iAP/Serial Port Profile (iSPP)

Application Programming Interface Reference Manual

Profile Version: 1.1

Release: 4.0.1
February 21, 2013

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 (Synchronous Connection-Oriented) Link layers. In addition to basic functionality at these layers, the Bluetooth Protocol Stack by Stonestreet One provides implementations of the Service Discovery Protocol (SDP), RFCOMM (the Radio Frequency serial COMMunications port emulator), and several of the Bluetooth Profiles. Program access to these layers, services, and profiles is handled via Application Programming Interface (API) calls.

This document focuses on the API reference that contains a description of all programming interfaces for the Bluetooth iOS Accessory Protocol (iAP) over the Serial Port Profile (SPP) provided by Bluetopia. Chapter 2 contains a description of the programming interfaces for this profile. Chapter 3 contains the header file name list for the Bluetooth iSPP library.

1.1 Scope

This reference manual provides information on the iAP/Serial Port Profile API. These APIs are available on the full range of platforms supported by Stonestreet One:

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

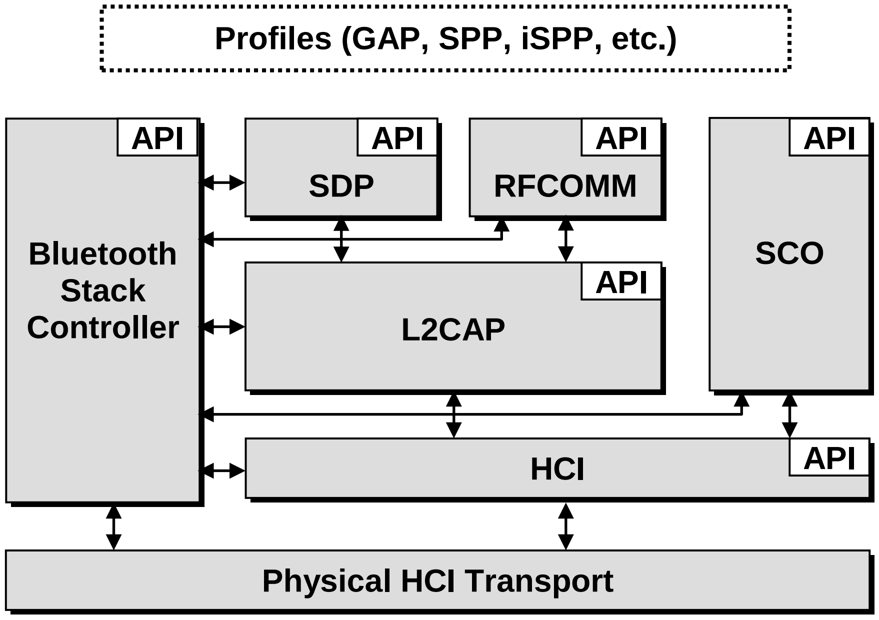

Figure 1-1 The Stonestreet One Bluetooth Protocol Stack

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. *Bluetopia® Protocol Stack, Application Programming Interface Reference Manual,* version 4.0.1, January 10, 2013.
30. *MFi Accessory Firmware Specification,* Release R42, November 30, 2010
31. *iPod Authentication Coprocessor Spec*, Version 2.0B R5, July 27, 2009

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
API	Application Programming Interface
BD_ADDR	Bluetooth Device Address
BR	Basic Rate

Term	Meaning
BT	Bluetooth
EDR	Enhanced Data Rate
HS	High Speed
iAP	iOS Accessory Protocol
iACP	iOS Apple Co-Processor
iSPP	iOS Accessory Protocol over Serial Port Profile
LE	Low Energy
LSB	Least Significant Bit
MSB	Most Significant Bit
SDP	Service Discovery Protocol
SPP	Serial Port Protocol
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

2. iAP/Serial Port Profile Programming Interface

The iAP/Serial Port Profile programming interface defines the protocols and procedures to be used to implement iAP over SPP (iSPP) capabilities. The iSPP commands are listed in section 2.1, the event callback prototype is described in section 2.2, and the iSPP events are itemized in section 2.3. The implementation of iSPP is provided as an extension of Bluetooth's SPP profile and thus an iSPP port can operate as a standard SPP port as well as handling the authentication process required by Apple.

