OBEX Object Push Profile (OBJP)

Application Programming Interface Reference Manual

Profile Version: 1.1

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 (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 OBEX Object Push Profile provided by Bluetopia. Chapter 2 contains a description of the programming interface for this profile. And, Chapter 3 contains the header file name list for the Bluetooth OBEX Object Push Profile 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

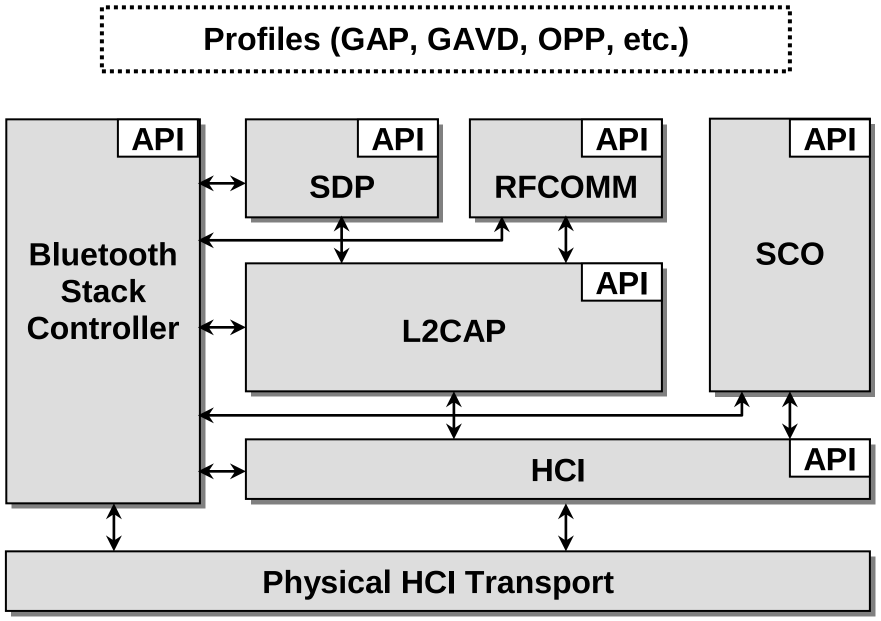

Figure 1-1 The Stonestreet One Bluetooth Protocol Stack

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

18. *Specification of the Bluetooth System, Volume 0, Master Table of Contents & Compliance Requirements,* version 4.0, June 30, 2010.
19. *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 4.0, June 30, 2010.
20. *Specification of the Bluetooth System, Volume 2, Core System Package [BR/EDR Controller Volume]*, version 4.0, June 30, 2010.
21. *Specification of the Bluetooth System, Volume 3, Core System Package [Host Volume]*, version 4.0, June 30, 2010.
22. *Specification of the Bluetooth System, Volume 4, Host Controller Interface [Transport Layer]*, version 4.0, June 30, 2010.
23. *Specification of the Bluetooth System, Volume 5, Core System Package [AMP Controller Volume]*, version 4.0, June 30, 2010.
24. *Specification of the Bluetooth System, Volume 6, Core System Package [Low Energy Controller Volume]*, version 4.0, June 30, 2010.
25. *Bluetooth Assigned Numbers,* version 1.1, February 22, 2001.
26. *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.
27. *Infrared Data Association, IrDA Object Exchange Protocol (IrOBEX) with Published Errata*, Version 1.2, April 1999.
28. *Bluetopia^{TM}Protocol Stack, Application Programming Interface Reference Manual,* version 4.0.1, April 5, 2012.

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
BR	Basic Rate
BSC	Bluetooth Stack Controller

Term	Meaning
BT	Bluetooth
EDR	Enhanced Data Rate
HS	High Speed
LE	Low Energy
LSB	Least Significant Bit
MSB	Most Significant Bit
SDP	Service Discovery Protocol
SPP	Serial Port Protocol
UAP	Upper Address Part
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

2. OBEX Object Push Profile Programming Interfaces

The OBEX Object Push Profile programming interface defines the protocols and procedures to be used to implement object push capabilities. The OBEX Object Push Profile commands are listed in section 2.1, the event callback prototype is described in section 2.2, and the OBEX Object Push Profile events are itemized in section 2.3. The actual prototypes and constants outlined in this section can be found in the **OBJPAPI.H** header file in the Bluetopia distribution.

