecting a Remote SDP Server connection that is still currently open. This function is used when the SDP Disconnect Mode is set to *dmManual* and an SDP Client Request has been issued. This function has no effect when used when the SDP Disconnect Mode is set to *dmAutomatic*. This function simply accepts the Bluetooth device address that has had an SDP Service Request issued. Upon completion of this function, there is no longer an L2CAP SDP Channel connection present between the local device and the Remote SDP Server.

Prototype:

int BTPSAPI **SDP_Disconnect_Server**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address of the Remote SDP Server for which the local device is currently connected

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Get_Server_Connection_Mode

This function provides a mechanism of querying the current server connection mode of the SDP layer.

Note:

The default server connection mode is sdAutomaticAccept.

Prototype:

int BTPSAPI **SDP_Get_Server_Connection_Mode**(unsigned int BluetoothStackID,
SDP_Server_Connection_Mode_t *SDPServerConnectionMode)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SDPServerConnectionMode	Pointer to return the current SDP server connection mode into. If this function returns success this will point to one of the following values: sdAutomaticAccept sdManualAccept

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Set_Server_Connection_Mode

This function provides a mechanism of setting the current server connection mode of the SDP layer.

Note:

The default server connection mode is sdAutomaticAccept.

If this function is called with the SDPServerConnectionMode set to sdManualAccept then the SDPConnectionEventCallback MUST be specified (i.e. cannot be NULL). Otherwise the SDPConnectionEventCallback is optional and may be NULL.

If this function is called with the SDPServerConnectionMode set to sdAutomaticAccept and the SDPConnectionEventCallback is specified (i.e. not NULL) then the caller can watch SDP connection events.

Prototype:

int BTPSAPI **SDP_Set_Server_Connection_Mode**(unsigned int BluetoothStackID,
SDP_Server_Connection_Mode_t SDPServerConnectionMode,
SDP_Connection_Event_Callback_t SDPConnectionEventCallback,
unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SDPServerConnectionMode	The server connection mode to set. This may be one of the following values: sdAutomaticAccept sdManualAccept
SDPConnectionEventCallback	A connection event callback function that can be specified to watch SDP connection events.
CallbackParameter	Callback parameter that will be passed to the connection event callback specified in the SDPConnectionEventCallback parameter whenever a SDP connection event occurs.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SDP_Connect_Request_Response

This function provides a mechanism of accepting an SDP connection request from a remote device. This function an only be used if the SDP server connection mode, set with the SDP_Set_Server_Connection_Mode(), is set to sdManualAccept.

Prototype:

int BTPSAPI **SDP_Connect_Request_Response**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	The BD_ADDR of the remote device whose SDP connection request is being accepted.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SDP_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.5 RFCOMM API

RFCOMM provides serial port emulation over top of the L2CAP protocol, which supports up to 60 simultaneous connections between two Bluetooth devices (or device-specific limits). RFCOMM emulates the nine circuits used in RS-232 serial communications utilizing a subset of the ETSI TS 07.10 standard (see applicable documents). The SPP (Serial Port Profile) is built on top of RFCOMM and for many users provides an easier to use interface. The RFCOMM commands are listed in section 2.5.1, the event callback prototype is described in section 2.5.2, and the RFCOMM events are itemized in section 2.5.3. The actual prototypes and constants outlined in this section can be found in the **RFCOMAPI.H** header file in the Bluetopia distribution.

2.5.1 RFCOMM Commands

The available RFCOMM command functions are listed in the table below and are described in the text which follows.

Function	Description
RFCOMM_Set_System_Parameters	Set up system-wide RFCOMM parameters.
RFCOMM_Get_System_Parameters	Retrieve system-wide RFCOMM parameters.
RFCOMM_Set_Data_Queuing_Parameters	Set system-wide RFCOMM data packet queuing parameters.
RFCOMM_Get_Data_Queuing_Parameters	Retrieve system-wide RFCOMM data packet queuing parameters.
RFCOMM_Register_Server_Channel	Register a server channel with RFCOMM.
RFCOMM_Un_Register_Server_Channel	Unregister an RFCOMM server channel.
RFCOMM_Open_Request	Instantiate an RFCOMM service channel with a remote RFCOMM server.

RFCOMM_Open_Response	Accept or reject an Open Request.
RFCOMM_Release_Request	Disconnect an RFCOMM channel.
RFCOMM_Send_Credits	Send flow control credits to an open RFCOMM channel.
RFCOMM_Send_Data	Send data on an open RFCOMM channel.
RFCOMM_Send_Data_With_Credits	Send flow control credits to an open RFCOMM channel in addition to the specified data (same RFCOMM packet).
RFCOMM_Parameter_Negotiation_Response	Send a response to a parameter negotiation response request.
RFCOMM_Test_Request	Send test data on an open RFCOMM channel
RFCOMM_Flow_Request	Control incoming data flow (i.e., turn on/off).
RFCOMM_Modem_Status	Send modem status information to remote RFCOMM entity.
RFCOMM_Line_Status_Change	Convey line status change information to the remote RFCOMM entity.
RFCOMM_Remote_Port_Negotiation_Request	Initiate a Remote Port Negotiation command.
RFCOMM_Remote_Port_Negotiation_Response	Respond to a Remote Port Negotiation request.
RFCOMM_Query_Remote_Port_Negotiation	Retrieve Remote RFCOMM entity's current Port Negotiation Parameters
RFCOMM_Get_Channel_Status	Retrieve current status of a specific Channel
RFCOMM_Query_Server_Channel_Present	Determine if there is a currently registered RFCOMM Server Channel for a specific Server Channel.

RFCOMM_Set_System_Parameters

This function is responsible for setting system-wide parameters. These parameters are used to control aspects of each Data Link Connection Identifier channel that is opened by the local or remote side. When a Server is registered, the current SystemParams are used as the parameters that are to be negotiated for that server connection.

Prototype:

int BTPSAPI **RFCOMM_Set_System_Parameters**(unsigned int BluetoothStackID,
    RFCOMM_System_Parameters_t *SystemParams)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

SystemParams The parameters to set. This is a structure defined as:

```
typedef struct
{
   Boolean_t              NegotiateParams;
   Word_t                 MaximumFrameSize;
   RFCOMM_Flow_Type_t     FlowType;
   Byte_t                 InitialCredits;
   Byte_t                 AcknowledgementTimer;
   Byte_t                 ResponseTimerForMultiplexer;
} RFCOMM_System_Parameters_t;
```

Where the MaximumFrameSize is expressed in bytes. Three defined constants which relate to frame size are:

RFCOMM_FRAME_SIZE_MINIMUM_VALUE
RFCOMM_FRAME_SIZE_MAXIMUM_VALUE
RFCOMM_FRAME_SIZE_DEFAULT_VALUE

AcknowledgementTimer is in seconds. Three defined constants which relate to it are:

RFCOMM_ACKNOWLEDGEMENT_TIMER_MINIMUM_VALUE
RFCOMM_ACKNOWLEDGEMENT_TIMER_MAXIMUM_VALUE
RFCOMM_ACKNOWLEDGEMENT_TIMER_DEFAULT_VALUE

ResponseTimerForMultiplexer also is in seconds. Three defined constants which relate to it are:

RFCOMM_RESPONSE_TIMER_MINIMUM_VALUE
RFCOMM_RESPONSE_TIMER_MAXIMUM_VALUE
RFCOMM_RESPONSE_TIMER_DEFAULT_VALUE

RFCOMM_Flow_Type_t is an enumeration with the following possible values:

ftCreditFlowNotAllowed,
ftCreditFlowPreferred,
ftCreditFlowMandatory,

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Get_System_Parameters

This function is used to retrieve system-wide parameters from a Bluetooth device. These parameters are used to control aspects of each Data Link Connection Identifier that are opened by the local or remote side.

Prototype:

int BTPSAPI **RFCOMM_Get_System_Parameters**(unsigned int BluetoothStackID,
    RFCOMM_System_Parameters_t *SystemParams)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SystemParams	The structure to return the parameters in. See the function RFCOMM_Set_System_Parameters for explanation of this structure.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Set_Data_Queuing_Parameters

This function is responsible for setting system-wide data queing parameters. These parameters are used to control the lower level data packet queing thresholds (to improve RAM usage). Specifically, these parameters are used to control aspects of the number of data packets (and only data packets) that can be queued into the lower level (per individual DLCI). This mechanism allows for the flexibility to limit the amount of RAM that is used for streaming type applications (where the remote side has a large number of credits that were granted).

<u>Notes:</u>

This function can only be called when there are NO active connections.

Setting both parameters to zero will disable the queuing mechanism. This means that the number of queued packets will only be limited via the amount of available RAM.

RFCOMM_Send_Credits() is not considered a data packet in terms of queuing. The only functions that count towards these values are:

- RFCOMM_Send_Data()
- RFCOMM_Send_Data_With_Credits()

Prototype:

int BTPSAPI **RFCOMM_Set_Data_Queuing_Parameters**(unsigned int BluetoothStackID, unsigned int MaximumNumberDataPackets, unsigned int QueuedDataPacketsThreshold)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
MaximumNumberDataPackets	The maximum number of data packets that can be queued into the lower layer simultaneously.
QueuedDataPacketsThreshold	The lower threshold limit that the lower layer should call back to inform RFCOMM that it can queue more data packets for transmission.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Get_Data_Queuing_Parameters

This function is responsible for querying the system-wide data queing parameters. These parameters are used to control the lower level data packet queing thresholds (to improve RAM usage). Specifically, these parameters are used to control aspects of the number of data packets (and only data packets) that can be queued into the lower level (per individual DLCI). This mechanism allows for the flexibility to limit the amount of RAM that is used for streaming type applications (where the remote side has a large number of credits that were granted).

Notes:

If both parameters are zero the the queuing mechanism is disabled. This means that the number of queued packets will only be limited via the amount of available RAM.

RFCOMM_Send_Credits() is not considered a data packet in terms of queuing. The only functions that count towards these values are:

- RFCOMM_Send_Data()
- RFCOMM_Send_Data_With_Credits()

Prototype:

int BTPSAPI **RFCOMM_Get_Data_Queuing_Parameters**(unsigned int BluetoothStackID,
unsigned int *MaximumNumberDataPackets,
unsigned int *QueuedDataPacketsThreshold)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
MaximumNumberDataPackets	Buffer that will contain the maximum number of data packets that can be queued into the lower layer simultaneously (if successful).
QueuedDataPacketsThreshold	Buffer that will contain the lower threshold limit that the lower layer should call back to inform RFCOMM that it can queue more data packets for transmission (if successful).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Register_Server_Channel

This function is used to register a server channel that the RFCOMM Layer is to providing services for. The channel is associated with the Bluetooth Protocol Stack, specified by the Bluetooth Stack ID, and a server program the run above the RFCOMM layer (e.g., the Serial Port Profile, SPP). After the channel is registered, all events that occur on the specified channel will be dispatched to the upper layer via the callback function provided.

Prototype:

int BTPSAPI **RFCOMM_Register_Server_Channel**(unsigned int BluetoothStackID, Byte_t ServerChannel, RFCOMM_Event_Callback_t RFCOMM_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
ServerChannel	The channel number that this server supports. This must be in the range of the following two constants: RFCOMM_MINIMUM_SERVER_CHANNEL_ID RFCOMM_MAXIMUM_SERVER_CHANNEL_ID
RFCOMM_Event_Callback	Function to call when events occur on this channel.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each event.

Return:

Positive, non-zero if successful. The return value will be the Server ID that can be passed to RFCOMM_Un_Register_Server_Channel to un-register the server.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_ADDING_SERVER_INFORMATION

Possible Events:

etOpen_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Un_Register_Server_Channel

This function is used to unregister a server channel that the RFCOMM Layer is providing services for. Upon completion of this function, all access to this RFCOMM channel will fail.

Prototype:

int BTPSAPI **RFCOMM_Un_Register_Server_Channel**(unsigned int BluetoothStackID, unsigned int ServerID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
ServerID	Server ID of the server that is to be un-registered.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Open_Request

This function is used to instantiate an RFCOMM service channel between the client application residing above this RFCOMM layer and a destination endpoint (server) that resides on the device associated with the Bluetooth BD_ADDR supplied. Only One L2CAP/ACL connection can exist between two RFCOMM entities, so this function will first check to see if an RFCOMM connection already exists between the two devices. If a connection already exists, then a new channel will be negotiated between the two devices over an existing L2CAP connection. If a connection does not exist, this function will initiate a L2CAP connection between the two devices on which the RFCOMM channel will be created in the future. If a connection was successfully initiated, the TEI (Terminal Endpoint Identifier) and DLCI (Data Link Connection Identifier) values are returned and must be supplied in future call to functions that are to operate on the connection.

Prototype:

int BTPSAPI **RFCOMM_Open_Request**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t Channel, Word_t *TEI, Byte_t *DLCI,
RFCOMM_Open_Parameters_t *OpenParams,
RFCOMM_Event_Callback_t RFCOMM_Event_Callback,
unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the Bluetooth device to establish the connection to.
Channel	Server channel to open on the remote device.

TEI	Returned Terminal Endpoint Identifier. Must be supplied on future calls for this channel.
DLCI	Returned Data Link Connection Identifier. This must be supplied on future calls for this channel.
OpenParams	Parameters to use in establishing the channel. These are passed in the following structure:

```
typedef struct
{
   Byte_t    OptionFlags;
   Word_t    MaximumFrameSize;
   Byte_t    InitialCredits;
} RFCOMM_Open_Parameters_t;
```

Where OptionFlags indicate whether either or both of the other two fields are defined for this channel. This is a bitmask which may have the follow bits:

RFCOMM_OPEN_PARAMS_OPTION_TYPE_MAX_FRAME_SIZE
RFCOMM_OPEN_PARAMS_OPTION_TYPE_INITIAL_CREDITS

InitialCredits is used for connections to channels with credit-based flow control capabilities.

RFCOMM_Event_Callback	Function to call when events occur on this channel.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each event.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_ADD_CONNECTION_I
NFORMATION
BTPS_ERROR_RFCOMM_UNABLE_TO_ADD_CHANNEL_
INFORMATION
BTPS_ERROR_RFCOMM_UNABLE_TO_CONNECT_TO_
REMOTE_DEVICE
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_MAX_FRAME_SIZE

Possible Events:

etOpen_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Open_Response

The following function is provided to allow a method for a server to accept or reject a connection request. When a connection is being established to a server, an etOpen_Indication is dispatched to the upper layer. The upper layer should examine the parameters that are being requested and supply an Accept or Reject for the connection via this function.

Prototype:

int BTPSAPI **RFCOMM_Open_Response**(unsigned int BluetoothStackID, Word_t TEI,
    Byte_t DLCI, Byte_t Accept)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier from etOpen_Indication event.
DLCI	Data Link Connection Identifier from etOpen_Indication event
Accept	Return TRUE or FALSE to indicate acceptance or rejection.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_CONNECT_TO_
REMOTE_DEVICE
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE

Possible Events:

etRelease_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Release_Request

This function is used to disconnect an RFCOMM channel that is currently open or in the process of being opened. This function takes as it parameters a Bluetooth Stack ID to identify the Bluetooth device that this command is associated with. The parameters TEI and Data Link Connection Identifier identify the channel that is to be disconnected.

Prototype:

int BTPSAPI **RFCOMM_Release_Request**(unsigned int BluetoothStackID, Word_t TEI, Byte_t DLCI)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel to release.
DLCI	Data Link Connection Identifier of channel to release.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Send_Credits

This function is used to send Credits to an RFCOMM channel that is currently open. This function takes as it parameters a Bluetooth Stack ID to identify the Bluetooth device that this command is associated with. The TEI and DLCI identify the channel to which the Credits are to be sent. The number of credits that are sent to the receiver will be added to the number of credits that are already available to the receiver. Note, this function is only available for those channels that have been configured to use credit-based flow control.

Prototype:

int BTPSAPI **RFCOMM_Send_Credits**(unsigned int BluetoothStackID, Word_t TEI, Byte_t DLCI, Byte_t Credits)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
Credits	Number of credits to issue to the receiver (cannot be zero).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

etCredit_Indication
etRelease_Indication
etTransport_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Send_Data

This function is used to send data to an RFCOMM channel that is currently open. The channel must be in a connected state and have the proper flow control requirements met before a successful data transfer can be expected.

Prototype:

int BTPSAPI **RFCOMM_Send_Data**(unsigned int BluetoothStackID, Word_t TEI,
Byte_t DLCI, Word_t Length, Byte_t *Data)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.

Length	Length of the data (cannot be zero).
Data	Data to send.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI
BTPS_ERROR_RFCOMM_CONTROL_MESSAGE_
CURRENTLY_PENDING
BTPS_ERROR_RFCOMM_FLOW_IS_DISABLED
BTPS_ERROR_RFCOMM_MAX_FRAME_SIZE_EXCEEDED
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

etDLCI_Data_Indication
etFlow_Indication
etFlow_Confirmation
etRelease_Indication
etTransport_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Send_Data_With_Credits

This function is used to send data to an RFCOMM channel that is currently open. The channel must be in a connected state and have the proper flow control requirements met before a successful data transfer can be expected. This function is also used to send Credits to the same RFCOMM channel. This function takes as it parameters a Bluetooth Stack ID to identify the Bluetooth device that this command is associated with. The TEI and DLCI identify the channel to which the Credits and data are to be sent. The number of credits that are sent to the receiver will be added to the number of credits that are already available to the receiver. This function also accepts data that will be sent on the channel (in the same RFCOMM packet). Note, this function is only available for those channels that have been configured to use credit-based flow control, and the credit parameter must be non-zero and this function must specify at least one byte of data to send.

Prototype:

int BTPSAPI **RFCOMM_Send_Data_With_Credits**(unsigned int BluetoothStackID,
Word_t TEI, Byte_t DLCI, Byte_t Credits, Word_t Length, Byte_t *Data)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
Credits	Number of credits to issue to the receiver (cannot be zero).
Length	Length of the data (cannot be zero).
Data	Data to send.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI
BTPS_ERROR_RFCOMM_CONTROL_MESSAGE_
CURRENTLY_PENDING
BTPS_ERROR_RFCOMM_FLOW_IS_DISABLED
BTPS_ERROR_RFCOMM_MAX_FRAME_SIZE_EXCEEDED
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

etDLCI_Data_Indication
etFlow_Indication
etFlow_Confirmation
etRelease_Indication
etTransport_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Parameter_Negotiation_Response

The following function is used to send a response to a DLCI parameter negotiation request (etDLCI_Param_Negotiation_Indication event). A parameter negotiation request as stated in the Bluetooth specification, can be received at any time. However, if a request is received after a channel is open, then the re-negotiation of the parameters that were accepted at the time the channel was opened, is optional.

Prototype:

int BTPSAPI **RFCOMM_Parameter_Negotiation_Response**(unsigned int BluetoothStackID, Word_t TEI, Byte_t DLCI, RFCOMM_PN_Data_t *ParamNegotiationData)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
ParamNegotiationData	A set of parameters that is being negotiated. The values received in the etDLCI_Param_Negotiation_Indication event should be examined and if they are acceptable, the response should return these values to the caller. If any parameter is not acceptable, the parameter should be changed to a value that is acceptable and returned to the caller. The parameters are passed in the following structure:

```
typedef struct
{
  Word_t                MaximumFrameSize;
  RFCOMM_Flow_Type_t    FlowType;
  Byte_t                Credits;
} RFCOMM_PN_Data_t;
```

where FlowType is one of the following values:

```
ftCreditFlowNotAllowed,
ftCreditFlowPreferred,
ftCreditFlowMandatory,
```

Return:

Zero if successful.

An error code if negative; one of the following values:

```
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
        WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI
```

Possible Events:

etRelease_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Test_Request

This function is used to send test data to RFCOMM multiplexer channel. This function has no purpose but to test to see if a remote end is responsive. The remote RFCOMM multiplexer will echo all data contained if the request back to the caller. The initiator will receive the data back via future etTest_Confirmation event

Prototype:

int BTPSAPI **RFCOMM_Test_Request**(unsigned int BluetoothStackID, Word_t TEI, Word_t Length, Byte_t *Data, RFCOMM_Event_Callback_t RFCOMM_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
Length	Length of the data.
Data	Data to send.
RFCOMM_Event_Callback	Function to call when etTest_Confirmation event occurs.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each event.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI
BTPS_ERROR_RFCOMM_FLOW_IS_DISABLED
BTPS_ERROR_RFCOMM_INVALID_MAX_FRAME_SIZE

Possible Events:

etTest_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Flow_Request

This function is used to control the flow of incoming data on an aggregate basis. The function requires a callback in order to receive confirmation that the state has changed. In this implementation, no data buffers reside in RFCOMM, so a request to halt the flow of data is sent to the remote entity. A confirmation must be received before the new state will become in effect. The TEI identifies the RFCOMM multiplexer that is being requested to halt flow. It should be noted that since the multiplexer is being halted, all DLCI (Data Link Connection Identifier) channels associated with that multiplexer will be halted with the exception of the multiplexer control channel (DLCI 0) on which the RFCOMM entities communicate.

Prototype:

int BTPSAPI **RFCOMM_Flow_Request**(unsigned int BluetoothStackID, Word_t TEI, Boolean_t ReceiverReady, RFCOMM_Event_Callback_t RFCOMM_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
ReceiverReady	Set to TRUE to allow flow between the RFCOMM entities.
RFCOMM_Event_Callback	Function to call with confirmation events.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each event.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI

Possible Events:

etFlow_Confirmation
etRelease_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Modem_Status

This function is used to convey modem status information between the RFCOMM entities. RFCOMM transparently passes the status information to the other entity and supplies the response for the command. RFCOMM will inspect the FC (Flow Control) bit of the Modem Status Byte and set the Flow State of the DLCI receiving the status information to the state reflected in the FC bit. This function operates on user DLCI and cannot be directed to the multiplexer control channel (DLCI 0). Confirmation of the delivery of the modem status information will be provided via the callback function that is assigned to the DLCI for which the status applies.

Prototype:

int BTPSAPI **RFCOMM_Modem_Status**(unsigned int BluetoothStackID, Word_t TEI,
 Byte_t DLCI, RFCOMM_Modem_Status_t *ModemStatus)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
ModemStatus	Status values to pass to the other RFCOMM entity. This is defined by the structure:

```
typedef struct
{
  Byte_t          ModemStatus;
  Boolean_t       BreakSignal;
  Byte_t          BreakLength;
} RFCOMM_Modem_Status_t;
```

where ModemStatus is defined by the ORing of the following bit masks values:

MODEM_STATUS_FC_BIT_MASK
MODEM_STATUS_RTC_BIT_MASK
MODEM_STATUS_RTR_BIT_MASK
MODEM_STATUS_IC_BIT_MASK
MODEM_STATUS_DV_BIT_MASK
MODEM_STATUS_BIT_MASK

	BreakLength is in units of 200 milliseconds (as defined by the constant: RFCOMM_BREAK_TIMEOUT_INTERVAL, which is in milliseconds). BreakLength only applies when BreakSignal is set to TRUE. This is used to send a break to the other RFCOMM entity. Constants defined that related to BreakLength are as follows: RFCOMM_BREAK_SIGNAL_DETECTED RFCOMM_BREAK_SIGNAL_MINIMUM RFCOMM_BREAK_SIGNAL_MAXIMUM

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI

Possible Events:

etModem_Status_Confirmation
etRelease_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Line_Status_Change

This function is used to convey line status change information between the RFCOMM entities. RFCOMM transparently passes the status information to the other entity and supplies a response for the message. RFCOMM does not inspect any bits of the LineStatus information, but rather passes the information to the upper layer for processing. This function operates on user DLCI and cannot be directed to the control channel (DLCI 0). Confirmation of the delivery of the line status information will be provided via the callback function that is assigned to the DLCI for which the status applies.

Prototype:

int BTPSAPI **RFCOMM_Line_Status_Change**(unsigned int BluetoothStackID,
Word_t TEI, Byte_t DLCI, Byte_t LineStatus)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
LineStatus	One or more conditions indicated by the following bit mask values: RFCOMM_LINE_STATUS_NO_ERROR_BIT_MASK RFCOMM_LINE_STATUS_OVERRUN_ERROR_BIT_MASK RFCOMM_LINE_STATUS_PARITY_ERROR_BIT_MASK RFCOMM_LINE_STATUS_FRAMING_ERROR_BIT_MASK

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI

Possible Events:

etRemote_Line_Status_Confirmation
etRelease_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Remote_Port_Negotiation_Request

This function is used to perform a Remote Port Negotiation. The Remote Port Negotiation command is used to exchange/retrieve port configuration usage information that may be useful to the upper layers. The command specifies the Baud Rate, software Flow Control information, etc. The usage of this command is optional.

Prototype:

int BTPSAPI **RFCOMM_Remote_Port_Negotiation_Request**(unsigned int BluetoothStackID, Word_t TEI, Byte_t DLCI, RFCOMM_RPN_Port_Data_t *PortData)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.

PortData	Parameters to re-negotiate, defined by the following structure:

```
typedef struct
{
  Byte_t     BaudRate;
  Byte_t     DataFormat;
  Byte_t     FlowControl;
  Byte_t     XOnCharacter;
  Byte_t     XOffCharacter;
  Word_t     ParameterMask;
} RFCOMM_RPN_Port_Data_t;
```

where BaudRate may be one of the following values:

RFCOMM_RPN_PARAMETER_BAUD_2400
RFCOMM_RPN_PARAMETER_BAUD_4800
RFCOMM_RPN_PARAMETER_BAUD_7200
RFCOMM_RPN_PARAMETER_BAUD_9600
RFCOMM_RPN_PARAMETER_BAUD_19200
RFCOMM_RPN_PARAMETER_BAUD_38400
RFCOMM_RPN_PARAMETER_BAUD_57600
RFCOMM_RPN_PARAMETER_BAUD_115200
RFCOMM_RPN_PARAMETER_BAUD_230400

DataFormat is built up from the following bit mask values, one from each section:

RFCOMM_RPN_PARAMETER_DATA_BITS_5
RFCOMM_RPN_PARAMETER_DATA_BITS_6
RFCOMM_RPN_PARAMETER_DATA_BITS_7
RFCOMM_RPN_PARAMETER_DATA_BITS_8

RFCOMM_RPN_PARAMETER_STOP_BITS_1
RFCOMM_RPN_PARAMETER_STOP_BITS_1_5 *(1.5)*

RFCOMM_RPN_PARAMETER_PARITY_DISABLED
RFCOMM_RPN_PARAMETER_PARITY_ODD
RFCOMM_RPN_PARAMETER_PARITY_EVEN
RFCOMM_RPN_PARAMETER_PARITY_MARK
RFCOMM_RPN_PARAMETER_PARITY_SPACE

The above bit mask values are already shifted to the proper bit position in the word. To access the sections of DataFormat, one may use the following masks:

RFCOMM_RPN_PARAMETER_DATA_FORMAT_DATA_
BITS_MASK
RFCOMM_RPN_PARAMETER_DATA_FORMAT_STOP_
BITS_MASK
RFCOMM_RPN_PARAMETER_DATA_FORMAT_PARITY_
MASK

FlowControl is built up from the following bit mask values:

RFCOMM_RPN_PARAMETER_FLOW_CONTROL_XON_
XOFF_ON_INPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_XON_
XOFF_ON_OUTPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_RTR_
ON_INPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_RTR_
ON_OUTPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_RTC_
ON_INPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_RTC_
ON_OUTPUT

or may be set to the following value:

RFCOMM_RPN_PARAMETER_FLOW_CONTROL_DISABLED

XOnCharacter and XoffCharacter may be any character. However, the following constants are defined for these:

RFCOMM_RPN_PARAMETER_DEFAULT_XON_CHARACTER
RFCOMM_RPN_PARAMETER_DEFAULT_XOFF_CHARACTER

ParameterMask indicates which portion(s) of the RFCOMM interface is being negotiated with this request; defined by the following bit mask values:

RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_BIT_RATE
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_DATA_BITS
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_STOP_BITS
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_PARITY
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_PARITY_TYPE
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_XON_CHARACTER
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_XOFF_CHARACTER
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_XON_XOFF_ON_INPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_XON_XOFF_ON_OUTPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_RTR_ON_INPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_RTR_ON_OUTPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_RTC_ON_INPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_
NEGOTIATE_RTC_ON_OUTPUT

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI

Possible Events:

etRemote_Port_Negotiation_Confirmation
etRelease_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Remote_Port_Negotiation_Response

The following function is used to respond to a Remote Port Negotiation Request. The Remote Port Negotiation command is used to exchange/retrieve port configuration usage information that may be useful to the upper layers. The command specifies the Baud Rate, software Flow Control information, etc. The usage of this command is mandatory if an etRemote_Port_Negotiation_Indication event is received.

Prototype:

int BTPSAPI **RFCOMM_Remote_Port_Negotiation_Response**(unsigned int BluetoothStackID, Word_t TEI, Byte_t DLCI, RFCOMM_RPN_Port_Data_t *PortData)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
PortData	Parameters to negotiate. The values received in the etRemote_Port_Negotiation_Indication event should be examined and if they are acceptable, the response should return these values to the caller. If any parameter is not acceptable, the parameter should be changed to a value that is acceptable and returned to the caller. See negotiation request command above for description of this data.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI

Possible Events:

etRelease_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Query_Remote_Port_Negotiation

This function is used to Query the Remote Side's Remote Port Negotiation Parameters.

Prototype:

int BTPSAPI **RFCOMM_Query_Remote_Port_Negotiation**(unsigned int BluetoothStackID, Word_t TEI, Byte_t DLCI)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_INVALID_TEI
BTPS_ERROR_RFCOMM_INVALID_DLCI

Possible Events:

etRelease_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Get_Channel_Status

This function is used to determine the current status of a specific RFCOMM Channel (even the Control Channel) for a specific Bluetooth device connection. This function is useful to determine when a RFCOMM Channel has been completely disconnected, as well as to determine when there is an outstanding message on a specific Channel (to aid with new connections).

Prototype:

int BTPSAPI **RFCOMM_Get_Channel_Status**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, Byte_t Channel, Boolean_t ServerChannel,
    RFCOMM_Channel_Status_t *RFCOMM_Channel_Status)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address of the remote Bluetooth device connection that the specified Server Channel is to be queried.
Channel	The RFCOMM Server Channel of the channel to query the status of. This value must be either: 0 (to query the control channel for the connection) or be a value between the following constants: RFCOMM_MINIMUM_SERVER_CHANNEL_ID RFCOMM_MAXIMUM_SERVER_CHANNEL_ID Note that this value is **NOT** a DLCI value but rather the Server Channel Number.
ServerChannel	Flag which specifies whether or not the RFCOMM Channel in question is a local RFCOMM Server (TRUE) or a remote RFCOMM connection (FALSE). Note that in either case, the Bluetooth address **MUST** specify the remotely connected Bluetooth device.
RFCOMM_Channel_Status	Pointer to a variable that is to receive the current status for the specified Channel. This value returned will be of the following values: rsTEIReady rsTEIDoesNotExist

rsTEIControlMessageOutstanding
rsTEIDisconnecting
rsDLCIDoesNotExist
rsDLCIReady
rsDLCIControlMessageOutstanding
rsDLCIDisconnecting

Return:

Zero if successful. Note that the RFCOMM_Channel_Status variable will only contain a valid value if this function returns success, otherwise the variable will contain an unknown value.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

RFCOMM_Query_Server_Channel_Present

This function is used to determine if there is an RFCOMM Server registered for the specified Server Channel.

Prototype:

int BTPSAPI **RFCOMM_Query_Server_Channel_Present**(unsigned int BluetoothStackID, Byte_t Channel, Boolean_t ServerChannelPresent)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
Channel	The RFCOMM Server Channel of the server port to determine the existence of. This value must be between the following values: RFCOMM_MINIMUM_SERVER_CHANNEL_ID RFCOMM_MAXIMUM_SERVER_CHANNEL_ID Note that this value is **NOT** a DLCI value but rather the Server Channel Number.
ServerChannelPresent	Buffer which will hold the Boolean return value which specifies whether a server is present (TRUE) or is not present (FALSE) for the specified Server Channel.

Return:

Zero if successful. Note that the ServerChannelPresent variable will only contain a valid value if this function returns success, otherwise the variable will contain an unknown value.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.5.2 RFCOMM Event Callback

The RFCOMM event callback is used in several of the RFCOMM commands for capturing RFCOMM events. This callback function is defined as follows:

RFCOMM_Event_Callback_t

Callback function for all RFCOMM events.

Prototype:

void (BTPSAPI ***RFCOMM_Event_Callback_t**)(unsigned int BluetoothStackID,
    RFCOMM_Event_Data_t *RFCOMM_Event_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
RFCOMM_Event_Data	The event that occurred. This is defined by the structure:

```
typedef struct
{
  RFCOMM_Event_Data_Type_t            RFCOMM_Event_Data_Type;
  DWord_t                             Event_Data_Length;
  union
  {
    RFCOMM_Open_Indication_Data_t          *RFCOMM_Open_Indication_Event_Data;
    RFCOMM_Open_Confirmation_Data_t        *RFCOMM_Open_Confirmation_Event_Data;
    RFCOMM_Release_Indication_Data_t       *RFCOMM_Release_Indication_Event_Data;
    RFCOMM_Data_Data_t                     *RFCOMM_Data_Indication_Event_Data;
    RFCOMM_Param_Negotiation_Data_t        *RFCOMM_Param_Negotiation_Indication_Event_Data;
    RFCOMM_Remote_Port_Negotiation_Data_t
             *RFCOMM_Remote_Port_Negotiation_Indication_Event_Data;
```

```
            RFCOMM_Remote_Port_Negotiation_Data_t
                        *RFCOMM_Remote_Port_Negotiation_Confirmation_Event_Data;
            RFCOMM_Remote_Line_Status_Data_t   *RFCOMM_Remote_Line_Status_Indication_Event_Data;
            RFCOMM_Remote_Line_Status_Confirmation_Data_t
                        *RFCOMM_Remote_Line_Status_Confirmation_Event_Data;
            RFCOMM_Modem_Status_Data_t             *RFCOMM_Modem_Status_Indication_Event_Data;
            RFCOMM_Modem_Status_Confirmation_Data_t
                        *RFCOMM_Modem_Status_Confirmation_Event_Data;
            RFCOMM_Test_Data_t                     *RFCOMM_Test_Confirmation_Event_Data;
            RFCOMM_Flow_Data_t                     *RFCOMM_Flow_Indication_Event_Data;
            RFCOMM_Flow_Confirmation_Data_t        *RFCOMM_Flow_Confirmation_Event_Data;
            RFCOMM_Credit_Indication_Data_t        *RFCOMM_Credit_Indication_Event_Data;
            RFCOMM_Non_Supported_Command_Data_t        *RFCOMM_Non_Supported_Command_Data;
            RFCOMM_Transport_Buffer_Empty_Data_t       *RFCOMM_Transport_Buffer_Empty_Data;
          } RFCOMM_Event_Data;
        } RFCOMM_Event_Data_t;
```

Where RFCOMM_Event_Data_Type one of the enumerations of the event types listed in the table in section 2.5.3, and each data structure in the union is described with its event in that section as well.

CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.5.3 RFCOMM Events

The events that can be generated by the RFCOMM portion of the Bluetooth Stack are listed in the table below and are described in the text that follows.

Event	Description
etOpen_Indication	Channel is being requested to the RFCOMM server.
etOpen_Confirmation	Channel has been opened with the remote RFCOMM server.
etRelease_Indication	Channel has been disconnected.
etDLCI_Data_Indication	Data has been received on the indicated channel.
etDLCI_Param_Negotiation_Indication	A request has been made to negotiate DLCI parameters for the channel.
etRemote_Port_Negotiation_Indication	A request has been made to query or re-negotiate the port parameters.

etRemote_Port_Negotiation_ Confirmation	Port negotiation response has been received.
etRemote_Line_Status_ Indication	Line status change request has been received.
etRemote_Line_Status_ Confirmation	Line status change has notification has been completed.
etModem_Status_Indication	Modem status change request has been received.
etModem_Status_ Confirmation	Modem status change notification has been completed.
etTest_Confirmation	Test data has been received.
etFlow_Indication	Flow control change request has been received.
etFlow_Confirmation	Flow control change has been completed.
etCredit_Indication	New flow control credits have received.
etNon_Supported_Command _Indication	A non-supported command has been received.
etTransport_Buffer_Empty_I ndication	Used to notify that RFCOMM has buffer space available for transmit data functions.

etOpen_Indication

Channel open request has been received by the RFCOMM server.

Return Structure:

```
typedef struct
{
  Word_t                TEI;
  Byte_t                DLCI;
  BD_ADDR_t             BD_ADDR;
  RFCOMM_PN_Data_t      DLCI_Parameters;
} RFCOMM_Open_Indication_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
BD_ADDR	Address of the requesting Bluetooth device.
DLCI_Parameters	Parameters for this link, defined in the following structure:

```
typedef struct
{
  Word_t                MaximumFrameSize;
  RFCOMM_Flow_Type_t    FlowType;
  Byte_t                Credits;
} RFCOMM_PN_Data_t;
```

where FlowType is one of the following values:

ftCreditFlowNotAllowed,
ftCreditFlowPreferred,
ftCreditFlowMandatory,

etOpen_Confirmation

Confirm that channel has been opened (or failed to open).

Return Structure:

```
typedef struct
{
  Word_t              TEI;
  Byte_t              DLCI;
  Byte_t              Result;
  RFCOMM_PN_Data_t    DLCI_Parameters;
} RFCOMM_Open_Confirmation_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
Result	Status of the open request. May be one of the following values: RFCOMM_CONNECT_RESULT_CONNECTION_SUCCESSFUL RFCOMM_CONNECT_RESULT_CONNECTION_TIMEOUT RFCOMM_CONNECT_RESULT_CONNECTION_REFUSED
DLCI_Parameters	Parameters for this link, defined in the following structure:

```
typedef struct
{
  Word_t                MaximumFrameSize;
  RFCOMM_Flow_Type_t    FlowType;
  Byte_t                Credits;
} RFCOMM_PN_Data_t;
```

where FlowType is one of the following values:

ftCreditFlowNotAllowed,
ftCreditFlowPreferred,
ftCreditFlowMandatory,

etRelease_Indication

A channel has been disconnected.

Return Structure:

```
typedef struct
{
  Word_t        TEI;
  Byte_t        DLCI;
} RFCOMM_Release_Indication_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.

etDLCI_Data_Indication

RFCOMM channel data has been received.

Return Structure:

```
typedef struct
{
  Word_t        TEI;
  Byte_t        DLCI;
  Word_t        DataLength;
  Byte_t        *Data;
} RFCOMM_Data_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
DataLength	Length of the data.
Data	Received data.

etDLCI_Param_Negotiation_Indication

Request to negotiate DLCI parameters for the channel has been received.

Return Structure:

```
typedef struct
{
  Word_t                TEI;
  Byte_t                DLCI;
  RFCOMM_PN_Data_t      Params;
} RFCOMM_Param_Negotiation_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.

Params	A set of parameters that is being negotiated. The values received in the etDLCI_Param_Negotiation_Indication event should be examined and if they are acceptable, the response should return these values to the caller. If any parameter is not acceptable, the parameter should be changed to a value that is acceptable and returned to the caller. The parameters are passed in the following structure:

```
typedef struct
{
  Word_t              MaximumFrameSize;
  RFCOMM_Flow_Type_t  FlowType;
  Byte_t              Credits;
} RFCOMM_PN_Data_t;
```

where FlowType is one of the following values:

```
ftCreditFlowNotAllowed,
ftCreditFlowPreferred,
ftCreditFlowMandatory,
```

etRemote_Port_Negotiation_Indication
etRemote_Port_Negotiation_Confirmation

Request has been received to return the Port Negotiation parameters, either from a query or a (re-)negotiation request (indication), or a response has been received (confirmation).

Return Structure:

```
typedef struct
{
  Word_t                  TEI;
  Byte_t                  DLCI;
  Boolean_t               ParameterRequest;
  RFCOMM_RPN_Port_Data_t  PortData;
} RFCOMM_Remote_Port_Negotiation_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
ParameterRequest	TRUE if this is a request (indication event) and FALSE if this is a confirmation.
PortData	Parameters to re-negotiate, defined by the following structure:

```
typedef struct
{
  Byte_t    BaudRate;
  Byte_t    DataFormat;
  Byte_t    FlowControl;
  Byte_t    XOnCharacter;
  Byte_t    XOffCharacter;
```

```
   Word_t     ParameterMask;
} RFCOMM_RPN_Port_Data_t;
```

where BaudRate may be one of the following values:

RFCOMM_RPN_PARAMETER_BAUD_2400
RFCOMM_RPN_PARAMETER_BAUD_4800
RFCOMM_RPN_PARAMETER_BAUD_7200
RFCOMM_RPN_PARAMETER_BAUD_9600
RFCOMM_RPN_PARAMETER_BAUD_19200
RFCOMM_RPN_PARAMETER_BAUD_38400
RFCOMM_RPN_PARAMETER_BAUD_57600
RFCOMM_RPN_PARAMETER_BAUD_115200
RFCOMM_RPN_PARAMETER_BAUD_230400

DataFormat is built up from the following bit mask values, one from each section:

RFCOMM_RPN_PARAMETER_DATA_BITS_5
RFCOMM_RPN_PARAMETER_DATA_BITS_6
RFCOMM_RPN_PARAMETER_DATA_BITS_7
RFCOMM_RPN_PARAMETER_DATA_BITS_8

RFCOMM_RPN_PARAMETER_STOP_BITS_1
RFCOMM_RPN_PARAMETER_STOP_BITS_1_5 *(1.5)*

RFCOMM_RPN_PARAMETER_PARITY_DISABLED
RFCOMM_RPN_PARAMETER_PARITY_ODD
RFCOMM_RPN_PARAMETER_PARITY_EVEN
RFCOMM_RPN_PARAMETER_PARITY_MARK
RFCOMM_RPN_PARAMETER_PARITY_SPACE

The above bit mask values are already shifted to the proper bit position in the word. To access the sections of DataFormat, one may use the following masks:

RFCOMM_RPN_PARAMETER_DATA_FORMAT_DATA_
BITS_MASK
RFCOMM_RPN_PARAMETER_DATA_FORMAT_STOP_
BITS_MASK
RFCOMM_RPN_PARAMETER_DATA_FORMAT_PARITY_
MASK

FlowControl is built up from the following bit mask values:

RFCOMM_RPN_PARAMETER_FLOW_CONTROL_XON_
XOFF_ON_INPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_XON_
XOFF_ON_OUTPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_RTR_
ON_INPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_RTR_
ON_OUTPUT
RFCOMM_RPN_PARAMETER_FLOW_CONTROL_RTC_
ON_INPUT

RFCOMM_RPN_PARAMETER_FLOW_CONTROL_RTC_ON_OUTPUT

or may be set to the following value:

RFCOMM_RPN_PARAMETER_FLOW_CONTROL_DISABLED

XOnCharacter and XoffCharacter may be any character. However, the following constants are defined for these:

RFCOMM_RPN_PARAMETER_DEFAULT_XON_CHARACTER
RFCOMM_RPN_PARAMETER_DEFAULT_XOFF_CHARACTER

ParameterMask indicates which portion(s) of the RFCOMM interface is being negotiated with this request; defined by the following bit mask values:

RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_BIT_RATE
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_DATA_BITS
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_STOP_BITS
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_PARITY
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_PARITY_TYPE
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_XON_CHARACTER
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_XOFF_CHARACTER
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_XON_XOFF_ON_INPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_XON_XOFF_ON_OUTPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_RTR_ON_INPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_RTR_ON_OUTPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_RTC_ON_INPUT
RFCOMM_RPN_PARAMETER_PARAMETER_MASK_NEGOTIATE_RTC_ON_OUTPUT

etRemote_Line_Status_Indication
etRemote_Line_Status_Confirmation

The line status change has been received or confirm the response to receipt.