2.1 iAP/Serial Port Profile Commands

The available iAP/Serial Port Profile command functions are listed in the table below and are described in the text that follows.

Function	Description
ISPP_Initialize	Initialize the ISPP module.
ISPP_Cleanup	Performs cleanup of the ISPP module.
ISPP_Open_Server_Port	Installs a server that will advertise iAP/SPP support.
ISPP_Close_Server_Port	Closes an iAP/SPP server port.
ISPP_Open_Port_Request_Response	Provides a response for a connection request.
ISPP_Register_SDP_Record	Adds a SDP record for a registered iAP/SPP port.
ISPP_Register_Raw_SDP_Record	Adds a SDP record for a registered iAP/SPP port with pre-formatted additional SDP Protocol Data information.
ISPP_Un_Register_SDP_Record	Removes the SDP record for an iAP/SPP port.
ISPP_Open_Remote_Port	Starts the connection process to a remote ISPP port.
ISPP_Close_Port	Disconnects a currently connected port.
ISPP_Start_Authorization	Starts the authentication process with a remote Apple device.
ISPP_Cancel_Authorization	Aborts an authentication process that is currently in progress.
ISPP_Open_Session_Request_Response	Submit a response to an open session request from the remote device.

ISPP_Send_Session_Data	Send session data to a remote device
ISPP_Cancel_Packet	Cancels a packet that has been submitted for transmission.
ISPP_Ack_Last_Session_Data_Packet	Acknowledges a previously received packet that was not immediately consumed.
ISPP_Send_Raw_Data	Sends raw Apple Lingo data to a remote device.
ISPP_Get_Port_Operating_Mode	Query the current operating mode of a specific iAP/SPP port.
ISPP_Data_Read[1]	Read data from a serial connection.
ISPP_Data_Write[1]	Send data on a serial connection.
ISPP_Change_Buffer_Size[1]	Change the default transmit/receive buffer sizes.
ISPP_Purge_Buffer[1]	Drop all data in an input/output buffer.
ISPP_Send_Break[1]	Notify the remote device of a break condition.
ISPP_Line_Status[1]	Send current line status to the remote side.
ISPP_Port_Status[1]	Send current modem/port control signals to the remote side.
ISPP_Send_Port_Information[1]	Send port parameters to be used to the remote side.
ISPP_Respond_Port_Information[1]	Respond to a send port information command from the remote side.
ISPP_Query_Remote_Port_Information[1]	Request current port parameters from the remote side.
ISPP_Respond_Query_Port_Information[1]	Reply to a request for current port parameters.
ISPP_Get_Configuration_Parameters[1]	Query RFCOMM frame size and default buffer sizes.
ISPP_Set_Configuration_Parameters[1]	Change RFCOMM frame size and default buffer sizes.
ISPP_Get_Port_Connection_State[1]	Query the current state of a specific iAP/SPP port connection.

[1] These functions are wrapper functions for the SPP counterparts and are only available when a port is operating in SPP mode. Refer to the SPP section of the Bluetopia Core API document for information on these functions.

ISPP_Initialize

The following function is used to Initialize the ISPP module.

Prototype:

int BTPSAPI **ISPP_Initialize**(unsigned int BluetoothStackID, void *ACP_Params)

Parameters:

BluetoothStackID	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize().
ACP_Params	Opaque data that is passed to the iACP transport module that is responsible for communicating with the Apple Authentication Coprocessor.

Return:

Zero if successful.

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

ISPP_ERROR_FAILED_TO_INITIALIZE
ISPP_ERROR_INSUFFICIENT_RESOURCES
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_INVALID_PARAMETER

ISPP_Cleanup

The following function performs any cleanup of this module when it is no longer needed.

Prototype:

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

Parameters:

BluetoothStackID	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize().

Return:

None

ISPP_Open_Server_Port

The following function registers an ISPP port that is capable of handling iAP.

Prototype:

int BTPSAPI **ISPP_Open_Server_Port**(unsigned int BluetoothStackID,
unsigned int ServerPort, ISPP_Event_Callback_t ISPP_Event_Callback,
unsigned long CallbackParameter)

Parameters:

BluetoothStackID	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
EventCallback	Function to call when events occur on this server.
CallbackParameter	A user-defined parameter (e.g., a tag value) that will be passed back to the user in the callback function.

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.

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

ISPP_ERROR_FAILED_TO_REGISTER_SERVER
ISPP_ERROR_INSUFFICIENT_RESOURCES
ISPP_ERROR_INVALID_PARAMETER
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID

Possible Events:

ietPort_Open_Request_Indication
ietPort_Open_Indication

ISPP_Close_Server_Port

This function is responsible for Unregistering an ISPP Server Port which was registered by a successful call to the ISPP_Open_Server_Port() function. Note that this function does NOT delete any SDP Service records (i.e., added via an ISPP_Register_SDP_Record() function call).

Prototype:

int BTPSAPI **ISPP_Close_Server_Port**(unsigned int BluetoothStackID,
unsigned int SerialPortID)

Parameters:

BluetoothStackID	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize().
SerialPortID	The identifier of the port to close. This is the value that was returned from the ISPP_Open_Server_Port() function.

Return:

Zero if successful.

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

ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_PARAMETER
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID

ISPP_Open_Port_Request_Response

This function is responsible for responding to requests to connect to an ISPP Server. If the connection is accepted and the port is to support iAP, the FIDInfoLength, FIDInfo and MaxRxPacketSize parameters must be specified.

Prototype:

int BTPSAPI **ISPP_Open_Port_Request_Response**(unsigned int BluetoothStackID, unsigned int SerialPortID, Boolean_t AcceptConnection, int FIDInfoLength, unsigned char *FIDInfo, int MaxRxPacketSize)

Parameters:

BluetoothStackID	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 ISPP_Open_Server_Port() function.
AcceptConnection	Boolean indicating if the pending connection should be accepted.
FIDInfoLength	Indicates the number of bytes that are contained in the FID Information structure.
FIDInfo	A pointer to the Full ID information that identifies the accessory and its capabilities. This information is specific to each application.
MaxRxPacketSize	This defines the maximum payload that can be accepted from a remote Apple Device. The value specified should match that value specified in the Accessory Info parameter of the FID Information data.

Return:

Zero if successful.

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

ISPP_ERROR_INSUFFICIENT_RESOURCES
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ietPort_Open_Indication

ISPP_Register_SDP_Record

This function provides a means to add a generic SDP Service Record to the SDP Database. By default, this function will add a 128 bit UUID to the SDP record that identifies this port as supporting the iAP protocol.

Notes:

1. This function should only be called with the SerialPortID that was returned from the ISPP_Open_Server_Port() function. This function should **never** be used with the Serial Port ID returned from the ISPP_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 ISPP_Un_Register_SDP_Record() to SDP_Delete_Service_Record(), and is defined as follows:

   ISPP_Un_Register_SDP_Record(__BluetoothStackID, __SerialPortID, __SDPRecordHandle)
3. If no UUID information is specified in the SDPServiceRecord Parameter, then the default SPP Service Classes 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 **ISPP_Register_SDP_Record**(unsigned int BluetoothStackID, unsigned int SerialPortID, SPP_SDP_Service_Record_t *SDPServiceRecord, char *ServiceName, DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID	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 ISPP_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;
} ISPP_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:

ISPP_ERROR_INSUFFICIENT_RESOURCES
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_INVALID_PARAMETER

ISPP_Register_Raw_SDP_Record

This function provides a means to add a generic raw SDP Service Record to the SDP Database. By default, this function will add a 128 bit UUID to the SDP record that identifies this port as supporting the iAP protocol.