Return Structure:

```
typedef struct
{
  Word_t       TEI;
  Byte_t       DLCI;
  Byte_t       LineStatus;
} RFCOMM_Remote_Line_Status_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
LineStatus	One or more conditions indicated by the following bit mask values: RFCOMM_LINE_STATUS_NO_ERROR_BIT_MASK RFCOMM_LINE_STATUS_OVERRUN_ERROR_BIT_MASK RFCOMM_LINE_STATUS_PARITY_ERROR_BIT_MASK RFCOMM_LINE_STATUS_FRAMING_ERROR_BIT_MASK

etRemote_Line_Status_Confirmation

The line status change has been received or confirm the response to receipt.

Return Structure:

```
typedef struct
{
  Word_t       TEI;
  Byte_t       DLCI;
} RFCOMM_Remote_Line_Status_Confirmation_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.

etModem_Status_Indication

A modem status change has been received.

Return Structure:

```
typedef struct
{
  Word_t                      TEI;
  Byte_t                      DLCI;
  RFCOMM_Modem_Status_t       ModemStatus;
} RFCOMM_Modem_Status_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.

ModemStatus	Status values received from the other RFCOMM entity. This is defined by the structure:

```
typedef struct
{
  Byte_t          ModemStatus;
  Boolean_t       BreakSignal;
  Byte_t          BreakLength;
} RFCOMM_Modem_Status_t;
```

where ModemStatus is defined by the ORing of the following bit masks values:

```
RFCOMM_MODEM_STATUS_FC_BIT_MASK
RFCOMM_MODEM_STATUS_RTC_BIT_MASK
RFCOMM_MODEM_STATUS_RTR_BIT_MASK
RFCOMM_MODEM_STATUS_IC_BIT_MASK
RFCOMM_MODEM_STATUS_DV_BIT_MASK
RFCOMM_MODEM_STATUS_BIT_MASK
```

Note: BreakLength (in Break Signal Intervals of 200ms) only applies when BreakSignal is set to TRUE. This is used to send a break to the other RFCOMM entity. The following constants are defined when using the BreakLength member:

```
RFCOMM_BREAK_TIMEOUT_INTERVAL
RFCOMM_BREAK_SIGNAL_DETECTED
RFCOMM_BREAK_SIGNAL_MINIMUM
RFCOMM_BREAK_SIGNAL_MAXIMUM
```

etModem_Status_Confirmation

Confirm that the modem status change has been processed.

Return Structure:

```
typedef struct
{
  Word_t          TEI;
  Byte_t          DLCI;
} RFCOMM_Modem_Status_Confirmation_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.

etTest_Confirmation

Confirm that the test data has been sent and responded to (or caused an error).

Return Structure:

```
typedef struct
{
  Word_t      TEI;
  Word_t      SequenceLength;
  Byte_t      *Sequence;
} RFCOMM_Test_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
SequenceLength	Length of the Sequence data.
Data	Actually data returned (echoed).

etFlow_Indication

Flow control change request has been received.

Return Structure:

```
typedef struct
{
  Word_t      TEI;
  Byte_t      DLCI;
  Boolean_t   ReceiverReady;
} RFCOMM_Flow_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
ReceiverReady	TRUE will resume flow between RFCOMM entities, FALSE will pause it.

etFlow_Confirmation

Flow control change request has been processed.

Return Structure:

```
typedef struct
{
  Word_t              TEI;
} RFCOMM_Flow_Confirmation_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.

etCredit_Indication

Indicate that additional flow control credit has been received.

Return Structure:

```
typedef struct
{
  Word_t      TEI;
  Byte_t      DLCI;
  Byte_t      NewCredits;
  DWord_t     TotalCredits;
} RFCOMM_Credit_Indication_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
NewCredits	Additional credits received.
TotalCredits	Current total of credits (new added to existing)

etNon_Supported_Command_Indication

A command was received which is not supported by this implementation of RFCOMM.

Return Structure:

```
typedef struct
{
  Word_t      TEI;
  Byte_t      DLCI;
  Byte_t      UnsupportedCommand;
} RFCOMM_Non_Supported_Command_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.
DLCI	Data Link Connection Identifier of channel.
UnsupportedCommand	Command received.

etTransport_Buffer_Empty_Indication

Used to notify that all data which has been buffered has been transmitted and that additional data write functions can resume if they had been disabled due to an channel buffer full condition..

Return Structure:

```
typedef struct
{
  Word_t      TEI;
  Byte_t      DLCI;
} RFCOMM_Transport_Buffer_Empty_Data_t;
```

Event Parameters:

TEI	Terminal Endpoint Identifier of channel.

DLCI	Data Link Connection Identifier of channel.

2.6 SCO API

The Synchronous Connection-Oriented link API provides capabilities for managing SCO Connections. This API layer consists of callbacks, described in section 2.6.1 and commands, described in section 2.6.2. The actual prototypes and constants outlined in this section can be found in the **SCOAPI.H** header file in the Bluetopia distribution.

2.6.1 SCO Event/Data Callbacks and Registration

The SCO callbacks available in the Bluetooth Protocol Stack API and the functions used to register and unregister them are listed in the table below and described in the text which follows.

Callback/Function	Description/Purpose
SCO_Connect_Request_Callback_t	Handle SCO Connection Requests.
SCO_Connection_Callback_t	Handle SCO Connection Actions.
SCO_Register_Synchronous_Connect_Request_Callback	Registers a eSCO and SCO Connection Request callback.
SCO_Register_Connect_Request_Callback	Register a connection request callback function with the SCO layer.
SCO_Un_Register_Callback	Undo a callback registration

The callback function is free to use the contents of the SCO Action Data **only** in the context of the callback. If the function requires the data for a longer period of time, then the callback function **must** copy them into another data buffer(s).

These callback functions 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 Needs to be noted however, that if the same Callback is installed more than once, then the callbacks will be called serially. Because of this, the processing in this functionshould be as efficient as possible. It should also be noted that these functions are 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 another SCO Action will not be processed while one of these function calls is outstanding).

NOTE: These functions MUST NOT Block and wait for events that can only be satisfied by receiving other Bluetooth Stack Events. A Deadlock WILL occur because other Callbacks might not be issued while one of these functions is currently outstanding.

SCO_Connect_Request_Callback_t

This is the prototype function for an SCO Connection Request Callback. This function will be called whenever an SCO Connection Request occurs within the Bluetooth Protocol Stack that is specified with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the SCO Connection Request Data associated with the SCO Connection Request that occurred, and the SCO Callback Parameter that was specified when this Callback was installed.

Note: A Connection can **only** be accepted/rejected in the context of this callback function. If the SCO_Accept_Connection function is **not** called during this callback (to accept or reject the connection) then there is no way to Accept/Reject the SCO Connection Request, and the SCO Connection Request will timeout on the originator's end and fail.

Prototype:

void (BTPSAPI ***SCO_Connect_Request_Callback_t**)(unsigned int BluetoothStackID,
SCO_Connect_Request_Data_t *SCO_Connect_Request_Data,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1] — Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

SCO_Connect_Request_Data — Data associated with this connection request. This data structure is defined as follows:

```
typedef struct
{
  BD_ADDR_t           BD_ADDR;
  Class_of_Device_t   Class_of_Device;
  unsigned int        SCO_Connection_ID;
  SCO_Link_Type_t     LinkType;
} SCO_Connect_Request_Data_t;
```

Where,

```
BD_ADDR             The address of the requesting device.
Class_of_Device     Class of the requesting device.
SCO_Connection_ID   Identifier for this connection which is
          passed to the SCO_Accept_Connection function.
LinkType            The link type of the connection request.
          Possible values are:
              ltSCO
              ltESCO
```

CallbackParameter — User-defined parameter (e.g., tag value) that was defined in the callback registration.

SCO_Connection_Callback_t

This is a dynamic callback function which is associated with an SCO Connection and receives notification when actions are taken on the connection, namely a successful connection or a disconnect. Callbacks of this type are passed to the following two functions:

SCO_Add_Connection	Initiate a connection to a remote device
SCO_Accept_Connection	Respond to request for a connection from a remote device.

This function passes to the caller the Bluetooth Stack ID, the SCO Action Data associated with the SCO Action that occurred, and the SCO Callback Parameter that was specified when this Callback was installed.

Prototype:

void (BTPSAPI ***SCO_Connection_Callback_t**)(unsigned int BluetoothStackID,
SCO_Event_Data_t *SCO_Event_Data, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCO_Event_Data	Event associated with this SCO Connection. This structure is defined as follows:

```
typedef struct {
  SCO_Event_Type_t                 SCO_Event_Type;
  Word_t                           SCO_Event_Data_Size;
  union
  {
    SCO_Connect_Result_Event_t  *SCO_Connect_Result_Event;
    SCO_Disconnect_Event_t      *SCO_Disconnect_Event;
    SCO_Data_Indication_Event_t *SCO_Data_Indication_Event;
    SCO_Transmit_Buffer_Empty_Event_t *
             SCO_Transmit_Buffer_Empty_Event;
    SCO_Synchronous_Connection_Changed_Event_t
             *SCO_Synchronous_Connection_Changed_Event;
  } SCO_Event_Data;
} SCO_Event_Data_t;
```

Where, the SCO_Event_Type is one of the following possible values are:

```
etSCO_Connect_Result
etSCO_Disconnect
etSCO_Data_Indication
etSCO_Transmit_Buffer_Empty_Indication
etSCO_Synchronous_Connection_Changed
```

And, the Event Data structures are defined below. These are associated, respectively, with the Event Types defined above.

```
typedef struct
{
  unsigned int       SCO_Connection_ID;
  BD_ADDR_t          BD_ADDR;
  unsigned int       Connection_Status;
  SCO_Link_Type_t    LinkType;
  Byte_t             Transmission_Interval;
  Byte_t             Retransmission_Window;
  Word_t             Rx_Packet_Length;
  Word_t             Tx_Packet_Length;
  Byte_t             Air_Mode;
} SCO_Connect_Result_Event_t;

typedef struct
{
   unsigned int       SCO_Connection_ID;
   BD_ADDR_t          BD_ADDR;
   unsigned int       Disconnection_Status;
} SCO_Disconnect_Event_t;

typedef struct
{
   unsigned int        SCO_Connection_ID;
   BD_ADDR_t           BD_ADDR;
   Byte_t              DataLength;
   Byte_t             *DataBuffer;
   Word_t              PacketStatus;
} SCO_Data_Indication_Event_t;

typedef struct
{
   unsigned int       SCO_Connection_ID;
   BD_ADDR_t          BD_ADDR;
} SCO_Transmit_Buffer_Empty_Event_t;

typedef struct
{
   unsigned int       SCO_Connection_ID;
   Byte_t             Status;
   Byte_t             Transmission_Interval;
   Byte_t             Retransmission_Window;
   Word_t             Rx_Packet_Length;
   Word_t             Tx_Packet_Length;
} SCO_Synchronous_Connection_Changed_Event_t;
```

Where the Connection_Status and Disconnection_Status are zero (0) for no error, otherwise they are HCI Error Codes (see section 2.2). Note, in the Data Event, the DataBuffer is not a pointer, but the actual data itself. Therefore the structure will be variable in size. A macro exists,

SCO_DATA_INDICATION_EVENT_SIZE(DataLength)

to assist in calculating the total size (in bytes) of the structure. The DataLength argument is the size (in bytes) of the amount of data that is or will be put into the Data Event structure.

CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

SCO_Register_Synchronous_Connect_Request_Callback

Registers a SCO and eSCO Connection Request Callback with the Bluetooth protocol stack identified by BluetoothStackID. If this call is successful, the callback function will be notified of subsequent Asynchronous eSCO and SCO Connection Requests.

Prototype:

int BTPSAPI **SCO_Register_Synchronous_Connect_Request_Callback**(unsigned int BluetoothStackID, SCO_Connect_Request_Callback_t SCO_Connect_Request_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCO_Connect_Request_Callback	Callback function that is to be registered. A Connection can ONLY be accepted/rejected in the context of this callback function This function MUST NOT Block and wait for events that can only be satisfied by Receiving other Bluetooth Stack Events. A Deadlock WILL occur because other Callbacks might not be issued while this function is currently outstanding. typedef void (BTPSAPI *SCO_Connect_Request_Callback_t)(unsigned int BluetoothStackID, SCO_Connect_Request_Data_t *SCO_Connect_Request_Data, unsigned long CallbackParameter);
CallbackParameter	User defined parameter that will be passed to the callback function when invoked.

Return:

Positive non-zero SCOCallbackID if successful.

Negative Error code if not successful.

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Register_Connect_Request_Callback

This function is responsible for registering a SCO Connection Request Callback with the specified Bluetooth Protocol Stack (specified via the BluetoothStackID parameter). Once this Callback is installed, the caller will be notified of asynchronous SCO Connection Requests when they occur.

Prototype:

int BTPSAPI **SCO_Register_Connect_Request_Callback**(unsigned int BluetoothStackID, SCO_Connect_Request_Callback_t SCO_Connect_Request_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCO_Connect_Request_Callback	User-supplied callback function.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive non-zero value if successful which is the registration ID (SCOCallbackID) that is used to unregister the Callback.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_CALLBACK_INFORMATION

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Un_Register_Callback

Remove a previously registered SCO Connection Request Callback.

Prototype:

int BTPSAPI **SCO_Un_Register_Callback**(unsigned int BluetoothStackID,
unsigned int SCOCallbackID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOCallbackID	Identifier returned from a successful callback registration.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DELETING_CALLBACK_INFORMATION

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

2.6.2 SCO Commands

The SCO layer API provides the commands listed in the table below, which are described in the text which follows.

Command	Description
SCO_Setup_Synchronous_Connection	Adds SCO and eSCO connection to specified Bluetooth device.
SCO_Add_Connection	Add an SCO Connection with a remote device.
SCO_Close_Connection	Close an SCO Connection.
SCO_Accept_Synchronous_Connection	Accepts or rejects a SSCO/eSCO Connection Request.
SCO_Accept_Connection	Accept or reject an SCO Connection request from a remote device.
SCO_Modify_Synchronous_Connection	Used to modify existing synchronous connection.
SCO_Send_Data	Send SCO data to an open SCO Connection (immediately).
SCO_Set_Queue_Threshold	Sets the current SCO Queuing Threshold information.
SCO_Get_Queue_Threshold	Queries the current SCO Queuing Threshold information.
SCO_Query_Packet_Information	Query the current HCI SCO Packet Size/Buffer Information.
SCO_Query_Data_Format	Query the current HCI SCO Data Format Information.
SCO_Change_Data_Format	Change the current HCI SCO Data Format Information.
SCO_Change_Buffer_Size	Change the current SCO Transmit Buffer (Queue) Size.
SCO_Purge_Buffer	Flush all Data queued in SCO Transmit Buffer.
SCO_Queue_Data	Queue Data into SCO Transmit Buffer.
SCO_Change_Packet_Information	Override the HCI SCO Packet Size/Buffer information that is used by the SCO layer.
SCO_Set_Connection_Mode	Sets SCO connection mode.
SCO_Set_Physical_Transport	Informs SCO module about the type of Physical Transport

Command	Description
	that will be use for SCO data.

SCO_Setup_Synchronous_Connection

This function adds an SCO and eSCO connection to the remote device specified by BD_ADDR. If successful the caller can pass the return value of this function to CO_Close_Connection() function. Note, there must already be an ACL connection to the specified Bluetooth device for this function to receive.

Prototype:

int BTPSAPI **SCO_Setup_Synchronous_Connection** (unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, SCO_Synchronous_Connection_Info_t *SynchronousConnectionInfo, SCO_Connection_Callback_t SCO_Connection_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address of the remote device to setup SCO/eSCO connection to.
SynchronousConnectionInfo	The connection parameters for the connection. To use defaults this parameter may be set to NULL.
SCO_Connection_Callback	Callback function to be installed for this connection. This is called when a SCO/eSCO event occurs on the specified SCO/eSCO connection.
CallbackParameter	Parameter that is passed to the callback function when a SCO/eSCO event occurs.

Return:

Non-zero, positive value on success which indicates the SCO/eSCO Connection ID of the specified Connection Link. Note that this does NOT mean that the SCO/eSCO Connection has been established in the case of a Accept. This information is returned in the specifed Connection Callback with the Connection Result.

Negative error code indicating a SCO/eSCO was not able to be established with the specified Bluetooth device.

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Add_Connection

This command is used to add an SCO Connection with another Bluetooth device. Note, there must already be an ACL Link with the Bluetooth device, or this request will fail.

Prototype:

int BTPSAPI **SCO_Add_Connection**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, SCO_Connection_Callback_t SCO_Connection_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the Bluetooth device to make the connection to.
SCO_Connection_Callback	Function to call to report connection status/actions to.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

If successful, a positive, non-zero value is returned which is the SCO Connection ID.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_MAX_SCO_CONNECTIONS
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etSCO_Connect_Result

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Close_Connection

Close an existing SCO Connection. Once this function is called for the specified SCO Connection ID, that SCO Connection is no Longer valid (if established) and the SCO Connection Callback that was registered with the Connection will no longer be called.

Prototype:

int BTPSAPI **SCO_Close_Connection**(unsigned int BluetoothStackID,
unsigned int SCOConnectionID, unsigned int Disconnect_Status);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOConnectionID	The identifier for this connection which was returned from a successful call to SCO_Add_Connection.
Disconnect_Status	The reason for the disconnection, which is one of the HCI Error Codes (see Section 2.2).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Accept_Synchronous_Connection

The following function is responsible for Accepting or Rejecting a SCO/eSCO Connection Request. This function CAN ONLY be called in the Context of a SCO/eSCO Connection Request Callback.

Prototype:

int BTPSAPI **SCO_Accept_Synchronous_Connection** (unsigned int BluetoothStackID, unsigned int SCOConnectionID, SCO_Synchronous_Connection_Info_t *SynchronousConnectionInfo, unsigned int RejectReason, SCO_Connection_Callback_t SCO_Connection_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOConnectionID	Obtained via the SCO_Connection_ID member of the SCO_Connect_Request Data member of the SCO Connect Request Event Data. This Data is specified in a SCO/eSCO Callback, so the caller will only be able to issue this function if a SCO/eSCO Callback has been installed.

SynchronousConnectionInfo	Required parameters of the connection, if set to NULL default values will be used.
RejectReason	Specifies whether or not the caller has Accepted or Rejected the SCO Connection Request. If this parameter is zero, then the SCO Request will be accepted, else this parameter represents the Rejection Reason (defined in the Bluetooth HCI specification Error Codes).
SCO_Connection_Callback	Callback function that is to be installed for the accepted SCO/eSCO connection. Ignored if the connection is being rejected, MUST be valid if the connection is being accepted. This Callback Function (and specified Callback Parameter) will be used when any SCO/eSCO Event occurs on the accepted SCO/eSCO Connection (if accepted).
CallbackParameter	Parameter to the callback function. Will be ignored if the connection is being reject, otherwise must be valid.

Return:

Zero if successful, meaning the connection has been accepted or rejected,.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_ACTION_NOT_ALLOWED
BTPS_ERROR_MAX_SCO_CONNECTIONS
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etSCO_Connect_Result

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Accept_Connection

This command is used to accept or reject a request from a remote Bluetooth device to establish an SCO Connection. This function *must* be called in the context of an SCO Connection Request Callback or it will have no effect.

Prototype:

int BTPSAPI **SCO_Accept_Connection**(unsigned int BluetoothStackID,
    unsigned int SCOConnectionID, unsigned int RejectReason,

SCO_Connection_Callback_t SCO_Connection_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOConnectionID	The unique identifier for this SCO Connection. This is provided to the SCO Connection Request Callback function.
RejectReason	If the connection is being accepted, this parameter is set to zero. If the connection is being rejected, this parameter is set to one of the HCI Error Codes (see Section 2.2).
SCO_Connection_Callback	Function to call to report connection status/actions to.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

If successful, a positive, non-zero value is returned which is the SCO Connection ID.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_MAX_SCO_CONNECTIONS
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Modify_Synchronous_Connection

This function is used to modify an existing synchronous connection. Note, only eSCO connections can be modified.

Prototype:

int BTPSAPI **SCO_Modify_Synchronous_Connection** (unsigned int BluetoothStackID,
unsigned int SCOConnectionID, Word_t MaxLatency, SCO_Retransmission_Effort_t RetransmissionEffort);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

SCOConnectionID	Connection ID obtained via a successful call to SCO_Setup_Synchronous_Connection().
MaxLatency	The value in milliseconds representing the upper limit of the sum of the synchronous interval.
RetransmissionEffort	The ReTransmissionEffort modes for a eSCO connection. May be one of the following: reNoRetransmissions reRetransmitOptimizePowerConsumption reRetransmitOptimizeLinkQuality reDontCare

Return:

If successful, a positive, non-zero value is returned. This means that the command was successfully sent to the device. The actual success of modifying the connection will be in the status of etSynchronous_Connection_Changed_Event returned from the SCO_Connection_Callback passed in during SCO_Setup_Synchronous_Connection().

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_INVALID_MODE

Possible Events:

etSynchronous_Connection_Changed_Event

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Send_Data

Send SCO Data to the specified SCO Connection. This function segments the data being sent into packet sizes that acceptable to the Bluetooth device.

Notes:

If this function returns BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE then the application must wait for the etSCO_Transmit_Buffer_Empty_Indication event and re-transmit the selected data.

Prototype:

int BTPSAPI **SCO_Send_Data**(unsigned int BluetoothStackID,
 unsigned int SCOConnectionID, Byte_t SCODataLength, Byte_t *SCOData)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOConnectionID	The unique identifier for this SCO Connection. This is provided to the SCO Connection Request Callback function.
SCODataLength	Length of the Data referenece by SCOData.
SCOData	Pointer to the data to send.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

etSCO_Transmit_Buffer_Empty_Indication
etSCO_Disconnect

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Set_Queue_Threshold

This command is used to set the SCO queue threshold. The queue threshold is globally applicable to all SCO connections.

Prototype:

int BTPSAPI **SCO_Set_Queue_Threshold** (unsigned int BluetoothStackID,
SCO_Queueing_Parameters_t *QueueingParameters)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
QueueingParameters	Pointer to structure that contains the queue threshold information to set. This structure is defined as follows:

```
typedef struct
{
  Word_t    QueueLimit;
  Word_t    LowThreshold;
} SCO_Queueing_Parameters_t;
```

where QueueLimit specifies the maximum number outstanding HCI SCO Packets that are acceptable for each SCO connection, and LowThreshold is the point after which a transmit buffer empty will be dispatched (if the buffer is ever considered fully full). QueueLimit must be less than or equal to the MaximumOutstandingSCOPackets member that is returned from the SCO_Query_Packet_Information() API and LowThreshold must be less than QueueLimit.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Get_Queue_Threshold

This command is used to get the current SCO queue threshold.

Prototype:

int BTPSAPI **SCO_Get_Queue_Threshold** (unsigned int BluetoothStackID,
SCO_Queueing_Parameters_t *QueueingParameters)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
QueueingParameters	Pointer to structure to return the current SCO queue threshold information. This structure is defined as follows:

```
typedef struct
{
  Word_t    QueueLimit;
  Word_t    LowThreshold;
} SCO_Queueing_Parameters_t;
```

where QueueLimit specifies the maximum number outstanding HCI SCO Packets that are acceptable for each SCO connection, and LowThreshold is the point after which a transmit buffer empty will be dispatched (if the buffer is ever considered fully full).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Query_Packet_Information

This command is used to query the current HCI SCO Packet/Buffer Information. The information returned from this function is applicable to ALL SCO Channels and cannot be different for individual SCO Channels.

Prototype:

int BTPSAPI **SCO_Query_Packet_Information**(unsigned int BluetoothStackID,
    SCO_Packet_Information_t *SCO_Packet_Information)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCO_Packet_Information	Pointer to buffer that is to receive the current SCO Packet information. This structure is defined as follows:

```
typedef struct
{
  unsigned int   MaximumOutstandingSCOPackets;
  unsigned int   MaximumSCOPacketSize;
} SCO_Packet_Information_t;
```

where MaximumOutstandingSCOPackets specifies the number outstanding HCI SCO Packets that are acceptable to the Bluetooth device (as reported by the Bluetooth device), and MaximumSCOPacketSize is the maximum size of an individual SCO Packet (in Bytes) that can be accepted by the Bluetooth device (as reported by the Bluetooth device).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

2. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Query_Data_Format

This command is used to query the current HCI SCO Data Format. The information returned from this function is applicable to ALL SCO Channels and cannot be different for individual SCO Channels.

Prototype:

int BTPSAPI **SCO_Query_Data_Format**(unsigned int BluetoothStackID,
SCO_Data_Format_t * SCO_Data_Format)

Parameters:

BluetoothStackID[1] — Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

SCO_Data_Format — Pointer to buffer that is to receive the current SCO data format information. This structure is defined as follows:

```
typedef struct
{
  SCO_Data_Encoding_Type_t    SCO_Data_Encoding_Type;
  SCO_Data_Encoding_Format_t  SCO_Data_Encoding_Format;
  SCO_PCM_Data_Sample_Size_t
          SCO_PCM_Data_Sample_Size;
  SCO_Air_Encoding_Type_t     SCO_Air_Encoding_Type;
} SCO_Data_Format_t;
```

where the SCO_Data_Encoding_Type member defines the encoding type of the input/output data and is defined to be one of the following types:

deLinearPCM
deuLaw
deALaw

The SCO_Data_Encoding_Format member defines the encoding format of the input/output data and is defined to be one of the following types:

ef1sComplement
ef2sComplement
efSignMagnitude

efUnsigned

The SCO_PCM_Data_Sample_Size member is valid only if the Data Encoding type specified Linear PCM Audio. When this member is valid it is defined to be one of the following types:

ds8Bit
ds16Bit

The SCO_Air_Encoding_Type member specifies the encoding type that is to be used over the Bluetooth Link (over the Air Encoding). This member is defined to be one of the following types:

aeCVSD
aeuLaw
aeALaw
aeNone

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Change_Data_Format

This command is used to change the current HCI SCO Data Format. The information that is changed with this function is applicable to ALL SCO Channels and cannot be different for individual SCO Channels. Note some of the formats that this function allows to be set may note be supported by all Bluetooth devices.

Prototype:

int BTPSAPI **SCO_Change_Data_Format**(unsigned int BluetoothStackID,
    SCO_Data_Format_t * SCO_Data_Format)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCO_Data_Format	Pointer to buffer that specifies the new SCO data format information. This structure is defined as follows:

```
typedef struct
{
  SCO_Data_Encoding_Type_t    SCO_Data_Encoding_Type;
  SCO_Data_Encoding_Format_t  SCO_Data_Encoding_Format;
  SCO_PCM_Data_Sample_Size_t
          SCO_PCM_Data_Sample_Size;
  SCO_Air_Encoding_Type_t     SCO_Air_Encoding_Type;
} SCO_Data_Format_t;
```

where the SCO_Data_Encoding_Type member defines the encoding type of the input/output data and is defined to be one of the following types:

deLinearPCM
deuLaw
deALaw

The SCO_Data_Encoding_Format member defines the encoding format of the input/output data and is defined to be one of the following types:

ef1sComplement
ef2sComplement
efSignMagnitude
efUnsigned

The SCO_PCM_Data_Sample_Size member is valid only if the Data Encoding type specified Linear PCM Audio. When this member is valid it is defined to be one of the following types:

ds8Bit
ds16Bit

The SCO_Air_Encoding_Type member specifies the encoding type that is to be used over the Bluetooth Link (over the Air Encoding). This member is defined to be one of the following types:

aeCVSD
aeuLaw
aeALaw
aeNone

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ HCI_RESPONSE_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Change_Buffer_Size

This command is used to change the buffer size of an outgoing SCO Transmit Buffer. This Buffer is set for an individual SCO Connection and is available for queuing SCO Data into. Once Data is queued into the SCO buffer, it will be sent automatically by the SCO Module to the Bluetooth device when required. This mechanism allows an application the ability to simply fill up a buffer (and keep the buffer occupied with data) and allowing the SCO Module to handle all Bluetooth Flow Control issues. The default value for the Buffer Size is zero which means NO queue is available. When there is no queue, NO data can be queued, only sent via the **SCO_Send_Data** function. The buffer size can be changed dynamically, however, changing the buffer size deletes all current information that is contained in the buffer. Therefore, the buffer size should only be changed when the application knows the buffer is empty.

Prototype:

int BTPSAPI **SCO_Change_Buffer_Size**(unsigned int BluetoothStackID,
    unsigned int SCOConnectionID, unsigned int TransmitBufferSize)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOConnectionID	The unique identifier for this SCO Connection. This is provided to the SCO Connection Request Callback function.
TransmitBufferSize	Size (in bytes) to change the SCO Output Buffer (Queue) size to.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Purge_Buffer

This command is used to clear the current contents of an outgoing SCO Transmit Buffer. This Buffer is active for an individual SCO Connection only and not all SCO Connections. Currently the only supported action is to delete all data that is currently present in the output buffer. Waiting for all data in the output buffer to be flushed is not supported. After this function is called, the SCO Output Data buffer is completely empty.

Prototype:

int BTPSAPI **SCO_Purge_Buffer** (unsigned int BluetoothStackID,
unsigned int SCOConnectionID, unsigned int PurgeBufferMask)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOConnectionID	The unique identifier for this SCO Connection. This is provided to the SCO Connection Request Callback function.
PurgeBufferMask	Mechanism with which to flush the Output buffer. Currently the following values are supported: SCO_PURGE_MASK_TRANSMIT_ABORT_BIT

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Queue_Data

This command is used to queued outgoing SCO Data into a previously established SCO Buffer. This buffer must have been established via a successful call to the **SCO_Change_Buffer_Size** function. Data queued into this buffer is sent to the Bluetooth device via the SCO Module. This eliminates the need for the application to worry about when (and how much) data to send to the Bluetooth device. The application, using this mechanism, only needs to keep the buffer updated with outgoing SCO Data and the SCO Module will take care of sending all SCO Data to the Module.

Note: If this function is unable to queue all of the data that was specified (via the SCODataLength parameter) because of a full Transmit Buffer condition, this function will return the number of bytes that were actually sent (zero or more, but less than the DataLength parameter value). When this happens (and **only** when this happens), the user can expect to be notified when the Transmit buffer is able to queue data again via the the etSCO_Transmit_Buffer_Empty_Indication SCO Event. This will allow the user a mechanism to know when the Transmit Buffer is empty so that more data can be sent.

Prototype:

int BTPSAPI **SCO_Queue_Data**(unsigned int BluetoothStackID,
    unsigned int SCOConnectionID, unsigned int SCODataLength, Byte_t *SCOData)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCOConnectionID	The unique identifier for this SCO Connection. This is provided to the SCO Connection Request Callback function.
SCODataLength	The number of data bytes to queue
SCOData	The data buffer that contains the data to queue

Return:

Positive or zero if successful indicating the number of data bytes actually queued. See note above, for situations when this value is less than SCODataLength.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Change_Packet_Information

This command is used to override the current HCI SCO Packet/Buffer Information. The information changed by this function is applicable to ALL SCO Channels and cannot be different for individual SCO Channels. This function is provided because it has been found that some Bluetooth HCI SCO implementations incorrectly report the parameters that can actually be used. This function allows the values that are used internally to differ from the values that are reported from the Bluetooth device via the HCI_Read_Buffer_Size HCI commands.

Prototype:

int BTPSAPI **SCO_Change_Packet_Information**(unsigned int BluetoothStackID,
SCO_Packet_Information_t *SCO_Packet_Information)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SCO_Packet_Information	Pointer to buffer that contains the new SCO Packet information. This structure is defined as follows:

```
typedef struct
{
   unsigned int   MaximumOutstandingSCOPackets;
   unsigned int   MaximumSCOPacketSize;
} SCO_Packet_Information_t;
```

where MaximumOutstandingSCOPackets specifies the number outstanding HCI SCO Packets that are acceptable to the Bluetooth device, and MaximumSCOPacketSize is the maximum size of an individual SCO Packet (in Bytes) that can be accepted by the Bluetooth device.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Set_Connection_Mode

This function is responsible for setting the SCO Connection Mode.

Prototype:

int BTPSAPI **SCO_Set_Connection_Mode**(unsigned int BluetoothStackID,
SCO_Connection_Mode_t ConnectionMode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

ConnectionMode	The second parameter is the SCO Connection Mode to set. May be one of the following: scmDisableConnections scmEnableConnections Specifying scmDisableConnections as the Connection Mode shall disconnect all currently on going connections and disallow all new connection requests.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SCO_Set_Physical_Transport

This function tells the SCO module about the type of Physical Transport that will be used to transport SCO Data. There is no defined way in the Bluetooth specification to determine this data as it depends on the physical Bluetooth Hardware configuration. The Physical Transport can ONLY be changed if there are NO active SCO connections.

Prototype:

int BTPSAPI **SCO_Set_Physcial_Transport**(unsigned int BluetoothStackID,
 SCO_Physical_Transport_t PhysicalTransport);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
PhysicalTransport	The Physical Transport value to set. Can be one of the following: sptCodec sptHCI

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SCO_NOT_INITIALIZED
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_MODE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

3. Profile Interfaces

The following Profile Interfaces are included in the Stonestreet One Bluetooth Stack Protocol at present and the sections they are documented in are:

3.1 GAP Programming Interface

The GAP (Generic Access Profile) programming interface provides features related to: (1) discovery of other Bluetooth devices, (2) link management aspects of connecting to those devices, and (3) using different levels of security. Commonly used data types are listed in section 3.1.1. Section 3.1.2 lists the GAP function calls. Section 3.1.3 lists the GAP event callback prototypes. Section 3.1.4 lists all supported GAP events. The actual prototypes and constants outlined in this section can be found in the **GAPAPI.H** header file in the Bluetopia distribution.

3.1.1 Commonly Used GAP Data Types

The following data types and structures are commonly used in the GAP functions. The list of data types covered in this section are listed in the table below.

Data Type	Description
GAP_Authentication_Information_t	Structure to hold GAP authentication information to be set and/or returned.
GAP_LE_Authentication_Response_ Information_t	Structure to hold GAP LE authentication information to be set and/or returned.

GAP_Authentication_Information_t

Structure to hold GAP authentication information to be set and/or returned. For GAP authentication types that are rejections, the Authentication_Data_Length member is set to zero and all data members can be ignored (since non are valid). Currently the Bonding_Type member of the IO_Capabilities member is ignored. The correct value is calculated and inserted automatically.

Structure:

```
typedef
{
  GAP_Authentication_Type_t  GAP_Authentication_Type;
  Byte_t                     Authentication_Data_Length;
  union
  {
    PIN_Code_t               PIN_Code;
    Link_Key_t               Link_Key;
    Boolean_t                Confirmation;
    DWord_t                  Passkey;
    GAP_Keypress_t           Keypress;
    GAP_Out_Of_Band_Data_t   Out_Of_Band_Data;
    GAP_IO_Capabilities_t    IO_Capabilities;
  } Authentication_Data;
} GAP_Authentication_Information_t;
```

Fields:

GAP_Authentication_Type_t	The different authentication methods that can be used and which member of the union should be used. Possible values are: atLinkKey atPINCode atUserConfirmation atPassKey atKeypress atOutOfBandData atIOCapabilities
Authentication_Data_Length	Length of authentication data. For rejected authentication types this value will be zero (0), and the data can/should be ignored.
PIN_Code_t	Up to 16 byte Personal Identification Number.
Link_Key_t	Up to 16 byte link key.
Confirmation	Used during user confirmation to specify the confirmation result.
Passkey	5 digit pass key (00, 000 – 99, 999)
Keypress	Specifies key press data. This value will be one of the following: kpEntryStarted kpDigitEntered kpDigitErased kpCleared kpEntryCompleted
Out_Of_Band_Data	Specifies out of band (OOB) data. This structure has the following format:

```
typedef struct
{
  Simple_Pairing_Hash_t
            Simple_Pairing_Hash;
  Simple_Pairing_Randomizer_t
            Simple_Pairing_Randomizer;
} GAP_Out_Of_Band_Data_t;
```

IO_Capabilities	Specifies I/O capabilities of the device. This value will be one of the following: icDisplayOnly icDisplayYesNo icKeyboardOnly icNoInputNoOutput

GAP_LE_Authentication_Response_Information_t

Structure to hold GAP LE authentication information to be set and/or returned. For GAP authentication types that are rejections, the Authentication_Data_Length member is set to zero and all data members can be ignored (since non are valid).

Structure:

```
typedef struct
{
  GAP_LE_Authentication_Response_Type_t   GAP_LE_Authentication_Type;
  Byte_t                                  Authentication_Data_Length;
  union
  {
    GAP_LE_Long_Term_Key_Information_t    Long_Term_Key_Information;
    GAP_LE_Pairing_Capabilities_t         Pairing_Capabilities;
    GAP_LE_OOB_Data_t                     Out_Of_Band_Data;
    DWord_t                               Passkey;
    Byte_t                                Error_Code;
    GAP_LE_Encryption_Information_t       Encryption_Information;
    GAP_LE_Identity_Information_t         Identity_Information;
    GAP_LE_Signing_Information_t          Signing_Information;
  } Authentication_Data;
} GAP_LE_Authentication_Response_Information_t;
```

Fields:

GAP_LE_Authentication_Type_t	The different authentication methods that can be used and which member of the union should be used. Possible values are: larLongTermKey larOutOfBandData larPairingCapabilities larPasskey larConfirmation larError larEncryptionInformation

	larIdentityInformation larSigningInformation
Authentication_Data_Length	Length of authentication data. For rejected authentication types this value will be zero (0), and the data can/should be ignored.

Long_Term_Key_Information	Contains the long term key information. This structure has the following format:
	typedef struct
	{
	  Byte_t Encryption_Key_Size;
	  Long_Term_Key_t Long_Term_Key;
	} **GAP_LE_Long_Term_Key_Information_t**;

Pairing_Capabilities	Specifies the pairing capabilities of the local host. This structure is defined as follows:
	typedef struct
	{
	  GAP_LE_IO_Capability_t IO_Capability;
	  Boolean_t OOB_Present;
	  GAP_LE_Bonding_Type_t Bonding_Type;
	  Boolean_t MITM;
	  Byte_t
	    Maximum_Encryption_Key_Size;
	  GAP_LE_Key_Distribution_t Receiving_Keys;
	  GAP_LE_Key_Distribution_t Sending_Keys;
	} **GAP_LE_Pairing_Capabilities_t**;

Out_Of_Band_Data	Specifies out of band (OOB) data. This structure has the following format: typedef struct { Encryption_Key_t OOB_Key; } **GAP_LE_OOB_Data_t**;
Passkey	6 digit pass key (000, 000 – 999, 999)
Error_Code	Specifies result of an on-going authentication procedure.

Encryption_Information	Specifies current encryption information. This structure has the following format:
	typedef struct
	{
	  Byte_t Encryption_Key_Size;
	  Long_Term_Key_t LTK;
	  Word_t EDIV;
	  Random_Number_t Rand;
	} **GAP_LE_Encryption_Information_t**;

Identity_Information	Specifies current identity information. This structure has the following format:

```
typedef struct
{
   Encryption_Key_t          IRK;
   GAP_LE_Address_Type_t     Address_Type;
   BD_ADDR_t                 Address;
} GAP_LE_Identity_Information_t;
```

Signing_Information — Specifies current signing information. This structure has the following format:

```
typedef struct
{
   Encryption_Key_t CSRK;
} GAP_LE_Signing_Information_t;
```

3.1.2 GAP Functions

The available GAP functions are listed in the table below and are described in the text that follows:

Function	Description
GAP_Set_Discoverability_Mode	Set the discoverability mode.
GAP_Query_Discoverability_Mode	Read the current discoverability mode.
GAP_Set_Connectability_Mode	Enable/disable connections to the local device.
GAP_Query_Connectability_Mode	Read the current connectability mode.
GAP_Set_Pairability_Mode	Enable/disable pairability mode.
GAP_Query_Pairability_Mode	Read the current the pairability mode.
GAP_Set_Authentication_Mode	Enable/disable authentication.
GAP_Query_Authentication_Mode	Read the current authentication mode.
GAP_Set_Encryption_Mode	Enable/disable encryption.
GAP_Cancel_Set_Encryption _Mode	Cancel any future callback notifications associated with changing the encryption mode.
GAP_Query_Encryption_Mode	Read the current encryption mode.
GAP_Authenticate_Remote_Device	Authenticate the indicated remote device
GAP_Cancel_Authenticate_Remote_Device	Cancel the authentication process on the indicated remote Bluetooth device.
GAP_Register_Remote_Authentication	Register a GAP Event Callback function to accept authentication requests from remote devices.
GAP_Un_Register_Remote_Authentication	Un-register a callback function for

	authentication requests.
GAP_Authentication_Response	Send the authentication information requested by a remote Bluetooth device.
GAP_Perform_Inquiry	Initiate an inquiry scan for other Bluetooth devices.
GAP_Cancel_Inquiry	Cancel an inquiry scan.
GAP_Set_Inquiry_Mode	Set the inquiry mode.
GAP_Query_Inquiry_Mode	Retrieve the inquiry mode.
GAP_Query_Remote_Device_Name	Retrieve the user-friendly name of a remote Bluetooth device.
GAP_Cancel_Query_Remote_Device_Name	Cancel any future callback notifications associated with a specific remote name request.
GAP_Query_Remote_Features	Retrieve features of the remote device.
GAP_Query_Remote_Version_Information	Retrieve version information of the remote device.
GAP_Initiate_Bonding	Initiate a bonding procedure of the type requested.
GAP_Cancel_Bonding	Cancel a bonding process that was previously started.
GAP_End_Bonding	Terminate a link established by a call to GAP_Initiate_Bonding.
GAP_Query_Local_BD_ADDR	Get the local Bluetooth device address.
GAP_Set_Class_Of_Device	Change the device class of the local Bluetooth device.
GAP_Query_Class_Of_Device	Read the current class of device of the local Bluetooth device.
GAP_Set_Local_Device_Name	Change the user-friendly name of the local Bluetooth device.
GAP_Query_Local_Device_Name	Read the current user-friendly name of the local Bluetooth device.
GAP_Disconnect_Link	Terminate an existing Bluetooth ACL connection.
GAP_Query_Connection_Handle	Query the ACL connection handle of a current connection to a remote Bluetooth device.
GAP_Query_Local_Out_Of_Band_Data	Retrive Out of band data from local

	device.
GAP_Refresh_Encryption_Key	Refesh the encryption key.
GAP_Read_Extended_Inquiry_Information	Get the extended inquiry information.
GAP_Write_Extended_Inquiry_Information	Write the extended inquiry information for the local device.
GAP_Convert_Extended_Inquiry_Response_Data	Convert the extended inquiry response data.
GAP_Parse_Extended_Inquiry_Response_Data	Parse the fields of the extended inquiry response data.
GAP_LE_Create_Connection	Scan and connect to a remote Bluetooth LE device.
GAP_LE_Cancel_Create_Connection	Cancel an on-going Bluetooth LE connection request.
GAP_LE_Disconnect	Disconnect from a currently connected Bluetooth LE device.
GAP_LE_Read_Remote_Features	Query the remote LE features of a currently connected Bluetooth LE device.
GAP_LE_Perform_Scan	Perform an active or passive scan for Bluetooth LE devices.
GAP_LE_Cancel_Scan	Cancel an on-going Bluetooth LE scan procedure.
GAP_LE_Set_Advertising_Data	Set the Bluetooth LE advertising data that is used when advertising is enabled.
GAP_LE_Convert_Advertising_Data	Convert the LE advertising data.
GAP_LE_Parse_Advertising_Data	Parse the fields of the advertising data.
GAP_LE_Set_Scan_Response_Data	Set the Bluetooth LE scan response data that is used when an active scan is detected.
GAP_LE_Convert_Scan_Response_Data	Convert the LE scan response data.
GAP_LE_Parse_Scan_Response_Data	Parse the fields of the scan response data.
GAP_LE_Advertising_Enable	Instruct the local Bluetooth LE device to begin advertising.
GAP_LE_Advertising_Disable	Instruct the local Bluetooth LE device to stop advertising.
GAP_LE_Generate_Non_Resolvable_Address	Generate a non-resolvable device address.
GAP_LE_Generate_Static_Address	Generate a static private address.

GAP_LE_Generate_Resolvable_Address	Generate a resolvable device address.
GAP_LE_Resolve_Address	Resolve a specified resolvable address.
GAP_LE_Set_Random_Address	Instruct local Bluetooth LE device to use the specified random address.
GAP_LE_Add_Device_To_White_List	Add one (or more) devices to the Bluetooth LE controller white list.
GAP_LE_Remove_Device_From_White_List	Remove one (or more) devices from the Bluetooth LE controller white list.
GAP_LE_Read_White_List_Size	Determine the number of devices the local Bluetooth LE controller can support in the controller white list.
GAP_LE_Set_Pairability_Mode	Set the GAP LE pairability mode for the local device.
GAP_LE_Register_Remote_Authentication	Register with the local GAP LE entity to receive authentication events.
GAP_LE_Un_Register_Remote_Authentication	Un-register for authentication events.
GAP_LE_Pair_Remote_Device	Begin a pairing process with the specified remote Bluetooth LE device (master only).
GAP_LE_Authentication_Response	Respond to a remote authentication request.
GAP_LE_Reestablish_Security	Re-establish previously established security.
GAP_LE_Request_Security	Request the master to re-establish security (slave only).
GAP_LE_Set_Fixed_Passkey	Allows a fixed passkey to be used when the local GAP LE entity is chosen to display a passkey during pairing.
GAP_LE_Query_Encryption_Mode	Query the encryption mode of a specified LE connection.
GAP_LE_Query_Connection_Handle	Query the connection handle of a specified LE connection.
GAP_LE_Generate_Long_Term_Key	Generate a long term key pairing key.
GAP_LE_Regenerate_Long_Term_Key	Re-generate a long term pairing key.
GAP_LE_Diversify_Function	Utility function which performs the diversify function which is used during key management.

GAP_LE_Connection_Parameter_Update_Request	Request that the master update the current connection parameters (slave only).
GAP_LE_Connection_Parameter_Update_Response	Respond to a request from a slave to update the connection parameters (master only).

GAP_Set_Discoverability_Mode

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

Prototype:

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

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Discoverability_Mode	Value that defines the limits to being discovered by other Bluetooth devices. The following modes are currently defined: dmNonDiscoverableMode dmLimitedDiscoverableMode dmGeneralDiscoverableMode
Max_Discoverable_Time	Length of time in seconds that the unit will be in the specified discoverable mode (not applicable for non discoverable mode).

Return:

Zero (0) if the discoverability mode was successfully changed**.**

Negative if an error occurred and the mode was not changed. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_MODE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_INTERNAL_ERROR
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Discoverability_Mode

This function allows a means to query the current discoverability mode parameters for the local Bluetooth device. The second parameter to this function is a pointer to a variable that will receive the current discoverability mode of the Bluetooth device, and the last parameter specifies a pointer to a variable that will receive the current discoverability mode maximum discoverability mode timeout value. Both of these parameters must be valid (i.e. non-NULL) and upon successful completion of this function will contain the current discoverability mode parameters of the local Bluetooth device.

Prototype:

int BTPSAPI **GAP_Query_Discoverability_Mode**(unsigned int BluetoothStackID,
    GAP_Discoverability_Mode_t *GAP_Discoverability_Mode,
    unsigned int *Max_Discoverable_Time);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Discoverability_Mode	Pointer to a buffer to receive the value that defines the current mode of discovery. The following modes are currently defined: dmNonDiscoverableMode dmLimitedDiscoverableMode dmGeneralDiscoverableMode
Max_Discoverable_Time	Pointer to a buffer to receive the length of time (in seconds) that the unit was specified to be the discoverable mode.

Return:

Zero (0) if the discoverability mode was successfully retrieved**.**

Negative if an error occurred. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Set_Connectability_Mode

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.

Prototype:

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

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Connectability_Mode	Value that defines the connectability mode (from other Bluetooth devices). The following modes are currently defined: cmNonConnectableMode cmConnectableMode

Return:

Zero (0) if the connectability mode was successfully changed**.**

Negative if an error occurred and the Mode was not changed. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_MODE
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Connectability_Mode

This function allows a means to query the current connectability mode for the local Bluetooth device. The second parameter to this function is a pointer to a variable that will receive the current connectability mode of the local Bluetooth device. The second parameter must be valid (i.e. non-NULL), and upon successful completion of this function will contain the current connectability mode of the local Bluetooth device.

Prototype:

int BTPSAPI **GAP_Query_Connectability_Mode**(unsigned int BluetoothStackID,
GAP_Connectability_Mode_t *GAP_Connectability_Mode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Connectability_Mode	Pointer to a buffer to receive the value that defines the current mode of connectability. The following modes are currently defined: cmNonConnectableMode cmConnectableMode

Return:

Zero (0) if the Connectability Mode was successfully retrieved**.**

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Set_Pairability_Mode

The following 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.

Note:

If secure simple pairing (SSP) pairing mode is specified, then SSP *MUST* be used for all pairing operations. The device can be placed into non pairable mode after this, however, if pairing is re-enabled, it *MUST* be set to pairable with SSP enabled.

Prototype:

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

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Pairability_Mode	Value that defines the pairability mode (to other Bluetooth devices). The following modes are currently defined: pmNonPairableMode pmPairableMode

pmPairableMode_EnableSecureSimplePairing

Return:

Zero (0) if the pairability mode was successfully changed**.**

Negative if an error occurred and the mode was not changed. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_MODE

Possible Events:

etAuthentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Pairability_Mode

This function is provided to allow a means to query the current pairability mode for the local Bluetooth device that is specified by the Bluetooth protocol stack that is associated with the specified Bluetooth stack ID. The second parameter to this function is a pointer to a variable that will receive the current pairability mode of the Bluetooth device. The second parameter must be valid (i.e. non-NULL), and upon successful completion of this function will contain the current pairability mode of the local Bluetooth device.

Prototype:

int BTPSAPI **GAP_Query_Pairability_Mode**(unsigned int BluetoothStackID,
    GAP_Pairability_Mode_t *GAP_Pairability_Mode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Pairability_Mode	Pointer to a buffer to receive the value that defines the current mode of pairability. The following modes are currently defined: pmNonPairableMode pmPairableMode pmPairableMode_EnableSecureSimplePairing

Return:

Zero (0) if the pairability mode was successfully retrieved**.**

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Set_Authentication_Mode

This function is provided to set the authentication 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 authentication mode to place the local Bluetooth device into.

Note:

If authentication is enabled for the local Bluetooth device, then this means that EVERY connection (both incoming and outgoing) will require authentication at the link level.

Link level authentication is not recommended for Bluetooth version 2.1 and greater (devices that support secure simple pairing (SSP)).

Prototype:

int BTPSAPI **GAP_Set_Authentication_Mode**(unsigned int BluetoothStackID,
GAP_Authentication_Mode_t GAP_Authentication_Mode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_ Authentication _Mode	Value that defines the authentication mode to set on the local device. The following modes are currently defined: amEnabled amDisabled

Return:

Zero (0) if the authentication mode was successfully changed**.**

Negative if an error occurred and the mode was not changed. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_INVALID_MODE
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etAuthentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_ Authentication _Mode

This function is provided to allow a means to query the current authentication mode for the Bluetooth device that is specified by the Bluetooth protocol stack that is associated with the specified Bluetooth stack ID. The second parameter to this function is a pointer to a variable that will receive the current authentication mode of the Bluetooth device. The second parameter must be valid (i.e. non-NULL), and upon successful completion of this function will contain the current authentication mode of the local Bluetooth device.

Note:

If Authentication is enabled for the local Bluetooth device, then this means that EVERY connection (both incoming and outgoing) will require authentication at the link level.

Link level authentication is not recommended for Bluetooth version 2.1 and greater (devices that support secure simple pairing (SSP)).

Prototype:

int BTPSAPI **GAP_Query_Authentication_Mode**(unsigned int BluetoothStackID,
    GAP_Authentication_Mode_t *GAP_Authentication_Mode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_ Authentication _Mode	Pointer to an area to receive the value that defines the current mode of Authentication. The following modes are currently defined: amDisabled amEnabled

Return:

Zero (0) if the authentication mode was successfully retrieved**.**

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Set_Encryption_Mode

This function is provided to allow the setting of the encryption modes for either:

- the local Bluetooth device (link level encryption – all connections)
- the specified (connected) Bluetooth device address

The second parameter specifies the Bluetooth device address to apply the encryption mode setting to (could be the local device or a connected remote device). The third parameter specifies the state of the encryption mode to change to. The final two parameters specify the GAP event callback to receive the encryption status when the encryption is changed. This callback will contain the actual status of the encryption change (success or failure). If the local device address is specified for the second parameter, then this function will set the specified encryption mode for ALL future link level connections. When the local device address is specified, the callback function and parameter are ignored, and the function return value indicates whether or not the encryption change was successful (for the local device for future connections). If the second parameter is NOT the local device address then this function will set the encryption mode at the link level for the specified Bluetooth link ONLY. A physical ACL link MUST already exist for this to work. The actual status of the encryption change for this link will be passed to the callback information that is required when using this function in this capacity. Because this function is asynchronous in nature (when specifying a non local device address), this function will notify the caller of the result via the installed callback. The caller is free to cancel the encryption mode change at any time by issuing the GAP_Cancel_Set_Encryption_Mode function and specifying the Bluetooth device address of the Bluetooth device that was specified in this call. It should be noted that when the callback is cancelled, the callback is the ONLY thing that is cancelled (i.e. the GAP module still changes the encryption for the link, it's just that NO callback will be issued).

Prototype:

int BTPSAPI **GAP_Set_Encryption_Mode**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, GAP_Encryption_Mode_t GAP_Encryption_Mode,
    GAP_Event_Callback_t GAP_Event_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_ Encryption _Mode	Value that defines the Encryption mode of the Bluetooth device. The following modes are currently defined: emEnabled emDisabled
GAP_Event_Callback	Callback function that will be used to dispatch result information to the upper layers.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the callbacks function.

Return:

Zero (0) if the encryption mode was successfully changed**.**

Negative if an error occurred and the mode was not changed. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etEncryption_Change_Result

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Cancel_Set_Encryption _Mode

This function is provided to cancel the future calling of a previously registered encryption mode callback that was installed via a successful call to the GAP_Set_Encryption_Mode function. This function DOES NOT cancel the changing of the encryption mode for the specified Bluetooth device, it ONLY cancels the callback notification. This function accepts as input the Bluetooth protocol stack ID of the Bluetooth device that the GAP_Set_Encryption_Mode function was previously issued, and the device address of the Bluetooth device that the previous call was called with. The BD_ADDR parameter MUST be valid, and cannot be the device address of the local Bluetooth device because the local encryption mode change does not use the callback mechanism.

Prototype:

int BTPSAPI **GAP_Cancel_Set_Encryption_Mode**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device that was previously issued with the GAP_Set_Encryption_Mode function.

Return:

Zero (0) if the encryption mode callback was successfully canceled**.**

Negative if an error occurred and the request was not canceled. Possible values are:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_NO_CALLBACK_REGISTERED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Encryption_Mode

This function is provided to allow a means to query the current encryption mode for the Bluetooth device that is specified. The second parameter to this function is the Bluetooth device address of the device to query the encryption state of. If the local Bluetooth device address is specified for this parameter then the encryption information that is returned represents the current encryption link level state of all future ACL connections (both incoming and outgoing). The third parameter to this function is a pointer to a variable that will receive the current encryption mode of the Bluetooth device/link. The third parameter to this function must be valid (i.e. non-NULL), and upon successful completion of this function will contain the current encryption mode for the Bluetooth device/link requested. If the second parameter is NOT the local device address, then this function will query the encryption mode on the current link level for the specified Bluetooth link (device must be connected). A physical ACL link MUST already exist for this to work (remote device address specified).

Prototype:

int BTPSAPI **GAP_Query_Encryption_Mode**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, GAP_Encryption_Mode_t *GAP_Encryption_Mode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device to which the encryption is to be retreived.
GAP_Encryption_Mode	Pointer to a buffer to receive the current encryption mode setting.

Return:

Zero (0) if the encryption mode request was successfully retrieved.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Authenticate_Remote_Device

This function is provided to allow a means to authenticate a remote device. This function accepts as input the Bluetooth protocol stack ID of the local Bluetooth device, the Bluetooth device address of the remote device to authenticate, and the GAP event callback (and callback parameter) information that is to be used during the authentication process to inform the caller of GAP authentication events and/or requests. Note that even if this function returns success, it does NOT mean that the specified remote device was successfully authenticated, it only that the authentication process has been started. Because this function is asynchronous in nature, this function will notify the caller of the result via the specified callback. The caller is free to cancel the authentication process at any time by calling the GAP_Cancel_Authenticate_Remote_Device function and specifying the Bluetooth device address of the Bluetooth device that was specified in this call. It should be noted that when the callback is cancelled, only the callback is cancelled (i.e. the GAP module still processes the authentication events only this callback will not be used during the remainder of the process).

Prototype:

int BTPSAPI **GAP_Authenticate_Remote_Device**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, GAP_Event_Callback_t GAP_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device to authenticate.
GAP_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the callback.

Return:

Zero (0) if the authentication process was successfully started.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_NO_CALLBACK_REGISTERED
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etAuthentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Cancel_Authenticate_Remote_Device

This function is provided to allow a means to cancel a current authentication process of a specified remote device. This function accepts as input the Bluetooth protocol stack ID of the local Bluetooth device and the Bluetooth device address of the remote device to cancel to the authentication process of.

Prototype:

int BTPSAPI **GAP_Authenticate_Remote_Device**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device on which to cancel the authentication.

Return:

Zero (0) if the cancellation request was successfully processed.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_NO_CALLBACK_REGISTERED
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Register_Remote_Authentication

This function is provided to allow a means to register a GAP event callback to accept remote authentication requests. This function accepts as input the GAP event callback information to register. It should be noted that ONLY ONE remote authentication callback can be installed per Bluetooth device. The caller can un-register the remote authentication callback that was registered with this function (if successful) by calling the GAP_Un_Register_Remote_Authentication function.

Note:

A remote authentication event is defined as an authentication event that was not requested by the local device (i.e. a pairing or authentication request issued from a remote device to the local device).

Prototype:

int BTPSAPI **GAP_Register_Remote_Authentication**(unsigned int BluetoothStackID, GAP_Event_Callback_t GAP_Event_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the remote authentication callback was successfully registered.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etAuthentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Un_Register_Remote_Authentication

This function is provided to allow a mechanism to un-register a previously registered GAP event callback for remote authentication events. This function accepts as input the Bluetooth stack ID of the Bluetooth device that the remote authentication callback was registered previously (via a successful call to the GAP_Register_Remote_Authentication function).

Prototype:

int BTPSAPI **GAP_Un_Register_Remote_Authentication**(unsigned int BluetoothStackID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

Return:

Zero (0) if the remote authentication callback was successfully un-registered.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Authentication_Response

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).

Note:

This function should be called to respond to authentication requests that were received via any of the installed callbacks:

- Bonding callback
- Authentication callback
- Remote authentication callback

Prototype:

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

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device that is being authenticated.
GAP_Authentication_Information	Pointer to a structure that holds authentication information.

Return:

Zero (0) if the remote authentication response was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etAuthentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Perform_Inquiry

This function is provided to allow a mechanism of starting an Inquiry scan procedure. The first parameter to this function is the Bluetooth protocol stack ID 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 (only valid in case a periodic inquiry scan is requested). The fifth parameter is the length of time to perform the inquiry procedure. The sixth parameter is the maximum number of responses to wait for. The final two parameters represent the callback function (and callback parameter) that is to be called when the specified inquiry has completed as well as when the individual inquiry results are found. This function returns zero if successful, or a negative return error code if an Inquiry was unable to be performed. Only a single inquiry scan can be performed at any given time. Calling this function while an outstanding inquiry scan is in progress will fail. The caller can call the GAP_Cancel_Inquiry function to cancel a currently executing inquiry procedure. The minimum and maximum inquiry parameters are optional and if specified represent the minimum and maximum periodic inquiry periods. The caller should set BOTH of these values to zero if a simple inquiry scan procedure is to be used (non-periodic). If these two parameters are specified, then these two parameters must satisfy the following formula:

MaximumPeriodLength > MinimumPeriodLength > InquiryLengthAll

Note:

All time values are specified in seconds.

The actual type of inquiry result that is returned in the specified callback depends on the current inquiry mode. The inquiry mode can be set with the GAP_Set_Inquiry_Mode function. The default inquiry mode is standard which returns the inquiry result via the etInquiry_Entry_Result event.

Prototype:

int BTPSAPI **GAP_Perform_Inquiry**(unsigned int BluetoothStackID,
GAP_Inquiry_Type_t GAP_Inquiry_Type, unsigned int MinimumPeriodLength, unsigned int MaximumPeriodLength, unsigned int InquiryLength,

unsigned int MaximumResponses, GAP_Event_Callback_t GAP_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Inquiry_Type	Type of inquiry scan to Perform. The currently defined values are: itGeneralInquiry itLimitedInquiry
MinimumPeriodLength	Mininum length of time to perform the inquiry procedure (in seconds). This parameter is only applicable if a periodic inquiry scan is required. This value must be in the following range: MINIMUM_MINIMUM_INQUIRY_PERIOD_LENGTH MAXIMUM_MINIMUM_INQUIRY_PERIOD_LENGTH
MaximumPeriodLength	Maxinum length of time to perform the inquiry procedure (in seconds). This parameter is only applicable if a periodic inquiry scan is required. This value must be in the following range (and satisfy the equation listed above: MINIMUM_MAXIMUM_INQUIRY_PERIOD_LENGTH MAXIMUM_MAXIMUM_INQUIRY_PERIOD_LENGTH
InquiryLength	Length of time to perform the inquiry procedure (in seconds). This value must be in the following range: MINIMUM_INQUIRY_LENGTH MAXIMUM_INQUIRY_LENGTH
MaximumResponses	Maximum number of responses to be received before the process is terminated. This value must be either: INFINITE_NUMBER_INQUIRY_RESPONSES or within the following range: MINIMUM_NUMBER_INQUIRY_RESPONSES MAXIMUM_NUMBER_INQUIRY_RESPONSES
GAP_Event_Callback	Pointer to a callback function that is used by the GAP layer to dispatch result information about the inquiry process.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the inquiry procedure was successfully started.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_DEVICE_HCI_ERROR

BTPS_ERROR_INVALID_MODE
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

etInquiry_Entry_Result

etInquiry_With_RSSI_Entry_Result

etExtended_Inquiry_Entry_Result

etInquiry_Result

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Cancel_Inquiry

This function is provided to allow a means of cancelling a currently active inquiry scan process that was started via a successful call to the GAP_Perform_Inquiry function. This function accepts as input the Bluetooth protocol stack that is associated with the Bluetooth device that is currently performing an inquiry scan procedure. This function returns zero if the inquiry process was able to be cancelled, or a negative return error code if there was an error. If this function returns success then the GAP event callback that was installed with the GAP_Perform_Inquiry function will NEVER be called (for the currently inquiry procedure).

Prototype:

int BTPSAPI **GAP_Cancel_Inquiry**(unsigned int BluetoothStackID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

Return:

Zero (0) if the inquiry process was successfully halted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Set_Inquiry_Mode

The following function is provided to set the inquiry 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 inquiry mode to place the local Bluetooth device into. This function returns zero if the inquiry mode was able to be successfully changed, otherwise this function returns a negative value which signifies an error condition.

Note:

The inquiry mode dictates how the local device will actually perform inquiries (and more importantly, how the results will be returned). The following table shows supported modes and the corresponding GAP inquiry result event for that mode. The following table shows the GAP inquiry result type that will be returned for each inquiry mode:

Inquiry Mode	GAP Inquiry Result Event Type
imStandard	etInquiry_Entry_Result
imRSSI	etInquiry_With_RSSI_Entry_Result
imExtended	etExtended_Inquiry_Entry_Result

Prototype:

int BTPSAPI **GAP_Set_Inquiry_Mode**(unsigned int BluetoothStackID,
    GAP_Inquiry_Mode_t GAP_Inquiry_Mode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Inquiry_Mode	Specifies the inquiry mode to use. Possible values: imStandard imRSSI imExtended

Return:

Zero (0) if the inquiry mode was successfully set.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Inquiry_Mode

The following function is provided to allow a means to query the current inquiry mode being used by the Bluetooth device that is specified by the Bluetooth protocol stack that is associated with the specified Bluetooth stack ID. The second parameter to this function is a pointer to a variable that will receive the current inquiry mode of the local Bluetooth device. The second parameter must be valid (i.e. non-NULL) and upon successful completion of this function will contain the current inquiry mode of the local Bluetooth device. This function will return zero on success, or a negative return error code if there was an error. If this function returns success, then the GAP inquiry mode will contain the current inquiry mode value.

Note:

The inquiry mode dictates how the local device will actually perform inquiries (and more importantly, how the results will be returned). The following table shows supported modes and the corresponding GAP inquiry result event for that mode. The following table shows the GAP inquiry result type that will be returned for each inquiry mode:

Inquiry Mode	GAP Inquiry Result Event Type
imStandard	etInquiry_Entry_Result
imRSSI	etInquiry_With_RSSI_Entry_Result
imExtended	etExtended_Inquiry_Entry_Result

Prototype:

int BTPSAPI **GAP_Query_Inquiry_Mode**(unsigned int BluetoothStackID,
GAP_Inquiry_Mode_t *GAP_Inquiry_Mode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_Inquiry_Mode	Pointer to a buffer to receive the current inquiry mode setting.

Return:

Zero (0) if the current inquiry mode was successfully retrieved.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Remote_Device_Name

This function is provided to allow a mechanism to query the user-friendly Bluetooth device name of the specified remote Bluetooth device. This function accepts as input the Bluetooth device address of the remote Bluetooth device to query the name of and the GAP event callback information that is to be used when the remote device name process has completed. This function returns zero if successful, or a negative return error code if the remote name request was unable to be submitted. If this function returns success, then the caller will be notified via the specified callback when the remote name information has been determined (or there was an error). This function cannot be used to determine the user-friendly name of the local Bluetooth device. The GAP_Query_Local_Name function should be used to query the user-friendly name of the local Bluetooth device. Because this function is asynchronous in nature (specifying a remote device address), this function will notify the caller of the result via the specified callback. The caller is free to cancel the remote name request at any time by issuing the GAP_Cancel_Query_Remote_Name function and specifying the Bluetooth device address of the Bluetooth device that was specified in the original call to this function. It should be noted that when the callback is cancelled, the operation is attempted to be cancelled and the callback is cancelled (i.e. the GAP module still might perform the remote name request, but no callback is ever issued).

Prototype:

int BTPSAPI **GAP_Query_Remote_Device_Name**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Event_Callback_t GAP_Event_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device of which the name is to be retrieved.
GAP_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_CALLBACK_INFORMATION

BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etRemote_Name_Result

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Cancel_Query_Remote_Device_Name

This function is provided to cancel the future dispatching of a GAP remote name result event callback that was installed via a successful call to the GAP_Query_Remote_Device_Name function. This function attempts to cancel the querying of the remote device's name and it will ALWAYS cancel the installed callback notification. This function accepts as input the device address of the Bluetooth device that the previous call to GAP_Query_Remote_Device_Name was issued with. The BD_ADDR parameter MUST be valid, and cannot be the device address of the local Bluetooth device because the local device name query does not use the callback mechanism (nor this function).

Prototype:

int BTPSAPI **GAP_Cancel_Query_Remote_Device_Name**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device of which the remote name is being retrieved (which should be cancelled).

Return:

Zero (0) if the remote device name query was successfully cancelled.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DELETING_CALLBACK_INFORMATION

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Remote_Features

This function is provided to allow a mechanism to query the LMP features of the specified remote Bluetooth device. This function accepts as input the Bluetooth device address of the remote Bluetooth device to query the LMP features of and the GAP event callback information that is to be used when the query LMP features process has completed. This function returns zero if successful, or a negative return error code if the query LMP features request was unable to be submitted. If this function returns success, then the caller will be notified via the specified callback when the remote LMP features information has been determined (or there was an error). This function cannot be used to determine the LMP features of the local Bluetooth device. Because this function is asynchronous in nature (specifying a remote device address), this function will notify the caller of the result via the specified callback.

Prototype:

int BTPSAPI **GAP_Query_Remote_Features**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, GAP_Event_Callback_t GAP_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the remote device
GAP_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the remote LMP feature request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etRemote_Features_Result

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Remote_Version_Information

The following function is provided to allow a mechanism to Query the Version information of the specified Bluetooth device. This function accepts as input the Bluetooth Protocol Stack ID of the Bluetooth device that is to issue the Version Request, the Remote Bluetooth device address that references the Remote Bluetooth device, and the GAP Event Callback Information that is to be used when the Remote Version Information has been determined. This function returns zero if successful, or a negative return error code if the Remote Version Request was unable to be submitted. If this function returns success, then the caller will be notified via the specified callback when the requested information has been determined (or if there was an error). NOTE: Because this function is asynchronous in nature , this function will notify the caller of the result via the installed Callback.

This function is provided to allow a mechanism to query the version information of the specified remote Bluetooth device. This function accepts as input the Bluetooth device address of the remote Bluetooth device to query the version information of and the GAP event callback information that is to be used when the query version process has completed. This function returns zero if successful, or a negative return error code if the query version request was unable to be submitted. If this function returns success, then the caller will be notified via the specified callback when the remote version information has been determined (or there was an error). This function cannot be used to determine the version information of the local Bluetooth device. Because this function is asynchronous in nature (specifying a remote device address), this function will notify the caller of the result via the specified callback.

Prototype:

int BTPSAPI **GAP_Query_Remote_Version_Information**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, GAP_Event_Callback_t GAP_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the remote device
GAP_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the remote version information request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etRemote_Version_Information_Result

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Initiate_Bonding

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

Prototype:

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

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device of which to Bond.

GAP_Bonding_Type	Type of Bonbding to perform. Currently the following are defined: btGeneral btDedicated btDedicated_ManualDisconnect
GAP_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP Event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for all callbacks.

Return:

Zero (0) if the Request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_ADDING_CALLBACK_INFORMATION
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

etAuthentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Cancel_Bonding

This function is provided to allow a means to cancel a bonding process that was started previously via a successful call to the GAP_Initiate_Bonding function (either dedicated or general). This function accepts the Bluetooth device address of the remote Bluetooth device that the bonding procedure was initiated with. This function terminates the ACL connection and guaranteed that NO further GAP Event Callbacks will be issued after this function has completed (if successful).

Prototype:

int BTPSAPI **GAP_Cancel_Bonding**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device of which to cancel Bonding.

Return:

Zero (0) if the request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DELETING_CALLBACK_INFORMATION
BTPS_ERROR_INVALID_MODE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_End_Bonding

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

Prototype:

int BTPSAPI **GAP_End_Bonding**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device of which to end bonding.

Return:

Zero (0) if the request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_MODE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Local_BD_ADDR

This function is responsible for querying (and reporting) the device address of the local Bluetooth device. The second parameter is a pointer to a buffer that is to receive the device address of the local Bluetooth device. If this function is successful, the buffer that the BD_ADDR parameter points to will be filled with the device address read from the local Bluetooth device. If this function returns a negative value, then the device address of the local Bluetooth device was NOT able to be queried (error condition).

Prototype:

int BTPSAPI **GAP_Query_Local_BD_ADDR**(unsigned int BluetoothStackID,
    BD_ADDR_t *BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Pointer to memory in which to receive the local device address.

Return:

Zero (0) if the request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Set_Class_Of_Device

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

Prototype:

int BTPSAPI **GAP_Set_Class_Of_Device**(unsigned int BluetoothStackID,
Class_of_Device_t Class_of_Device);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Class_of_Device	Structure that holds the class of device information.

Return:

Zero (0) if the request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Class_Of_Device

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

Prototype:

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

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Class_of_Device	Pointer to a structure to receive the class of device information.

Return:

Zero (0) if the request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Set_Local_Device_Name

This function is provided to allow the changing of the device name of the local Bluetooth device. The Name parameter must be a pointer to a NULL terminated ASCII string of at most MAX_NAME_LENGTH (not counting the trailing NULL terminator). This function will return zero if the local device name was successfully changed, or a negative return error code if there was an error condition.

Note:

The format of the local device name is a NULL terminated UTF-8 string.

Prototype:

int BTPSAPI **GAP_Set_Local_Device_Name**(unsigned int BluetoothStackID, char *Name);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Name	Pointer to a buffer to containing the local device name.

Return:

Zero (0) if the Request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Local_Device_Name

This function is responsible for querying (and reporting) the user friendly name of the local Bluetooth device. The final parameters to this function specify the buffer and buffer length of the buffer that is to receive the local device name. The NameBufferLength parameter should be at least (MAX_NAME_LENGTH+1) to hold the maximum allowable device name (plus a single character to hold the NULL terminator). If this function is successful, this function returns zero, and the buffer that NameBuffer points to will be filled with a NULL terminated ASCII representation of the local device name. If this function returns a negative value, then the local device name was NOT able to be queried (error condition).

Note:

The format of the local device name is a NULL terminated UTF-8 string.

Prototype:

int BTPSAPI **GAP_Query_Local_Device_Name**(unsigned int BluetoothStackID,
    unsigned int NameBufferLength, char *NameBuffer);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
NameBufferLength	Size of vuffer to receive local device name.
NameBuffer	Pointer to a buffer to receive the local device name.

Return:

Zero (0) if the request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Disconnect_Link

The following function is provided to allow a means to terminate an existing connection (ACL) that was established by any BR/EDR Bluetooth protocol stack mechanism. This function accepts the Bluetooth device address of the remote Bluetooth device to disconnect. This function terminates any ACL connection that was established. If this function is successful, then the caller can expect each layer of the Bluetooth protocol stack that was dependent upon the specified connection to clean up correctly and dispatch all necessary disconnection callbacks.

Note:

This function will only disconnect BR/EDR connections. It will not disconnect Bluetooth LE connections.

Prototype:

int BTPSAPI **GAP_Disconnect_Link**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device of which to terminate the link.

Return:

Zero (0) if the Request was successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Connection_Handle

The following function is provided to allow a means to query the ACL connection handle of a connection to a remote Bluetooth device. If a connection exists to the remote device specified, the ACL connection handle is returned in the buffer passed to this function. This function will return zero on success, or a negative return error code if there was an error. If this function returns success, then the Connection_Handle variable will contain the current ACL connection handle for the connection to the specified Bluetooth device address.

Note:

This function is only for BR/EDR connections. This function will NOT return connection handles for Bluetooth LE connections.

Prototype:

int BTPSAPI **GAP_Query_Connection_Handle**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Word_t *Connection_Handle);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device of which to query the connection handle.
Connection_Handle	Pointer to a variable that will receive the connection handle associated with the specified Bluetooth device address.

Return:

Zero (0) if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_NOT_CONNECTED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Query_Local_Out_Of_Band_Data

The following function is provided for Local devices that support Out of Band (OOB) pairing using a technology such as near field communications (NFC). It is used to obtain the simple pairing hash (C) and the simple pairing randomizer (R) which are intended to be transferred to a remote device using OOB.

Note:

A new value for C and R are created each time this call is made. Each OOB transfer will have unique C and R values so that after each OOB transfer this function should be called to obtain a new set for the next OOB transfer.

These values are not kept on device reset or device power off in which case a call to this function should be invoked during one time initialization.

Prototype:

int BTPSAPI **GAP_Query_Local_Out_Of_Band_Data**(unsigned int BluetoothStackID,
GAP_Out_Of_Band_Data_t *OutOfBandData);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OutOfBandData	Pointer to a buffer that is to receive the Out Of Band Data that the local device has generated.

Return:

Zero (0) if the OOB data was successfully retreived.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Refresh_Encryption_Key

The following function is provided to allow the host to cause the Controller to refresh the encryption by pausing the current encryption and then resuming the encryption.

Note:

This function is asynchronous in nature and will notify the caller of the completion of a refresh via the specified callback. This operation cannot be cancelled (other than a disconnect occurring).

Prototype:

int BTPSAPI **GAP_Refresh_Encryption_Key**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_Event_Callback_t GAP_Event_Callback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the remote device
GAP_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP Event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for all callbacks.

Return:

Zero (0) if the Refresh encryption process was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Read_Extended_Inquiry_Information

The following function is provided to allow the local host to read the Extended Inquiry Response Information currently stored in the controller. This is the data that the controller will return when it returns an extended inquiry response to a remote device. This function will return zero if successful, or a negative return error code if there was an error condition. If this function returns success, then the Extended_Inquiry_Response_Data member will be filled in with the correct data.

Note:

The GAP_Parse_Extended_Inquiry_Response_Data() function can be used to parse the Extended Inquiry Response Data for easy parsing (if required).

Prototype:

int BTPSAPI **GAP_Read_Extended_Inquiry_Information**(unsigned int BluetoothStackID,
    Byte_t *FEC_Required,
    Extended_Inquiry_Response_Data_t *Extended_Inquiry_Response_Data);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FEC_Required	Specifies whether FEC is required or not.
Extended_Inquiry_Response_Data	Buffer that is to receive the actual Extended Inquiry Response Data that the local Bluetooth device is currently using. This buffer must be at least 240 bytes in length.

Return:

Zero (0) if the Extended Inquiry Response data was successfully read. The Extended_Inquiry_Response_Data buffer will be populated with the Extended Inquiry Response data.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Write_Extended_Inquiry_Information

The following function is provided to allow the local host to write the extended inquiry information to be stored in the controller. This is the data that the controller will return when it returns an extended inquiry response to a remote device. This function will return zero if successful, or a negative return error code if there was an error condition.

Prototype:

int BTPSAPI **GAP_Write_Extended_Inquiry_Information**(
unsigned int BluetoothStackID, Byte_t FEC_Required,
Extended_Inquiry_Response_Data_t *Extended_Inquiry_Response_Data);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FEC_Required	Specifies whether FEC is required or not.
Extended_Inquiry_Response_Data	Buffer that contains the actual Extended Inquiry Response Data that the local Bluetooth device is to begin using. This buffer must be at least 240 bytes in length.

Return:

Zero (0) if the Extended Inquiry Response data was successfully written.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_HCI_ERROR

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_Convert_Extended_Inquiry_Response_Data

The following function is provided to allow a simple mechanism to convert a GAP_Extended_Inquiry_Response_Data_t to the raw Extended_Inquiry_Response_Data_t. This second parameter *MUST* point to the maximum sized Extended Inquiry Response Buffer size (EXTENDED_INQUIRY_RESPONSE_DATA_SIZE). This function will return the number of successfully converted items (zero or more), or a negative error code if there was an error.

Note:

This function will populate the entire Extended_Inquiry_Response_Data_t buffer (all EXTENDED_INQUIRY_RESPONSE_DATA_SIZE bytes). If the specified information is smaller than the full Extended Inquiry Response Data size, the resulting buffer will be padded with zeros.

Prototype:

int BTPSAPI **GAP_Convert_Extended_Inquiry_Response_Data**(
    GAP_Extended_Inquiry_Response_Data_t *GAP_Extended_Inquiry_Response_Data,
    Extended_Inquiry_Response_Data_t *Extended_Inquiry_Response_Data);

Parameters:

GAP_Extended_Inquiry_Response_Data	Pointer to the Parsed Extended Inquiry data that is to be converted.
Extended_Inquiry_Response_Data	Buffer that is to receive the actual Extended Inquiry Response Data from the parsed Extended Inquiry Data. This buffer must be at least 240 bytes in length.

Return:

Non-negative if successful. This value represents the number of valid Extended Inquiry Response data fields that were successfully parsed.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

GAP_Parse_Extended_Inquiry_Response_Data

The following function is a utility function that exists to parse the specified Extended_Inquiry_Response_Data_t information into GAP_Extended_Inquiry_Response_Data_t structure (for ease of parsing). This function accepts as the first parameter the Extended_Inquiry_Response_Data_t to parse, followed by a pointer to a GAP_Extended_Inquiry_Response_Data_t that will receive the Parsed data. The final parameter, if specified, *MUST* specify the maximum number of entries that can be parsed, as well as the actual Entry array to parse the entries into (on input).

Note:

If this function is called with a NULL passed as the final parameter, then, this function will simply calculate the number of Extended Inquiry Data Information Entries that will be required to hold the parsed information. If the final parameter is NOT NULL then it *MUST* contain the maximum number of entries that can be supported (specified via the Number_Data_Entries member) and the Data_Entries member must point to memory that contains (at least) that many members).

This function will return BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE if there was not enough Data Entries specified (via the Number_Data_Entries member) to satisfy the parsing of the actual Extended Inquiry Response Data.

Prototype:

int BTPSAPI **GAP_Parse_Extended_Inquiry_Response_Data**(
    Extended_Inquiry_Response_Data_t *Extended_Inquiry_Response_Data,
    GAP_Extended_Inquiry_Response_Data_t *GAP_Extended_Inquiry_Response_Data);

Parameters:

Extended_Inquiry_Response_Data	Buffer that contains the actual Extended Inquiry Response Data that is to be parsed. This buffer must be at least 240 bytes in length.
GAP_Extended_Inquiry_Response_Data	Pointer to the Parsed Extended Inquiry data that has been parsed.

Return:

Non-negative if successful. This value represents the number of valid Extended Inquiry Response data fields that were successfully parsed.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

GAP_LE_Create_Connection

The following function is provided to allow the local host the ability to create a connection to a remote device using the Bluetooth LE radio. The connection process is asynchronous in nature and the caller will be notified via the GAP LE event callback function (specified in this function) when the connection completes. This function will return zero if successful, or a negative return error code if there was an error condition.

Note:

The Bluetooth LE connection process is not like a Bluetooth BR/EDR connection. Once a connection request is submitted (via this function), it will stay active and will not time out. The connection process is over when either the connection is made OR the caller calls the GAP_LE_Cancel_Create_Connection() function.

This function allows the use of the Bluetooth white-list and can be used to specify a specific set of devices to connect to.

Prototype:

int BTPSAPI **GAP_LE_Create_Connection**(unsigned int BluetoothStackID,
unsigned int ScanInterval, unsigned int ScanWindow,
GAP_LE_Filter_Policy_t InitatorFilterPolicy,
GAP_LE_Address_Type_t RemoteAddressType, BD_ADDR_t *RemoteDevice,
GAP_LE_Address_Type_t LocalAddressType,
GAP_LE_Connection_Parameters_t *ConnectionParameters, GAP_LE_Event_Callback_t GAP_LE_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
ScanInterval	Scan interval to use when scanning for the device(s) to connect. This value is specified in milli-seconds. This value must be within the range: MINIMUM_LE_SCAN_INTERVAL MAXIMUM_LE_SCAN_INTERVAL
ScanWindow	Scan window to use when scanning for the device(s) to connect. This value is specified in milli-seconds. This value must be within the range: MINIMUM_LE_SCAN_WINDOW MINIMUM_LE_SCAN_WINDOW
InitatorFilterPolicy	Filter policy to apply when scanning. Valid values are: fpNoFilter fpWhiteList If the white-list filter is specified then the remote device address (and address type) are ignored.
RemoteAddressType	Specifies the type of the remote device address to connect with (if not using white-list filter). Valid values are: latPublic latRandom
RemoteDevice	Specifies the remote device address to connect with. This value is required if no filter is specified as the filter policy.
LocalAddressType	Specifies the type of the address the local device is to use when connecting to the remote device. Valid values are: latPublic latRandom

ConnectionParameters	Specifies the parameters to use when actually establishing the connection to the remote device. This structure is defined as follows:

```
typedef struct
{
   Word_t Connection_Interval_Min;
   Word_t Connection_Interval_Max;
   Word_t Slave_Latency;
   Word_t Supervision_Timeout;
   Word_t Minimum_Connection_Length;
   Word_t Maximum_Connection_Length;
} GAP_LE_Connection_Parameters_t;
```

Note that ALL parameters are specified in milli-seconds except the Slave_Latency parameter which is specified in connection events.

where, Connection_Interval_Min is specified in milli-seconds and must be between:

MINIMUM_MINIMUM_CONNECTION_INTERVAL
MAXIMUM_MINIMUM_CONNECTION_INTERVAL

Note the default minimum connection interval is defined by the constant:

DEFAULT_MINIMUM_CONNECTION_INTERVAL

and, Connection_Interval_Max is specified in milli-seconds and must be between:

MINIMUM_MAXIMUM_CONNECTION_INTERVAL
MAXIMUM_MAXIMUM_CONNECTION_INTERVAL

Note the default maximum connection interval is defined by the constant:

DEFAULT_MAXIMUM_CONNECTION_INTERVAL

and, Slave_Latency is specified in number of connection events and must be between:

MINIMUM_SLAVE_LATENCY
MAXIMUM_SLAVE_LATENCY

Note the default slave latency is defined by the constant:

DEFAULT_SLAVE_LATENCY

and, Supervision_Timeout is specified in milli-seconds and must be between:

MINIMUM_LINK_SUPERVISION_TIMEOUT
MAXIMUM_LINK_SUPERVISION_TIMEOUT

Note the default link supervision timeout is defined by the constant:

	DEFAULT_LINK_SUPERVISION_TIMEOUT and, the Minimum_Connection_Length and Maximum_Connection_Length parameters are specified in milli-seconds and represent the expected minimum and maximum connection events for the connection. These values must be between: MINIMUM_CONNECTION_EVENT_LENGTH MAXIMUM_CONNECTION_EVENT_LENGTH
GAP_LE_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP LE Event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the callback.

Return:

Zero (0) if the connection request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_CONNECTION_PARAMETERS
BTPS_ERROR_RANDOM_ADDRESS_IN_USE
BTPS_ERROR_CREATE_CONNECTION_OUTSTANDING
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Connection_Complete

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Cancel_Create_Connection

The following function is provided to allow the local host the ability to cancel (end) a connection process. This function does not disconnect a connected device, it merely stops the connection process (scanning and connecting). This function will return zero if successful, or a negative return error code if there was an error condition.

Note:

The Bluetooth LE connection process is not like a Bluetooth BR/EDR connection. Once a connection request is submitted (via the GAP_LE_Create_Connection function), it will stay active and will not time out. The connection process is over when either the connection is made OR the caller calls this function.

Prototype:

int BTPSAPI **GAP_LE_Cancel_Create_Connection**(unsigned int BluetoothStackID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

Return:

Zero (0) if the connection process was successfully cancelled.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Disconnect

The following function is provided to allow the local host the ability to disconnect a currently connected LE device. This function will return zero if successful, or a negative return error code if there was an error condition.

Prototype:

int BTPSAPI **GAP_LE_Disconnect**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Specifies the remote device address of the currently connected device to disconnect.

Return:

Zero (0) if the disconnection request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Disconnection_Complete

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Read_Remote_Features

The following function is provided to allow the local host the ability to determine the remote LMP features of a connected remote device. This function will return zero if successful, or a negative return error code if there was an error condition.

Note:

This function will not create an LE ACL connection to the specified device. The LE ACL connection to the specified remote device must already exist before calling this function.

Prototype:

int BTPSAPI **GAP_LE_Read_Remote_Features**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Specifies the remote device address of the currently connected device to query the remote LMP features.

Return:

Zero (0) if the read remote features request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED

BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Remote_Features_Result

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Perform_Scan

The following function is provided to allow the local host the ability to begin an LE scanning procedure. This procedure is similar in concept to the inquiry procedure in Bluetooth BR/EDR in that it can be used to discover devices that have been instructed to advertise. This function will return zero if successful, or a negative return error code if there was an error condition.

Note:

There can only be a single scan being performed at any given time. The caller must call the GAP_LE_Cancel_Scan() function to stop a currently active scan process.

The scan interval and scan window parameters are specified in milli-seconds and MUST satisfy the following equation:

Scan Window <= Scan Interval

Note that if the scan window equals the scan interval than continuous scanning is specified.

Prototype:

int BTPSAPI **GAP_LE_Perform_Scan**(unsigned int BluetoothStackID,
GAP_LE_Scan_Type_t ScanType, unsigned int ScanInterval,
unsigned int ScanWindow, GAP_LE_Address_Type_t LocalAddressType,
GAP_LE_Filter_Policy_t FilterPolicy, Boolean_t FilterDuplicates,
GAP_LE_Event_Callback_t GAP_LE_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
ScanType	Specifies the type of scan to perform. This value must be one of the following: stPassive stActive

ScanInterval	Specifies interval to use while scanning. This value must be be between (and satisfy the equation listed above): MINIMUM_LE_SCAN_INTERVAL MAXIMUM_LE_SCAN_INTERVAL
ScanWindow	Specifies window to use while scanning. This value must be be between (and satisfy the equation listed above): MINIMUM_LE_SCAN_WINDOW MAXIMUM_LE_SCAN_WINDOW
LocalAddressType	Specifies the type of the address the local device is to use when scanning. Valid values are: latPublic latRandom
FilterPolicy	Filter policy to apply when scanning. Valid values are: fpNoFilter fpWhiteList
FilterDuplicates	Specifies whether or not the host controller is to filter duplicate scan responses.
GAP_LE_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP LE Event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the callback.

Return:

Zero (0) if the scan procedure was successfully started.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_SCAN_ACTIVE
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Advertising_Report

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Cancel_Scan

The following function is provided to allow the local host the ability to cancel (stop) an on-going scan procedure. This function will return zero if successful, or a negative return error code if there was an error condition.

Prototype:

int BTPSAPI **GAP_LE_Cancel_Scan**(unsigned int BluetoothStackID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

Return:

Zero (0) if the scan procedure was successfully cancelled.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Set_Advertising_Data

The following function is provided to allow the local host the ability to set the advertising data that is used during the advertising procedure (started via the GAP_LE_Advertising_Enable function). This function will return zero if successful, or a negative return error code if there was an error condition.

Note:

Advertising data consists of zero or more tuples that consist of:

- Type (byte)
- Length (byte)
- Data (zero or more bytes)

Also note that the advertising data itself is a fixed length. If the list of the tuples of the advertising data is not long enough to fill the required advertising length then bytes containing the binary value zero (0x00) should be used to pad the data (until the end of the required advertising data size).

Prototype:

int BTPSAPI **GAP_LE_Set_Advertising_Data**(unsigned int BluetoothStackID,
unsigned int Length, Advertising_Data_t *Advertising_Data);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Length	Number of significant advertising data bytes contained in the advertising data.
Advertising_Data	Pointer to a buffer that contains the advertising data. This buffer must be at least: ADVERTISING_DATA_MAXIMUM_SIZE bytes long. Note that the length parameter specifies the actual number of bytes that are valid. The remaining bytes should be padded with zero's.

Return:

Zero (0) if the advertising data was successfully set.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Convert_Advertising_Data

The following function is provided to allow a simple mechanism to convert a GAP_LE_Advertising_Data_t to the raw Advertising_Data_t format (packed format). This second parameter *MUST* point to the maximum sized advertising data buffer size (ADVERTISING_DATA_SIZE). This function will return the number of successfully converted items (zero or more), or a negative error code if there was an error.

Note:

This function will populate the entire Advertising_Data_t buffer (all ADVERTISING_DATA_SIZE bytes). If the specified information is smaller than the full advertising data size, the resulting buffer will be padded with binary zero bytes (0x00).

Prototype:

int BTPSAPI **GAP_LE_Convert_Advertising_Data**(
    GAP_LE_Advertising_Data_t *GAP_LE_Advertising_Data,
    Advertising_Data_t *Advertising_Data);

Parameters:

GAP_LE_Advertising_Data	Pointer to the parsed advertising data that is to be converted.
Advertising_Data	Buffer that is to receive the actual advertising data from the parsed advertising data. This buffer must be at least: ADVERTISING_DATA_SIZE bytes in length.

Return:

Non-negative if successful. This value represents the number of valid advertising data fields that were successfully parsed.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

GAP_LE_Parse_Advertising_Data

The following function is a utility function that exists to parse the specified Advertising_Data_t information into GAP_LE_Advertising_Data_t structure (for ease of parsing). This function accepts as the first parameter the Advertising_Data_t to parse, followed by a pointer to a GAP_LE_Advertising_Data_t that will receive the parsed data. The final parameter, if specified, *MUST* specify the maximum number of entries that can be parsed, as well as the actual entry array to parse the entries into (on input).

Note:

If this function is called with a NULL passed as the final parameter, then, this function will simply calculate the number of advertising data entries that will be required to hold the parsed information. If the final parameter is NOT NULL then it *MUST* contain the maximum number of entries that can be supported (specified via the Number_Data_Entries member) and the Data_Entries member must point to memory that contains (at least) that many members).

This function will return BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE if there was not enough data entries specified (via the Number_Data_Entries member) to satisfy the parsing of the actual advertising data.

Prototype:

int BTPSAPI **GAP_LE_Parse_Advertising_Data(**Advertising_Data_t *Advertising_Data, GAP_LE_Advertising_Data_t *GAP_LE_Advertising_Data);

Parameters:

Advertising_Data	Buffer that contains the actual advertising data that is to be parsed. This buffer must be at least: ADVERTISING_DATA_SIZE bytes long. Note that if the advertising occupies less data bytes than the data should be padded with zero bytes (0x00).
GAP_LE_Advertising_Data	Pointer to the parsed advertising data that has been parsed. Note that if this parameter is not NULL then the Number_Data_Entries member must contain the number of data entries that the Data_Entries member points to (to receive the parsed data information.

Return:

Non-negative if successful. This value represents the number of valid advertising data fields that were successfully parsed.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

GAP_LE_Set_Scan_Response_Data

The following function is provided to allow the local host the ability to set the scan response data that is used during the advertising procedure (started via the GAP_LE_Advertising_Enable function). This function will return zero if successful, or a negative return error code if there was an error condition.

Note:

Scan response data consists of zero or more tuples that consist of:

- Type (byte)

- Length (byte)
- Data (zero or more bytes)

Also note that the scan response data itself is a fixed length. If the list of the tuples of the response data is not long enough to fill the required response length then bytes containing the binary value zero (0x00) should be used to pad the data (until the end of the required response data size).

Prototype:

int BTPSAPI **GAP_LE_Set_Scan_Response_Data**(unsigned int BluetoothStackID, unsigned int Length, Scan_Response_Data_t *Scan_Response_Data);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Length	Number of significant advertising data bytes contained in the advertising data.
Scan_Response_Data	Pointer to a buffer that contains the response data. This buffer must be at least: SCAN_RESPONSE_DATA_MAXIMUM_SIZE bytes long. Note that the length parameter specifies the actual number of bytes that are valid. The remaining bytes should be padded with zero's.

Return:

Zero (0) if the scan response data was successfully set.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Convert_Scan_Response_Data

The following function is provided to allow a simple mechanism to convert a GAP_LE_Advertising_Data_t to the raw Scan_Response_Data_t format (packed format). This second parameter *MUST* point to the maximum sized scan response data buffer size (SCAN_RESPONSE_DATA_SIZE). This function will return the number of successfully converted items (zero or more), or a negative error code if there was an error.

Note:

This function will populate the entire Scan_Response_Data_t buffer (all SCAN_RESPONSE_DATA_SIZE bytes). If the specified information is smaller than the full scan response data size, the resulting buffer will be padded with binary zero bytes (0x00).

Prototype:

int BTPSAPI **GAP_LE_Convert_Scan_Response_Data**(
    GAP_LE_Advertising_Data_t *GAP_LE_Advertising_Data,
    Scan_Response_Data_t *Scan_Response_Data);

Parameters:

GAP_LE_Advertising_Data	Pointer to the parsed advertising data that is to be converted.
Scan_Response_Data	Buffer that is to receive the actual scan response data from the parsed advertising data. This buffer must be at least: SCAN_RESPONSE_DATA_SIZE bytes in length.

Return:

Non-negative if successful. This value represents the number of valid advertising data fields that were successfully parsed.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

GAP_LE_Parse_Scan_Response_Data

The following function is a utility function that exists to parse the specified Scan_Response_Data_t information into GAP_LE_Advertising_Data_t structure (for ease of parsing). This function accepts as the first parameter the Scan_Response_Data_t to parse, followed by a pointer to a GAP_LE_Advertising_Data_t that will receive the parsed data. The final parameter, if specified, *MUST* specify the maximum number of entries that can be parsed, as well as the actual entry array to parse the entries into (on input).

Note:

If this function is called with a NULL passed as the final parameter, then, this function will simply calculate the number of advertising data entries that will be required to hold the parsed information. If the final parameter is NOT NULL then it *MUST* contain the maximum number of entries that can be supported (specified via the Number_Data_Entries member) and the Data_Entries member must point to memory that contains (at least) that many members).

This function will return BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE if there was not enough data entries specified (via the Number_Data_Entries member) to satisfy the parsing of the actual scan response data.

Prototype:

int BTPSAPI **GAP_LE_Parse_Scan_Response_Data(**
    Scan_Response_Data_t *Scan_Response_Data,
    GAP_LE_Advertising_Data_t *GAP_LE_Advertising_Data);

Parameters:

Scan_Response_Data	Buffer that contains the actual scan response data that is to be parsed. This buffer must be at least: SCAN_RESPONSE_DATA_SIZE bytes long. Note that if the scan rsponse occupies less data bytes than the data should be padded with zero bytes (0x00).
GAP_LE_Advertising_Data	Pointer to the parsed scan response data that has been parsed. Note that if this parameter is not NULL then the Number_Data_Entries member must contain the number of data entries that the Data_Entries member points to (to receive the parsed data information.

Return:

Non-negative if successful. This value represents the number of valid advertising data fields that were successfully parsed.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_BUFFER_SPACE

Possible Events:

GAP_LE_Advertising_Enable

The following function is provided to allow the local host the ability to begin an advertising procedure. An advertising procedure is required to allow a remote Bluetooth LE device to connect with the local device. The connectability mode and parameters are set via the connectability parameters passed to this function. This function also accepts the advertising parameters to apply while advertising. This function also accepts callback information that will be used to inform the caller (asynchronously) when a remote LE device (master) connects to the local LE device (slave). This function will return zero if successful, or a negative return error code if there was an error condition.

Note:

The advertising data and scan response data should be set before this function is called to enable advertising.

Prototype:

```
int BTPSAPI GAP_LE_Advertising_Enable(unsigned int BluetoothStackID,
    Boolean_t EnableScanResponse,
    GAP_LE_Advertising_Parameters_t *GAP_LE_Advertising_Parameters,
    GAP_LE_Connectability_Parameters_t *GAP_LE_Connectability_Parameters,
    GAP_LE_Event_Callback_t GAP_LE_Event_Callback,
    unsigned long CallbackParameter);
```

Parameters:

BluetoothStackID[1]
: Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

EnableScanResponse
: Flag which specifies whether or not the device should send scan response data in response to a scan request.

GAP_LE_Advertising_Parameters
: Pointer to advertising parameters that control how the advertising is performed. This structure is defined as follows:

```
typedef struct
{
   Word_t                  Advertising_Interval_Min;
   Word_t                  Advertising_Interval_Max;
   Byte_t                  Advertising_Channel_Map;
   GAP_LE_Filter_Policy_t  Scan_Request_Filter;
   GAP_LE_Filter_Policy_t  Connect_Request_Filter;
} GAP_LE_Advertising_Parameters_t;
```

where, Advertising_Interval_Min is specified in milli-seconds and must be between:

MINIMUM_ADVERTISING_INTERVAL
MAXIMUM_ADVERTISING_INTERVAL

and, Advertising_Interval_Max is specified in milli-seconds and must be between:

MINIMUM_ADVERTISING_INTERVAL
MAXIMUM_ADVERTISING_INTERVAL

and, Advertising_Channel_Map is a bit-mask that consists of one or more of the following values:

GAP_LE_ADVERTISING_CHANNEL_MAP_USE_CHANNEL_37
GAP_LE_ADVERTISING_CHANNEL_MAP_USE_CHANNEL_38
GAP_LE_ADVERTISING_CHANNEL_MAP_USE_CHANNEL_39

alternately, to use all channels, the following constant can be used:

GAP_LE_ADVERTISING_CHANNEL_MAP_
USE_ALL_CHANNELS

and, Scan_Request_Filter specifies the filter to apply governing how the device is to respond to scan requests (if enabled). This is one of the following values:

fpNoFilter
fpWhiteList

and, Connect_Request_Filter specifies the filter to apply governing how the device is to respond to connection requests. This is one of the following values:

fpNoFilter
fpWhiteList

GAP_LE_Connectability_Parameters	Specifies the connectability parameters to use while advertising. This structure is defined as follows:

```
typedef struct
{
   GAP_LE_Connectability_Mode_t
                Connectability_Mode;
   GAP_LE_Address_Type_t
                Own_Address_Type;
   GAP_LE_Address_Type_t
                Direct_Address_Type;
   BD_ADDR_t
                Direct_Address;
} GAP_LE_Connectability_Parameters_t;
```

where, Connectability_Mode defines the actual connectability mode. This is one of the following values:

lcmNonConnectable
lcmConnectable
lcmDirectConnectable

and, Own_Address_Type specifies the address to use for the connection. This is one of the following values:

latPublic
latRandom

and, Direct_Address_Type specifies the address to use when lcmDirectConnectable is used (it is not used

	for the other connectability modes). This is one of the following values: latPublic latRandom and, Direct_Address specifies the direct address to use when lcmDirectConnectable is used (it is not used for the other connectability modes).
GAP_LE_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP LE event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the callback.

Return:

Zero (0) if the scan procedure was successfully cancelled.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Connection_Complete

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Advertising_Disable

The following function is provided to allow the local host the ability to cancel (stop) an on-going advertising procedure. This function will return zero if successful, or a negative return error code if there was an error condition.

Prototype:

int BTPSAPI **GAP_LE_Advertising_Disable**(unsigned int BluetoothStackID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

Return:

Zero (0) if the advertising procedure was successfully stopped.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Generate_Non_Resolvable_Address

The following function is provided to allow the local host the ability to generate a non resolvable address. The output of this function can then be used as a random address for connection purposes. This function will return zero if successful, or a negative return error code if there was an error condition.

Notes:

If this function is successful, the address that is generated can be passed to the GAP_LE_Set_Random_Address function and used by the local device.

Prototype:

int BTPSAPI **GAP_LE_Generate_Non_Resolvable_Address**(
unsigned int BluetoothStackID, BD_ADDR_t *NonResolvableAddress_Result);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
NonResolvableAddress_Result	Buffer that will receive the generated non resolvable address upon successful execution of this function.

Return:

Zero (0) if the a non resolvable address was successfully generated.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Generate_Static_Address

The following function is provided to allow the local host the ability to generate a static private address. The output of this function can then be used as a random address for connection purposes. This function will return zero if successful, or a negative return error code if there was an error condition.

Notes:

The Bluetooth Specification has defined that a static private address shall only change once per power cycle. It is the requirement of the application that this function is only used to generate a new static address once per power cycle

If this function is successful, the address that is generated can be passed to the GAP_LE_Set_Random_Address function and used by the local device.

Prototype:

int BTPSAPI **GAP_LE_Generate_Static_Address**(unsigned int BluetoothStackID,
BD_ADDR_t *StaticAddress_Result);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
StaticAddress_Result	Buffer that will receive the generated static address upon successful execution of this function.

Return:

Zero (0) if the a non resolvable address was successfully generated.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE

BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Generate_Resolvable_Address

The following function is provided to allow the local host the ability to generate a resolvable address. The output of this function can then be used as a random address for connection purposes. This function will return zero if successful, or a negative return error code if there was an error condition.

Notes:

If this function is successful, the address that is generated can be passed to the GAP_LE_Set_Random_Address function and used by the local device.

Prototype:

int BTPSAPI **GAP_LE_Generate_Resolvable_Address**(unsigned int BluetoothStackID, Encryption_Key_t *IRK, BD_ADDR_t *ResolvableAddress_Result);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
IRK	Identity resolving key (IRK) that is used to generate the resolvable address.
ResolvableAddress_Result	Buffer that will receive a generated resolvable address upon successful execution of this function.

Return:

Zero (0) if the a resolvable address was successfully generated.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Resolve_Address

The following function is provided to allow the local host the ability to check to see if a specified Identity Resolving Key (IRK) and a specified resolvable address can be resolved. This function will return a BOOLEAN TRUE value if the address was able to be resolved or FALSE if it was not.

Notes:

If this function is successful, the address that is generated can be passed to the GAP_LE_Set_Random_Address function and used by the local device.

Prototype:

int BTPSAPI **GAP_LE_Resolve_Address**(unsigned int BluetoothStackID,
Encryption_Key_t *IRK, BD_ADDR_t ResolvableAddress);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
IRK	Identity resolving key (IRK) that is used to resolve the resolvable address.
ResolvableAddress	Bluetooth address that represents the resolvable address that is attempting to be resolved (using the specified IRK).

Return:

TRUE if the a resolvable address was successfully resolved.

FALSE if the address was not able to be resolved.

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Set_Random_Address

The following function is provided to allow the local host the ability to set the random address used by the local device. This function will return zero if successful, or a negative return error code if there was an error condition.

Prototype:

int BTPSAPI **GAP_LE_Set_Random_Address**(unsigned int BluetoothStackID,
    BD_ADDR_t RandomAddress);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
RandomAddress	Actual random address value to set in the local device.

Return:

Zero (0) if the a resolvable address was successfully generated.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_RANDOM_ADDRESS_IN_USE
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
    SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Add_Device_To_White_List

The following function is provided to allow the local host the ability to add one (or more) devices to the white list maintained by the local device. This function will attempt to add as many devices as possible (from the specified list) and will return the number of devices added. The GAP_LE_Read_White_List_Size function can be used to determine how many devices the local device supports in the white list (simultaneously). This function will return zero if successful, or a negative return error code if there was an error condition.

Notes:

The final parameter will contain, on successful completion of this function, the total number of devices that were written to the device white list.

The white list cannot be changed while a scan or connection is in progress. If this function is called while a scan or connection is active, the following error code will be returned:

GAP_LE_ERROR_WHITE_LIST_IN_USE

Prototype:

int BTPSAPI **GAP_LE_Add_Device_To_White_List**(unsigned int BluetoothStackID, unsigned int DeviceCount, GAP_LE_White_List_Entry_t *WhiteListEntries, unsigned int *AddedDeviceCount);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
DeviceCount	Total number of device list entries that are pointed to by the WhiteListEntries buffer.
WhiteListEntries	Buffer that contains one or more individual white list device entries to write to the local device. This buffer must point to (at least) DeviceCount entries. The structure of an individual white list entry is as follows:

```
typedef struct
{
  GAP_LE_Address_Type_t     Address_Type;
  BD_ADDR_t                 Address;
} GAP_LE_White_List_Entry_t;
```

where, Address_Type defines the type of the address that is represented by this entry. This is one of the following values:

latPublic
latRandom

and Address is the actual device address of the device to write to the white list.

AddedDeviceCount	Upon successful execution of this function contains the total number of white list entries that were successfully written to the device white list.

Return:

Zero (0) if at least one device was written to the white list.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_RANDOM_ADDRESS_IN_USE
GAP_LE_ERROR_WHITE_LIST_IN_USE
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Remove_Device_From_White_List

The following function is provided to allow the local host the ability to remove one (or more) devices from the white list maintained by the local device. This function will attempt to delete as many devices as possible (from the specified list) and will return the number of devices deleted. The GAP_LE_Read_White_List_Size function can be used to determine how many devices the local device supports in the white list (simultaneously). This function will return zero if successful, or a negative return error code if there was an error condition.

Notes:

If the device count parameter is specified as zero then the entire white list will be deleted. In this case the final parameter will be set to zero and NOT the number of devices that were deleted.

The final parameter will contain, on successful completion of this function, the total number of devices that were deleted from the device white list.

The white list cannot be changed while a scan or connection is in progress. If this function is called while a scan or connection is active, the following error code will be returned:

GAP_LE_ERROR_WHITE_LIST_IN_USE

Prototype:

int BTPSAPI **GAP_LE_Remove_Device_From_White_List**(
unsigned int BluetoothStackID, unsigned int DeviceCount, GAP_LE_White_List_Entry_t *WhiteListEntries,
unsigned int *RemovedDeviceCount);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
DeviceCount	Total number of device list entries that are pointed to by the WhiteListEntries buffer. If this value is specified as zero then the next parameter is ignored and all devices are removed from the white list.
WhiteListEntries	Buffer that contains one or more individual white list device entries to remove from the local device. This buffer must point to (at least) DeviceCount entries. The structure of an individual white list entry is as follows:

```
typedef struct
{
      GAP_LE_Address_Type_t    Address_Type;
      BD_ADDR_t                Address;
   } GAP_LE_White_List_Entry_t;
```

where, Address_Type defines the type of the address that is represented by this entry. This is one of the following values:

latPublic
latRandom

and Address is the actual device address of the device to remove from the white list.

AddedDeviceCount	Upon successful execution of this function contains the total number of white list entries that were successfully removed from the device white list.

Return:

Zero (0) if at least one device was removed from the white list.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
GAP_LE_ERROR_WHITE_LIST_IN_USE
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Read_White_List_Size

The following function is provided to allow the local host the ability to determine the total number of devices that can be be present in the white list (simultaneously) on the local device. This function will return zero if successful, or a negative return error code if there was an error condition.

Prototype:

int BTPSAPI **GAP_LE_Read_White_List_Size**(unsigned int BluetoothStackID,
    unsigned int *WhiteListSize);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

WhiteListSize	Total number of device list entries are supported by the local device. This value is the number of entried NOT the number of white list entry buffer size in bytes.

Return:

Zero (0) if the white list size was able to be successfully retrieved.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Set_Pairability_Mode

The following function is provided to allow the local host the ability to change the pairability mode used by the local host. This function will return zero if successful, or a negative return error code if there was an error condition.

Prototype:

int BTPSAPI **GAP_LE_Set_Pairability_Mode**(unsigned int BluetoothStackID,
GAP_LE_Pairability_Mode_t PairableMode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
PairableMode	Pairability mode to set. This value is one of: lpmNonPairableMode lpmPairableMode

Return:

Zero (0) if the pairability mode was successfully set.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED

BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Register_Remote_Authentication

This function is provided to allow a means to register a GAP LE event callback to accept remote authentication requests. This function accepts as input the GAP LE event callback information to register. It should be noted that ONLY ONE remote authentication callback can be installed per Bluetooth device. The caller can un-register the remote authentication callback that was registered with this function (if successful) by calling the GAP_LE_Un_Register_Remote_Authentication function.

Note:

A remote authentication event is defined as an authentication event that was not requested by the local device (i.e. a pairing or authentication request issued from a remote device to the local device).

Prototype:

int BTPSAPI **GAP_LE_Register_Remote_Authentication**(unsigned int BluetoothStackID,
GAP_LE_Event_Callback_t GAP_LE_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
GAP_LE_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP LE event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the remote authentication callback was successfully registered.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED

BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Authentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Un_Register_Remote_Authentication

This function is provided to allow a mechanism to un-register a previously registered GAP LE event callback for remote authentication events. This function accepts as input the Bluetooth stack ID of the Bluetooth device that the remote authentication callback was registered previously (via a successful call to the GAP_LE_Register_Remote_Authentication function).

Prototype:

int BTPSAPI **GAP_LE_Un_Register_Remote_Authentication**(
unsigned int BluetoothStackID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

Return:

Zero (0) if the remote authentication callback was successfully un-registered.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Pair_Remote_Device

This function is provided to allow a means to pair with a remote, connected, device. This function accepts the device address of the currently connected device to pair with, followed by the pairing capabilities of the local device. This function also accepts as input the GAP LE event callback information to use during the pairing process. This function returns zero if successful or a negative error code if there was an error.

Note:

This function can only be issued by the master of the connection (the initiator of the connection). The reason is that a slave can only request a security procedure, it cannot initiate a security procedure.

This function will not create an LE ACL connection to the specified device. The LE ACL connection to the specified remote device must already exist before calling this function.

Prototype:

```
int BTPSAPI GAP_LE_Pair_Remote_Device(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, GAP_LE_Pairing_Capabilities_t *Capabilities,
    GAP_LE_Event_Callback_t GAP_LE_Event_Callback,
    unsigned long CallbackParameter);
```

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the connected device to pair with.
Capabilities	Pointer to a buffer that holds the pairing capabilities of the local host. This structure is defined as follows:

```
typedef struct
{
  GAP_LE_IO_Capability_t       IO_Capability;
  Boolean_t                    OOB_Present;
  GAP_LE_Bonding_Type_t        Bonding_Type;
  Boolean_t                    MITM;
  Byte_t                       Maximum_Encryption_Key_Size;
  GAP_LE_Key_Distribution_t    Receiving_Keys;
  GAP_LE_Key_Distribution_t    Sending_Keys;
} GAP_LE_Pairing_Capabilities_t;
```

where, IO_Capability defines the I/O capabilities of the host. This is one of the following values:

```
licDisplayOnly
licDisplayYesNo
licKeyboardOnly
licNoInputNoOutput
licKeyboardDisplay
```

and, OOB_Present is a flag that specifies whether the host contains out of band (OOB) data.

and, Bonding_type defines the type of bonding being requested. This is one of the following values:

lbtNoBonding
lbtBonding

and, MITM specifies whether man in the middle (MITM) protection is requested.

and, Maximum_Encryption_Key_Size specifies the largest size of the encryption key that is required.

and, Receiving_Keys and Sending_Keys members define the keys that the host would like to receive or send to the device (respectively). These structures are defined as follows:

```
typedef struct
{
  Boolean_t Encryption_Key;
  Boolean_t Identification_Key;
  Boolean_t Signing_Key;
} GAP_LE_Key_Distribution_t;
```

where, each member is a flag that specifies whether that particular key type is requested.

GAP_LE_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP LE event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the pairing request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Authentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Authentication_Response

This function is provided to allow a mechanism for the local device to respond to GAP LE authentication events. This function is used to specify the authentication information for the specified Bluetooth device. This function accepts as input, the Bluetooth protocol stack ID of the Bluetooth device that has requested the authentication action, and the authentication response information (specified by the caller).

Note:

This function should be called to respond to authentication requests that were received via any of the installed callbacks:

- Pairing callback
- Remote authentication callback

Prototype:

int BTPSAPI **GAP_LE_Authentication_Response**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR,
GAP_LE_Authentication_Response_Information_t *GAP_LE_Authentication_Information);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the Bluetooth device that is being authenticated.
GAP_LE_Authentication_Information	Pointer to a structure that holds authentication information.

Return:

Zero (0) if the remote authentication response was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_PAIRING_NOT_ACTIVE
BTPS_ERROR_DEVICE_HCI_ERROR
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Authentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Reestablish_Security

This function is provided to allow a means to re-establish security credentials that were previously valid. This function performs differently depending upon if the local device is a master or a slave to the device specified. If the local device is a master then this function will process the specified security parameters and attempt to re-authenticate the device. If the local device is a slave then this function will request the master to re-establish the security. The reason for the differing behavior is that the slave can only request security be initiated, it cannot initiate the security process itself. This function returns zero if successful or a negative error code if there was an error.

Note:

This function will not create an LE ACL connection to the specified device. The LE ACL connection to the specified remote device must already exist before calling this function.

Prototype:

int BTPSAPI **GAP_LE_Reestablish_Security**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, GAP_LE_Security_Information_t *SecurityInformation,
    GAP_LE_Event_Callback_t GAP_LE_Event_Callback,
    unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the connected device to re-establish security with.
SecurityInformation	Pointer to a buffer that holds the security information required to re-establish the security. This structure is defined as follows:

```
typedef struct
{
  Boolean_t Local_Device_Is_Master;
  union
  {
    GAP_LE_Slave_Security_Information_t
               Slave_Information;
    GAP_LE_Master_Security_Information_t Master_Information;
  } Security_Information;
} GAP_LE_Security_Information_t;
```

where, Local_Device_Is_master is a flag that specifies whether or not the local device is the master or the slave of this connection.

If the local device IS NOT the master (i.e. this parameter is FALSE), then the Slave_Information structure needs to be populated. The format of the Slave_Information member is defined as:

```
typedef struct
{
   GAP_LE_Bonding_Type_t Bonding_Type;
   Boolean_t             MITM;
} GAP_LE_Slave_Security_Information_t;
```

and, contains the required security parameters that the slave is requesting (should match prior security establishment).

If the local device IS the master (i.e. this parameter is TRUE), then the Master_Information structure needs to be populated. The format of the Master_Information member is defined as:

```
typedef struct
{
   Byte_t            Encryption_Key_Size;
   Long_Term_Key_t   LTK;
   Word_t            EDIV;
   Random_Number_t   Rand;
} GAP_LE_Master_Security_Information_t;
```

and, contains the required security parameters that the master is requesting (should match prior security establishment).

GAP_LE_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP LE event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the re-establish security request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Authentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Request_Security

This function is provided to allow a means for a slave device to request that the master (of the connection) perform a pairing operation or re-establishing prior security. This function can only be called by a slave device. The reason for this is that the slave can only request security be initiated, it cannot initate the security process itself. This function returns zero if successful or a negative error code if there was an error.

Note:

This function will not create an LE ACL connection to the specified device. The LE ACL connection to the specified remote device must already exist before calling this function.

Prototype:

int BTPSAPI **GAP_LE_Request_Security**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, GAP_LE_Bonding_Type_t Bonding_Type, Boolean_t MITM,
GAP_LE_Event_Callback_t GAP_LE_Event_Callback,
unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the connected device to request security from.
Bonding_Type	The required bonding type for the security being requested. This value is one of the following: lbtNoBonding lbtBonding
MITM	Flag that specifies whether man in the middle (MITM) protection is required.
GAP_LE_Event_Callback	Pointer to a callback function to be used by the GAP layer to dispatch GAP LE event information for this request.
CallbackParameter	User defined value to be used by the GAP layer as an input parameter for the specified callback.

Return:

Zero (0) if the security request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Authentication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Set_Fixed_Passkey

This function is provided to allow a means for a fixed passkey to be used whenever the local Bluetooth device is chosen to display a passkey during a pairing operation. This fixed passkey is only used when the local Bluetooth device is chosen to display the passkey, based on the remote I/O capabilies and the local I/O capabilities.

Prototype:

int BTPSAPI **GAP_LE_Set_Fixed_Passkey**(unsigned int BluetoothStackID,
DWord_t *Fixed_Display_Passkey);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Fixed_Display_Passkey	Optional pointer to the fixed display passkey to use. If this parameter is NULL, then a fixed display passkey that was previously set using this function is no longer used. If this parameter is non-NULL then the passkey that it points to is used for all future pairing operations where the local Bluetooth device displays the passkey.

Return:

Zero (0) if the fixed passkey was successfully configured.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Query_Encryption_Mode

This function is provided to allow a means to query the current encryption mode for the LE connection that is specified.

Prototype:

int BTPSAPI **GAP_LE_Query_Encryption_Mode**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, GAP_Encryption_Mode_t *GAP_Encryption_Mode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the connected device to query the link encryption mode.
GAP_Encryption_Mode	Pointer to store the link encryption mode. This parameter is not optional, and can not be NULL. If this function returns success this will point to one of the following values: emDisabled emEnabled

Return:

Zero (0) if the encryption mode for the specified connection was successfully obtained.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Query_Connection_Handle

The following function is provided to allow a means to query the LE connection handle of a connection to a remote Bluetooth Low Energy device. If a connection exists to the remote device specified, the LE connection handle is returned in the buffer passed to this function. This function will return zero on success, or a negative return error code if there was an error. If this function returns success, then the Connection_Handle variable will contain the current LE connection handle for the LE connection to the specified Bluetooth device address.

Note:

This function is only for LE connections. This function will NOT return connection handles for Bluetooth BR/EDR connections.

Prototype:

int BTPSAPI **GAP_LE_Query_Connection_Handle**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, Word_t *Connection_Handle);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth Low Energy device of which to query the connection handle.
Connection_Handle	Pointer to a variable that will receive the connection handle associated with the specified Bluetooth device address.

Return:

Zero (0) if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_NOT_CONNECTED
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Query_Connection_Parameters

The following function is provided to allow a means to query the LE connection parameters for a connection to a remote Bluetooth Low Energy device. If a connection exists to the remote device specified, the current LE connection parameters are returned in the structure passed to this function. This function will return zero on success, or a negative return error code if there was an error. If this function returns success, then the Current_Connection_Parameters variable will contain the current LE connection parameters for the LE connection to the specified Bluetooth device address.

Prototype:

int BTPSAPI **GAP_LE_Query_Connection_Parameters**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR,
GAP_LE_Current_Connection_Parameters_t *Current_Connection_Parameters);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth Low Energy device of which to query the connection handle.
Current_Connection_Parameters	Pointer to a structure that will receive the connection parameters for the connection to the specified device.

Return:

Zero (0) if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_DEVICE_NOT_CONNECTED
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Generate_Long_Term_Key

This function is provided to allow a means for creating a new long term key (LTK). This function accepts the diversifying hiding key (DHK) and the encryption rook key (ER). Using these inputs this function generates the long term key (LTK), the diversifier (DIV), and the encrypted diversifier (EDIV) values. This function returns zero if successful or a negative error code if there was an error.

Prototype:

int BTPSAPI **GAP_LE_Generate_Long_Term_Key(**unsigned int BluetoothStackID, Encryption_Key_t *DHK, Encryption_Key_t *ER, Long_Term_Key_t *LTK_Result, Word_t *DIV_Result, Word_t *EDIV_Result, Random_Number_t *Rand_Result);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the connected device to generate the long term key (LTK) for.
DHK	Diversifying hiding key used as input to generate the long term key (LTK).
ER	Encryption root key (ERK) used with the DHK to generate the long term key (LTK).
LTK_Result	Pointer to a buffer that will receive the generated long term key (LTK).
DIV_Result	Pointer to a buffer that will receive the diversifier (DIV) that was used to generate the long term key (LTK).
EDIV_Result	Pointer to a buffer that will receive the encrypted diversifier (EDIV) that was used to generate the long term key (LTK).
Rand_Result	Pointer to a buffer that will receive the random number that was used to generate the long term key (LTK).

Return:

Zero (0) if the long term key (LTK) was successfully generated.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Regenerate_Long_Term_Key

This function is provided to allow a means for re-generating a long term key (LTK) given the required security parameters. This function accepts the diversifying hiding key (DHK), the encryption rook key (ER), the encrypted diversifier (EDIV), and a random number (Rand). Using these inputs this function re-generates a long term key (LTK). This function returns zero if successful or a negative error code if there was an error.

Prototype:

int BTPSAPI **GAP_LE_Regenerate_Long_Term_Key(**unsigned int BluetoothStackID,
Encryption_Key_t *DHK, Encryption_Key_t *ER, Word_t EDIV,
Random_Number_t *Rand, Long_Term_Key_t *LTK_Result);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Bluetooth device address of the connected device to generate the long term key (LTK) for.
DHK	Diversifying hiding key used as input to re-generate the long term key (LTK).
ER	Encryption root key (ERK) that will be used to re-generate the long term key (LTK).
EDIV	Encrypted diversifier (EDIV) that will be used to re-generate the long term key (LTK).
Rand	Random number that will be used to during the re-generation process.
LTK_Result	Pointer to a buffer that will receive the generated long term key (LTK).

Return:

Zero (0) if the long term key (LTK) was successfully re-generated.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Diversify_Function

The following function is provided to allow a means of performing the Diversify Function, D1, as specified in the Bluetooth 4.0 specification, Volume 3, Part H, section 5.2.2.1 of the Core specification. This function accepts the input Encryption Key, the D and R values, and a pointer to place the encryption key result. This function returns zero if succesfull or a negative error code.

Prototype:

int BTPSAPI **GAP_LE_Diversify_Function(**unsigned int BluetoothStackID,
Encryption_Key_t *Key, Word_t DIn, Word_t RIn, Encryption_Key_t *Result);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Key	Encryption key used as input to the diversify function.
DIn	D value used as input to the diversify function.
RIn	R value used as input to the diversify function.
Result	Pointer to a buffer that will receive the generated encryption key.

Return:

Zero (0) if the diversify function completed successfully.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Connection_Parameter_Update_Request

The following function is provided to allow a means for a slave device to request that the master update the connection parameters. This function can only be issued by the slave device. This function is asynchronous in nature because the master has to accept the parameter request. This function returns zero if succesfull or a negative error code.

Note:

All connection parameters to this function are specified in milli-seconds except the slave latency which is specified in number of connection events.

Prototype:

int BTPSAPI **GAP_LE_Connection_Parameter_Update_Request(**
unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR,
Word_t Connection_Interval_Min, Word_t Connection_Interval_Max,
Word_t Slave_Latency, Word_t Supervision_Timeout);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the connected device that the slave is requesting the connection parameter update of.
Connection_Interval_Min	Minimum requested connection interval. This value is specified in milli-seconds and must be between: MINIMUM_MINIMUM_CONNECTION_INTERVAL MAXIMUM_MINIMUM_CONNECTION_INTERVAL Note the default minimum connection interval is defined by the constant: DEFAULT_MINIMUM_CONNECTION_INTERVAL
Connection_Interval_Max	Maximum requested connection interval. This value is specified in milli-seconds and must be between: MINIMUM_MAXIMUM_CONNECTION_INTERVAL MAXIMUM_MAXIMUM_CONNECTION_INTERVAL Note the default maximum connection interval is defined by the constant: DEFAULT_MAXIMUM_CONNECTION_INTERVAL
Slave_Latency	Requested slave latency. This value is specified in number of connection events and must be between: MINIMUM_SLAVE_LATENCY MAXIMUM_SLAVE_LATENCY Note the default slave latency is defined by the constant: DEFAULT_SLAVE_LATENCY

Supervision_Timeout	Requested supervision timeout. This value is specified in milli-seconds and must be between: MINIMUM_LINK_SUPERVISION_TIMEOUT MAXIMUM_LINK_SUPERVISION_TIMEOUT Note the default link supervision timeout is defined by the constant: DEFAULT_LINK_SUPERVISION_TIMEOUT

Return:

Zero (0) if the connection update request was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Connection_Parameter_Update_Response

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Connection_Parameter_Update_Response

The following function is provided to allow a means for a master device to respond to a connection update request from a slave that has requested an update to the connection parameters. This function can only be issued by the master device. This function returns zero if succesfull or a negative error code.

Note:

If the connection parameters are accepted, then:

- the slave is notified of the connection parameters that were accepted
- the new connection parameters are applied to the connection

Prototype:

int BTPSAPI **GAP_LE_Connection_Parameter_Update_Response(**
unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, Boolean_t Accept,
GAP_LE_Connection_Parameters_t *ConnectionParameters);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Device address of the connected device that the master is responding to the connection parameter update of.
Accept	Flag that specifies whether the slave requested parameters were accepted. If this value is FALSE then the next parameter is ignored. If TRUE, the next parameter specifies the new connection parameters.
ConnectionParameters	Specifies the new, accepted, connection parameters of the connection to the remote device. This structure is defined as follows:

```
typedef struct
{
  Word_t Connection_Interval_Min;
  Word_t Connection_Interval_Max;
  Word_t Slave_Latency;
  Word_t Supervision_Timeout;
  Word_t Minimum_Connection_Length;
  Word_t Maximum_Connection_Length;
} GAP_LE_Connection_Parameters_t;
```

Note that ALL parameters are specified in milli-seconds except the Slave_Latency parameter which is specified in connection events.

where, Connection_Interval_Min is specified in milli-seconds and must be between:

MINIMUM_MINIMUM_CONNECTION_INTERVAL
MAXIMUM_MINIMUM_CONNECTION_INTERVAL

Note the default minimum connection interval is defined by the constant:

DEFAULT_MINIMUM_CONNECTION_INTERVAL

and, Connection_Interval_Max is specified in milli-seconds and must be between:

MINIMUM_MAXIMUM_CONNECTION_INTERVAL
MAXIMUM_MAXIMUM_CONNECTION_INTERVAL

Note the default maximum connection interval is defined by the constant:

DEFAULT_MAXIMUM_CONNECTION_INTERVAL

and, Slave_Latency is specified in number of connection events and must be between:

MINIMUM_SLAVE_LATENCY

MAXIMUM_SLAVE_LATENCY

Note the default slave latency is defined by the constant:

DEFAULT_SLAVE_LATENCY

and, Supervision_Timeout is specified in milli-seconds and must be between:

MINIMUM_LINK_SUPERVISION_TIMEOUT
MAXIMUM_LINK_SUPERVISION_TIMEOUT

Note the default link supervision timeout is defined by the constant:

DEFAULT_LINK_SUPERVISION_TIMEOUT

and, the Minimum_Connection_Length and Maximum_Connection_Length parameters are specified in milli-seconds and represent the expected minimum and maximum connection events for the connection. These values must be between:

MINIMUM_CONNECTION_EVENT_LENGTH
MAXIMUM_CONNECTION_EVENT_LENGTH

Return:

Zero (0) if the connection update response was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GAP_LE_Update_Connection_Parameters

The following function is provided to allow a means for a master device to attempt to update the connection parameters for an LE connection. This function can only be issued by the master device. This function returns zero if succesfull or a negative error code.

Prototype:

int BTPSAPI **GAP_LE_Update_Connection_Parameters**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, GAP_LE_Connection_Parameters_t *ConnectionParameters);

Parameters:

BluetoothStackID[1] — Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.

BD_ADDR — Device address of the connected device that the master is responding to the connection parameter update of.

ConnectionParameters — Specifies the new connection parameters to attempt to apply to the connection to the remote device. This structure is defined as follows:

```
typedef struct
{
  Word_t Connection_Interval_Min;
  Word_t Connection_Interval_Max;
  Word_t Slave_Latency;
  Word_t Supervision_Timeout;
  Word_t Minimum_Connection_Length;
  Word_t Maximum_Connection_Length;
} GAP_LE_Connection_Parameters_t;
```

Note that ALL parameters are specified in milli-seconds except the Slave_Latency parameter which is specified in connection events.

where, Connection_Interval_Min is specified in milli-seconds and must be between:

MINIMUM_MINIMUM_CONNECTION_INTERVAL
MAXIMUM_MINIMUM_CONNECTION_INTERVAL

Note the default minimum connection interval is defined by the constant:

DEFAULT_MINIMUM_CONNECTION_INTERVAL

and, Connection_Interval_Max is specified in milli-seconds and must be between:

MINIMUM_MAXIMUM_CONNECTION_INTERVAL
MAXIMUM_MAXIMUM_CONNECTION_INTERVAL

Note the default maximum connection interval is defined by the constant:

DEFAULT_MAXIMUM_CONNECTION_INTERVAL

and, Slave_Latency is specified in number of connection events and must be between:

MINIMUM_SLAVE_LATENCY
MAXIMUM_SLAVE_LATENCY

Note the default slave latency is defined by the constant:

DEFAULT_SLAVE_LATENCY

and, Supervision_Timeout is specified in milli-seconds and must be between:

MINIMUM_LINK_SUPERVISION_TIMEOUT
MAXIMUM_LINK_SUPERVISION_TIMEOUT

Note the default link supervision timeout is defined by the constant:

DEFAULT_LINK_SUPERVISION_TIMEOUT

and, the Minimum_Connection_Length and Maximum_Connection_Length parameters are specified in milli-seconds and represent the expected minimum and maximum connection events for the connection. These values must be between:

MINIMUM_CONNECTION_EVENT_LENGTH
MAXIMUM_CONNECTION_EVENT_LENGTH

Return:

Zero (0) if the connection update response was successfully submitted.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_GAP_NOT_INITIALIZED
BTPS_ERROR_LOCAL_CONTROLLER_DOES_NOT_
SUPPORT_LE
BTPS_ERROR_INSUFFICIENT_RESOURCES
BTPS_ERROR_INVALID_DEVICE_ROLE_MODE
BTPS_ERROR_FEATURE_NOT_CURRENTLY_ACTIVE

Possible Events:

etLE_Connection_Parameter_Updated

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

3.1.3 GAP Event Callbacks

There is one event callback prototype for all callback events in GAP for BR/EDR. These callbacks may be permanent (set in place for long periods of time) or dynamic (active only for getting the results of one query). The callback prototype is defined below.

GAP_Event_Callback_t

The following declared type represents the Prototype Function for the GAP event callback. This function will be called whenever a callback has been registered for the specified GAP action that is associated with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the GAP event data of the specified event, and the GAP event callback parameter that was specified when this callback was installed. The caller is free to use the contents of the GAP event 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 needs to be noted however, that if the same callback is installed more than once, then the callbacks will be called serially. Because of this, the processing in this function should be as efficient as possible. It should also 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 other GAP events will not be processed while this function call is outstanding).

Note: This function MUST NOT Block and wait for events that can only be satisfied by receiving other GAP events. A deadlock WILL occur because NO GAP event callbacks will be issued while this function is currently outstanding.

Prototype:

void (BTPSAPI ***GAP_Event_Callback_t**)(unsigned int BluetoothStackID,
GAP_Event_Data_t *GAP_Event_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GAP_Event_Data	Pointer to the passed event data. This structure has the following format:

```
typedef
{
  GAP_Event_Type_t   Event_Data_Type;
  Word_t             Event_Data_Size;
  union
  {
    GAP_Inquiry_Event_Data_t
              *GAP_Inquiry_Event_Data;
    GAP_Encryption_Mode_Event_Data_t
              *GAP_Encryption_Mode_Event_Data;
    GAP_Authentication_Event_Data_t
              *GAP_Authentication_Event_Data;
    GAP_Remote_Name_Event_Data_t
              *GAP_Remote_Name_Event_Data;
    GAP_Inquiry_Entry_Event_Data_t
              *GAP_Inquiry_Entry_Event_Data;
    GAP_Inquiry_With_RSSI_Entry_Event_Data_t
              *GAP_Inquiry_With_RSSI_Entry_Event_Data;
    GAP_Extended_Inquiry_Entry_Event_Data_t
              *GAP_Extended_Inquiry_Entry_Event_Data;
    GAP_Encryption_Refresh_Complete_Event_Data_t
              *GAP_Encryption_Refresh_Complete_Event_Data;
    GAP_Remote_Features_Event_Data_t
              *GAP_Remote_Features_Event_Data;
    GAP_Remote_Version_Information_Event_Data_t
              *GAP_Remote_Version_Information_Event_Data;
  } Event_Data;
} GAP_Event_Data_t;
```

where, GAP_Event_Type_t is an enumerated type with the values listed in the table in section 3.1.4.

CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

There is one event callback prototype for all callback events in GAP for LE. These callbacks may be permanent (set in place for long periods of time) or dynamic (active only for getting the results of one query). The callback prototype is defined below.

GAP_LE_Event_Callback_t

The following declared type represents the Prototype Function for the GAP LE event callback. This function will be called whenever a callback has been registered for the specified GAP LE action that is associated with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the GAP LE event data of the specified event, and the GAP LE event callback parameter that was specified when this callback was installed. The caller is free to use the contents of the GAP LE event 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 needs to be noted however, that if the same callback is installed more than once, then the callbacks will be called serially. Because of this, the processing in this function should be as efficient as possible. It should also 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 other GAP LE events will not be processed while this function call is outstanding).

Note: This function MUST NOT Block and wait for events that can only be satisfied by receiving other GAP LE events. A deadlock WILL occur because NO GAP LE event callbacks will be issued while this function is currently outstanding.

Prototype:

void (BTPSAPI ***GAP_LE_Event_Callback_t**)(unsigned int BluetoothStackID,
GAP_LE_Event_Data_t *GAP_LE_Event_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GAP_LE_Event_Data	Pointer to the passed event data. See definition in section 3.1.4
CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

3.1.4 GAP Events

The events that can be generated by the GAP profile portion of the Bluetooth Stack are listed in the table below and are described in the text that follows.

Event	Description
etInquiry_Result	Notify the host of the result of a completed inquiry (including all found devices).
etEncryption_Change_Result	Notify the host of a device link encryption change.

etAuthentication	Notify the host of a GAP authentication event.
etRemote_Name_Result	Notify the host of the result of a completed remote name request.
etInquiry_Entry_Result	Notify the host of an individual inquiry result.
etInquiry_With_RSSI_Entry_ Result	Notify the host of an individual inquiry result with RSSI information.
etExtended_Inquiry_Entry_Result	Notify the host of an individual inquiry result that contains Extended Inquiry Result information.
etEncryption_Refresh_Complete	Notify the host of the result of a completed encryption refresh request.
etRemote_Features_Result	Notify the host of the result of a completed remote features request.
etRemote_Version_Information_ Result	Notify the host of the result of a completed remote version information request.
etLE_Remote_Features_Result	Notify the host of the result of a completed LE remote features request.
etLE_Advertising_Report	Notify the host of an individual advertising report that was received during a scanning procedure.
etLE_Connection_Complete	Notify the host that a device is now connected.
etLE_Disconnection_Complete	Notify the host that a device is no longer connected.
etLE_Encryption_Change	Notify the host of a LE device link encryption change.
etLE_Encryption_Refresh_Complete	Notify the host of the result of a completed LE encryption refresh request.
etLE_Authentication	Notify the host of a GAP LE authentication event.
etLE_Connection_Parameter_ Update_Request	Notify the host of a connection parameter update request (received by master from a connected slave).
etLE_Connection_Parameter_ Update_Response	Notify the host of the status of a connection parameter update request (received by slave from a connected master).
etLE_Connection_Parameter_Updat ed	Notify the host of a change in the connection parameters for a specified connection.

etInquiry_Result

This event is dispatched when the Inquiry procedure is complete (normally, and not when cancelled). This event uses the following structure to hold the GAP inquiry event data to return all returned inquiry results once the Inquiry is complete.

Structure:

```
typedef struct
{
  Word_t              Number_Devices;
  GAP_Inquiry_Data_t  *GAP_Inquiry_Data;
} GAP_Inquiry_Event_Data_t;
```

Fields:

Number_Devices — Number of Inquiry data entries that the GAP_Inquiry_Data member points to (if non-zero).

GAP_Inquiry_Data — Pointer to an array of GAP Inquiry data structures. Each structure has the following format:

```
typedef struct
{
  BD_ADDR_t           BD_ADDR;
  Byte_t              Page_Scan_Repetition_Mode;
  Byte_t              Page_Scan_Period_Mode;
  Byte_t              Page_Scan_Mode;
  Class_of_Device_t   Class_of_Device;
  Word_t              Clock_Offset;
} GAP_Inquiry_Data_t;
```

where BD_ADDR is the address of the Bluetooth device,

Page_Scan_Repetition_Mode memberrepresents the Page Scan Modes that the remote device supports. The currently defined values are:

HCI_PAGE_SCAN_REPETITION_MODE_R0
HCI_PAGE_SCAN_REPETITION_MODE_R1
HCI_PAGE_SCAN_REPETITION_MODE_R2

The Page_Scan_Period_Mode member defines the Page Scan Period Mode that the remote device is using. The currently defined values are:

HCI_PAGE_SCAN_PERIOD_MODE_P0
HCI_PAGE_SCAN_PERIOD_MODE_P1
HCI_PAGE_SCAN_PERIOD_MODE_P2

The Page_Scan_Mode member defines the Page Scan Mode that the remote device is using. The currently defined values are:

Bluetooth Version 1.1

HCI_PAGE_SCAN_MODE_MANDATORY

HCI_PAGE_SCAN_MODE_OPTIONAL_I
HCI_PAGE_SCAN_MODE_OPTIONAL_II
HCI_PAGE_SCAN_MODE_OPTIONAL_III

Bluetooth Version 1.2

HCI_PAGE_SCAN_MODE_MANDATORY_STANDARD_SCAN
HCI_PAGE_SCAN_MODE_OPTIONAL_INTERLACED_SCAN

The Clock_Offset member defines the clock offset of the remote device. Bits 16 to 2 represent the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid.

The Class_of_Device member is a bit maks that determines the Bluetooth Class of Device that the device is using. See the HCI_Read_Class_of_Device command for a complete listing of feature bits.

etEncryption_Change_Result

This event is dispatched when the link level encryption status for a specific device completes (either successfully or with an error).

Structure:

```
typedef
{
  BD_ADDR_t                  Remote_Device;
  Byte_t                     Encryption_Change_Status;
  GAP_Encryption_Mode_t      Encryption_Mode;
} GAP_Encryption_Mode_Event_Data_t;
```

Fields:

BD_ADDR	Address of the Bluetooth device.
Encryption_Change_Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)
Encryption_Mode	The supported encryption mode types that the Bluetooth device can be set to. Possible Values are: emDisabled emEnabled

etAuthentication

This event is dispatched whenever an authentication event occurs. The authentication information itself contains the type of authentication that actually occurred.

Structure:

```
typedef struct
{
  GAP_Authentication_Event_Type_t              GAP_Authentication_Event_Type;
  BD_ADDR_t                                    Remote_Device;
  union
  {
    Byte_t                                     Authentication_Status;
    Byte_t                                     Secure_Simple_Pairing_Status;
    Boolean_t                                  Remote_IO_Capabilities_Known;
    GAP_Authentication_Event_Link_Key_Info_t   Link_Key_Info;
    DWord_t                                    Numeric_Value;
    GAP_Keypress_t                             Keypress_Type;
    GAP_IO_Capabilites_t                       IO_Capabilities;
  } Authentication_Event_Data;
} GAP_Authentication_Event_Data_t;
```

Fields:

GAP_Authentication_Event_Type — Specifies the data member of the struct that is valid. Possible values and the accompanying data are:

Authentication Event Type	Accompanying data
atLinkKeyRequest	No further data
atPINCodeRequest	No further data
atAuthenticationStatus	Authentication_Status
atLinkKeyCreation	Link_Key_Info
atKeypressNotification	Keypress_Type
atUserConfirmationRequest	Numeric_Value
atPasskeyNotification	Numeric_Value
atPasskeyRequest	No further data
etRemoteOutOfBandDataRequest	No further data
atIOCapabilityRequest	Remote_IO_Capabilities_Known
atIOCapabilityResponse	IO_Capabilities
atSecureSimplePairingComplete	Secure_Simple_Pairing_Status

BD_ADDR — Bluetooth address of the remote device.

Link_Key_Info — Link key authentication information calculated for the remote device. This structure is defined as follows:

```
typedef
```

```
{
  Link_Key_t   Link_Key;
  Byte_t       Key_Type;
} GAP_Authentication_Event_Link_Key_Info_t;
```

where, Key_Type is defined to be one of the following:

```
HCI_LINK_KEY_TYPE_COMBINATION_KEY
HCI_LINK_KEY_TYPE_LOCAL_UNIT_KEY
HCI_LINK_KEY_TYPE_REMOTE_UNIT_KEY
HCI_LINK_KEY_TYPE_DEBUG_COMBINATION_KEY
HCI_LINK_KEY_TYPE_UNAUTHENTICATED_
             COMBINATION_KEY
HCI_LINK_KEY_TYPE_AUTHENTICATED_
             COMBINATION_KEY
HCI_LINK_KEY_TYPE_CHANGED_COMBINATION_KEY
HCI_LINK_KEY_TYPE_INVALID_KEY_TYPE
```

Numeric_Value	Passkey or User Confirmation authentication information sent from the remote device.
Keypress_Type	Keypress type authentication information sent from the remote device. This value will be one of the following: kpEntryStarted kpDigitEntered kpDigitErased kpCleared kpEntryCompleted
IO_Capabilities	I/O capabilities authentication information sent from the remote device. This value will be one of the following: icDisplayOnly icDisplayYesNo icKeyboardOnly icNoInputNoOutput

etRemote_Name_Result

This event is dispatched when a remote name result operation is completed (either successfully or with an error.

Structure:

```
typedef
{
  Byte_t          Remote_Name_Status;
  BD_ADDR_t       Remote_Device;
  char           *Remote_Name;
} GAP_Remote_Name_Event_Data_t;
```

Fields:

Remote_Name_Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)

Remote_Device	The Bluetooth device address of the device queried.
Remote_Name	The user-friendly name of the remote device in a null-terminated string.

etInquiry_Entry_Result

This event is dispatched whenever a remote device is discovered during an inquiry procedure AND the local inquiry mode is set to imStandard (which is the default).

Structure:

```
typedef struct
{
  BD_ADDR_t           BD_ADDR;
  Byte_t              Page_Scan_Repetition_Mode;
  Byte_t              Page_Scan_Period_Mode;
  Byte_t              Page_Scan_Mode;
  Class_of_Device_t   Class_of_Device;
  Word_t              Clock_Offset;
} GAP_Inquiry_Entry_Event_Data_t;
```

Fields:

BD_ADDR	Address of the Bluetooth device.
Page_Scan_Repetition_Mode	Part of the supported Page Scan Modes that the remote device supports. The currently defined values are: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2
Page_Scan_Period_Mode	Current setting of this parameter. Possible values are: HCI_PAGE_SCAN_PERIOD_MODE_P0 HCI_PAGE_SCAN_PERIOD_MODE_P1 HCI_PAGE_SCAN_PERIOD_MODE_P2
Page_Scan_Mode	The other part of the supported Page Scan Modes that the remote device supports. The currently defined values are: **Bluetooth Version 1.1** HCI_PAGE_SCAN_MODE_MANDATORY HCI_PAGE_SCAN_MODE_OPTIONAL_I HCI_PAGE_SCAN_MODE_OPTIONAL_II HCI_PAGE_SCAN_MODE_OPTIONAL_III **Bluetooth Version 1.2** HCI_PAGE_SCAN_MODE_MANDATORY_ STANDARD_SCAN HCI_PAGE_SCAN_MODE_OPTIONAL_ INTERLACED_SCAN

Clock_Offset	Bits 16 to 2 of the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid.
Class_of_Device	Bit mask list of features that determine the class of device for this Bluetooth device. See the HCI_Read_Class_of_Device command for a complete listing of feature bits.

etInquiry_With_RSSI_Entry_Result

This event is dispatched whenever a remote device is discovered during an inquiry procedure AND the local inquiry mode is set to imRSSI.

Structure:

```
typedef struct
{
  BD_ADDR_t           BD_ADDR;
  Byte_t              Page_Scan_Repetition_Mode;
  Byte_t              Page_Scan_Period_Mode;
  Class_of_Device_t   Class_of_Device;
  Word_t              Clock_Offset;
  Byte_t              RSSI;
} GAP_Inquiry_With_RSSI_Entry_Event_Data_t;
```

Fields:

BD_ADDR	Address of the Bluetooth device.
Page_Scan_Repetition_Mode	Part of the supported Page Scan Modes that the remote device supports. The currently defined values are: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2
Page_Scan_Period_Mode	Current setting of this parameter. Possible values are: HCI_PAGE_SCAN_PERIOD_MODE_P0 HCI_PAGE_SCAN_PERIOD_MODE_P1 HCI_PAGE_SCAN_PERIOD_MODE_P2
Clock_Offset	Bits 16 to 2 of the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid.
Class_of_Device	Bit mask list of features that determine the class of device for this Bluetooth device. See the HCI_Read_Class_of_Device command for a complete listing of feature bits.
RSSI	RSSI value returned for the remote device.

etExtended_Inquiry_Entry_Result

This event is dispatched whenever a remote device is discovered during an inquiry procedure AND the local inquiry mode is set to imExtended.

Structure:

```
typedef struct
{
  BD_ADDR_t                              BD_ADDR;
  Byte_t                                 Page_Scan_Repetition_Mode;
  Byte_t                                 Page_Scan_Period_Mode;
  Class_of_Device_t                      Class_of_Device;
  Word_t                                 Clock_Offset;
  Byte_t                                 RSSI;
  GAP_Extended_Inquiry_Response_Data_t   Extended_Inquiry_Response_Data;
  Extended_Inquiry_Response_Data_t      *Raw_Extended_Inquiry_Response_Data;
} GAP_Extended_Inquiry_Entry_Event_Data_t;
```

Fields:

BD_ADDR	Address of the Bluetooth device.
Page_Scan_Repetition_Mode	Part of the supported Page Scan Modes that the remote device supports. The currently defined values are: HCI_PAGE_SCAN_REPETITION_MODE_R0 HCI_PAGE_SCAN_REPETITION_MODE_R1 HCI_PAGE_SCAN_REPETITION_MODE_R2
Page_Scan_Period_Mode	Current setting of this parameter. Possible values are: HCI_PAGE_SCAN_PERIOD_MODE_P0 HCI_PAGE_SCAN_PERIOD_MODE_P1 HCI_PAGE_SCAN_PERIOD_MODE_P2
Clock_Offset	Bits 16 to 2 of the difference between the master and slave device clocks, mapped to bits 14 to 0 of this parameter (i.e., computed from ((clock_slave – clock_master) ShiftRight 2). Bit 15 (MSB) is the Clock_Offset_Valid flag which is 1 if the offset value is valid.
Class_of_Device	Bit mask list of features that determine the class of device for this Bluetooth device. See the HCI_Read_Class_of_Device command for a complete listing of feature bits.
RSSI	RSSI value returned for the remote device.
Extended_Inquiry_Response_Data	Container structure which contains the parsed Extended Inquiry Result data. This structure contains a count and a pointer to a list of each individual Extended Inquiry Result items.

Raw_Extended_Inquiry_Response_Data	Pointer to the actual, raw, un-parsed, Extended Inquiry Result data that was returned during the inquiry procedure.

etEncryption_Refresh_Result

This event is dispatched when the link level encryption refresh status for a specific device completes (either successfully or with an error).

Structure:

```
typedef struct
{
  BD_ADDR_t Remote_Device;
  Byte_t    Status;
} GAP_Encryption_Refresh_Complete_Event_Data_t;
```

Fields:

Remote_Device	Address of the Bluetooth device for which the event is valid.
Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)

etRemote_Features_Result

This event is dispatched when a remote device features request for a specific device completes (either successfully or with an error). Please see the description for the etRead_Remote_Supported_Features_Complete_Event event in the HCI section for a more detailed description of the parameters of the his event.

Structure:

```
typedef struct
{
  Byte_t          Status;
  BD_ADDR_t       BD_ADDR;
  LMP_Features_t  Features;
  Byte_t          Page_Number;
  Byte_t          Maximum_Page_Number;
} GAP_Remote_Features_Event_Data_t;
```

Fields:

Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)
BD_ADDR	Address of the Bluetooth device for which the event is valid.
Features	LMP features of the remote device.
Page_Number	LMP page number that the returned LMP features are located.
Maximum_Page_Number	Largest LMP page number that the remote device supports (for LMP feature requests).

etRemote_Version_Information_Result

This event is dispatched when a remote device version information request for a specific device completes (either successfully or with an error). Please see the description for the etRead_Remote_Version_Information_Complete_Event event in the HCI section for a more detailed description of the parameters of the his event.

Structure:

```
typedef struct
{
   Byte_t          Status;
   BD_ADDR_t       BD_ADDR;
   Byte_t          LMP_Version;
   Word_t          Manufacturer_ID;
   Word_t          LMP_Subversion;
} GAP_Remote_Version_Information_Event_Data_t;
```

Fields:

Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)
BD_ADDR	Address of the Bluetooth device for which the event is valid.
LMP_Version	LMP version of the remote device.
Manufacturer_ID	LMP manufacturer ID of the remote device.
LMP_Subversion	LMP subversion of the remote device.

etLE_Remote_Features_Result

This event is dispatched when a remote device features request for a specific LE device completes (either successfully or with an error). Please see the description for the meRead_Remote_Used_Features_Complete_Event LE meta event in the HCI section for a more detailed description of the parameters of the his event.

Structure:

```
typedef struct
{
   Byte_t            Status;
   BD_ADDR_t         BD_ADDR;
   LE_Features_t LE_ Features;
} GAP_LE_Remote_Features_Event_Data_t;
```

Fields:

Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)
BD_ADDR	Remote Bluetooth device address for which this event is valid for.
Features	LE LMP features of the remote device.

etLE_Advertising_Report

This event is dispatched when either an advertising report or a scan response report is received during a scan procedure. This event will contain all of the parsed report data, as well as the original, un-parsed data bytes that make up the report itself.

Notes:

This event contains both the raw report data (simple array of bytes), and the parsed report data. Because the format of scan response data and advertising data is the same, the same container structure can be used to represent the parsed data. The parsed data is simply an array of elements that break out each invidual tuple of the data. Each tuple consists of:

- data type
- data length
- data

Structure:

```
typedef struct
{
  GAP_LE_Advertising_Report_Type_t      Advertising_Report_Type;
  GAP_LE_Address_Type_t                 Address_Type;
  BD_ADDR_t                             BD_ADDR;
  Byte_t                                RSSI;
  GAP_LE_Advertising_Data_t             Advertising_Data;
  Byte_t                                Raw_Report_Length;
  Byte_t                               *Raw_Report_Data;
} GAP_LE_Advertising_Report_Data_t;
```

Fields:

Advertising_Report_Type	Specifies the actual type of report that was received. This value is one of the following: rtConnectableUndirected rtConnectableDirected rtScannableUndirected rtNonConnectableUndirected rtScanResponse
Address_Type	Specifies the device address type of the address that the report was received from.
BD_ADDR	Remote Bluetooth device address for which this event is valid for.
RSSI	RSSI value returned for the remote device.
Advertising_Data	Parsed report data. This member contains each individual element of the report accessable by simple array logic. This member has the following format: typedef struct {

```
   unsigned int                    Number_Data_Entries;
   GAP_LE_Advertising_Data_Entry_t *Data_Entries;
} GAP_LE_Advertising_Data_t;
```

	where Data_Entries is a pointer to an array that contains Number_Data_Entries entries of parsed data. Each element of the array has the following format and represents an individual tuple (type, length, data) of the report data:

```
typedef struct
{
   DWord_t    AD_Type;
   Byte_t     AD_Data_Length;
   Byte_t    *AD_Data_Buffer;
} GAP_LE_Advertising_Data_Entry_t;
```

Raw_Report_Length	Specifies the size (in bytes) of the actual raw report that was received.
Raw_Report_Data	Pointer to a buffer that contains the actual raw report data bytes that were received. This buffer will be Raw_Report_Length bytes in length.

etLE_Connection_Complete

This event is dispatched when a remote device is connected to the local device. This can occur by one of two mechanisms:

- LE device calling the GAP_LE_Create_Connection function
- LE device calling the GAP_LE_Advertising_Enable function (and allowing connections)

Note that whenever this event is dispatched, if the device was advertising, the advertising process is stopped. If the connection was established via calling the GAP_LE_Create_Connection function then the connection process is stopped (this means that if multiple devices were specified in the white list they will not continued to be have connection attempts).

Structure:

```
typedef struct
{
   Byte_t                              Status;
   Boolean_t                           Master;
   GAP_LE_Address_Type_t               Peer_Address_Type;
   BD_ADDR_t                           Peer_Address;
   GAP_LE_Current_Connection_Parameters_t Current_Connection_Parameters;
} GAP_LE_Connection_Complete_Event_Data_t;
```

Fields:

Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)

Master	Flag that denotes whether the local device is the master of the connection.
Peer_Address_Type	Denotes the address type of the device that is now connected. This is one of the following values: latPublic latRandom
Peer_Address	Remote Bluetooth device address for which this event is valid for.
Current_Connection_Parameters	Structure that contains the connection parameters for the connection.

etLE_Disconnection_Complete

This event is dispatched when a remote device is disconnected from the local device.

Structure:

```
typedef struct
{
  Byte_t                  Status;
  Byte_t                  Reason;
  GAP_LE_Address_Type_t   Peer_Address_Type;
  BD_ADDR_t               Peer_Address;
} GAP_LE_Disconnection_Complete_Event_Data_t;
```

Fields:

Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)
Reason	Disconnection reason (HCI error code - see HCI error codes in section 2.2)
Peer_Address_Type	Denotes the address type of the device that is now disconnected. This is one of the following values: latPublic latRandom
Peer_Address	Remote Bluetooth device address for which this event is valid for.

etLE_Encryption_Change

This event is dispatched when the encryption mode for a specific connected device occurs.

Structure:

```
typedef struct
{
  BD_ADDR_t                BD_ADDR;
  Byte_t                   Encryption_Change_Status;
  GAP_Encryption_Mode_t    Encryption_Mode;
} GAP_LE_Encryption_Change_Event_Data_t;
```

Fields:

BD_ADDR	Remote Bluetooth device address for which this event is valid for.
Encryption_Change_Status	Status of the encryption change.
Encryption_Mode	Denotes the current encryption mode. This is one of the following values: emDisabled emEnabled

etLE_Encryption_Refresh_Complete

This event is dispatched when the active encryption for a connected device is refreshed.

Structure:

```
typedef struct
{
  BD_ADDR_t BD_ADDR;
  Byte_t    Status;
} GAP_LE_Encryption_Refresh_Complete_Event_Data_t;
```

Fields:

BD_ADDR	Remote Bluetooth device address for which this event is valid for.
Status	Zero if successful or negative of HCI error code if problem occurred (see HCI error codes in section 2.2)

etLE_Authentication

This event is dispatched whenever an LE authentication event occurs. The authentication information itself contains the type of authentication that actually occurred.

Structure:

```
typedef struct
{
  GAP_LE_Authentication_Event_Type_t   GAP_LE_Authentication_Event_Type;
  BD_ADDR_t                            BD_ADDR;
  union
  {
    GAP_LE_Key_Request_Info_t                  Long_Term_Key_Request;
    GAP_LE_Pairing_Capabilities_t              Pairing_Request;
    GAP_LE_Security_Request_t                  Security_Request;
    GAP_LE_Confirmation_Request_t              Confirmation_Request;
    GAP_LE_Pairing_Status_t                    Pairing_Status;
    GAP_LE_Encryption_Request_Information_t    Encryption_Request_Information;
    GAP_LE_Encryption_Information_t            Encryption_Information;
    GAP_LE_Identity_Information_t              Identity_Information;
    GAP_LE_Signing_Information_t               Signing_Information;
    GAP_LE_Security_Establishment_Complete_t   Security_Establishment_Complete;
  } Authentication_Event_Data;
} GAP_LE_Authentication_Event_Data_t;
```

Fields:

GAP_LE_Authentication_Event_Type Specifies the data member of the struct that is valid. Possible values and the accompanying data are:

Authentication Event Type	Accompanying data
latLongTermKeyRequest	Long_Term_Key_Request
latSecurityRequest	Security_Request
latPairingRequest	Pairing_Request
latConfirmationRequest	Confirmation_Request
latPairingStatus	Pairing_Status
latEncryptionInformationRequest	Encryption_Request_Information
latIdentityInformationRequest	No further data
latSigningInformationRequest	No further data
latEncryptionInformation	Encryption_Information
latIdentityInformation	Identity_Information
latSigningInformation	Signing_Information
latSecurityEstablishmentComplete	Security_Establishment_Complete

BD_ADDR Bluetooth address of the remote device.

Long_Term_Key_Request	Long term key request information. This structure is defined as follows:

```
typedef struct
{
  Random_Number_t   Rand;
  Word_t            EDIV;
} GAP_LE_Key_Request_Info_t;
```

where, Rand is the random number, and EDIV is the encrypted diversifier that should be used to generate the key.

Pairing_Request	Pairing capabilities of the remote device. This structure is defined as follows:

```
typedef struct
{
  GAP_LE_IO_Capability_t      IO_Capability;
  Boolean_t                   OOB_Present;
  GAP_LE_Bonding_Type_t       Bonding_Type;
  Boolean_t                   MITM;
  Byte_t               Maximum_Encryption_Key_Size;
  GAP_LE_Key_Distribution_t Receiving_Keys;
  GAP_LE_Key_Distribution_t Sending_Keys;
} GAP_LE_Pairing_Capabilities_t;
```

where, IO_Capability defines the I/O capabilities of the host. This is one of the following values:

licDisplayOnly
licDisplayYesNo
licKeyboardOnly
licNoInputNoOutput
licKeyboardDisplay

and, OOB_Present is a flag that specifies whether the host contains out of band (OOB) data.

and, Bonding_type defines the type of bonding being requested. This is one of the following values:

lbtNoBonding
lbtBonding

and, MITM specifies whether man in the middle (MITM) protection is requested.

and, Maximum_Encryption_Key_Size specifies the largest size of the encryption key that is required.

and, Receiving_Keys and Sending_Keys members define the keys that the host would like to receive or send to the device (respectively). These structures are defined as follows:

```
typedef struct
{
  Boolean_t Encryption_Key;
  Boolean_t Identification_Key;
  Boolean_t Signing_Key;
} GAP_LE_Key_Distribution_t;
```

where, each member is a flag that specifies whether that particular key type is requested.

Security_Request

Defines the requested security parameters. This structure has the following format:

```
typedef struct
{
  GAP_LE_Bonding_Type_t     Bonding_Type;
  Boolean_t                 MITM;
} GAP_LE_Security_Request_t;
```

where, Bonding_Type defines the requested bonding type. This value is one of:

lbtNoBonding
lbtBonding

and, MITM is a flag that specifies whether man in the middle (MITM) protection is required.

Confirmation_Request

Specifies the required request type. This structure is defined as follows:

```
typedef struct
{
  GAP_LE_Confirmation_Request_Type_t  Request_Type;
  DWord_t                             Display_Passkey;
  Byte_t                  Negotiated_Encryption_Key_Size;
} GAP_LE_Confirmation_Request_t;
```

where, Request_Type defines the requested confirmation type. This value is one of:

crtNone
crtPasskey
crtDisplay
crtOOB

and, Display_Passkey represents the six digit passkey (000, 000 – 999, 999) to display if the requested confirmation type is crtDisplay. Note that this member is valid only if the type is crtDisplay.

and, Negotiated_Encryption_Key_Size represents the negotiated encryption key size.

Pairing_Status

Specifies the pairing status that has occurred. This structure has the following format:

```
typedef struct
{
   Boolean_t      Remote_Initiated;
   Byte_t         Status;
   Byte_t         Negotiated_Encryption_Key_Size;
} GAP_LE_Pairing_Status_t;
```

where, Remote_Initiated is a flag which specifies whether or not he remote device initiated the pairing.

and, Status represents the pairing status. This is one of the following values:

```
GAP_LE_PAIRING_STATUS_NO_ERROR
GAP_LE_PAIRING_STATUS_DISCONNECTED
GAP_LE_PAIRING_STATUS_LOCAL_RESOURCES
GAP_LE_PAIRING_STATUS_PROTOCOL_TIMEOUT
GAP_LE_PAIRING_STATUS_PASSKEY_ENTRY_FAILE
            D
GAP_LE_PAIRING_STATUS_OOB_NOT_AVAILABLE
GAP_LE_PAIRING_STATUS_AUTHENTICATION_
            REQUIREMENTS
GAP_LE_PAIRING_STATUS_CONFIRM_VALUE_FAILE
            D
GAP_LE_PAIRING_STATUS_PAIRING_NOT_SUPPORT
            ED
GAP_LE_PAIRING_STATUS_ENCRYPTION_KEY_SIZE
GAP_LE_PAIRING_STATUS_COMMAND_NOT_
            SUPPORTED
GAP_LE_PAIRING_STATUS_UNSPECIFIED_REASON
GAP_LE_PAIRING_STATUS_REPEATED_ATTEMPTS
GAP_LE_PAIRING_STATUS_INVALID_PARAMETERS
```

and, Negotiated_Encryption_Key_Size represents the negotiated encryption key size.

Encryption_Request_Information Specifies requested encryption information. This structure is defined as follows:

```
typedef struct
{
   Byte_t Encryption_Key_Size;
} GAP_LE_Encryption_Request_Information_t;
```

where, Encrypton_Key_Size represents the encryption key size of the remote device.

Encryption_Information Specifies the encryption parameters. This structure is defined as follows:

```
typedef struct
{
   Byte_t               Encryption_Key_Size;
   Long_Term_Key_t      LTK;
   Word_t               EDIV;
   Random_Number_t    Rand;
} GAP_LE_Encryption_Information_t;
```

where, Encryption_Key_Size represents the encryption key size (in bytes).

and, LTK represents the long term key.

and, EDIV represents the encrypted diversifier.

and, Rand represents the random number.

Identity_Information — Specifies the current identity information. This structure has the following format:

```
typedef struct
{
   Encryption_Key_t            IRK;
   GAP_LE_Address_Type_t       Address_Type;
   BD_ADDR_t                   Address;
} GAP_LE_Identity_Information_t;
```

where, IRK represents the identity resolving key.

and, Address_Type specifies the address type of the remote device. This is one of the following values:

```
latPublic
latRandom
```

and, Address specifies the address of the remote device.

Signing_Information — Specifies the device signing information. This structure has the following format:

```
typedef struct
{
   Encryption_Key_t CSRK;
} GAP_LE_Signing_Information_t;
```

Security_Establishment_Complete — Specifies that the security process has completed. This structure has the following format:

```
typedef struct
{
   Byte_t Status;
} GAP_LE_Security_Establishment_Complete_t;
```

where, Status defines the status of the completed security process. This is one of the following values:

```
GAP_LE_SECURITY_ESTABLISHMENT_STATUS_
              CODE_NO_ERROR
GAP_LE_SECURITY_ESTABLISHMENT_STATUS_
              CODE_LONG_TERM_KEY_ERROR
GAP_LE_SECURITY_ESTABLISHMENT_STATUS_
              CODE_EDIV_RAND_INVALID
```

GAP_LE_SECURITY_ESTABLISHMENT_STATUS_
CODE_DEVICE_TRIED_TO_REPAIR
GAP_LE_SECURITY_ESTABLISHMENT_STATUS_
CODE_LINK_DISCONNECTED
GAP_LE_SECURITY_ESTABLISHMENT_STATUS_
CODE_TIMEOUT

etLE_Connection_Parameter_Update_Request

This event is dispatched when the remote slave device is requesting a connection parameter update. This event is only dispatched to the master device (of the connection) because the master of the connection is the only device that can change the connection parameters.

Structure:

```
typedef struct
{
  BD_ADDR_t BD_ADDR;
  Word_t    Conn_Interval_Min;
  Word_t    Conn_Interval_Max;
  Word_t    Slave_Latency;
  Word_t    Conn_Supervision_Timeout;
} GAP_LE_Connection_Parameter_Update_Request_Event_Data_t;
```

Fields:

BD_ADDR	Remote Bluetooth device address for which this event is valid for.
Conn_Interval_Min	Minimum value for the the connection interval (in milli-seconds). This should fall within the range: MINIMUM_MINIMUM_CONNECTION_INTERVAL MAXIMUM_MINIMUM_CONNECTION_INTERVAL
Conn_Interval_Max	This should be greater than or equal to Conn_Interval_Min. This value is also specified in milli-seconds and shall fall within the range: MINIMUM_MAXIMUM_CONNECTION_INTERVAL MAXIMUM_MAXIMUM_CONNECTION_INTERVAL
Slave_Latency	Slave latency for connection. This value is specified in number of connection events and should be in range: MINIMUM_SLAVE_LATENCY MAXIMUM_SLAVE_LATENCY The default slave latency is specified by the constant: DEFAULT_SLAVE_LATENCY
Conn_Supervision_Timeout	Supervision timeout for LE link. This value is in milli-seconds and should be in range: MINIMUM_LINK_SUPERVISION_TIMEOUT

MAXIMUM_LINK_SUPERVISION_TIMEOUT

The default supervision timeout is specified by the constant:

DEFAULT_LINK_SUPERVISION_TIMEOUT

etLE_Connection_Parameter_Update_Response

This event is dispatched when the remote master device has processed a connection parameter update request (issued by the slave device). This event is only dispatched to the slave device (of the connection) because the master of the connection is the only device that can change the connection parameters.

Structure:

```
typedef struct
{
  BD_ADDR_t BD_ADDR;
  Boolean_t      Accepted;
} GAP_LE_Connection_Parameter_Update_Response_Event_Data_t;
```

Fields:

BD_ADDR	Remote Bluetooth device address for which this event is valid for.
Accepted	Boolean value that specifies whether or not the master accepted (and applied) the requested connection parameter updates.

etLE_Connection_Parameter_Updated

This event is dispatched when the connection parameters for a connection have been updated. This event is dispatched to both the master and the slave device of the connection.

Structure:

```
typedef struct
{
  Byte_t                                  Status;
  BD_ADDR_t                               BD_ADDR;
  GAP_LE_Current_Connection_Parameters_t  Current_Connection_Parameters;
} GAP_LE_Connection_Parameter_Updated_Event_Data_t;
```

Fields:

Status	Contains the status of the connection parameter update.
BD_ADDR	Remote Bluetooth device address for which this event is valid for.
Current_Connection_Parameters	Structure that contains the new connection parameters for the connection.

3.2 SPP Programming Interface

The SPP (Serial Port Profile) programming interface provides all features required for serial port emulation utilizing the RFCOMM protocol. Section 3.2.1 lists the SPP function calls. Section 3.2.2 lists the SPP event callback prototypes. Section 3.2.3 lists all supported SPP events. The actual prototypes and constants outlined in this section can be found in the **SPPAPI.H** header file in the Bluetopia distribution.

3.2.1 SPP Commands

The available SPP command functions are listed in the table below and are described in the text which follows.

Function	Description
SPP_Open_Server_Port	Establish server port to wait for connections
SPP_Close_Server_Port	Close an open port
SPP_Open_Port_Request_Response	Respond to a port open request from the remote device.
SPP_Register_SDP_Record	Add a generic SDP Service Record to the SDP database
SPP_Register_Raw_SDP_Record	Add a generic SDP Service Record to the SDP database with only pre-parsed attribute data possibly added to the protocol data.
SPP_Open_Remote_Port	Open a serial port to a remote device.
SPP_Close_Port	Close either a server port or a remote port.
SPP_Data_Read	Read data from a serial connection.
SPP_Data_Write	Send data on a serial connection.
SPP_Change_Buffer_Size	Change the default transmit/receive buffer sizes.
SPP_Purge_Buffer	Drop all data in an input/output buffer.
SPP_Send_Break	Notify the remote device of a break condition.
SPP_Line_Status	Send current line status to the remote side.
SPP_Port_Status	Send current modem/port control signals to the remote side.
SPP_Send_Port_Information	Send port parameters to be used to the remote side.
SPP_Respond_Port_Information	Respond to a send port information command from the remote side.
SPP_Query_Remote_Port_Information	Request current port parameters from the remote side.
SPP_Respond_Query_Port_Information	Reply to a request for current port parameters.

SPP_Get_Configuration_Parameters	Query RFCOMM frame size and default buffer sizes.
SPP_Set_Configuration_Parameters	Change RFCOMM frame size and default buffer sizes.
SPP_Get_Server_Connection_Mode	Query the current server connection mode.
SPP_Set_Server_Connection_Mode	Change the current server connection mode.
SPP_Get_Port_Connection_State	Query the current state of a specific SPP Port connection.
SPP_Set_Queuing_Parameters	Change the current lower level queuing parameters.
SPP_Get_Queuing_Parameters	Query the current lower level queuing parameters.
SPP_Query_Server_Present	Determine if there is currently a registered Serial Port Profile Server Port for a specific RFCOMM Server Port.

SPP_Open_Server_Port

This function is responsible for establishing a Serial Port Server which will wait for a connection to occur on the port established by this function.

Prototype:

int BTPSAPI **SPP_Open_Server_Port**(unsigned int BluetoothStackID,
unsigned int ServerPort, SPP_Event_Callback_t SPP_Event_Callback,
unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
ServerPort	Port number to use. This must fall in the range defined by the following constants: SPP_PORT_NUMBER_MINIMUM SPP_PORT_NUMBER_MAXIMUM
SPP_Event_Callback	Function to call when events occur on this port.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive, non-zero if successful. The return value will be the SerialPortID for the server port that was successfully opened. *This* is the value that should be used in all subsequent function calls (except another SPP_Open_Server_Port( ) call).

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etPort_Open_Request_Indication

etPort_Open_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Close_Server_Port

This function is responsible for Unregistering a Serial Port Server which was registered by a successful call to the SPP_Open_Server_Port( ) function. Note, this function does NOT delete any SDP Service Record Handles (i.e., added via a SPP_Register_SDP_Record( ) function call).

Prototype:

int BTPSAPI **SPP_Close_Server_Port**(unsigned int BluetoothStackID,
unsigned int SerialPortID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port to close. This is the value that was returned from the SPP_Open_Server_Port( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Open_Port_Request_Response

This function is responsible for responding to requests to connect to a Serial Port Server.

Notes:

1. When using this feature Bluetopia requires that a response be sent to a device requesting a connection within sixty seconds. If a response is not sent within this time a negative response will be sent to the device. Since this timeout is implementation specific the requesting device may timeout and disconnect sooner then Bluetopia.

Prototype:

int BTPSAPI **SPP_Open_Port_Request_Response**(unsigned int BluetoothStackID, unsigned int SerialPortID, Boolean_t AcceptConnection)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) function.
AcceptConnection	Boolean indicating if the pending connection should be accepted.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SPP_NOT_INITIALIZED

Possible Events:

etPort_Open_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Register_SDP_Record

This function provides a means to add a generic SDP Service Record to the SDP Database.

Notes:

1. This function should only be called with the SerialPortID that was returned from the SPP_Open_Server_Port( ) function. This function should **never** be used with the Serial Port ID returned from the SPP_Open_Remote_Port( ) function.

2. The Service Record Handle that is returned from this function will remain in the SDP Record Database until it is deleted by calling the SDP_Delete_Service_Record( ) function. A Macro is provided to delete the Service Record from the SDP Database. This Macro maps SPP_Un_Register_SDP_Record**( )** to SDP_Delete_Service_Record(), and is defined as follows:

SPP_Un_Register_SDP_Record(__BluetoothStackID, __SerialPortID, __SDPRecordHandle)

3. If no UUID information is specified in the SDPServiceRecord Parameter, then the default SPP Service Class's are added. Any Protocol Information that is specified (if any) will be added in the Protocol Attribute *after* the default SPP Protocol List (L2CAP and RFCOMM).
4. The Service Name is always added at Attribute ID 0x0100. A Language Base Attribute ID List is created that specifies that 0x0100 is UTF-8 Encoded, English Language.

Prototype:

int BTPSAPI **SPP_Register_SDP_Record**(unsigned int BluetoothStackID, unsigned int SerialPortID, SPP_SDP_Service_Record_t *SDPServiceRecord, char *ServiceName, DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) function.
SDPServiceRecord	Any additional Service Discovery Protocol information to be added to the record for this serial port server. This is a structured defined as:
ServiceName	Name to appear in the SDP Database for this service.
SDPServiceRecordHandle	Returned handle to the SDP Database entry which may be used to remove the entry at a later time.