Notes:

1. This function should only be called with the SerialPortID that was returned from the ISPP_Open_Server_Port() function. This function should **never** be used with the Serial Port ID returned from the ISPP_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 ISPP_Un_Register_SDP_Record() to SDP_Delete_Service_Record(), and is defined as follows:

   ISPP_Un_Register_SDP_Record(__BluetoothStackID, __SerialPortID, __SDPRecordHandle)
3. If no UUID information is specified in the SDPServiceRecord Parameter, then the default SPP Service Classes 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 **ISPP_Register_Raw_SDP_Record**(unsigned int BluetoothStackID, unsigned int SerialPortID, ISPP_SDP_Raw_Service_Record_t *SDPServiceRecord, char *ServiceName, DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID	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 ISPP_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;
} ISPP_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:

ISPP_ERROR_INSUFFICIENT_RESOURCES
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_INVALID_PARAMETER

ISPP_Open_Remote_Port

This function is used to open a remote an ISPP port on the specified Remote Device.

Prototype:

int BTPSAPI **ISPP_Open_Remote_Port**(unsigned int BluetoothStackID,
BD_ADDR_t BD_ADDR, unsigned int ServerPort, int FIDInfoLength,
unsigned char *FIDInfo, int MaxRxPacketSize,
SPP_Event_Callback_t SPP_Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID	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.
FIDInfoLength	Indicates the number of bytes that are contained in the FID Information structure.
FIDInfo	A pointer to the Full ID information that identifies the accessory and its capabilities. This information is specific to each application.
MaxRxPacketSize	This defines the maximum payload that can be accepted from a remote Apple Device. The value specified should match that value specified in the Accessory Info parameter of the FID Information data.
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:

ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INSUFFICIENT_RESOURCES
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ietPort_Open_Confirmation

ISPP_Close_Port

This function is used to close an ISPP Port that was previously opened with the ISPP_Open_Server_Port() function ***or*** the ISPP_Open_Remote_Port() function. This function does **not** unregister an ISPP Server Port from the system, it only disconnects any connection that is currently active on the port. The ISPP_Close_Server_Port() function can be used to Unregister the ISPP Server Port.

Prototype:

int BTPSAPI **ISPP_Close_Port**(unsigned int BluetoothStackID, unsigned int SerialPortID)

Parameters:

BluetoothStackID	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 ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.

Return:

Zero if successful.

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

ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_PARAMETER
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ISPP_Start_Authorization

The following function is used to start the Authentication Process with a remote Apple device.

Prototype:

int BTPSAPI **ISPP_Start_Authorization**(unsigned int BluetoothStackID,
unsigned int SerialPortID)

Parameters:

BluetoothStackID	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 ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.

Return:

Zero if successful.

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

ISPP_ERROR_NOT_ALLOWED
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ietPort_Process_Status

ISPP_Cancel_Authorization

The following function is used to abort an Authentication Process that is currently in progress.

Prototype:

int BTPSAPI **ISPP_Cancel_Authorization**(unsigned int BluetoothStackID,
unsigned int SerialPortID)

Parameters:

BluetoothStackID	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 ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.

Return:

Zero if successful.

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

ISPP_ERROR_NOT_ALLOWED
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_PARAMETER

ISPP_Open_Session_Request_Response

The following function is used to respond to an Open Session Request from a remote Apple device.

Prototype:

int BTPSAPI **ISPP_Open_Session_Request_Response**(unsigned int BluetoothStackID, unsigned int SerialPortID, unsigned short SessionID, Boolean_t Accept)

Parameters:

BluetoothStackID	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 ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.
SessionID	identifies the ID of the Session that is being established.
Accept	indicates whether the user wants to accept or reject the session request.

Return:

Zero if successful.

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

ISPP_ERROR_NOT_ALLOWED
ISPP_ERROR_INVALID_SESSION_ID
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ietPort_Session_Data_Indication

ISPP_Send_Session_Data

The following function is used to send session data to a remote device that is referenced by the specified SessionID.

Prototype:

int BTPSAPI **ISPP_Send_Session_Data**(unsigned int BluetoothStackID,
unsigned int SerialPortID, unsigned short SessionID, unsigned short DataLength,
unsigned char *DataPtr)

Parameters:

BluetoothStackID	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 ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.
SessionID	Identifies the session for which the data is to be associated with.
DataLength	Specifies the number of bytes of data that is to be sent.
DataPtr	Pointer to the session data that is to be sent.

Return:

Positive, non-zero if successful. The return value is a Packet Identifier that identifies the session data packet. This is the value that should be used in a call to ISPP_Cancel_Packet(). Each packet that is sent will be acknowledged by the remote device. Upon receiving an acknowledgment, a confirmation callback will be issued specifying the identifier of the packet that was acknowledged. Automatic retransmission of unacknowledged data is handled by this module.

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

ISPP_ERROR_NOT_ALLOWED
ISPP_ERROR_INVALID_SESSION_ID
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ietPort_Send_Session_Data_Confirmation

ISPP_Cancel_Packet

The following function is used to cancel a packet that has been queued for sending, but has yet to be sent.

Prototype:

int **BTPSAPI ISPP_Cancel_Packet**(unsigned int BluetoothStackID,
unsigned int SerialPortID, unsigned int PacketID)

Parameters:

BluetoothStackID	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize().
SerialPortID	The port on which the packet is queued. This is the value that was returned from the ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.
PacketID	The packet ID that was returned from a call to ISPP_Send_Session_Data() or ISPP_Send_Raw_Data().

Return:

Positive, non-zero if successful. The return value is a Packet Identifier that identifies the data packet.

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

ISPP_ERROR_NOT_ALLOWED
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_PACKET_ID
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ietPort_Send_Raw_Data_Confirmation

ISPP_Ack_Last_Session_Data_Packet

The following function is used to acknowledge a session data packet that was not automatically acknowledged in the iSPP Event Callback.

Prototype:

int **BTPSAPI ISPP_Ack_Last_Session_Data_Packet**(unsigned int BluetoothStackID,
unsigned int SerialPortID, Word_t SessionID)

Parameters:

BluetoothStackID	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize().
SerialPortID	The port on which the packet is queued. This is the value that was returned from the ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.

SessionID	Identifies the session for which the data is to be associated with.

Return:

Zero if successful.

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

ISPP_ERROR_NOT_ALLOWED
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_SESSION_ID
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ietPort_Send_Raw_Data_Confirmation

ISPP_Send_Raw_Data

The following function is provided to allow for the creation and sending of Lingo packets that are not currently supported by this module.

Prototype:

int BTPSAPI **ISPP_Send_Raw_Data**(unsigned int BluetoothStackID, unsigned int SerialPortID, unsigned char Lingo, unsigned char CommandID, unsigned short TransactionID, unsigned short PacketDataLength, unsigned char *PacketDataPtr)

Parameters:

BluetoothStackID	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 ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.
Lingo	Lingo identifier of the packet.
CommandID	Command ID associated with the data packet.
TransactionID	For response packets, this identifies the data packet that is being responded to. For command packets, this must be set to zero.
PacketLength	Specifies the number of bytes of data that is to be sent
PacketDataPtr	Pointer to the data associated with the Lingo and Command ID.

Return:

Positive, non-zero if successful. The return value is a Packet Identifier that identifies the data packet. This is the value that should be used in a call to ISPP_Cancel_Packet(). The module does not handle any retransmissions of raw data packets. Once the packet has been sent, an event will be dispatched which includes the packet identifier.

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

ISPP_ERROR_NOT_ALLOWED
ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_PARAMETER

Possible Events:

ietPort_Send_Raw_Data_Confirmation

ISPP_Get_Port_Operating_Mode

This function is used to query the lower layer about the current operating mode of a specified active port.

Prototype:

int BTPSAPI **ISPP_Get_Port_Operating_Mode**(unsigned int BluetoothStackID, unsigned int SerialPortID, Port_Operating_Mode_t *OperatingMode)

Parameters:

BluetoothStackID	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 ISPP_Open_Server_Port() or ISPP_Open_Remote_Port() function.
OperatingMode	Pointer to a variable that will receive the mode information.