```
typedef struct
{
   unsigned int          NumberServiceClassUUID;
   SDP_UUID_Entry_t      *SDPUUIDEntries;
   SDP_Data_Element_t    *ProtocolList;
} SPP_SDP_Service_Record_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED

BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Register_Raw_SDP_Record

This function provides a means to add a generic SDP Service Record to the SDP Database. The only difference with the **SPP_Register_SDP_Record**() API is that any additional protocol information to add to the SDP record must be in pre-parsed format.

Notes:

1. This function should only be called with the SerialPortID that was returned from the SPP_Open_Server_Port( ) function. This function should **never** be used with the Serial Port ID returned from the SPP_Open_Remote_Port( ) function.
2. The Service Record Handle that is returned from this function will remain in the SDP Record Database until it is deleted by calling the SDP_Delete_Service_Record( ) function. A Macro is provided to delete the Service Record from the SDP Database. This Macro maps SPP_Un_Register_SDP_Record**( )** to SDP_Delete_Service_Record(), and is defined as follows:

SPP_Un_Register_SDP_Record(__BluetoothStackID, __SerialPortID, __SDPRecordHandle)

3. If no UUID information is specified in the SDPServiceRecord Parameter, then the default SPP Service Class's are added. Any Protocol Information that is specified (if any) will be added in the Protocol Attribute *after* the default SPP Protocol List (L2CAP and RFCOMM).
4. The Service Name is always added at Attribute ID 0x0100. A Language Base Attribute ID List is created that specifies that 0x0100 is UTF-8 Encoded, English Language.

Prototype:

int BTPSAPI **SPP_Register_Raw_SDP_Record**(unsigned int BluetoothStackID,
unsigned int SerialPortID,
SPP_SDP_Raw_Service_Record_t *SDPServiceRecord,
char *ServiceName,
DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) function.

SDPServiceRecord	Any additional Service Discovery Protocol information to be added to the record for this serial port server. This is a structured defined as: `typedef struct { unsigned int NumberServiceClassUUID; SDP_UUID_Entry_t *SDPUUIDEntries; unsigned int NumberOfProtocolDataListUUIDOffsets; Word_t *ProtocolDataListUUIDOffsets; unsigned int ProtocolDataListLength; Byte_t *ProtocolDataList; } SPP_SDP_Raw_Service_Record_t;`
ServiceName	Name to appear in the SDP Database for this service.
SDPServiceRecordHandle	Returned handle to the SDP Database entry which may be used to remove the entry at a later time.

```
typedef struct
{
   unsigned int          NumberServiceClassUUID;
   SDP_UUID_Entry_t *SDPUUIDEntries;
   unsigned int          NumberOfProtocolDataListUUIDOffsets;
   Word_t                *ProtocolDataListUUIDOffsets;
   unsigned int          ProtocolDataListLength;
   Byte_t                *ProtocolDataList;
} SPP_SDP_Raw_Service_Record_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Open_Remote_Port

This function is used to open a remote serial port on the specified Remote Device.

Prototype:

int BTPSAPI **SPP_Open_Remote_Port**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, unsigned int ServerPort,
    SPP_Event_Callback_t SPP_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the Bluetooth device to connect with.
ServerPort	The remote device's server port ID to connect with.

SPP_Event_Callback	Function to call when events occur on this port.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive, non-zero if successful. The return value will be the SerialPortID for the port that was successfully opened. This is the value that should be used in all subsequent function calls.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED
RFCOMM_UNABLE_TO_CONNECT_TO_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE

Possible Events:

etPort_Open_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Close_Port

This function is used to close a Serial Port that was previously opened with the SPP_Open_Server_Port( ) function ***or*** the SPP_Open_Remote_Port( ) function. This function does **not** unregister a SPP Server Port from the system, it only disconnects any connection that is currently active on the Server Port. The SPP_Close_Server_Port() function can be used to Unregister the SPP Server Port.

Prototype:

int BTPSAPI **SPP_Close_Port**(unsigned int BluetoothStackID, unsigned int SerialPortID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port to close. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Data_Read

This function is used to read serial data from the specified serial connection. The SerialPortID that is passed to this function **must** have been established by either accepting a Serial Port Connection (callback from the SPP_Open_Server_Port() function) or by initiating a Serial Port Connection (via calling the SPP_Open_Remote_Port() function and having the remote side accept the connection).

Prototype:

int BTPSAPI **SPP_Data_Read**(unsigned int BluetoothStackID, unsigned int SerialPortID, Word_t DataBufferSize, Byte_t *DataBuffer)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
DataBufferSize	The size of the data buffer to be used for reading
DataBuffer	The data buffer that may be used to hold the read data

Return:

Positive or Zero if successful. Indicates the number of data bytes actually read in (zero if no data is available at the time of the call).

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etClose_Port_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Data_Write

This function is used to send data to the specified Serial Connection. The SerialPortID that is passed to this function **must** have been established by either accepting a Serial Port Connection (callback from the SPP_Open_Server_Port() function) or by initiating a Serial Port Connection (via calling the SPP_Open_Remote_Port() function and having the remote side accept the connection).

Note: If this function is unable to send all of the data that was specified (via the DataLength parameter) because of a full Transmit Buffer condition, this function will return the number of bytes that were actually sent (zero or more, but less than the DataLength parameter value). When this happens (and **only** when this happens), the user can expect to be notified when the Serial Port is able to send data again via the the etPort_Transmit_Buffer_Empty_Indication SPP Event. This will allow the user a mechanism to know when the Transmit Buffer is empty so that more data can be sent.

Prototype:

int BTPSAPI **SPP_Data_Write**(unsigned int BluetoothStackID, unsigned int SerialPortID, Word_t DataLength, Byte_t *DataBuffer)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
DataLength	The number of data bytes to send
DataBuffer	The data buffer that contains the data to send

Return:

Positive or zero if successful indicating the number of data bytes actually sent. See note above, for situations when this value is less than DataLength.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etClose_Port_Indication
etPort_Transmit_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Change_Buffer_Size

This function is provided to allow the programmer a means to change the default transmit and receive buffer sizes. Note, this function causes ALL data in each buffer to be lost. This function clears each data buffer so that all the available data buffer is available to be used.

Prototype:

int BTPSAPI **SPP_Change_Buffer_Size**(unsigned int BluetoothStackID,
    unsigned int SerialPortID, unsigned int ReceiveBufferSize,
    unsigned int TransmitBufferSize)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
ReceiveBufferSize	Size of the receive buffer.
TransmitBufferSize	Size of the transmit buffer.

Some handy constants that relate to buffer sizes are:

SPP_BUFFER_SIZE_MINIMUM
SPP_BUFFER_SIZE_MAXIMUM
SPP_BUFFER_SIZE_DEFAULT
SPP_BUFFER_SIZE_CURRENT

Where SPP_BUFFER_SIZE_CURRENT means to keep the indicated buffer at its current size.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Purge_Buffer

This function allows the programmer a mechanism for either aborting (dropping) all data present in either an input or an output buffer, or a means to wait until all data present in the output buffer has been transmitted.

Prototype:

int BTPSAPI **SPP_Purge_Buffer**(unsigned int BluetoothStackID, unsigned int SerialPortID, unsigned int PurgeBufferMask)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
PurgeBufferMask	Operation indicator, defined by the following bit mask values: SPP_PURGE_MASK_TRANSMIT_ABORT_BIT SPP_PURGE_MASK_RECEIVE_ABORT_BIT SPP_PURGE_MASK_TRANSMIT_FLUSH_BIT It should be noted that the SPP_PURGE_MASK_TRANSMIT_ABORT_BIT and the SPP_PURGE_MASK_TRANSMIT_FLUSH_BIT mask values can not be specified concurrently (i.e. they are mutually exclusive). If the flush is requested and this function returns BTPS_ERROR_SPP_BUFFER_EMPTY then a SPP Event Callback will not be issued because there is no data currently queued. Otherwise, if this function returns zero (success) and a flush is requested then the SPP Event Callback will be issued when the transmit buffer is empty.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER

BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED
BTPS_ERROR_SPP_BUFFER_EMPTY

Possible Events:

etPort_Transmit_Buffer_Empty_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Send_Break

This function allows the programmer a means to notify the remote side of the serial connection of a break condition.

Prototype:

int BTPSAPI **SPP_Send_Break**(unsigned int BluetoothStackID, unsigned int SerialPortID, unsigned int BreakTimeout)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
BreakTimeout	Length of the break detected in milliseconds. The following three constants are defined that relate to this parameter: SPP_BREAK_SIGNAL_DETECTED SPP_BREAK_SIGNAL_MINIMUM SPP_BREAK_SIGNAL_MAXIMUM

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etClose_Port_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Line_Status

This function provides a means to send the existing state of the Line Status to the remote side.

Prototype:

int BTPSAPI **SPP_Line_Status**(unsigned int BluetoothStackID, unsigned int SerialPortID, unsigned int SPPLineStatusMask)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
SPPLineStatusMask	Status to send. Built up from the following bit mask values: SPP_LINE_STATUS_OVERRUN_ERROR_BIT_MASK SPP_LINE_STATUS_PARITY_ERROR_BIT_MASK SPP_LINE_STATUS_FRAMING_ERROR_BIT_MASK Or one may send the following value: SPP_LINE_STATUS_NO_ERROR_VALUE

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etPort_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Port_Status

This function is used to send the existing state of all modem/port control signals to the remote side.

Prototype:

int BTPSAPI **SPP_Port_Status**(unsigned int BluetoothStackID, unsigned int SerialPortID, unsigned int PortStatus)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
PortStatus	Port status bits. Value is built up from the following constants: SPP_PORT_STATUS_RTS_CTS_BIT SPP_PORT_STATUS_DTR_DSR_BIT SPP_PORT_STATUS_RING_INDICATOR_BIT SPP_PORT_STATUS_CARRIER_DETECT_BIT Or the status may be cleared with the following constant: SPP_PORT_STATUS_CLEAR_VALUE

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etPort_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Send_Port_Information

This function provides a means to inform the remote side of the serial port parameters that are to be used.

Prototype:

int BTPSAPI **SPP_Send_Port_Information**(unsigned int BluetoothStackID,
unsigned int SerialPortID, SPP_Port_Information_t *SPPPortInformation)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
SPPPortInformation	The port parameters to be passed to the remote side, defined by the following structure:

```
typedef struct
{
   unsigned int        PortInformationMask;
   unsigned int        BaudRate;
   unsigned int        DataBits;
   SPP_Stop_Bits_t     StopBits;
   SPP_Parity_t        Parity;
   Byte_t              XOnCharacter;
   Byte_t              XOffCharacter;
   unsigned int        FlowControlMask;
} SPP_Port_Information_t;
```

where PortInformationMask defines which of the port parameters are set, defined by the following bit mask values:

SPP_PORT_INFORMATION_BAUD_RATE_BIT
SPP_PORT_INFORMATION_DATA_BITS_BIT
SPP_PORT_INFORMATION_STOP_BITS_BIT
SPP_PORT_INFORMATION_PARITY_BIT
SPP_PORT_INFORMATION_XON_CHARACTER_BIT
SPP_PORT_INFORMATION_XOFF_CHARACTER_BIT
SPP_PORT_INFORMATION_FLOW_CONTROL_BIT

Or it may be set to the following constant:

SPP_PORT_INFORMATION_NONE_VALUE

BaudRate can be one of the following values:

SPP_BAUD_RATE_MINIMUM
SPP_BAUD_RATE_MAXIMUM

SPP_BAUD_RATE_2400
SPP_BAUD_RATE_4800
SPP_BAUD_RATE_7200
SPP_BAUD_RATE_9600
SPP_BAUD_RATE_19200
SPP_BAUD_RATE_38400
SPP_BAUD_RATE_57600
SPP_BAUD_RATE_115200

SPP_BAUD_RATE_230400

DataBits can be one of the following values:

SPP_DATA_BITS_MINIMUM
SPP_DATA_BITS_MAXIMUM

SPP_DATA_BITS_5
SPP_DATA_BITS_6
SPP_DATA_BITS_7
SPP_DATA_BITS_8

StopBits can be one of the following values:

sbOneStopBit
sbOneOneHalfStopBit

Parity can be one of the following values:

ptNone
ptOdd
ptEven
ptMark
ptSpace

XOnCharacter and XoffCharacter may be any character. However, the following constants are defined in RFCOMM and may be useful for these:

RFCOMM_RPN_PARAMETER_DEFAULT_XON_CHARACTER
RFCOMM_RPN_PARAMETER_DEFAULT_XOFF_CHARACTER

FlowControlMask is built up from the following bit mask values:

SPP_FLOW_CONTROL_XON_XOFF_INPUT_ENABLED_BIT
SPP_FLOW_CONTROL_XON_XOFF_OUTPUT_ENABLED_BIT
SPP_FLOW_CONTROL_CTS_INPUT_ENABLED_BIT
SPP_FLOW_CONTROL_RTS_OUTPUT_ENABLED_BIT
SPP_FLOW_CONTROL_DSR_INPUT_ENABLED_BIT
SPP_FLOW_CONTROL_DTR_OUTPUT_ENABLED_BIT

or may be set to the following value:

SPP_FLOW_CONTROL_DISABLED_VALUE

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etPort_Send_Port_Information_Confirmation

etPort_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Respond_Port_Information

This function provides a means to respond to a Serial Port Parameters Indication from the remote side.

Prototype:

int BTPSAPI **SPP_Respond_Port_Information**(unsigned int BluetoothStackID, unsigned int SerialPortID, SPP_Port_Information_t *SPPPortInformation)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
SPPPortInformation	Acceptable port information. See description of this structure above in the SPP_Send_Port_Information( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etPort_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Query_Remote_Port_Information

This function provides a means to query the existing Serial Port Parameters from the remote side

Prototype:

int BTPSAPI **SPP_Query_Remote_Port_Information**(unsigned int BluetoothStackID, unsigned int SerialPortID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etPort_Query_Port_Information_Confirmation

etClose_Port_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Respond_Query_Port_Information

This function is used to respond to the etPort_Query_Port_Information_Indication event.

Prototype:

int BTPSAPI **SPP_Respond_Query_Port_Information**(unsigned int BluetoothStackID, unsigned int SerialPortID, SPP_Port_Information_t *SPPPortInformation)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) or SPP_Open_Remote_Port( ) function.
SPPPortInformation	Current port information. See description of this structure above in the SPP_Send_Port_Information( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_SPP_PORT_NOT_OPENED

Possible Events:

etPort_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Get_Configuration_Parameters

This function is used to determine the current SPP parameters that are being used. These parameters are the RFCOMM Frame size that is to be used for incoming/outgoing connections and the size (in bytes) of the default transmit and receive buffers that are used. The transmit and receive buffer sizes are the sizes that are used by default for newly opened SPP Ports (either client or server). The programmer is free to use the SPP_Change_Buffer_Size() function to change the transmit and receive buffer sizes for an existing SPP Port (either client or server).

Prototype:

int BTPSAPI **SPP_Get_Configuration_Parameters**(unsigned int BluetoothStackID, SPP_Configuration_Params_t *SPPConfigurationParams)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SPPConfigurationParams	Pointer to a structure to receive the configuration information.

```
typedef struct
{
   Word_t              MaximumFrameSize;
   unsigned int        TransmitBufferSize;
   unsigned int              ReceiveBufferSize;
} SPP_Configuration_Params_t;
```

where, the MaximumFrameSize is between:

SPP_FRAME_SIZE_MINIMUM
SPP_FRAME_SIZE_MAXIMUM

And TransmitBufferSize and ReceiveBufferSize is between:

SPP_BUFFER_SIZE_MINIMUM
SPP_BUFFER_SIZE_MAXIMUM

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Set_Configuration_Parameters

This function is used to change the current SPP parameters that are to be used for future SPP Ports that are opened. These parameters are the RFCOMM Frame size that is to be used for incoming/outgoing connections and the size (in bytes) of the default transmit and receive buffers that are used. The transmit and receive buffer sizes are the sizes that are used by default for newly opened SPP Ports (either client or server). The programmer is free to use the SPP_Change_Buffer_Size() function to change the transmit and receive buffer sizes for an existing SPP Port (either client or server). This function cannot be called if there exists ANY active SPP Client of Server. In other words, these parameters can only changed when there are no active SPP Server Ports or SPP Client Ports open. Note that for all of the parameters there exists special constants which indicate to use the currently configured parameters.

Prototype:

int BTPSAPI **SPP_Set_Configuration_Parameters**(unsigned int BluetoothStackID,
SPP_Configuration_Params_t *SPPConfigurationParams)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

SPPConfigurationParams	Pointer to a structure that contains the new configuration information.

```
typedef struct
{
  Word_t              MaximumFrameSize;
  unsigned int        TransmitBufferSize;
  unsigned int        ReceiveBufferSize;
} SPP_Configuration_Params_t;
```

where, the MaximumFrameSize is between:

SPP_FRAME_SIZE_MINIMUM
SPP_FRAME_SIZE_MAXIMUM or
SPP_FRAME_SIZE_CURRENT

And TransmitBufferSize and ReceiveBufferSize is between:

SPP_BUFFER_SIZE_MINIMUM
SPP_BUFFER_SIZE_MAXIMUM or
SPP_BUFFER_SIZE_CURRENT

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_SPP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Get_Server_Connection_Mode

This function is responsible for allowing a mechanism to query the SPP Server Connection Mode.

Prototype:

int BTPSAPI **SPP_Get_Server_Connection_Mode**(unsigned int BluetoothStackID, unsigned int SerialPortID, SPP_Server_Connection_Mode_t *SPPServerConnectionMode)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) function.
SPPServerConnectionMode	Pointer to a variable to receive the current Server Connection Mode. The following modes are currently defined. smAutomaticAccept smAutomaticReject smManualAccept

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SPP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Set_Server_Connection_Mode

This function is responsible for allowing a mechanism to change the SPP Server Connection Mode.

Prototype:

int BTPSAPI **SPP_Set_Server_Connection_Mode**(unsigned int BluetoothStackID, unsigned int SerialPortID, SPP_Server_Connection_Mode_t SPPServerConnectionMode)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SerialPortID	The port this command applies to. This is the value that was returned from the SPP_Open_Server_Port( ) function.
SPPServerConnectionMode	The new Server Connection Mode being set. The following modes are currently defined. smAutomaticAccept smAutomaticReject smManualAccept

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SPP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Get_Port_Connection_State

This function is used to determine the current status of a specific SPP Port/RFCOMM Channel for a specific Bluetooth device connection. This function is useful to determine when a RFCOMM Channel has been completely disconnected, as well as to determine when there is an outstanding message on a specific SPP Port/RFCOMM Channel (to aid with new connections).

Prototype:

int BTPSAPI **SPP_Get_Port_Connection_State**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t Channel, Boolean_t LocalPort,
SPP_Port_Connection_State_t *SPP_Port_Connection_State)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Bluetooth device address of the remote Bluetooth device connection that the specified Server Channel is to be queried.
ServerPort	The SPP Port number of the port to query the status of. This value must be either: 0 (to determine if a connection is possible) or be a value between the following constants: SPP_PORT_NUMBER_MINIMUM SPP_PORT_NUMBER_MAXIMUM Note that this value is **NOT** a SPP Port ID (returned from any of the SPP Open functions).
LocalPort	Flag which specifies whether or not the SPP Port in question is a local SPP Server (TRUE) or a remote SPP Port connection (FALSE). Note that in either case, the Bluetooth address **MUST** specify the remotely connected Bluetooth device.
SPP_Port_Connection_State	Pointer to a variable that is to receive the current status for the specified Port. This value returned will be of the following values:

csPortNotPresent
csPortBusy
csPortDisconnecting
csPortReady

Return:

Zero if successful. Note that the SPP_Port_Connection_State variable will only contain a valid value if this function returns success, otherwise the variable will contain an unknown value.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

2. 1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Set_Queuing_Parameters

This function is responsible for setting the lower level data queing parameters. These parameters are used to control the lower level data packet queing thresholds (to improve RAM usage). Specifically, these parameters are used to control aspects of the number of data packets that can be queued into the lower level (per individual channel). This mechanism allows for the flexibility to limit the amount of RAM that is used for streaming type applications (where the remote side has a large number of credits that were granted).

Notes:

This function can only be called when there are NO active connections.

Setting both parameters to zero will disable the queuing mechanism. This means that the number of queued packets will only be limited via the amount of available RAM.

These parameters do not affect the transmit and receive buffers and do not affect any frame sizes and/or credit logic. These parameters ONLY affect the number of simultaneous data packets queued into the lower level.

Prototype:

int BTPSAPI **SPP_Set_ Queuing_Parameters**(unsigned int BluetoothStackID, unsigned int MaximumNumberDataPackets, unsigned int QueuedDataPacketsThreshold)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

MaximumNumberDataPackets	The maximum number of data packets that can be queued into the lower layer simultaneously.
QueuedDataPacketsThreshold	The lower threshold limit that the lower layer should call back to signify that it can queue more data packets for transmission.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Get_Queuing_Parameters

This function is responsible for querying the lower level data queing parameters. These parameters are used to control the lower level data packet queing thresholds (to improve RAM usage). Specifically, these parameters are used to control aspects of the number of data packets that can be queued into the lower level (per individual channel). This mechanism allows for the flexibility to limit the amount of RAM that is used for streaming type applications (where the remote side has a large number of credits that were granted).

Notes:

If both parameters are zero the the queuing mechanism is disabled. This means that the number of queued packets will only be limited via the amount of available RAM.

These parameters do not affect the transmit and receive buffers and do not affect any frame sizes and/or credit logic. These parameters ONLY affect the number of simultaneous data packets queued into the lower level.

Prototype:

int BTPSAPI **SPP_Get_Queuing_Parameters**(unsigned int BluetoothStackID,
unsigned int *MaximumNumberDataPackets,
unsigned int *QueuedDataPacketsThreshold)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

MaximumNumberDataPackets	Buffer that will contain the maximum number of data packets that can be queued into the lower layer simultaneously (if successful).
QueuedDataPacketsThreshold	Buffer that will contain the lower threshold limit that the lower layer should call back to signify that it can queue more data packets for transmission (if successful).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

SPP_Query_Server_Present

This function is responsible for determining if a Serial Port Profile server has been registered (via a successful call to the SPP_Open_Server_Port() function) for the specified RFCOMM server port.

Prototype:

int BTPSAPI **SPP_Query_Server_Present**(unsigned int BluetoothStackID,
Byte_t ServerPort, Boolean_t *ServerPresent)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
ServerPort	The SPP Port number of the port to query the existence of. This value must be a value between the following constants: SPP_PORT_NUMBER_MINIMUM SPP_PORT_NUMBER_MAXIMUM Note that this value is **NOT** a SPP Port ID (returned from any of the SPP Open functions).
ServerPresent	Buffer which will hold the Boolean return value which specifies whether a server is present (TRUE) or is not present (FALSE) for the specified Server Channel.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_SPP_NOT_INITIALIZED
BTPS_ERROR_RFCOMM_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

3.2.2 SPP Event Callback Protoype

The event callback functions mentioned in the SPP commands all accept the callback function described by the following prototype.

SPP_Event_Callback_t

Prototype of callback function passed in one of the SPP open commands.

Prototype:

```
void (BTPSAPI *SPP_Event_Callback_t)(unsigned int BluetoothStackID,
    SPP_Event_Data_t *SPP_Event_Data, unsigned long CallbackParameter)
```

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
SPP_Event_Data	Data describing the event for which the callback function is called. This is defined by the following struture:

```
typedef struct
{
  SPP_Event_Type_t          Event_Data_Type;
  Word_t                    Event_Data_Size;
  union
  {
    SPP_Open_Port_Indication_Data_t
              *SPP_Open_Port_Indication_Data;
    SPP_Open_Port_Confirmation_Data_t
              *SPP_Open_Port_Confirmation_Data;
    SPP_Close_Port_Indication_Data_t
              *SPP_Close_Port_Indication_Data;
    SPP_Port_Status_Indication_Data_t
              *SPP_Port_Status_Indication_Data;
```

```
        SPP_Data_Indication_Data_t
                *SPP_Data_Indication_Data;
        SPP_ Transmit_Buffer_Empty_Indication_Data_t
                *SPP_ Transmit_Buffer_Empty_Indication_Data;
        SPP_Line_Status_Indication_Data_t
                *SPP_Line_Status_Indication_Data;
        SPP_Send_Port_Information_Indication_Data_t
                *SPP_Send_Port_Information_Indication_Data;
        SPP_Send_Port_Information_Confirmation_Data_t
                *SPP_Send_Port_Information_Confirmation_Data;
        SPP_Query_Port_Information_Indication_Data_t
                *SPP_Query_Port_Information_Indication_Data;
        SPP_Query_Port_Information_Confirmation_Data_t
                *SPP_Query_Port_Information_Confirmation_Data;
        SPP_Open_Port_Request_Indication_Data_t
                *SPP_Open_Port_Request_Indication_Data;
    } Event_Data;
} SPP_Event_Data_t;
```

where, Event_Data_Type one of the enumerations of the event types listed in the table in section 3.2.3, and each data structure in the union is described with its event in that section as well.

CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

3.2.3 SPP Events

The possible SPP events from the Bluetooth stack are listed in the table below and are described in the text which follows:

Event	Description
etPort_Open_Indication	Indicate that a Remote Port Open connection has been made.
etPort_Open_Confirmation	Confirm that a Port Open request has been responded to or errored out.
etPort_Close_Port_Indication	Indicate that a port has been closed (unregistered).
etPort_Status_Indication	Indicate that a change in port status has been received.

etPort_Data_Indication	Indicate that data has arrived on a port.
etPort_Transmit_Buffer_Empty_Indication	Indicate when the Transmit Buffer is Empty (only if the Transmit Buffer was completely full or the SPP_Purge_Buffer() function was called with the option to flush the transmit buffer).
etPort_Line_Status_Indication	Indicate that a change in line status has been received.
etPort_Send_Port_Information_Indication	Indicate that a remote device's port parameters have been received (start of negotiation of parameters).
etPort_Send_Port_Information_Confirmation	Confirm that a response has been received to a send port parameters command.
etPort_Query_Port_Information_Indication	Indicate that a request to send current port parameters has been received.
etPort_Query_Port_Information_Confirmation	Confirm that a response has been received to a request to send current port parameters.
etPort_Open_Request_Indication	Indicate that a Remote Port Open request has been received.

etPort_Open_Indication

Indicate that a Remote Port Open connection has been made.

Return Structure:

```
typedef struct
{
   unsigned int      SerialPortID;
   BD_ADDR_t         BD_ADDR;
} SPP_Open_Port_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
BD_ADDR	Address of the Bluetooth device.

etPort_Open_Confirmation

Confirm that a Port Open request has been responded to or errorred out.

Return Structure:

```
typedef struct
{
  unsigned int        SerialPortID;
  unsigned int        PortOpenStatus;
} SPP_Open_Port_Confirmation_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
PortOpenStatus	Status of the open request, one of the following values: SPP_OPEN_PORT_STATUS_SUCCESS SPP_OPEN_PORT_STATUS_CONNECTION_TIMEOUT SPP_OPEN_PORT_STATUS_CONNECTION_REFUSED SPP_OPEN_PORT_STATUS_UNKNOWN_ERROR

etPort_Close_Port_Indication

Indicate that a port has been closed (unregistered).

Return Structure:

```
typedef struct
{
  unsigned int        SerialPortID;
} SPP_Close_Port_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.

etPort_Status_Indication

Indicate that a change in port status has been received.

Return Structure:

```
typedef struct
{
  unsigned int          SerialPortID;
  unsigned int          PortStatus;
  SPP_Break_Status_t    BreakStatus;
  unsigned int          BreakTimeout;
} SPP_Port_Status_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
PortStatus	The current status of the port sent from the remote side; a bit mask that may contain one or more of the following bits: SPP_PORT_STATUS_RTS_CTS_BIT SPP_PORT_STATUS_DTR_DSR_BIT SPP_PORT_STATUS_RING_INDICATOR_BIT

	SPP_PORT_STATUS_CARRIER_DETECT_BIT
BreakStatus	One of the following values: bsBreakCleared bsBreakReceived
BreakTimeout	Value of the Break Timeout, in seconds, if BreakStatus is set to bsBreakReceived.

etPort_Data_Indication

Indicate that data has arrived on a port. Call SPP_Data_Read( ) to retrieve.

Return Structure:

```
typedef struct
{
  unsigned int      SerialPortID;
  Word_t            DataLength;
} SPP_Data_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
DataLength	Length of the data which is waiting to be read.

et Port_Transmit_Buffer_Empty_Indication

Indicate when the Transmit Buffer is Empty (only if the Transmit Buffer was completely full or the SPP_Purge_Buffer() function was called with the option to flush the transmit buffer). This Event is ONLY dispatched when one of two conditions exist:

- The Transmit Buffer has been filled to capacity. This condition can be determined by checking the return value from the SPP_Data_Write() function. When SPP_Data_Write() returns a value greater than or equal to zero AND less than the number of bytes that were requested to be transmitted, the Transmit Buffer is considered full. No more data can be sent through the Serial Port until this event is received (for the specified Port).
- The SPP_Purge_Buffer() function was called and SPP_PURGE_MASK_TRANSMIT_FLUSH_BIT was specified. If this bit was specified and the SPP_Purge_Buffer() function returned zero (success) then this event will be generated when the transmit buffer is empty.

Return Structure:

```
typedef struct
{
  unsigned int      SerialPortID;
} SPP_Transmit_Buffer_Empty_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.

etPort_Line_Status_Indication

Indicate that a change in line status has been received.

Return Structure:

```
typedef struct
{
  unsigned int        SerialPortID;
  unsigned int        SPPLineStatusMask;
} SPP_Line_Status_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
SPPLineStatusMask	Status bits, which may contain one or more of the following bit mask values: SPP_LINE_STATUS_OVERRUN_ERROR_BIT_MASK SPP_LINE_STATUS_PARITY_ERROR_BIT_MASK SPP_LINE_STATUS_FRAMING_ERROR_BIT_MASK Or one may the following value: SPP_LINE_STATUS_NO_ERROR_VALUE

etPort_Send_Port_Information_Indication

Indicate that a remote device's port parameters have been received (start of negotiation of parameters).

Return Structure:

```
typedef struct
{
  unsigned int              SerialPortID;
  SPP_Port_Information_t    SPPPortInformation;
} SPP_Send_Port_Information_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
SPPPortInformation	The port parameters from the remote side, defined by the following structure:

```
typedef struct
{
  unsigned int        PortInformationMask;
  unsigned int        BaudRate;
  unsigned int        DataBits;
  SPP_Stop_Bits_t     StopBits;
  SPP_Parity_t        Parity;
  Byte_t              XOnCharacter;
  Byte_t              XOffCharacter;
  unsigned int        FlowControlMask;
} SPP_Port_Information_t;
```

where PortInformationMask defines which of the port parameters are set, defined by the following bit mask values:

SPP_PORT_INFORMATION_BAUD_RATE_BIT
SPP_PORT_INFORMATION_DATA_BITS_BIT
SPP_PORT_INFORMATION_STOP_BITS_BIT
SPP_PORT_INFORMATION_PARITY_BIT
SPP_PORT_INFORMATION_XON_CHARACTER_BIT
SPP_PORT_INFORMATION_XOFF_CHARACTER_BIT
SPP_PORT_INFORMATION_FLOW_CONTROL_BIT

Or it may be set to the following constant:

SPP_PORT_INFORMATION_NONE_VALUE

BaudRate can be one of the following values:

SPP_BAUD_RATE_MINIMUM
SPP_BAUD_RATE_MAXIMUM

SPP_BAUD_RATE_2400
SPP_BAUD_RATE_4800
SPP_BAUD_RATE_7200
SPP_BAUD_RATE_9600
SPP_BAUD_RATE_19200
SPP_BAUD_RATE_38400
SPP_BAUD_RATE_57600
SPP_BAUD_RATE_115200
SPP_BAUD_RATE_230400

DataBits can be one of the following values:

SPP_DATA_BITS_MINIMUM
SPP_DATA_BITS_MAXIMUM

SPP_DATA_BITS_5
SPP_DATA_BITS_6
SPP_DATA_BITS_7
SPP_DATA_BITS_8

StopBits can be one of the following values:

sbOneStopBit
sbOneOneHalfStopBit

Parity can be one of the following values:

ptNone
ptOdd
ptEven
ptMark
ptSpace

XOnCharacter and XoffCharacter may be any character. However, the following constants are defined in RFCOMM and may be useful for these:

RFCOMM_RPN_PARAMETER_DEFAULT_XON_CHARACTER

RFCOMM_RPN_PARAMETER_DEFAULT_XOFF_CHARACTER

FlowControlMask may contain one or more of the following bit mask values:

SPP_FLOW_CONTROL_XON_XOFF_INPUT_ENABLED_BIT
SPP_FLOW_CONTROL_XON_XOFF_OUTPUT_ENABLED_BIT
SPP_FLOW_CONTROL_CTS_INPUT_ENABLED_BIT
SPP_FLOW_CONTROL_RTS_OUTPUT_ENABLED_BIT
SPP_FLOW_CONTROL_DSR_INPUT_ENABLED_BIT
SPP_FLOW_CONTROL_DTR_OUTPUT_ENABLED_BIT

Or may be set to the following value:

SPP_FLOW_CONTROL_DISABLED_VALUE

etPort_Send_Port_Information_Confirmation

Confirm that a response has been received to a send port parameters command.

Return Structure:

```
typedef struct
{
  unsigned int               SerialPortID;
  SPP_Port_Information_t     SPPPortInformation;
} SPP_Send_Port_Information_Confirmation_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
SPPPortInformation	Port parameters. See etPort_Send_Port_Information_Indication event for a complete listing of this structure.

etPort_Query_Port_Information_Indication

Indicate that a request to send current port parameters has been received.

Return Structure:

```
typedef struct
{
  unsigned int     SerialPortID;
} SPP_Query_Port_Information_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.

etPort_Query_Port_Information_Confirmation

Confirm that a response has been received to a request to send current port parameters.

Return Structure:

```
typedef struct
{
  unsigned int           SerialPortID;
  SPP_Port_Information_t SPPPortInformation;
} SPP_Query_Port_Information_Confirmation_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
SPPPortInformation	Port parameters. See etPort_Send_Port_Information_Indication event for a complete listing of this structure.

etPort_Open_Request_Indication

Indicate that a Remote Port Open request has been received.

Notes:

1. When using this feature Bluetopia requires that a response be sent to a device requesting a connection within sixty seconds. If a response is not sent within this time a negative response will be sent to the device. Since this timeout is implementation specific the requesting device may timeout and disconnect sooner then Bluetopia.