Return:

Zero if successful.

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

ISPP_ERROR_INVALID_SERIAL_PORT_ID
ISPP_ERROR_INVALID_BLUETOOTH_STACK_ID
ISPP_ERROR_NOT_INITIALIZED
ISPP_ERROR_INVALID_PARAMETER

2.2 iAP/Serial Port Profile Event Callback Prototypes

The event callback functions mentioned in the iAP/Serial Port Profile Registration or Connection commands all accept the callback function described by the following prototype.

ISPP_Event_Callback_t

Prototype of callback function passed in one of the commands that register a callback.

Prototype:

```
void (BTPSAPI *ISPP_Event_Callback_t)(unsigned int BluetoothStackID,
    ISPP_Event_Data_t *ISPP_Event_Data, unsigned long CallbackParameter)
```

Parameters:

BluetoothStackID	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize().
ISPP_Event_Data	Data describing the event for which the callback function is called. This is defined by the following structure:

```
typedef struct
{
  ISPP_Event_Type_t   Event_Data_Type;
  Word_t              Event_Data_Size;
  union
  {
     SPP_Open_Port_Indication_Data_t
           *ISPP_Open_Port_Indication_Data;
     SPP_Open_Port_Confirmation_Data_t
           *ISPP_Open_Port_Confirmation_Data;
     SPP_Close_Port_Indication_Data_t
           *ISPP_Close_Port_Indication_Data;
     SPP_Port_Status_Indication_Data_t
           *ISPP_Port_Status_Indication_Data;
     SPP_Data_Indication_Data_t
           *ISPP_Data_Indication_Data;
     SPP_Transmit_Buffer_Empty_Indication_Data_t
           *ISPP_Transmit_Buffer_Empty_Indication_Data;
     SPP_Line_Status_Indication_Data_t
           *ISPP_Line_Status_Indication_Data;
     SPP_Send_Port_Information_Indication_Data_t
           *ISPP_Send_Port_Information_Indication_Data;
     SPP_Send_Port_Information_Confirmation_Data_t
           *ISPP_Send_Port_Information_Confirmation_Data;
     SPP_Query_Port_Information_Indication_Data_t
           *ISPP_Query_Port_Information_Indication_Data;
     SPP_Query_Port_Information_Confirmation_Data_t
           *ISPP_Query_Port_Information_Confirmation_Data;
     SPP_Open_Port_Request_Indication_Data_t
           *ISPP_Open_Port_Request_Indication_Data;

     ISPP_Process_Status_Data_t
           *ISPP_Process_Status;
     ISPP_Session_Open_Indication_Data_t
           *ISPP_Session_Open_Indication;
     ISPP_Session_Close_Indication_Data_t
           *ISPP_Session_Close_Indication;
     ISPP_Session_Data_Indication_Data_t
           *ISPP_Session_Data_Indication;
```

```
            ISPP_Send_Session_Data_Confirmation_Data_t
                  *ISPP_Send_Session_Data_Confirmation;
            ISPP_Raw_Data_Indication_Data_t
                  *ISPP_Raw_Data_Indication;
            ISPP_Send_Raw_Data_Confirmation_Data_t
                  *ISPP_Send_Raw_Data_Confirmation;
         } Event_Data;
      } ISPP_Event_Data_t;
```

where, Event_Data_Type is one of the enumerations of the event types listed in the table in section 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.

2.3 iAP/Serial Port Profile Events

The possible iAP/Serial Port Profile events from the Bluetooth stack are listed in the table below and are described in the text that follows:

Event	Description
ietPort_Open_Indication	Indicates that a connection has been made to a Server that was previously registered.
ietPort_Open_Confirmation	Indicates that a connection to a remote device is complete. The status value in the ISPP_Open_Port_Confirmation_Data structure should be examined to determine if the connection was successful.
ietPort_Close_Port_Indication	Indicates that the connection to a remote device has been terminated.
ietPort_Open_Request_Indication	Indicates that a remote device is attempting to connect to a registered server port.
ietPort_Process_Status	Provides status information about the identification and authorization process
ietPort_Open_Session_Indication	Indicates that the request has been received to establish a data session.
ietPort_Close_Session_Indication	Indicates that a data session has been terminated.
ietPort_Session_Data_Indication	Indicates that data has been received for an established session.
ietPort_Send_Session_Data_Confirmation	Indicates that a session data packet has been sent and acknowledged.

ietPort_Raw_Data_Indication	Indicates a data packet that is not handled by this module has been received from the remote device.
ietPort_Send_Raw_Data_Confirmation	Indicates a data packet that was queued has been transmitted.
ietPort_Status_Indication[1]	Indicate that a change in port status has been received.
ietPort_Data_Indication[1]	Indicate that data has arrived on a port.
ietPort_Transmit_Buffer_Empty_Indication[1]	Indicate when the Transmit Buffer is Empty (only if the Transmit Buffer was completely full or the ISPP_Purge_Buffer() function was called with the option to flush the transmit buffer).
ietPort_Line_Status_Indication[1]	Indicate that a change in line status has been received.
ietPort_Send_Port_Information_Indication[1]	Indicate that a remote device's port parameters have been received (start of negotiation of parameters).
ietPort_Send_Port_Information_Confirmation[1]	Confirm that a response has been received to a send port parameters command.
ietPort_Query_Port_Information_Indication[1]	Indicate that a request to send current port parameters has been received.
ietPort_Query_Port_Information_Confirmation[1]	Confirm that a response has been received to a request to send current port parameters.

[1] These events are dispatched by SPP and forwarded by this module. Refer to the SPP section of the Bluetopia Core API documentation for further information.

ietPort_Open_Indication

The following structure is associated with the ietPort_Open_Indication. This module wraps the SPP Open Port Indication structure.

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.

ietPort_Open_Confirmation

The following structure is associated with the ietPort_Open_Confirmation. This module wraps the SPP Open Port Confirmation structure.

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: ISPP_OPEN_PORT_STATUS_SUCCESS ISPP_OPEN_PORT_STATUS_CONNECTION_TIMEOUT ISPP_OPEN_PORT_STATUS_CONNECTION_REFUSED ISPP_OPEN_PORT_STATUS_UNKNOWN_ERROR

ietPort_Close_Port_Indication

The following structure is associated with the ietPort_Close_Port_Indication. This module wraps the SPP Close Port Indication structure.

Return Structure:

```
typedef struct
{
   unsigned int   SerialPortID;
} SPP_Close_Port_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.

ietPort_Open_Request_Indication

The following structure is associated with the ietPort_Open_Request_Indication. This module wraps the SPP Open Request Indication structure.

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.

ietPort_Process_Status

The following structure is associated with the ietPort_Process_Status event. The structure provides information about the Apple Identification and Authentication process.

Return Structure:

```
typedef struct
{
   unsigned int      SerialPortID;
   Process_State_t   ProcessState;
   Byte_t            Status;
} ISPP_Process_Status_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
ProcessState	State of the process, one of the following values: psStartIdentificationProcess psIdentificationProcess psIdentificationProcessComplete psStartAuthenticationProcess psAuthenticationProcess psAuthenticationProcessComplete
Status	Status of the current state, one of the following vales: IAP_PROCESS_STATUS_SUCCESS IAP_PROCESS_STATUS_ERROR_RETRYING IAP_PROCESS_STATUS_TIMEOUT_HALTING IAP_PROCESS_STATUS_GENERAL_FAILURE IAP_PROCESS_STATUS_PROCESS_FAILURE IAP_PROCESS_STATUS_PROCESS_FAILURE_RETRYING

ietPort_Open_Session_Indication

The following structure is associated with the ietPort_Open_Session_Indication event.