Return Structure:

```
typedef struct
{
  unsigned int   SerialPortID;
  BD_ADDR_t      BD_ADDR;
} SPP_Open_Port_Request_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
BD_ADDR	Address of the Bluetooth device.

3.3 GOEP Programming Interface

The GOEP (Generic Object Exchange Profile) programming interface defines the protocols and procedures to be used to implement Object Exchange (OBEX) capabilities such as folder synchronization, file transfer, and Object Push activities. The GOEP commands are listed in section 3.3.1, the event callback prototype is described in section 3.3.2, and the GOEP events are itemized in section 3.3.3. The actual prototypes and constants outlined in this section can be found in the **GOEPAPI.H** header file in the Bluetopia distribution.

3.3.1 GOEP Commands

The available GOEP command functions are listed in the table below and are described in the text which follows.

Function	Description
GOEP_Open_Server_Port	Establish server port to wait for connections
GOEP_Close_Server_Port	Close an open port
GOEP_Open_Port_Request_Response	Respond to a port open request from the remote device.
GOEP_Register_SDP_Record	Add a generic SDP Service Record to the SDP database
GOEP_Register_Raw_SDP_Record	Adds a generic SDP Service Record to the SDP database with only pre-parsed protocol list data possibly added by the caller.
GOEP_Open_Remote_Port	Open a serial port to a remote device.
GOEP_Close_Port	Close either a server port or a remote port.
GOEP_Connect_Request	Request a connection with a remote OBEX server.
GOEP_Disconnect_Request	Close an OBEX server connection.
GOEP_Put_Request	Push a data Object to a remote OBEX server.
GOEP_Get_Request	Pull a data Object from a remote OBEX server
GOEP_Set_Path_Request	Set the current folder for Put/Get Requests.
GOEP_Abort_Request	Abort the current Put/Get Request.
GOEP_Command_Response	Send a response back to the remote OBEX entity (typically the client of the connection).
GOEP_Get_Server_Connection_Mode	Query the current Server Connection Mode.
GOEP_Set_Server_Connection_Mode	Change the current Server Connection Mode.
GOEP_Find_Application_Parameter_Header_By_Tag_ID	Traverses hidApplicationParameters Header types and attempts to match the Tag ID

GOEP_Find_Header	Scans through an array of headers for the header ID type that was specified.
GOEP_Generate_Digest_Nonce	Generates the MD5 Hash of the two pieces required for OBEX Authentication.

GOEP_Open_Server_Port

This function is responsible for establishing a GOEP Port Server (OBEX server) which will wait for a connection to occur on the port established by this function.

Prototype:

int BTPSAPI **GOEP_Open_Server_Port**(unsigned int BluetoothStackID,
unsigned int ServerPort, Word_t MaxPacketLength, GOEP_Event_Callback_t
GOEP_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
ServerPort	Port number to use. This must fall in the range defined by the following constants: SPP_PORT_NUMBER_MINIMUM SPP_PORT_NUMBER_MAXIMUM
MaxPacketLength	Max packet length that will be accepted by this server.
GOEP_Event_Callback	Function to call when events occur on this port.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive, non-zero if successful. The return value will be the GOEP_ID for the server port that was successfully opened. *This* is the value that should be used in all subsequent function calls (except another GOEP_Open_Server_Port( ) call).

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Port_Open_Request_Indication

etOBEX_Port_Open_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Close_Server_Port

This function is responsible for Unregistering a Serial Port Server which was registered by a successful call to the GOEP_Open_Server_Port( ) function. Note, this function does NOT delete any SDP Service Record Handles (i.e., added via a GOEP_Register_SDP_Record( ) function call).

Prototype:

int BTPSAPI **GOEP_Close_Server_Port**(unsigned int BluetoothStackID,
unsigned int GOEP_ID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port to close. This is the value that was returned from the GOEP_Open_Server_Port( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Open_Port_Request_Response

This function is responsible for responding to requests to connect to a OBEX Port Server.

Notes:

1. When using this feature Bluetopia requires that a response be sent to a device requesting a connection within sixty seconds. If a response is not sent within this time a negative response will be sent to the device. Since this timeout is implementation specific the requesting device may timeout and disconnect sooner then Bluetopia.

Prototype:

int BTPSAPI **GOEP_Open_Port_Request_Response**(unsigned int BluetoothStackID, unsigned int GOEP_ID, Boolean_t AcceptConnection)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port this command applies to. This is the value that was returned from the GOEP_Open_Server_Port( ) function.
AcceptConnection	Boolean indicating if the pending connection should be accepted.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Port_Open_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Register_SDP_Record

This function provides a means to add a generic SDP Service Record to the SDP Database.

Notes:

1. This function should only be called with the GOEP_ID that was returned from the GOEP_Open_Server_Port( ) function. This function should **never** be used with the Serial Port ID returned from the GOEP_Open_Remote_Port( ) function.
2. The Service Record Handle that is returned from this function will remain in the SDP Record Database until it is deleted by calling the SDP_Delete_Service_Record( ) function. A Macro is provided to delete the Service Record from the SDP Database. This Macro maps GOEP_Un_Register_SDP_Record**( )** to SDP_Delete_Service_Record(), and is defined as follows:

GOEP_Un_Register_SDP_Record(__BluetoothStackID, __GOEP_ID, __SDPRecordHandle)

3. There must be UUID Information specified in theSDPServiceRecord Parameter, however protocol information is completely optional. Any Protocol Information that is specified (if any) will be added in the Protocol Attribute AFTER the default OBEX Protocol List (L2CAP, RFCOMM, and OBEX).
4. The Service Name is always added at Attribute ID 0x0100. A Language Base Attribute ID List is created that specifies that 0x0100 is UTF-8 Encoded, English Language.

Prototype:

int BTPSAPI **GOEP_Register_SDP_Record**(unsigned int BluetoothStackID, unsigned int GOEP_ID, GOEP_SDP_Service_Record_t *SDPServiceRecord, char *ServiceName, DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port this command applies to. This is the value that was returned from the GOEP_Open_Server_Port( ) function.
SDPServiceRecord	Any additional Service Discovery Protocol information to be added to the record for this serial port server. This is a structured defined as:

```
typedef struct
{
   unsigned int            NumberServiceClassUUID;
   SDP_UUID_Entry_t        *SDPUUIDEntries;
   SDP_Data_Element_t      *ProtocolList;
} GOEP_SDP_Service_Record_t;
```

ServiceName	Name to appear in the SDP Database for this service.
SDPServiceRecordHandle	Returned handle to the SDP Database entry which may be used to remove the entry at a later time.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Register_Raw_SDP_Record

This function provides a means to add a generic SDP Service Record to the SDP Database. This function is identical to the **GOEP_Register_SDP_Record**() with the exception that any additional information to be added to the Protocol Attribute must be in a pre-parsed format.

Notes:

1. This function should only be called with the GOEP_ID that was returned from the GOEP_Open_Server_Port( ) function. This function should **never** be used with the Serial Port ID returned from the GOEP_Open_Remote_Port( ) function.
2. The Service Record Handle that is returned from this function will remain in the SDP Record Database until it is deleted by calling the SDP_Delete_Service_Record( ) function. A Macro is provided to delete the Service Record from the SDP Database. This Macro maps GOEP_Un_Register_SDP_Record**( )** to SDP_Delete_Service_Record(), and is defined as follows:

GOEP_Un_Register_SDP_Record(__BluetoothStackID, __GOEP_ID, __SDPRecordHandle)

3. There must be UUID Information specified in theSDPServiceRecord Parameter, however protocol information is completely optional. Any Protocol Information that is specified (if any) will be added in the Protocol Attribute AFTER the default OBEX Protocol List (L2CAP, RFCOMM, and OBEX).
4. The Service Name is always added at Attribute ID 0x0100. A Language Base Attribute ID List is created that specifies that 0x0100 is UTF-8 Encoded, English Language.

Prototype:

int BTPSAPI **GOEP_Register_Raw_SDP_Record**(unsigned int BluetoothStackID,
 unsigned int GOEP_ID,
 GOEP_SDP_Raw_Service_Record_t *SDPServiceRecord,
 char *ServiceName,
 DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port this command applies to. This is the value that was returned from the GOEP_Open_Server_Port( ) function.
SDPServiceRecord	Contains any additional Service Discovery Protocol information to be added to the record for this serial port server. This is a structured defined as:

```
typedef struct
{
   unsigned int           NumberServiceClassUUID;
   SDP_UUID_Entry_t      *SDPUUIDEntries;
   unsigned int           NumberOfProtocolDataListUUIDOffsets;
   Word_t                *ProtocolDataListUUIDOffsets;
   unsigned int           ProtocolDataListLength;
   Byte_t                *ProtocolDataList;
} GOEP_SDP_Raw_Service_Record_t;
```

ServiceName	Name to appear in the SDP Database for this service.
SDPServiceRecordHandle	Returned handle to the SDP Database entry which may be used to remove the entry at a later time.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Open_Remote_Port

This function is used to open a remote serial port on the specified Remote Device.

Prototype:

int BTPSAPI **GOEP_Open_Remote_Port**(unsigned int BluetoothStackID,
    BD_ADDR_t BD_ADDR, unsigned int ServerPort, Word_t MaxPacketLength
    GOEP_Event_Callback_t GOEP_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the Bluetooth device to connect with.
ServerPort	The remote device's server port ID to connect with.
MaxPacketLength	The largest packet that will be sent on this connection. Each side must support a minimum of 255 bytes, and cannot have a

	packet size greater than 64K-1 bytes. These constraints are defined as the constants: OBEX_PACKET_LENGTH_MINIMUM OBEX_PACKET_LENGTH_MAXIMUM
GOEP_Event_Callback	Function to call when events occur on this port.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive, non-zero if successful. The return value will be the GOEP_ID for the port that was successfully opened. *This* is the value that should be used in all subsequent function calls.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED
RFCOMM_UNABLE_TO_CONNECT_TO_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE

Possible Events:

etOBEX_Port_Open_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Close_Port

This function is used to close a GOEP Port that was previously opened with the GOEP_Open_Server_Port( ) function ***or*** the GOEP_Open_Remote_Port( ) function. This function does **not** unregister a GOEP Server Port from the system, it only disconnects any connection that is currently active on the Server Port. The GOEP_Close_Server_Port() function can be used to Unregister the GOEP Server Port.

Prototype:

int BTPSAPI **GOEP_Close_Port**(unsigned int BluetoothStackID, unsigned int GOEP_ID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

GOEP_ID	The port to close. This is the value that was returned from the GOEP_Open_Server_Port( ) or GOEP_Open_Remote_Port( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

```
BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED
```

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Connect_Request

Make a connection to a remote OBEX Server.

Prototype:

int BTPSAPI **GOEP_Connect_Request**(unsigned int BluetoothStackID,
  unsigned int GOEP_ID, OBEX_Header_List_t *Header_List);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port to use for the connection.
Header_List	A pointer to an array of optional headers. This parameter is defined by the following structure:

```
typedef struct
{
   Byte_t                          NumberOfHeaders;
   OBEX_Header_t                   *Headers;
} OBEX_Header_List_t;
```

where OBEX_Header_t is defined as:

```
typedef struct
{
   OBEX_Header_ID_t                OBEX_Header_ID;
   OBEX_Header_Type_t              OBEX_Header_Type;
   union
   {
      Byte_t                      OneByteValue;
      DWord_t                     FourByteValue;
      OBEX_Byte_Sequence_t        ByteSequence;
      OBEX_Word_Sequence_t        UnicodeText;
   } Header_Value;
} OBEX_Header_t;
```

where OBEX_Header_ID may be one of the following enumeration values:

hidCount, hidName, hidType, hidLength, hidTime, hidDescription, hidTarget, hidHTTP, hidBody, hidEndOfBody, hidWho, hidConnectionID, hidApplicationParameters, hidAuthenticationChallenge, hidAuthenticationResponse, hidObjectClass

and OBEX_Header_Type defines the format of the header and may be one of the following enumeration values:

htUnsignedInteger1Byte
htUnsignedInteger4Byte
htNullTerminatedUnicodeText
htByteSequence

The Header_Value union contains the value for fixed length formats or pointers to variable length format headers. The sequence structures shown in this union are defined as:

```
typedef struct
{
   Word_t           DataLength;
   Byte_t           *ValuePointer;
} OBEX_Byte_Sequence_t;
```

```
typedef struct
{
   Word_t           DataLength;
   Word_t           *ValuePointer;
} OBEX_Word_Sequence_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Connect_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Disconnect_Request

Break a connection made with GOEP_Connect_Request( ). This function may be called from either the client or the server side of the connection.

Prototype:

int BTPSAPI **GOEP_Disconnect_Request**(unsigned int BluetoothStackID,
    unsigned int GOEP_ID, OBEX_Header_List_t *Header_List);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port to close the connection on. This is the value that was returned from either the GOEP_Open_Remote_Port( ) or GOEP_Open_Server_Port( ) function.
Header_List	A pointer to an array of optional headers. See GOEP_Connect_Request( ) for a description of the headers.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Disconnect_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Put_Request

Push a data Object onto the remote OBEX server. The body of the object is contained in the Header_List passed.

Notes:

1. A file can be deleted on the Server with the Put Request by placing the name of the file in the Name header (hidName) and omitting a body (hidBody).
2. An empty folder may be deleted in the same manner as the file delete in Note 1. On some servers, it may also be possible to delete a folder with files in it by this method, but others may not allow this operation, returning a "Precondition Failed" (0xCC) response code.

Prototype:

int BTPSAPI **GOEP_Put_Request**(unsigned int BluetoothStackID, unsigned int GOEP_ID, Boolean_t Final, OBEX_Header_List_t *Header_List);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port to send the Put Request to. This is the value that was returned from the GOEP_Open_Remote_Port( ) function.
Final	Flag which indicates if this is the last packet of the Put sequence or not.
Header_List	A pointer to an array of OBEX headers. This is the data to send. See GOEP_Connect_Request( ) for a description of the headers.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Put_Confirmation
etOBEX_Disconnect_Indication
etOBEX_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Get_Request

Pull a data Object from the remote OBEX server.

Prototype:

int BTPSAPI **GOEP_Get_Request**(unsigned int BluetoothStackID, unsigned int GOEP_ID, Boolean_t Final, OBEX_Header_List_t *Header_List);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port to send the Get Request to. This is the value that was returned from the GOEP_Open_Remote_Port( ) function.
Final	Flag which indicates when all the headers have been sent over and the Server should start sending the object data.
Header_List	A pointer to an optional array of OBEX headers. This is the data to be retrieved, and is only optional on the final call. See GOEP_Connect_Request( ) for a description of the headers.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Get_Confirmation
etOBEX_Disconnect_Indication
etOBEX_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Set_Path_Request

Change the current folder on the Server for subsequent Put and Get Requests. If a folder name is supplied that doesn't exist on the Server, a new folder will be created before the Server changes to that folder.

Prototype:

int BTPSAPI **GOEP_Set_Path_Request**(unsigned int BluetoothStackID, unsigned int GOEP_ID, Byte_t Flags, OBEX_Header_List_t *Header_List);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port to send the Set Path Request to. This is the value that was returned from the GOEP_Open_Remote_Port( ) function.
Flags	Flags to control folder navigation and creation. Possible values are: OBEX_SET_PATH_FLAGS_BACKUP_MASK OBEX_SET_PATH_FLAGS_NO_CREATE_MASK
Header_List	A pointer to an array of OBEX headers. The path to change to should be provided in a hidName type header. See GOEP_Connect_Request( ) for a description of the headers.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Set_Path_Confirmation
etOBEX_Disconnect_Indication
etOBEX_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Abort_Request

Abort any Get or Put Request in progress.

Prototype:

int BTPSAPI **GOEP_Abort_Request**(unsigned int BluetoothStackID, unsigned int GOEP_ID, OBEX_Header_List_t *Header_List);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

GOEP_ID	The port to send the Abort Request to. This is the value that was returned from either the GOEP_Open_Remote_Port( ) or the GOEP_Open_Server_Port( ) function.
Header_List	A pointer to an array of OBEX headers. See GOEP_Connect_Request( ) for a description of the headers.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Abort_Confirmation
etOBEX_Disconnect_Indication
etOBEX_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Command_Response

Return a response to a GOEP command.

Prototype:

int BTPSAPI **GOEP_Command_Response**(unsigned int BluetoothStackID,
unsigned int GOEP_ID, Byte_t ResponseCode, OBEX_Header_List_t *Header_List);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port to send the Comamnd Response to. This is the value that was provided in the event being responded to.
ResponseCode	Response code to return to the requester. This code is a logical ORing of the Final status flag (0x80 or the constant: OBEX_FINAL_BIT) with one of the following possible status values (all less than 0x7F).

OBEX_CONTINUE_RESPONSE
OBEX_OK_RESPONSE
OBEX_CREATED_RESPONSE
OBEX_ACCEPTED_RESPONSE

OBEX_NON_AUTHORITATIVE_INFORMATION_RESPONSE
OBEX_NO_CONTENT_RESPONSE
OBEX_RESET_CONTENT_RESPONSE
OBEX_PARTIAL_CONTENT_RESPONSE
OBEX_MULTIPLE_CHOICES_RESPONSE
OBEX_MOVED_PERMANETLY_RESPONSE
OBEX_MOVED_TEMPORARILY_RESPONSE
OBEX_SEE_OTHER_RESPONSE
OBEX_NOT_MODIFIED_RESPONSE
OBEX_USE_PROXY_RESPONSE
OBEX_BAD_REQUEST_RESPONSE
OBEX_UNAUTHORIZED_RESPONSE
OBEX_PAYMENT_REQUIRED_RESPONSE
OBEX_FORBIDDEN_RESPONSE
OBEX_NOT_FOUND_RESPONSE
OBEX_METHOD_NOT_ALLOWED_RESPONSE
OBEX_NOT_ACCEPTABLE_RESPONSE
OBEX_PROXY_AUTHENTICATION_REQUIRED_RESPONSE
OBEX_REQUEST_TIMEOUT_RESPONSE
OBEX_CONFLICT_RESPONSE
OBEX_GONE_RESPONSE
OBEX_LENGTH_REQUIRED_RESPONSE
OBEX_PRECONDITION_FAILED_RESPONSE
OBEX_REQUESTED_ENTITY_TOO_LARGE_RESPONSE
OBEX_REQUESTED_URL_TOO_LARGE_RESPONSE
OBEX_UNSUPORTED_MEDIA_TYPE_RESPONSE
OBEX_INTERNAL_SERVER_ERROR_RESPONSE
OBEX_NOT_IMPLEMENTED_RESPONSE
OBEX_BAD_GATEWAY_RESPONSE
OBEX_SERVICE_UNAVAILABLE_RESPONSE
OBEX_GATEWAY_TIMEOUT_RESPONSE
OBEX_HTTP_VERSION_NOT_SUPPORTED_RESPONSE
OBEX_DATABASE_FULL_RESPONSE
OBEX_DATABASE_LOCKED_RESPONSE

Header_List	Optional list of headers to be passed with the command response (e.g., return data object requested).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etOBEX_Disconnect_Indication
etOBEX_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Get_Server_Connection_Mode

This function is responsible for allowing a mechanism to query the OBEX Port Server Connection Mode.

Prototype:

int BTPSAPI **GOEP_Get_Server_Connection_Mode**(unsigned int BluetoothStackID, unsigned int GOEP_ID, SPP_Server_Connection_Mode_t *ServerConnectionMode)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port this command applies to. This is the value that was returned from the GOEP_Open_Server_Port( ) function.
ServerConnectionMode	Pointer to a variable to receive the current Server Connection Mode. The following modes are currently defined. smAutomaticAccept smAutomaticReject smManualAccept

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Set_Server_Connection_Mode

This function is responsible for allowing a mechanism to change the OBEX Port Server Connection Mode.

Prototype:

int BTPSAPI **GOEP_Set_Server_Connection_Mode**(unsigned int BluetoothStackID, unsigned int GOEP_ID, SPP_Server_Connection_Mode_t ServerConnectionMode)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
GOEP_ID	The port this command applies to. This is the value that was returned from the GOEP_Open_Server_Port( ) function.
ServerConnectionMode	The new Server Connection Mode being set. The following modes are currently defined. smAutomaticAccept smAutomaticReject smManualAccept

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

GOEP_Find_Application_Parameter_Header_By_Tag_ID

Given a pointer to a list of headers this function will traverse the hidApplicationParameters Header types and match the Tag ID to one of the Triplets.

Prototype:

OBEX_Application_Parameters_t *BTPSAPI **GOEP_Find_Application_Parameter_Header_By_Tag_ID**(OBEX_Header_List_t *HeaderListPtr, Byte_t TagID)

Parameters:

HeaderListPtr	Pointer to list of OBEX Headers
TagID	The Tag ID to attempt to match in the header.

Return:

If successful, pointer to the OBEX Application Parameters structure which was matched. Its specification is below:

```
typedef __PACKED_STRUCT_BEGIN__ struct
{
  Byte_t Tag;
  Byte_t Length;
  Byte_t Value[1];
} __PACKED_STRUCT_END__ OBEX_Application_Parameters_t;
```

If not found or an error occurs, NULL is returned.

Possible Events:

Notes:

GOEP_Find_Header

The following function is used to scan through an array of headers for the header ID type that was specified.

Prototype:

int BTPSAPI **GOEP_Find_Header**(OBEX_Header_ID_t HeaderID,
OBEX_Header_List_t *ListPtr)

Parameters:

HeaderID	Header ID to search for. May be one of the following: hidCount hidName hidType hidLength hidTime hidDescription hidTarget hidHTTP hidBody hidEndOfBody hidWho hidConnectionID hidApplicationParameters hidAuthenticationChallenge hidAuthenticationResponse hidObjectClass
ListPtr	Pointer to header list to search for HeaderID.

Return:

If successful returns the index into Header list of the matched Header.

If not successful, returns negative value.

Possible Events:

Notes:

GOEP_Generate_Digest_Nonce

The following function is used to generate the MD5 Hash of the two pieces required for OBEX Authentication. The two pieces refer to the first part of the data to be MD5 hashed before the OBEX Delimeter and the second part of the data to be MD5 hashed after the OBEX Delimeter. The OBEX Delimeter used by this function is defined as:

OBEX_DIGEST_CHALLENGE_RESPONSE_NONCE_MD5_DELIMETER_BYTE

The first and second parts *MUST* be specified and cannot of zero length. The MD5 Hash is returned (as an OBEX_Nonce_t) in the buffer passed as the final parameter to this function (this parameter also *MUST* be specified and cannot be NULL). NOTE, as an example (using simple ASCII strings):

```
GOEP_Generate_Digest_Nonce(4, "ABCD", 5, "WXYZ", &N);
```

would calculate the MD5 Hash of the following 9 bytes:

```
ABCD:WXYZ
```

and return this in the buffer pointed to by N. Note that the ':' character is assumed to be the Delimeter constant mentioned above.

Prototype:

int BTPSAPI **GOEP_Generate_Digest_Nonce**(unsigned int PreDelimeterLength, Byte_t *PreDelimeterData, unsigned int PostDelimeterLength, Byte_t *PostDelimeterData, OBEX_Nonce_t *OutputNonce)

Parameters:

PreDelimeterLength	Number of bytes in the byte array pointed to by PreDelimeterData.
PreDelimeterData	The byte array buffer that holds the piece that will MD5 hashed before the OBEX Delimeter.
PostDelimeterLength	Number of bytes in the byte array pointed to by PostDelimeterData.
PostDelimeterData	The byte array buffer that holds the piece that will MD5 hashed afer the OBEX Delimeter.
OutputNonce	Buffer to hold the returned MD5 hash. Must not be NULL.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INSUFFICIENT_RESOURCES

Possible Events:

Notes:

3.3.2 GOEP Event Callback Protoype

The event callback functions mentioned in the GOEP Open commands all accept the callback function described by the following prototype.

GOEP_Event_Callback_t

Prototype of callback function passed in one of the GOEP open commands.

Prototype:

void (BTPSAPI ***GOEP_Event_Callback_t**)(unsigned int BluetoothStackID,
GOEP_Event_Data_t *GOEP_Event_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1] Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

GOEP_Event_Data Data describing the event for which the callback function is called. This is defined by the following struture:

```
typedef struct
{
  OBEX_Event_Data_Type_t                Event_Data_Type;
  Word_t                                Event_Data_Size;
  union
  {
    OBEX_Port_Open_Indication_Data_t       *OBEX_Port_Open_Indication_Data;
    OBEX_Port_Open_Confirmation_Data_t     *OBEX_Port_Open_Confirmation_Data;
    OBEX_Port_Close_Indication_Data_t      *OBEX_Port_Close_Indication_Data;
    OBEX_Connect_Indication_Data_t         *OBEX_Connect_Indication_Data;
    OBEX_Connect_Confirmation_Data_t       *OBEX_Connect_Confirmation_Data;
    OBEX_Disconnect_Indication_Data_t      *OBEX_Disconnect_Indication_Data;
    OBEX_Disconnect_Confirmation_Data_t    *OBEX_Disconnect_Confirmation_Data;
    OBEX_Put_Indication_Data_t             *OBEX_Put_Indication_Data;
    OBEX_Put_Confirmation_Data_t           *OBEX_Put_Confirmation_Data;
    OBEX_Get_Indication_Data_t             *OBEX_Get_Indication_Data;
    OBEX_Get_Confirmation_Data_t           *OBEX_Get_Confirmation_Data;
    OBEX_Set_Path_Indication_Data_t        *OBEX_Set_Path_Indication_Data;
    OBEX_Set_Path_Confirmation_Data_t      *OBEX_Set_Path_Confirmation_Data;
    OBEX_Abort_Indication_Data_t           *OBEX_Abort_Indication_Data;
    OBEX_Abort_Confirmation_Data_t         *OBEX_Abort_Confirmation_Data;
    OBEX_Port_Open_Request_Indication_Data_t
              *OBEX_Port_Open_Request_Indication_Data;
  } Event_Data;
} GOEP_Event_Data_t;
```

where, Event_Data_Type is one of the enumerations of the event types listed in the table in section 3.3.3, and each data structure in the union is described with its event in that section as well.

CallbackParameter User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

3.3.3 GOEP Events

The possible GOEP events from the Bluetooth stack are listed in the table below and are described in the text which follows:

Event	Description
etOBEX_Port_Open_Indication	Indicate that a Remote Port Open connection has been made.
etOBEX_Port_Open_Confirmation	Confirm that a Port Open request has been responded to or has errored out.
etOBEX_Port_Open_Request_Indication	Indicate that a Remote Port Open request has been received.
etOBEX_Port_Close_Indication	Indicate that a port has been closed (unregistered).
etOBEX_Connect_Indication	Indicate that a Connect Request has been received.
etOBEX_Connect_Confirmation	Confirm that a Connect Request has been responded to or has errored out
etOBEX_Disconnect_Indication	Indicate that a Disconnect Request has been received.
etOBEX_Disconnect_Confirmation	Confirm that a Disconnect Request has been responded to or has errored out
etOBEX_Put_Indication	Indicate that a Put Request has been received.
etOBEX_Put_Confirmation	Confirm that a Put Request has been responded to or has errored out
etOBEX_Get_Indication	Indicate that a Get Request has been received.
etOBEX_Get_Confirmation	Confirm that a Get Request has been responded to or has errored out
etOBEX_Set_Path_Indication	Indicate that a Set Path Request has been received.
etOBEX_Set_Path_Confirmation	Confirm that a Set Path Request has been responded to or has errored out
etOBEX_Abort_Indication	Indicate that an Abort Request has been received.
etOBEX_Abort_Confirmation	Confirm that an Abort Request has been responded to or has errored out

Several of the events return a Response_Code. This code is a logical ORing of the Final status flag (0x80 or constant: OBEX_FINAL_BIT) with one of the following possible status values (all less than 0x7F).

OBEX_CONTINUE_RESPONSE
OBEX_OK_RESPONSE
OBEX_CREATED_RESPONSE
OBEX_ACCEPTED_RESPONSE
OBEX_NON_AUTHORITATIVE_INFORMATION_RESPONSE
OBEX_NO_CONTENT_RESPONSE
OBEX_RESET_CONTENT_RESPONSE
OBEX_PARTIAL_CONTENT_RESPONSE
OBEX_MULTIPLE_CHOICES_RESPONSE
OBEX_MOVED_PERMANETLY_RESPONSE
OBEX_MOVED_TEMPORARILY_RESPONSE
OBEX_SEE_OTHER_RESPONSE
OBEX_NOT_MODIFIED_RESPONSE
OBEX_USE_PROXY_RESPONSE
OBEX_BAD_REQUEST_RESPONSE
OBEX_UNAUTHORIZED_RESPONSE
OBEX_PAYMENT_REQUIRED_RESPONSE
OBEX_FORBIDDEN_RESPONSE
OBEX_NOT_FOUND_RESPONSE
OBEX_METHOD_NOT_ALLOWED_RESPONSE
OBEX_NOT_ACCEPTABLE_RESPONSE
OBEX_PROXY_AUTHENTICATION_REQUIRED_RESPONSE
OBEX_REQUEST_TIMEOUT_RESPONSE
OBEX_CONFLICT_RESPONSE
OBEX_GONE_RESPONSE
OBEX_LENGTH_REQUIRED_RESPONSE
OBEX_PRECONDITION_FAILED_RESPONSE
OBEX_REQUESTED_ENTITY_TOO_LARGE_RESPONSE
OBEX_REQUESTED_URL_TOO_LARGE_RESPONSE
OBEX_UNSUPORTED_MEDIA_TYPE_RESPONSE
OBEX_INTERNAL_SERVER_ERROR_RESPONSE
OBEX_NOT_IMPLEMENTED_RESPONSE
OBEX_BAD_GATEWAY_RESPONSE
OBEX_SERVICE_UNAVAILABLE_RESPONSE
OBEX_GATEWAY_TIMEOUT_RESPONSE
OBEX_HTTP_VERSION_NOT_SUPPORTED_RESPONSE
OBEX_DATABASE_FULL_RESPONSE
OBEX_DATABASE_LOCKED_RESPONSE

etOBEX_Port_Open_Indication

Indicate that a Remote Port Open connection has been made.

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  BD_ADDR_t             BD_ADDR;
} OBEX_Port_Open_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
BD_ADDR	Address of the Bluetooth device making the request.

etOBEX_Port_Open_Confirmation

Confirm that a Port Open request has been responded to or has errored out.

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  unsigned int          PortOpenStatus;
} OBEX_Port_Open_Confirmation_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
PortOpenStatus	One of the following possible status values: GOEP_OPEN_PORT_STATUS_SUCCESS GOEP_OPEN_PORT_STATUS_CONNECTION_TIMEOUT GOEP_OPEN_PORT_STATUS_CONNECTION_REFUSED GOEP_OPEN_PORT_STATUS_UNKNOWN_ERROR

etOBEX_Port_Close_Indication

Indicate that a port has been closed (unregistered).

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
} OBEX_Port_Close_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.

etOBEX_Connect_Indication

Indicate that a Connect Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int           GOEP_ID;
  Byte_t                 Version_Number;
  Word_t                 Max_Packet_Length;
  OBEX_Header_List_t     Header_List;
} OBEX_Connect_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Version_Number	Version of the OBEX used by the connection requester.
Max_Packet_Length	The maximum packet length supported by the requester. This is non-negotiable and may be different than what the responder supports. Each side must support a minimum of 255 bytes, and cannot have a packet size greater than 64K-1 bytes. These constraints are defined as the constants: OBEX_PACKET_LENGTH_MINIMUM OBEX_PACKET_LENGTH_MAXIMUM
Header_List	Optional list of headers passed with the Connect Request.

etOBEX_Connect_Confirmation

Confirm that a Connect Request has been responded to or has errored out

Return Structure:

```
typedef struct
{
  unsigned int           GOEP_ID;
  Byte_t                 Response_Code;
  Byte_t                 Version_Number;
  Byte_t                 Flags;
  Word_t                 Max_Packet_Length;
  OBEX_Header_List_t     Header_List;
} OBEX_Connect_Confirmation_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Response_Code	One of the values indicated near the beginning of this section.
Version_Number	Version of the OBEX used by the connection requester.
Flags	Used to indicate whether the Server can support multiple connections or not. Possible values are as follows: OBEX_CONNECTION_FLAGS_RESPONSE_MULTIPLE_IRLMP_CONNECTIONS
Max_Packet_Length	The maximum packet length supported by the requester. This is non-negotiable and may be different than what the responder

	supports. Each side must support a minimum of 255 bytes, and cannot have a packet size greater than 64K-1 bytes.
Header_List	Optional list of headers passed with the Connect Request.

etOBEX_Disconnect_Indication

Indicate that a Disconnect Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  OBEX_Header_List_t    Header_List;
} OBEX_Disconnect_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Header_List	Optional list of headers passed with the Disconnect Request.

etOBEX_Disconnect_Confirmation

Confirm that a Disconnect Request has been responded to or has errored out

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  Byte_t                Response_Code;
  OBEX_Header_List_t    Header_List;
} OBEX_Disconnect_Confirmation_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Response_Code	One of the values indicated near the beginning of this section.
Header_List	Optional list of headers passed with the Disconnect Request.

etOBEX_Put_Indication

Indicate that a Put Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  Boolean_t             Final_Flag;
  OBEX_Header_List_t    Header_List;
} OBEX_Put_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Final_Flag	Whether this is the last packet in a multi-packet Put Request or not.
Header_List	List of headers. The body of the object being pushes is included (hidBody type header).

etOBEX_Put_Confirmation

Confirm that a Put Request has been responded to or has errored out

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  Byte_t                Response_Code;
  OBEX_Header_List_t    Header_List;
} OBEX_Put_Confirmation_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Response_Code	One of the values indicated near the beginning of this section.
Header_List	List of headers passed with the Put Request.

etOBEX_Get_Indication

Indicate that a Get Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  Boolean_t             Final_Flag;
  OBEX_Header_List_t    Header_List;
} OBEX_Get_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.

Final_Flag	Whether this is the last packet in a multi-packet Get Request or not.
Header_List	List of headers.

etOBEX_Get_Confirmation

Confirm that a Get Request has been responded to or has errored out

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  Byte_t                Response_Code;
  OBEX_Header_List_t    Header_List;
} OBEX_Get_Confirmation_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Response_Code	One of the values indicated near the beginning of this section.
Header_List	Optional list of headers.

etOBEX_Set_Path_Indication

Indicate that a Set Path Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  Boolean_t             CreateDirectory;
  Boolean_t             Backup;
  OBEX_Header_List_t    Header_List;
} OBEX_Set_Path_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
CreateDirectory	Whether the folder indicated (in the Header_List) should be created if it doesn't exist.
Backup	Go back up one level in the directory tree.
Header_List	List of headers sent with the Set Path Request, e.g., the name (hidName) of the Path.

etOBEX_Set_Path_Confirmation

Confirm that a Set Path Request has been responded to or has errored out

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  Byte_t                Response_Code;
  OBEX_Header_List_t    Header_List;
} OBEX_Set_Path_Confirmation_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Response_Code	One of the values indicated near the beginning of this section.
Header_List	List of headers passed with the Set Path Request, e.g., the name (hidName) of the Path.

etOBEX_Abort_Indication

Indicate that an Abort Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  OBEX_Header_List_t    Header_List;
} OBEX_Abort_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Header_List	Optional list of headers passed with the Abort Request.

etOBEX_Abort_Confirmation

Confirm that an Abort Request has been responded to or has errored out

Return Structure:

```
typedef struct
{
  unsigned int          GOEP_ID;
  Byte_t                Response_Code;
  OBEX_Header_List_t    Header_List;
} OBEX_Abort_Confirmation_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
Response_Code	One of the values indicated near the beginning of this section.
Header_List	Optional list of headers passed with the Abort Request.

etOBEX_Port_Open_Request_Indication

Indicate that a Remote Port Open request has been received.

Notes:

1. When using this feature Bluetopia requires that a response be sent to a device requesting a connection within sixty seconds. If a response is not sent within this time a negative response will be sent to the device. Since this timeout is implementation specific the requesting device may timeout and disconnect sooner then Bluetopia.

Return Structure:

```
typedef struct
{
   unsigned int      GOEP_ID;
   BD_ADDR_t         BD_ADDR;
} OBEX_Port_Open_Request_Indication_Data_t;
```

Event Parameters:

GOEP_ID	Identifier of the GOEP server connection.
BD_ADDR	Address of the Bluetooth device.

3.4 OTP Programming Interface

The OTP (Object Transfer Protocol) programming interface defines the protocols and procedures to be used to perform File Transfer Protocol (FTP) and Object Transfer Protocol functions called out in the Bluetooth Profile specification. The OTP commands are listed in section 3.4.1, the response codes are listed in section 3.4.2, the event callback prototype is described in section 3.4.3, and the OTP events are itemized in section 3.4.4. The actual prototypes and constants outlined in this section can be found in the **OTPAPI.H** header file in the Bluetopia distribution.

3.4.1 OTP Commands/Responses

The available OTP Command and Response functions are listed in the table below and are described in the text which follows.

Function	Description
OTP_Open_Server_Port	Establish server port to wait for connections
OTP_Close_Server_Port	Close an open port
OTP_Open_Port_Request_Response	Respond to an open request from the remote device.
OTP_Register_SDP_Record	Add a generic SDP Service Record to the SDP database.
OTP_Register_Raw_SDP_Record	Adds a generic SDP Service Record to the SDP database with only pre-parsed protocol list data possibly added by the caller.

OTP_Open_Remote_Port	Open an OBEX connection to a remote device
OTP_Close_Port	Close either a server port or a remote port
OTP_Client_Connect	Make a connection with a remote OBEX server
OTP_Client_Disconnect	Close an OBEX server connection
OTP_Client_Get_Directory	Get a directory listing of the current folder from the remote OBEX file browing server
OTP_Client_Get_Object	Pull a data Object from a remote OBEX server
OTP_Client_Put_Object_Request	Request permission to push an Object into a remote OBEX server
OTP_Client_Put_Sync_Object_Request	Request permission to push an Object into a remote OBEX Sync server
OTP_Client_Put_Object	Push a data Object into a remote OBEX server, after receiving confirmation/permission via a _Request
OTP_Client_Set_Path	Create, delete or set the current folder on the OBEX server
OTP_Client_Delete_Object_Request	Delete an Object from a remote OBEX server
OTP_Client_Delete_Sync_Object_Request	Delete an Object from a remote OBEX Sync server
OTP_Client_Abort_Request	Abort the current request to the server
OTP_Connect_Response	Respond to the OTP client for a Connect command
OTP_Get_Directory_Request_Response	Respond to the OTP client for a Get Directory command
OTP_Set_Path_Response	Respond to the OTP client for a Set Path command
OTP_Abort_Response	Respond to the OTP client for an Abort command
OTP_Get_Object_Response	Respond to the OTP client for a Get Object command
OTP_Delete_Object_Response	Respond to the OTP client for a Delete Object command
OTP_Delete_Sync_Object_Response	Respond to the OTP client for a Delete Object command on a Sync Server
OTP_Put_Object_Response	Respond to the OTP client for a Put Object command
OTP_Put_Sync_Object_Response	Respond to the OTP client for a Put Object command on a Sync Server
OTP_Get_Server_Connection_Mode	Query the current Server Connection Mode.

OTP_Set_Server_Connection_Mode	Change the current Server Connection Mode.

OTP_Open_Server_Port

This function is responsible for establishing a OTP Port Server which will wait for a connection to occur on the port established by this function.

Prototype:

int BTPSAPI **OTP_Open_Server_Port**(unsigned int BluetoothStackID,
 Byte_t ServerPort, OTP_Target_t Target, Word_t MaxPacketLength,
 OTP_Event_Callback_t OTP_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
ServerPort	Port number to use. This must fall in the range defined by the following constants: SPP_PORT_NUMBER_MINIMUM SPP_PORT_NUMBER_MAXIMUM
Target	The service on the remote server to which the connection is targeted. May be one of the following values: tInbox tFileBrowser tIRSync
MaxPacketLength	The largest packet that will be sent/received on this connection. Each side must support a minimum of 255 bytes, and cannot have a packet size greater than 64K-1 bytes. These constraints are defined as the constants: OTP_PACKET_LENGTH_MINIMUM OTP_PACKET_LENGTH_MAXIMUM
OTP_Event_Callback	Function to call when events occur on this port.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive, non-zero if successful. The return value will be the OTP_ID for the server port that was successfully opened. *This* is the value that should be used in all subsequent function calls (except another OTP_Open_Server_Port( ) call).

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING

BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_ALREADY_CONNECTED
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Open_Request_Indication

etOTP_Port_Open_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Close_Server_Port

This function is responsible for Unregistering a Serial Port Server which was registered by a successful call to the OTP_Open_Server_Port( ) function. Note, this function does NOT delete any SDP Service Record Handles (i.e., added via a OTP_Register_SDP_Record( ) function call).

Prototype:

int BTPSAPI **OTP_Close_Server_Port**(unsigned int BluetoothStackID,
    unsigned int OTP_ID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port to close. This is the value that was returned from the OTP_Open_Server_Port( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Open_Port_Request_Response

This function is responsible for responding to requests to connect to a OTP Port Server.

Notes:

1. When using this feature Bluetopia requires that a response be sent to a device requesting a connection within sixty seconds. If a response is not sent within this time a negative response will be sent to the device. Since this timeout is implementation specific the requesting device may timeout and disconnect sooner then Bluetopia.

Prototype:

int BTPSAPI **OTP_Open_Port_Request_Response**(unsigned int BluetoothStackID, unsigned int OTP_ID, Boolean_t AcceptConnection)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port this command applies to. This is the value that was returned from the OTP_Open_Server_Port( ) function.
AcceptConnection	Boolean indicating if the pending connection should be accepted.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_OTP_NOT_INITIALIZED

Possible Events:

etOTP_Port_Open_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Register_SDP_Record

This function provides a means to add a generic SDP Service Record to the SDP Database.

Notes:

1. This function should only be called with the OTP_ID that was returned from the OTP_Open_Server_Port( ) function. This function should **never** be used with the Serial Port ID returned from the OTP_Open_Remote_Port( ) function.
2. The Service Record Handle that is returned from this function will remain in the SDP Record Database until it is deleted by calling the SDP_Delete_Service_Record( ) function. A Macro is provided to delete the Service Record from the SDP Database. This Macro maps OTP_Un_Register_SDP_Record**( )** to SDP_Delete_Service_Record(), and is defined as follows:

OTP_Un_Register_SDP_Record(__BluetoothStackID, __OTPID, __SDPRecordHandle)

3. There must be UUID Information specified in theSDPServiceRecord Parameter, however protocol information is completely optional. Any Protocol Information that is specified (if any) will be added in the Protocol Attribute AFTER the default OTP Protocol List (L2CAP, RFCOMM, and OTP).
4. The Service Name is always added at Attribute ID 0x0100. A Language Base Attribute ID List is created that specifies that 0x0100 is UTF-8 Encoded, English Language.

Prototype:

int BTPSAPI **OTP_Register_SDP_Record**(unsigned int BluetoothStackID, unsigned int OTP_ID, OTP_SDP_Service_Record_t *SDPServiceRecord, char *ServiceName, DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port this command applies to. This is the value that was returned from the OTP_Open_Server_Port( ) function.
SDPServiceRecord	Any additional Service Discovery Protocol information to be added to the record for this serial port server. This is a structured defined as:

```
typedef struct
{
   unsigned int             NumberServiceClassUUID;
   SDP_UUID_Entry_t         *SDPUUIDEntries;
   SDP_Data_Element_t       *ProtocolList;
} OTP_SDP_Service_Record_t;
```

ServiceName	Name to appear in the SDP Database for this service.
SDPServiceRecordHandle	Returned handle to the SDP Database entry which may be used to remove the entry at a later time.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Register_Raw_SDP_Record

This function provides a means to add a generic SDP Service Record to the SDP Database. This function is identical to the **OTP_Register_SDP_Record**() with the exception that any additional information to be added to the Protocol Attribute must be in a pre-parsed format.

Notes:

5. This function should only be called with the OTP_ID that was returned from the OTP_Open_Server_Port( ) function. This function should **never** be used with the Serial Port ID returned from the OTP_Open_Remote_Port( ) function.
6. The Service Record Handle that is returned from this function will remain in the SDP Record Database until it is deleted by calling the SDP_Delete_Service_Record( ) function. A Macro is provided to delete the Service Record from the SDP Database. This Macro maps OTP_Un_Register_SDP_Record**( )** to SDP_Delete_Service_Record(), and is defined as follows:

OTP_Un_Register_SDP_Record(__BluetoothStackID, __OTPID, __SDPRecordHandle)

7. There must be UUID Information specified in theSDPServiceRecord Parameter, however protocol information is completely optional. Any Protocol Information that is specified (if any) will be added in the Protocol Attribute AFTER the default OTP Protocol List (L2CAP, RFCOMM, and OTP).
8. The Service Name is always added at Attribute ID 0x0100. A Language Base Attribute ID List is created that specifies that 0x0100 is UTF-8 Encoded, English Language.

Prototype:

int BTPSAPI **OTP_Register_Raw_SDP_Record**(unsigned int BluetoothStackID,
    unsigned int OTP_ID,

```
        OTP_SDP_Raw_Service_Record_t *SDPServiceRecord,
        char *ServiceName,
        DWord_t *SDPServiceRecordHandle)
```

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port this command applies to. This is the value that was returned from the OTP_Open_Server_Port( ) function.
SDPServiceRecord	Any additional Service Discovery Protocol information to be added to the record for this serial port server. This is a structured defined as:

```
typedef struct
{
   unsigned int          NumberServiceClassUUID;
   SDP_UUID_Entry_t *SDPUUIDEntries;
   unsigned int         NumberOfProtocolDataListUUIDOffsets;
   Word_t              *ProtocolDataListUUIDOffsets;
   unsigned int         ProtocolDataListLength;
   Byte_t              *ProtocolDataList;
} OTP_SDP_Raw_Service_Record_t;
```

ServiceName	Name to appear in the SDP Database for this service.
SDPServiceRecordHandle	Returned handle to the SDP Database entry which may be used to remove the entry at a later time.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Open_Remote_Port

This function is used to open a remote serial port on the specified Remote Device.

Prototype:

int BTPSAPI **OTP_Open_Remote_Port**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, Byte_t ServerPort, Word_t MaxPacketLength
OTP_Event_Callback_t OTP_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
BD_ADDR	Address of the Bluetooth device to connect with.
ServerPort	The remote device's server port ID to connect with.
MaxPacketLength	The largest packet that will be sent/received on this connection. Each side must support a minimum of 255 bytes, and cannot have a packet size greater than 64K-1 bytes. These constraints are defined as the constants: OTP_PACKET_LENGTH_MINIMUM OTP_PACKET_LENGTH_MAXIMUM
OTP_Event_Callback	Function to call when events occur on this port.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function with each packet.

Return:

Positive, non-zero if successful. The return value will be the OTP_ID for the port that was successfully opened. *This* is the value that should be used in all subsequent function calls.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
RFCOMM_UNABLE_TO_CONNECT_TO_REMOTE_DEVICE
BTPS_ERROR_RFCOMM_UNABLE_TO_COMMUNICATE_
WITH_REMOTE_DEVICE
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_ALREADY_CONNECTED
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Open_Confirmation

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Close_Port

This function is used to close a OTP Port that was previously opened with the OTP_Open_Server_Port( ) function or the OTP_Open_Remote_Port( ) function. This function does **not** unregister a OTP Server Port from the system, it only disconnects any connection that is currently active on the Server Port. The OTP_Close_Server_Port() function can be used to Unregister the OTP Server Port.

Prototype:

int BTPSAPI **OTP_Close_Port**(unsigned int BluetoothStackID, unsigned int OTP_ID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port to close. This is the value that was returned from the OTP_Open_Server_Port( ) or OTP_Open_Remote_Port( ) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Connect

Make a connection to a remote OBEX Server.

Prototype:

int BTPSAPI OTP_Client_Connect(unsigned int BluetoothStackID, unsigned int OTP_ID,
    OTP_Target_t Target, OTP_Digest_Challenge_t *DigestChallenge,
    OTP_Digest_Response_t *DigestResponse);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port to use.
Target	The service on the remote server to which the connection is targeted. May be one of the following values: tUnknown tInbox tFileBrowser tIRSync
DigestChallenge	Used along with DigestResponse to pass Authentication Request and Response information between Server and Clients. These parameters should be set to NULL if authentication is not in use. This data item is the following structure:

```
typedef struct
{
   Byte_t          Nonce[OTP_DIGEST_MAXIMUM_NONCE_LENGTH];
   Byte_t          OptionalParametersMask;
   Byte_t          Options;
   unsigned int    RealmLength;
   Byte_t          RealmCharacterSet;
   char            Realm[OTP_DIGEST_MAXIMUM_REALM_LENGTH];
} OTP_Digest_Challenge_t;
```

The Nonce field is mandatory and must be 16 bytes in length. The Realm value has been limited to 50 bytes in this implementation (as defined by the constants shown).