Return Structure:

```
typedef struct
{
   unsigned int   SerialPortID;
   Word_t         SessionID;
   Byte_t         ProtocolIndex;
   Word_t         MaxPacketPayloadSize;
} ISPP_Session_Open_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
SessionID	Identifies the session that was opened.
ProtocolIndex	Identifies the protocol that will be used over the port

MaxPacketPayloadSize	Identifies the Max payload that can be received on the port.

ietPort_Close_Session_Indication

The following structure is associated with the ietPort_Close_Session_Indication event.

Return Structure:

```
typedef struct
{
   unsigned int   SerialPortID;
   Word_t         SessionID;
} ISPP_Session_Close_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
SessionID	Identifies the session that was opened.

ietPort_Session_Data_Indication

The following structure is associated with the ietPort_Session_Data_Indication event.

Return Structure:

```
typedef struct
{
   unsigned int    SerialPortID;
   Word_t          SessionID;
   Word_t          DataLength;
   Byte_t         *DataPtr;
   Boolean_t      *PacketConsumed;
} ISPP_Session_Data_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
SessionID	Identifies the session that was opened.
DataLength	Indicates the number of bytes that was received.
DataPtr	Pointer to the data that was received.
PacketConsumed	Flag returned to indicate that the packet was consumed by the upper layer. If the value returned is TRUE, the packet is ACKed and packet flow continues. If the value returned is FALSE, the packet will not be ACKed and the packet flow will be suspended. If the packet is not consumed, the upper layer must maintain the pointer to the data and the length of the data so that it can be consumed when resources become available. Once the packet is consumed, a call to ISPP_Ack_Last_Session_Data_Packet() must be called to release the packet data. This will ACK the pending packet and

allows the flow of packets again. NOTE: Delaying the ACK of a packet for too long of a period may cause the connection to the iOS device to be dropped.

ietPort_Send_Session_Data_Confirmation

The following structure is associated with the ietPort_Send_Session_Data_Confirmation event.

Return Structure:

```
typedef struct
{
   unsigned int    SerialPortID;
   unsigned int    PacketID;
   Word_t          SessionID;
   Byte_t          Status;
} ISPP_Send_Session_Data_Confirmation_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
PacketID	Identifies the packet that has been acknowledged.
SessionID	Identifies the session that was opened.
Status	Status of the packet delivery, one of the following values: PACKET_SEND_STATUS_SENT PACKET_SEND_STATUS_ACKNOWLEDGED PACKET_SEND_STATUS_FAILED PACKET_SEND_STATUS_CANCELED

ietPort_Raw_Data_Indication

The following structure is associated with the ietPort_Raw_Data_Indication event.

Return Structure:

```
typedef struct
{
   unsigned int    SerialPortID;
   Byte_t          Lingo;
   Byte_t          Command;
   Word_t          DataLength;
   Byte_t         *DataPtr;
} ISPP_Raw_Data_Indication_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
Lingo	Lingo for the associated data.
Command	Command ID associated with the data.

TransactionID	ID used to reference the packet.
DataLength	Number of data bytes in the packet.
DataPtr	Pointer to the data received

ietPort_Send_Raw_Data_Confirmation

The following structure is associated with the ietPort_Send_Raw_Data_Confirmation event.

Return Structure:

```
typedef struct
{
   unsigned int     SerialPortID;
   unsigned int     PacketID;
   unsigned short   TransactionID;
   Byte_t           Status;
} ISPP_Send_Raw_Data_Confirmation_Data_t;
```

Event Parameters:

SerialPortID	The port this event applies to.
PacketID	Identifies the packet that has been acknowledged.
TransactionID	Identifies the transaction ID of packet .
Status	Status of the packet delivery, one of the following values: PACKET_SEND_STATUS_SENT PACKET_SEND_STATUS_ACKNOWLEDGED PACKET_SEND_STATUS_FAILED PACKET_SEND_STATUS_CANCELED

3. File Distributions

The header files that are distributed with the ISPP Library are listed in the table below.

File	Contents/Description
IAPTypes.h	iAP API Type definitions
ISPPAPI.h	iAP/Serial Port Profile API
SS1BTISP.h	Bluetooth iAP/SPP Include file