The OptionalParametersMask is a set of bits that define which of the Optional parameters is filled in this structure (if the bit is set). This parameter is a logical ORing of the following bit constants:

OTP_DIGEST_CHALLENGE_OPTIONAL_PARAMETERS_MASK_OPTIONS
OTP_DIGEST_CHALLENGE_OPTIONAL_PARAMETERS_MASK_REALM

The following values are legal in the Options field:

OTP_DIGEST_CHALLENGE_OPTIONS_USER_ID_IN_RESPONSE_BIT
OTP_DIGEST_CHALLENGE_OPTIONS_ACCESS_MODE_READ_ONLY_BIT

Possible values for the RealmCharacterSet are:
OTP_REALM_CHARACTER_SET_ASCII
OTP_REALM_CHARACTER_SET_ISO88591

OTP_REALM_CHARACTER_SET_ISO88592
OTP_REALM_CHARACTER_SET_ISO88593
OTP_REALM_CHARACTER_SET_ISO88594
OTP_REALM_CHARACTER_SET_ISO88595
OTP_REALM_CHARACTER_SET_ISO88596
OTP_REALM_CHARACTER_SET_ISO88597
OTP_REALM_CHARACTER_SET_ISO88598
OTP_REALM_CHARACTER_SET_ISO88599
OTP_REALM_CHARACTER_SET_UNICODE

DigestResponse This is defined by the following structure:

```
typedef struct
{
   Byte_t         RequestDigest[OTP_DIGEST_MAXIMUM_REQUEST_DIGEST_LENGTH];
   Byte_t         OptionalParametersMask;
   unsigned int   UserIDLength;
   Byte_t         UserID[OTP_DIGEST_MAXIMUM_USER_ID_LENGTH];
   Byte_t         Nonce[OTP_DIGEST_MAXIMUM_NONCE_LENGTH];
} OTP_Digest_Response_t;
```

The RequestDigest field is mandatory and must be 16 bytes and, similarly, the UserID has been limited in size in this implementation (as defined by the constants shown).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Connect_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Disconnect

Break a connection made with OTP_Client_Connect( ).

Prototype:

int BTPSAPI **OTP_Client_Disconnect**(unsigned int BluetoothStackID,
unsigned int OTP_ID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port on which to close the connection. This is the value that was returned from either the OTP_Open_Remote_Port( ).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Disconnect_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Get_Directory

Get a directory listing of the current folder from the remote OBEX file browing server.

Prototype:

int BTPSAPI **OTP_Client_Get_Directory**(unsigned int BluetoothStackID,
unsigned int OTP_ID, char *Name);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.

Name	A pointer to a ASCIIZ string that identifies the name of the directory that is to be retreived. When specifying the Name, No path information is allowed. When retreiving a directory listing, the SETPATH function should be used to set the current directory. This function is then called with the Name parameter set to NULL to pull the current directory. If the Name parameter is not NULL, then Name must point to a ASCIIZ string of the name of a sub-directory that exists off the current directory. It must also be noted that when the Name parameter is used, a sub-directory listing will be returned for the directory specified, however, the current directory will remain the same and will not be changed to the sub-directory specified.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Get_Directory_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Get_Object

Pull a data Object from a remote OBEX server.

Prototype:

int BTPSAPI **OTP_Client_Get_Object**(unsigned int BluetoothStackID,
unsigned int OTP_ID, char *Type, char *Name, unsigned long UserInfo);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize

OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
Type	A pointer to a NULL terminated string that describes the type of object to be retreived
Name	A pointer to a NULL terminated string that specifies the Name of the Object that is to be retreived. *It should be noted that when connected to an OBEX File Browser Service, the Type parameter is optional. When connected to the OBEX Inbox, the Name parameter is optional.*
UserInfo	A user-defined parameter that will be passed back in the event callback.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Get_Object_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Put_Object_Request

Request permission to save or create an object on the remote OBEX server.

Prototype:

int BTPSAPI **OTP_Client_Put_Object_Request**(unsigned int BluetoothStackID,
unsigned int OTP_ID, Boolean_t CreateOnly, unsigned int Length, char *Type,
char *Name, unsigned long UserInfo);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
CreateOnly	Specifies whether or not this request is being made as an introduction to putting an object (CreateOnly equals FALSE), or to simply create an object of zero length (CreateOnly equals TRUE).
Length	The Length (in Bytes) of the actual Object that is to be placed on the Remote Server.
Type	A pointer to a NULL terminated string that describes the type of object to be retreived. This is NULL for files or a string that defines the Object Type (for example "text/x-vCard" to put a vCard Object). This field is only used if the Target is not a File Browser.
Name	A pointer to a NULL terminated string that specifies the Name of the Object that is to be sent.
UserInfo	A user-defined parameter that will be passed back in the event callback.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Put_Object_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Put_Sync_Object_Request

Request permission to save an object on the remote OBEX Sync server. This function differs from the normal Put Object function in that this function allows a Synchronization Anchor to be specified. Note that this function does not allow the specification of the type of Object that is being placed on the remote OBEX Sync server. The type of object is inferred from the the path of the name of the object (e.g. "/telecom/pb" as the path means the object is a vCARD) as per the IRSync specification.

Prototype:

int BTPSAPI **OTP_Client_Put_Sync_Object_Request**(unsigned int BluetoothStackID, unsigned int OTP_ID, unsigned int Length, char *Type, SyncAnchor_t *SyncAnchor, unsigned long UserInfo);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
Length	The Length (in Bytes) of the actual Object that is to be placed on the Remote Server.
Name	A pointer to a NULL terminated string that specifies the Name of the Object that is to be sent.
SyncAnchor	A pointer to structure that contains the Synchronization Anchor information for the Object. This structure is defined as:
UserInfo	A user-defined parameter that will be passed back in the event callback.

```
typedef struct
{
  Boolean_t         TimestampUsed;
  OTP_TimeDate_t Timestamp;
  Boolean_t         ChangeCountUsed;
  DWord_t           ChangeCount;
} SyncAnchor_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA

BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Put_Sync_Object_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Put_Object

Send a data Object to the remote OTP server. This can only be called after a sucessful response from a call to OTP_Client_Put_Object_Request() or OTP_Client_Put_Sync_Object_Request().

Prototype:

int BTPSAPI **OTP_Client_Put_Object**(unsigned int BluetoothStackID, unsigned int OTP_ID, unsigned int DataLength, Byte_t *Data, Boolean_t Final, unsigned long UserInfo);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port to send the Put Request to. This is the value that was returned from the OTP_Open_Remote_Port( ) function.
DataLength	The number of bytes being passed in this call in the Data parameter.
Data	Data to be sent for this object in this call.
Final	Flag which indicates if this is the last packet of the Put sequence or not.
UserInfo	A user-defined parameter that will be passed back in the event callback.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA

BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Put_Object_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Set_Path

Create, delete or set the current folder on the OBEX server. If a folder name is supplied that doesn't exist on the Server, a new folder will be created before the Server changes to that folder.

Prototype:

int BTPSAPI **OTP_Client_Set_Path**(unsigned int BluetoothStackID, unsigned int OTP_ID, char *Name, Boolean_t Backup, Boolean_t Create);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port to send the Get Request to. This is the value that was returned from the OTP_Open_Remote_Port( ) function.
Name	A pointer to a NULL terminated string of the path to the sub-directory referenced from the current directory.
Backup	Go back up one level in the directory structure. When this is set to TRUE, it takes priority over the Name parameter which is ignored in this situation.
Create	Whether or not to create the directory if it does not already exist. The Name parameter *must* be supplied if TRUE.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_ALREADY_CONNECTED

BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Set_Path_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Delete_Object_Request

Delete an Object from a remote OBEX server.

Prototype:

int BTPSAPI **OTP_Client_Delete_Object_Request**(unsigned int BluetoothStackID, unsigned int OTP_ID, char *Name);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
Name	A pointer to a NULL terminated string that indicates the object to be deleted.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Delete_Object_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Delete_Sync_Object_Request

Delete an Object from a remote OBEX Sync server.

Prototype:

int BTPSAPI **OTP_Client_Delete_Sync_Object_Request**(unsigned int BluetoothStackID, unsigned int OTP_ID, char *Name, SyncAnchor_t *SyncAnchor, Boolean_t HardDelete);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
Name	A pointer to a NULL terminated string that indicates the object to be deleted.
SyncAnchor	A pointer to a Syncronization Anchor to use. This member only has meaning if the SyncAnchor type is of type Change Counter. This action then allows the remote OBEX Sync entity the ability to allow/reject the delete based on the remote OBEX Sync servers Current Change Count for the Object. This value should be the expected Change Count of the Object AFTER the delete is successful (i.e. not the current Change Count value). This structure is defined as:

```
typedef struct
{
   Boolean_t         TimestampUsed;
   OTP_TimeDate_t    Timestamp;
   Boolean_t         ChangeCountUsed;
   DWord_t           ChangeCount;
} SyncAnchor_t;
```

HardDelete	A Boolean_t flag which specifies whether the delete is Hard Delete (TRUE) or Soft Delete (FALSE).

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID

BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Delete_Sync_Object_Response
etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Client_Abort_Request

Abort the current request to the server.

Prototype:

int BTPSAPI **OTP_Client_Abort**(unsigned int BluetoothStackID, unsigned int OTP_ID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_ALREADY_CONNECTED
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Abort_Response

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Connect_Response

Respond to the OTP client for a Connect command.

Prototype:

int BTPSAPI **OTP_Connect_Response**(unsigned int BluetoothStackID,
unsigned int OTP_ID, Boolean_t Accept, OTP_Digest_Challenge_t *DigestChallenge,
OTP_Digest_Response_t *DigestResponse);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
Accept	Whether to accept the connection or not.
DigestChallenge	With DigestResponse are used for authentification. If authentification is not being used, both parameters are set to NULL. See The OTP_Client_Connect( ) command for information the data structure of this parameter.
DigestResponse	See The OTP_Client_Connect( ) command for information the data structure of this parameter.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_ALREADY_CONNECTED
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Get_Directory_Request_Response

Respond to the OTP client for a Get Directory command.

Prototype:

int BTPSAPI **OTP_Get_Directory_Request_Response**(unsigned int BluetoothStackID, unsigned int OTP_ID, OTP_DirectoryInfo_t *DirInfo, Byte_t ResponseCode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
DirInfo	The parameter DirEntry is a pointer to an array of directory entrystructures. Each entry in the array contains information about a file or directory entry that is to be sent in response to the request. It is important to note that the stack receives the directory information as an array of structures, and will convert this information into XML format prior to sending to information to the remote client. The process of converting the data to XML and sending all of the information to the remote client may require multiple requests and responses from the client and server. The lower layer stack will handle all of these additionaltransactions without any further interaction from the application. Since the directory transfer process may take some time to complete, the data pointed to by the parameter DirInfo must be preserved until the transfer process is complete. When the DirInfo information is no longer needed by the lower stack, a Callback will be generated with the etOTP_Free_Directory_Information event to inform the application that the directory transfer process is complete and the data can be freed. The structures used for this parameter are defined as follows:

```
typedef struct
{
   Boolean_t            ParentDirectory;
   unsigned int         NumberEntries;
   OTP_ObjectInfo_t     *ObjectInfo;
} OTP_DirectoryInfo_t;
```

Where ObjectInfo is an array of the following structures:

```
typedef struct
{
  OTP_ObjectType_t    ObjectType;
  Word_t              FieldMask;
  unsigned int        NameLength;
  char                Name[OTP_OBJECT_INFO_MAXIMUM_NAME_LENGTH];
  unsigned int        Size;
  unsigned int        TypeLength;
  char                Type[OTP_OBJECT_INFO_MAXIMUM_TYPE_LENGTH];
  OTP_TimeDate_t      Modified;
  OTP_TimeDate_t      Created;
  OTP_TimeDate_t      Accessed;
  Word_t              Permission;
  unsigned int        OwnerLength;
  char                Owner[OTP_OBJECT_INFO_MAXIMUM_OWNER_LENGTH];
  unsigned int        GroupLength;
  char                Group[OTP_OBJECT_INFO_MAXIMUM_GROUP_LENGTH];
} OTP_ObjectInfo_t;
```

Note the limits on the character arrays. The Bluetooth and OBEX specifications do not impose a limit, but to accommodate operating systems with memory limitations, this implement has imposed the limits shown by the constants. Entries longer than this will be truncated to the limits.

The ObjectType field can take on any of the following values:

otUnknown, otFolder, otFile, otvCard, otvCalander, otObject, otFileFolder

The FieldMask field is an ORing of bits which indicate what information has been filled in. The bitmask constants are:

```
OTP_OBJECT_INFO_MASK_CLEAR
OTP_OBJECT_INFO_MASK_NAME
OTP_OBJECT_INFO_MASK_SIZE
OTP_OBJECT_INFO_MASK_TYPE
OTP_OBJECT_INFO_MASK_MODIFIED
OTP_OBJECT_INFO_MASK_CREATED
OTP_OBJECT_INFO_MASK_ACCESSED
OTP_OBJECT_INFO_MASK_USER_PERMISSION
OTP_OBJECT_INFO_MASK_GROUP_PERMISSION
OTP_OBJECT_INFO_MASK_OTHER_PERMISSION
OTP_OBJECT_INFO_MASK_OWNER
OTP_OBJECT_INFO_MASK_GROUP
```

The Modified, Created, and Accessed date/time fields are defined by the following structure, where time is on a 24-hr clock and the UTC_Time flag indicates if the time is universal time vs. local time.

```
typedef struct
{
  Word_t        Year;
```

```
   Word_t       Month;
   Word_t       Day;
   Word_t       Hour;
   Word_t       Minute;
   Word_t       Second;
   Boolean_t    UTC_Time;
} OTP_TimeDate_t;
```

The Permissions field is an ORing of bits from the following list of defined permissions:

OTP_USER_PERMISSION_READ
OTP_USER_PERMISSION_WRITE
OTP_USER_PERMISSION_DELETE
OTP_GROUP_PERMISSION_READ
OTP_GROUP_PERMISSION_WRITE
OTP_GROUP_PERMISSION_DELETE
OTP_OTHER_PERMISSION_READ
OTP_OTHER_PERMISSION_WRITE
OTP_OTHER_PERMISSION_DELETE

ResponseCode	The parameter ResponseCode is used to notify the remote client of its ability to satisfy the request. If the ResponseCode value is non-Zero, then the information pointed to by the DirInfo parameter is considered invalid and the ResponseCode value represents the OBEX result code that identifies the reason why the request was not processed. The possible ResponseCode values are listed earlier in this section (before the first _Response) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Free_Directory_Information

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Set_Path_Response

Respond to the OTP client for a Set Path command.

Prototype:

int BTPSAPI **OTP_Set_Path_Response**(unsigned int BluetoothStackID,
unsigned int OTP_ID, Byte_t ResponseCode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
ResponseCode	The parameter ResponseCode is used to notify the remote client of its ability to satisfy the request. The possible ResponseCode values are listed earlier in this section (before the first _Response) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Abort_Response

Respond to the OTP client for an Abort command. Since it is impossible to refuse an abort request, there are no additional parameters, like a ResponseCode. This response is simply an acknowledgement.

Prototype:

int BTPSAPI **OTP_Abort_Response**(unsigned int BluetoothStackID,
unsigned int OTP_ID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Get_Object_Response

Respond to the OTP client for a Get Object command, i.e., sent the Object.

Prototype:

int BTPSAPI **OTP_Get_Object_Response**(unsigned int BluetoothStackID,
unsigned int OTP_ID, unsigned int BytesToSend, unsigned int ResponseCode,
unsigned long UserInfo);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
BytestoSend	When the request was made, the Server received a Get Request event which included a pointer to a buffer where the data was to be loaded. This buffer was referenced in the structure OTP_Info_t. The number of bytes that was loaded into this buffer is what is placed into BytestoSend.
ResponseCode	The parameter ResponseCode is used to notify the remote client of its ability to satisfy the request. If the ResponseCode value is non-Zero, then any other information is considered invalid and the ResponseCode value represents the OBEX result code that identifies the reason why the request was not processed. The possible ResponseCode values are listed earlier in this section (before the first _Response) function.
UserInfo	A user-defined parameter that will be passed back in the next Get Request event.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Delete_Object_Response

Respond to the OTP client for a Delete Object command.

Prototype:

int BTPSAPI **OTP_Delete_Object_Response**(unsigned int BluetoothStackID,
unsigned int OTP_ID, Byte_t ResponseCode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
ResponseCode	The parameter ResponseCode is used to notify the remote client of its ability to satisfy the request. If the ResponseCode value is non-Zero, then any other information is considered invalid and the ResponseCode value represents the OBEX result code that identifies the reason why the request was not processed. The possible ResponseCode values are listed earlier in this section (before the first _Response) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Delete_Sync_Object_Response

Respond to the OTP client for a Delete Sync Object command.

Prototype:

int BTPSAPI **OTP_Delete_Sync_Object_Response**(unsigned int BluetoothStackID,
unsigned int OTP_ID, Byte_t ResponseCode , char *UID, SyncAnchor_t *SyncAnchor);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
ResponseCode	The parameter ResponseCode is used to notify the remote client of its ability to satisfy the request. If the ResponseCode value is non-Zero, then any other information is considered invalid and the ResponseCode value represents the OBEX result code that identifies the reason why the request was not processed. The possible ResponseCode values are listed earlier in this section (before the first _Response) function.
UID	A pointer to a NULL terminated ASCII string that specifies the local UID of the Object that was deleted.
SyncAnchor	A pointer to the SyncAnchor to return in the delete response (either Change Count or Timestamp) . This structure is defined as:

```
typedef struct
{
   Boolean_t          TimestampUsed;
   OTP_TimeDate_t Timestamp;
   Boolean_t          ChangeCountUsed;
   DWord_t            ChangeCount;
} SyncAnchor_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Put_Object_Response

Respond to the OTP client for a Put Object command.

Prototype:

int BTPSAPI **OTP_Put_Object_Response**(unsigned int BluetoothStackID,
 unsigned int OTP_ID, Byte_t ResponseCode);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
ResponseCode	The parameter ResponseCode is used to notify the remote client of its ability to satisfy the request. If the ResponseCode value is non-Zero, then any other information is considered invalid and the ResponseCode value represents the OBEX result code that identifies the reason why the request was not processed. The possible ResponseCode values are listed earlier in this section (before the first _Response) function.

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Put_Sync_Object_Response

Respond to the OTP client for a Put Sync Object command.

Prototype:

int BTPSAPI **OTP_Put_Sync_Object_Response**(unsigned int BluetoothStackID,
unsigned int OTP_ID, Byte_t ResponseCode , char *UID, SyncAnchor_t *SyncAnchor);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The OBEX connection on which to issue the request. This is the value that was returned from the OTP_Open_Remote_Port() function.
ResponseCode	The parameter ResponseCode is used to notify the remote client of its ability to satisfy the request. If the ResponseCode value is non-Zero, then any other information is considered invalid and the ResponseCode value represents the OBEX result code that identifies the reason why the request was not processed. The possible ResponseCode values are listed earlier in this section (before the first _Response) function.
UID	A pointer to a NULL terminated ASCII string that specifies the local UID of the Object that was deleted.
SyncAnchor	A pointer to the SyncAnchor to return in the delete response (either Change Count or Timestamp). This structure is defined as:

```
typedef struct
{
   Boolean_t          TimestampUsed;
   OTP_TimeDate_t Timestamp;
   Boolean_t          ChangeCountUsed;
   DWord_t            ChangeCount;
} SyncAnchor_t;
```

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_OTP_NOT_INITIALIZED
BTPS_ERROR_OTP_REQUEST_OUTSTANDING
BTPS_ERROR_OTP_ERROR_PARSING_DATA
BTPS_ERROR_OTP_NO_CONNECTION
BTPS_ERROR_OTP_ACTION_NOT_ALLOWED

Possible Events:

etOTP_Port_Close_Indication

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Get_Server_Connection_Mode

This function is responsible for allowing a mechanism to query the OTP Port Server Connection Mode.

Prototype:

int BTPSAPI **OTP_Get_Server_Connection_Mode**(unsigned int BluetoothStackID, unsigned int OTP_ID, SPP_Server_Connection_Mode_t *ServerConnectionMode)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port this command applies to. This is the value that was returned from the OTP_Open_Server_Port( ) function.
ServerConnectionMode	Pointer to a variable to receive the current Server Connection Mode. The following modes are currently defined. smAutomaticAccept smAutomaticReject smManualAccept

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_OTP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

OTP_Set_Server_Connection_Mode

This function is responsible for allowing a mechanism to change the OTP Port Server Connection Mode.

Prototype:

int BTPSAPI **OTP_Set_Server_Connection_Mode**(unsigned int BluetoothStackID,
unsigned int OTP_ID, SPP_Server_Connection_Mode_t ServerConnectionMode)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_ID	The port this command applies to. This is the value that was returned from the OTP_Open_Server_Port( ) function.
ServerConnectionMode	The new Server Connection Mode being set. The following modes are currently defined. smAutomaticAccept smAutomaticReject smManualAccept

Return:

Zero if successful.

An error code if negative; one of the following values:

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_OTP_NOT_INITIALIZED

Possible Events:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

3.4.2 Response Codes for OTP Operations

The following codes are a direct mapping of the OBEX Response Codes. These are possible values for a number of the following _Response functions as well as the event handling structures described in section 3.4.3 .

OTP_CONTINUE_RESPONSE
OTP_OK_RESPONSE
OTP_CREATED_RESPONSE
OTP_ACCEPTED_RESPONSE
OTP_NON_AUTHORITATIVE_INFORMATION_RESPONSE
OTP_NO_CONTENT_RESPONSE
OTP_RESET_CONTENT_RESPONSE
OTP_PARTIAL_CONTENT_RESPONSE
OTP_MULTIPLE_CHOICES_RESPONSE
OTP_MOVED_PERMANETLY_RESPONSE
OTP_MOVED_TEMPORARILY_RESPONSE

OTP_SEE_OTHER_RESPONSE
OTP_NOT_MODIFIED_RESPONSE
OTP_USE_PROXY_RESPONSE
OTP_BAD_REQUEST_RESPONSE
OTP_UNAUTHORIZED_RESPONSE
OTP_PAYMENT_REQUIRED_RESPONSE
OTP_FORBIDDEN_RESPONSE
OTP_NOT_FOUND_RESPONSE
OTP_METHOD_NOT_ALLOWED_RESPONSE
OTP_NOT_ACCEPTABLE_RESPONSE
OTP_PROXY_AUTHENTICATION_REQUIRED_RESPONSE
OTP_REQUEST_TIMEOUT_RESPONSE
OTP_CONFLICT_RESPONSE
OTP_GONE_RESPONSE
OTP_LENGTH_REQUIRED_RESPONSE
OTP_PRECONDITION_FAILED_RESPONSE
OTP_REQUESTED_ENTITY_TOO_LARGE_RESPONSE
OTP_REQUESTED_URL_TOO_LARGE_RESPONSE
OTP_UNSUPORTED_MEDIA_TYPE_RESPONSE
OTP_INTERNAL_SERVER_ERROR_RESPONSE
OTP_NOT_IMPLEMENTED_RESPONSE
OTP_BAD_GATEWAY_RESPONSE
OTP_SERVICE_UNAVAILABLE_RESPONSE
OTP_GATEWAY_TIMEOUT_RESPONSE
OTP_HTTP_VERSION_NOT_SUPPORTED_RESPONSE
OTP_DATABASE_FULL_RESPONSE
OTP_DATABASE_LOCKED_RESPONSE

3.4.3 OTP Event Callback Protoype

The event callback functions mentioned in the OTP Open commands all accept the callback function described by the following prototype.

OTP_Event_Callback_t

Prototype of callback function passed in one of the OTP open commands.

Prototype:

void (BTPSAPI ***OTP_Event_Callback_t**)(unsigned int BluetoothStackID, OTP_Event_Data_t *OTP_Event_Data, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OTP_Event_Data	Data describing the event for which the callback function is called. This is defined by the following struture:

```
typedef struct
{
  OTP_Event_Data_Type_t                    Event_Data_Type;
```

```
        Word_t                                      Event_Data_Size;
        union
        {
          OTP_Port_Open_Indication_Data_t          *OTP_Port_Open_Indication_Data;
          OTP_Port_Open_Confirmation_Data_t        *OTP_Port_Open_Confirmation_Data;
          OTP_Port_Close_Indication_Data_t         *OTP_Port_Close_Indication_Data;
          OTP_Connect_Request_Data_t               *OTP_Connect_Request_Data;
          OTP_Connect_Response_Data_t              *OTP_Connect_Response_Data;
          OTP_Disconnect_Request_Data_t            *OTP_Disconnect_Request_Data;
          OTP_Disconnect_Response_Data_t           *OTP_Disconnect_Response_Data;
          OTP_Set_Path_Request_Data_t              *OTP_Set_Path_Request_Data;
          OTP_Set_Path_Response_Data_t             *OTP_Set_Path_Response_Data;
          OTP_Abort_Request_Data_t                 *OTP_Abort_Request_Data;
          OTP_Abort_Response_Data_t                *OTP_Abort_Response_Data;
          OTP_Get_Directory_Request_Data_t         *OTP_Get_Directory_Request_Data;
          OTP_Get_Directory_Response_Data_t        *OTP_Get_Directory_Response_Data;
          OTP_Put_Object_Request_Data_t            *OTP_Put_Object_Request_Data;
          OTP_Put_Object_Response_Data_t           *OTP_Put_Object_Response_Data;
          OTP_Get_Object_Request_Data_t            *OTP_Get_Object_Request_Data;
          OTP_Get_Object_Response_Data_t           *OTP_Get_Object_Response_Data;
          OTP_Delete_Object_Request_Data_t         *OTP_Delete_Object_Request_Data;
          OTP_Delete_Object_Response_Data_t        *OTP_Delete_Object_Response_Data;
          OTP_Free_Directory_Information_Data_t    *OTP_Free_Directory_Information_Data;
          OTP_Port_Open_Request_Indication_Data_t *OTP_Port_Open_Request_Indication_Data;
        } Event_Data;
      } OTP_Event_Data_t;
```

where, Event_Data_Type is one of the enumerations of the event types listed in the table in section 3.4.3, and each data structure in the union is described with its event in that section as well.

CallbackParameter	User-defined parameter (e.g., tag value) that was defined in the callback registration.

Return:

Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

3.4.4 OTP Events

The possible OTP events from the Bluetooth stack are listed in the table below and are described in the text which follows:

Event	Description
etOTP_Port_Open_Indication	Indicate that a Remote Port Open connection has been made

etOTP_Port_Open_Confirmation	Confirm that a Port Open request has been responded to or has errored out
etOTP_Port_Open_Request_Indication	Indicate that a Remote Port Open request has been received
etOTP_Port_Close_Port_Indication	Indicate that a port has been closed (unregistered)
etOTP_Connect_Request	Indicate that a Connect Request has been received
etOTP_Connect_Response	Indicate that a Connect Response has been received
etOTP_Disconnect_Request	Indicate that a Disconnect Request has been received
etOTP_Disconnect_Response	Indicate that a Disconnect Response has been received
etOTP_Set_Path_Request	Indicate that a Set Path Request has been received
etOTP_Set_Path_Response	Indicate that a Set Path Response has been received
etOTP_Abort_Request	Indicate that a Abort Request has been received
etOTP_Abort_Response	Indicate that a Abort Response has been received
etOTP_Delete_Object_Request	Indicate that a Delete Object Request has been received
etOTP_Delete_Sync_Object_Request	Indicate that a Delete Sync Object Request has been received
etOTP_Delete_Object_Response	Indicate that a Delete Object Response has been received
etOTP_Delete_Sync_Object_Response	Indicate that a Delete Sync Object Response has been received
etOTP_Put_Object_Request	Indicate that a Put Object Request has been received
etOTP_Put_Sync_Object_Request	Indicate that a Put sync Object Request has been received
etOTP_Put_Object_Response	Indicate that a Put Object Response has been received
etOTP_Put_Sync_Object_Response	Indicate that a Put Sync Object Response has been received
etOTP_Get_Object_Request	Indicate that a Get Object Request has been received
etOTP_Get_Object_Response	Indicate that a Get Object Response has been received
etOTP_Get_Directory_Request	Indicate that a Get Directory Request has been received
etOTP_Get_Directory_Response	Indicate that a Get Directory Response has been received
etOTP_Free_Directory_Information	Indicate that it is now safe to free up the DirInfo data provided in OTP_Get_Directory_Response()

Several of the events return a Response_Code. These are listed just before the first _Response function in the section 3.4.1 .

etOTP_Port_Open_Indication

Indicate that a Remote Port Open connection has been made.

Return Structure:

```
typedef struct
{
  unsigned int                OTP_ID;
  BD_ADDR_t                   BD_ADDR;
} OTP_Port_Open_Indication_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
BD_ADDR	Address of the Bluetooth device.

etOTP_Port_Open_Confirmation

Confirm that a Port Open request has been responded to or has errored out.

Return Structure:

```
typedef struct
{
  unsigned int             OTP_ID;
  unsigned int             PortOpenStatus;
} OTP_Port_Open_Confirmation_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
PortOpenStatus	Status of the request. May be one of the following values: OTP_OPEN_PORT_STATUS_SUCCESS OTP_OPEN_PORT_STATUS_CONNECTION_TIMEOUT OTP_OPEN_PORT_STATUS_CONNECTION_REFUSED OTP_OPEN_PORT_STATUS_UNKNOWN_ERROR

etOTP_Port_Open_Request_Indication

Indicate that a Remote Port Open request has been received.

Notes:

1. When using this feature Bluetopia requires that a response be sent to a device requesting a connection within sixty seconds. If a response is not sent within this time a negative response will be sent to the device. Since this timeout is implementation specific the requesting device may timeout and disconnect sooner then Bluetopia.

Return Structure:

```
typedef struct
{
  unsigned int                  OTP_ID;
  BD_ADDR_t                     BD_ADDR;
} OTP_Port_Open_Request_Indication_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
BD_ADDR	Address of the Bluetooth device.

etOTP_Port_Close_Port_Indication

Indicate that a port has been closed (unregistered).

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  unsigned long         UserInfo;
} OTP_Port_Close_Indication_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
UserInfo	User-define value passed in the command.

etOTP_Connect_Request

Indicate that a Connect Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int                  OTP_ID;
  OTP_Target_t                  Target;
  OTP_Digest_Challenge_t        *DigestChallenge;
  OTP_Digest_Response_t         *DigestResponse;
} OTP_Connect_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
Target	The service which is being requested. May be one of the following values: tUnknown tInbox tFileBrowser tIRSync

DigestChallenge	With DigestResponse are used for authentification. If authentification is not being used, both parameters are set to NULL. See The OTP_Client_Connect( ) command for information the data structure of this parameter.
DigestResponse	See The OTP_Client_Connect( ) command for information the data structure of this parameter.

etOTP_Connect_Response

Indicate that a Connect Response has been received.

Return Structure:

```
typedef struct
{
   unsigned int              OTP_ID;
   Byte_t                    ResponseCode;
   OTP_Target_t              Target;
   OTP_Digest_Challenge_t    *DigestChallenge;
   OTP_Digest_Response_t     *DigestResponse;
} OTP_Connect_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.
Target	The service which is being requested. May be one of the following values: tUnknown tInbox tFileBrowser tIRSync
DigestChallenge	With DigestResponse are used for authentification. If authentification is not being used, both parameters are set to NULL. See The OTP_Client_Connect( ) command for information the data structure of this parameter.
DigestResponse	See The OTP_Client_Connect( ) command for information the data structure of this parameter.

etOTP_Disconnect_Request

Indicate that a Disconnect Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  unsigned long         UserInfo;
} OTP_Disconnect_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
UserInfo	User-defined value that was possibly passed in the currently executing Request Command.

etOTP_Disconnect_Response

Indicate that a Disconnect Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  Byte_t                ResponseCode;
} OTP_Disconnect_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.

etOTP_Set_Path_Request

Indicate that a Set Path Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int       OTP_ID;
  Boolean_t       Backup;
  Boolean_t       Create;
  char               *Folder;
} OTP_Set_Path_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
Backup	Whether to go back up one level in the directory tree. If present, the Folder field is ignored.
Create	Whether to allow the folder (sub-directory) to be created if it doesn’t exist.

Folder	A pointer to the NULL terminated name of the folder (sub-directory) to change to, relative to the current directory.

etOTP_Set_Path_Response

Indicate that a Set Path Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  Byte_t                ResponseCode;
} OTP_Set_Path_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.

etOTP_Abort_Request

Indicate that a Abort Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  unsigned long         UserInfo;
} OTP_Abort_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
UserInfo	User-defined value that was possibly passed in the currently executing Request Command.

etOTP_Abort_Response

Indicate that a Abort Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  Byte_t                ResponseCode;
} OTP_Abort_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.

ResponseCode	Returned response. See the list of response codes in section 3.4.2.

etOTP_Delete_Object_Request

Indicate that a Delete Object Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  OTP_ObjectInfo_t      ObjectInfo;
} OTP_Delete_Object_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ObjectInfo	Information on the object to be deleted. See the description in the OTP_Get_Directory_Request_Response( ) function.

etOTP_Delete_Sync_Object_Request

Indicate that a Delete Object Sync Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int                  OTP_ID;
  OTP_Sync_Request_Params_t     SyncParams;
  OTP_ObjectInfo_t              ObjectInfo;
} OTP_Delete_Sync_Object_Request_Data_t;
```

Event Parameters:

OTP_ID — Identifier of the OTP server connection.

SyncParams — Synchronization information regarding the item that is being deleted. This structure is defined as:

```
typedef struct
{
  Boolean_t HardDelete;
  SyncAnchor_t SyncAnchor;
} OTP_Sync_Request_Params_t;
```

ObjectInfo — Information on the object to be deleted. See the description in the OTP_Get_Directory_Request_Response( ) function.

etOTP_Delete_Object_Response

Indicate that a Delete Object Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int              OTP_ID;
  Byte_t                    ResponseCode;
} OTP_Delete_Object_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.

etOTP_Delete_Sync_Object_Response

Indicate that a Delete Object Sync Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int                  OTP_ID;
  OTP_Sync_Response_Params_t    SyncParams;
  OTP_ObjectInfo_t              ObjectInfo;
} OTP_Delete_Sync_Object_Response_Data_t;
```

Event Parameters:

OTP_ID — Identifier of the OTP server connection.

SyncParams — Synchronization information regarding the item that was deleted. This structure is defined as:

```
typedef struct
{
   SyncAnchor_t SyncAnchor;
   Byte_t     UID[OTP_SYNC_UID_MAXIMUM_LENGTH];
} OTP_Sync_Response_Params_t;
```

ObjectInfo — Information on the object to be deleted. See the description in the OTP_Get_Directory_Request_Response( ) function.

etOTP_Put_Object_Request

Indicate that a Put Object Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int              OTP_ID;
  Byte_t                    Phase;
  OTP_ObjectInfo_t          ObjectInfo;
  unsigned int              DataLength;
  Byte_t                    *DataPtr;
  unsigned long             UserInfo;
} OTP_Put_Object_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
Phase	Indicates whether this is the first request, continuation, or the final Request in the Put Object Transaction. Possible values are: OTP_OBJECT_PHASE_FIRST OTP_OBJECT_PHASE_LAST OTP_OBJECT_PHASE_CONTINUE
ObjectInfo	Information on the object to put. See the description in the OTP_Get_Directory_Request_Response( ) function.
DataLength	Length of the buffer pointed to by Data.
Data	Pointer to a buffer to containing the actual object data.
UserInfo	User-defined value that was passed in the command.

etOTP_Put_Sync_Object_Request

Indicate that a Put Sync Object Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int                    OTP_ID;
  Byte_t                          Phase;
  OTP_ObjectInfo_t                ObjectInfo;
  OTP_Sync_Request_Params_t       SyncParams;
  unsigned int                    DataLength;
  Byte_t                          *DataPtr;
  unsigned long                   UserInfo;
} OTP_Put_Sync_Object_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
Phase	Indicates whether this is the first request, continuation, or the final Request in the Put Object Transaction. Possible values are:

	OTP_OBJECT_PHASE_FIRST OTP_OBJECT_PHASE_LAST OTP_OBJECT_PHASE_CONTINUE
ObjectInfo	Information on the object to put. See the description in the OTP_Get_Directory_Request_Response( ) function.

SyncParams	Synchronization information regarding the item that is being deleted. This structure is defined as:
	typedef struct
	{
	Boolean_t HardDelete;
	SyncAnchor_t SyncAnchor;
	} OTP_Sync_Request_Params_t;

DataLength	Length of the buffer pointed to by Data.
Data	Pointer to a buffer to containing the actual object data.
UserInfo	User-defined value that was passed in the command.

etOTP_Put_Object_Response

Indicate that a Put Object Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int       OTP_ID;
  Byte_t             ResponseCode;
  unsigned int       BufferSize;
  unsigned long      UserInfo;
} OTP_Put_Object_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.
BufferSize	Amount of data that can be accepted in the buffer when sending the next Put Object Request.
UserInfo	User-defined value that was passed in the command.

etOTP_Put_Sync_Object_Response

Indicate that a Put Sync Object Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  Byte_t                ResponseCode;
  unsigned int          BufferSize;
  unsigned long         UserInfo;
} OTP_Put_Sync_Object_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.
SyncParams	Synchronization information regarding the item that was put. This structure is defined as:

```
typedef struct
{
   SyncAnchor_t SyncAnchor;
   Byte_t    UID[OTP_SYNC_UID_MAXIMUM_LENGTH];
} OTP_Sync_Response_Params_t;
```

BufferSize	Amount of data that can be accepted in the buffer when sending the next Put Object Request.
UserInfo	User-defined value that was passed in the command.

etOTP_Get_Object_Request

Indicate that a Get Object Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  Byte_t                Phase;
  OTP_ObjectInfo_t      ObjectInfo;
  unsigned int          BufferSize;
  Byte_t                *BufferPtr;
  unsigned long         UserInfo;
} OTP_Get_Object_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.
Phase	Indicates whether this is the first request, continuation, or the final Request in the Get Object Transaction. Possible values are:

OTP_OBJECT_PHASE_FIRST
OTP_OBJECT_PHASE_LAST
OTP_OBJECT_PHASE_CONTINUE

ObjectInfo	Information on the directory to get the listing for. See the description in the OTP_Get_Directory_Request_Response( ) function.
BufferSize	Amount of data that can be accepted in the buffer when sending the next Get Object Request.
Buffer	Pointer to a buffer to return the object data in.
UserInfo	User-defined value that was passed in the command.

etOTP_Get_Object_Response

Indicate that a Get Object Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int          OTP_ID;
  Byte_t                ResponseCode;
  Byte_t                Phase;
  OTP_ObjectInfo_t      ObjectInfo;
  unsigned int          BufferSize;
  Byte_t                *BufferPtr;
  unsigned long         UserInfo;
} OTP_Get_Object_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.
Phase	Indicates whether this is the first request, continuation, or the final Request in the Get Object Transaction. Possible values are: OTP_OBJECT_PHASE_FIRST OTP_OBJECT_PHASE_LAST OTP_OBJECT_PHASE_CONTINUE
ObjectInfo	Information on the directory to get the listing for. See the description in the OTP_Get_Directory_Request_Response( ) function.
BufferSize	Length of the buffer pointed to by Buffer.
Buffer	Pointer to a buffer to return the object data in.
UserInfo	User-defined value that was passed in the command.

etOTP_Get_Directory_Request

Indicate that a Get Directory Request has been received.

Return Structure:

```
typedef struct
{
  unsigned int            OTP_ID;
  unsigned int            NameLength;
  char                    *Name;
} OTP_Get_Directory_Request_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
NameLength	Length of the Name string;
Name	Name of the directory to retrieve the listing for. This is a sub-directory relative to the current path.

etOTP_Get_Directory_Response

Indicate that a Get Directory Response has been received.

Return Structure:

```
typedef struct
{
  unsigned int            OTP_ID;
  Byte_t                  ResponseCode;
  Byte_t                  Phase;
  OTP_DirectoryInfo_t     DirInfo;
} OTP_Get_Directory_Response_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
ResponseCode	Returned response. See the list of response codes in section 3.4.2.
Phase	Indicates whether this is the first request, continuation, or the final Request in the Get Directory Transaction. Possible values are: OTP_OBJECT_PHASE_FIRST OTP_OBJECT_PHASE_LAST OTP_OBJECT_PHASE_CONTINUE
DirInfo	Information that is returned. See the description in the OTP_Get_Directory_Response( ) function.

etOTP_Free_Directory_Information

Indicate that it is now safe to free up the DirInfo data provided in OTP_Get_Directory_Response( ).

Return Structure:

```
typedef struct
{
  unsigned int            OTP_ID;
  OTP_DirectoryInfo_t     DirectoryInfo;
} OTP_Free_Directory_Information_Data_t;
```

Event Parameters:

OTP_ID	Identifier of the OTP server connection.
DirectoryInfo	Pointer to the data that can be freed up. This value is the DirectoryInfo pointer that was passed into the OTP module when the Directory Respnonse was submitted.

4. File Distributions

The header files that are distributed with the Bluetooth Stack library are listed in the table below.

File	Contents/Description
BSCAPI.h	Bluetooth Stack Controller API definitions
BTAPITyp.h	Definition of API calling convention (symbol BTPSAPI)
BTErrors.h	Definition of error codes (BTPS_ERROR_... constants)
BTPSCFG.h	Prebuilt Bluetooth Stack configuration definitions
BTTypes.h	General Bluetooth type definitions
GAPAPI.h	Generic Access Profile API definitions
GOEPAPI.h	Generic Object Exchange Profile API definitions
HCIAPI.h	Host Controller Interface API definitions
HCICommT.h	Serial Comm port types for the HCI layer implementation
HCITypes.h	Supporting types, macros and constants for the HCI API
HCIUSBT.h	Universal Serial Bus types for the HCI layer implementation
L2CAPAPI.h	Logical Link Control and Adaption Protocol API definitions
L2CAPTyp.h	Supporting types, macros and constants for the L2CAP API
OBXTypes.h	Supporting types, macros and constants for OBEX API.
OTPAPI.h	Object Transfer Protocol API definitions.
RFCOMAPI.h	Radio Frequency Communications API definitions
RFCOMMT.h	Supporting types, macros and constants for the RFCOMM API
SCOAPI.H	Sychronous Connection-Oriented API definitions
SDPAPI.H	Service Discovery Protocol API definitions
SDPTypes.h	Supporting types, macros and constants for the SDP API
SMTypes.h	Supporting types, macros, and constants for LE security manager
SPPAPI.h	Serial Port Profile API definitions
SS1BTPS.h	Bluetooth Protocol Stack Include file