2.1 OBEX Object Push Profile Commands

The available OBEX Object Push Profile command functions are listed in the table below and are described in the text that follows.

Function	Description
OBJP_Open_Server	Open an Object Push Server.
OBJP_Close_Server	Close an open Object Push Server.
OBJP_Close_Server_Connection	Close an Object Push connection to a local server.
OBJP_Server_Connect_Request_Response	Respond to a connect request from the remote device.
OBJP_Register_Server_SDP_Record	Add a generic Object Push SDP Service Record to the SDP database
OBJP_Open_Remote_Object_Server	Open a remote Object Push Server.
OBJP_Close_Client	Terminate connection to a remote Object Push Server.
OBJP_Get_Default_Object	Retrieve the Default Object from the specified Remote Object Push Server
OBJP_Put_Object	Send the specified Object to the specified Remote Object Push Server
OBJP_Abort	Abort ANY currently outstanding Object Push Client Request
OBJP_Change_Reject_Request_Mask	Change the currently active Reject Request Mask
OBJP_Get_Server_Mode	Provides a mechanism to query the current Object Push Server Mode.
OBJP_Set_Server_Mode	Provides a mechanism to change the current Object Push Server Mode.

OBJP_Open_Server

This function is responsible for Opening an Object Push Server on the specified Bluetooth SPP Serial Port. Once an Object Push Server is opened, it can only be Un-Registered via a call to the OBJP_Close_Server() function (passing the return value from this function).

Prototype:

int BTPSAPI **OBJP_Open_Server**(unsigned int BluetoothStackID, unsigned int ServerPort,
char *InBoxDirectory, char *DefaultBusinessCardFileName,
unsigned long SupportedObjectsMask, unsigned long RejectRequestsMask,
OBJP_Server_Event_Callback_t EventCallback, unsigned long CallbackParameter);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
ServerPort	SPP port number to use. This must fall in the range defined by the following constants: SPP_PORT_NUMBER_MINIMUM SPP_PORT_NUMBER_MAXIMUM
InBoxDirectory	Pointer to a NULL terminated Wide string that specifies the Local Directory Path of the directory to use as the INBOX for the Object Server
DefaultBusinessCardFileName	Pointer to a NULL terminated Wide string that specifies the default business card file name, which must be specified and must be located in the INBOX directory.
SupportedObjectsMask	The Supported Object List specifies the Bit Mask of supported Objects that can be pushed to this server. This parameter must specify at least (vCard 2.1 support - or Any Object Support).
RejectRequestsMask	Specifies the object requests that are to be rejected by this object server. This parameter can be zero, which means do NOT reject ANY object requests for the Object Push Server.
EventCallback	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 value if successful or a negative return error code if an error occurs. A successful return code will be an Object Push ID that can be used to reference the Opened Object Push Server in ALL other Object Push Server functions.

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

BTOBJP_ERROR_INSUFFICIENT_RESOURCES
BTOBJP_ERROR_INVALID_DEFAULT_BUSINESS_CARD
BTOBJP_ERROR_INVALID_ROOT_DIRECTORY
BTOBJP_ERROR_NOT_INITIALIZED

BTOBJP_ERROR_INVALID_PARAMETER

If zero, there was an error opening the OTP port and the port was not opened successfully.

Possible Events:

etOBJP_Server_Connect_Request_Indication

etOBJP_Server_Connect_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.

OBJP_Close_Server

This function is responsible for un-registering an Object Push Server (which was registered by a successful call to the OBJP_Open_Server() function). Note that this function does NOT delete any SDP Service Record Handles.

Prototype:

int BTPSAPI **OBJP_Close_Server**(unsigned int BluetoothStackID, unsigned int OBJPID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The Object Push ID to close. This is the value that was returned from the OBJP_Open_Server( ) function.

Return:

Zero if successful.

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

BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_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.

OBJP_Close_Server_Connection

This function closes an Object Push connection by a remote client to a local server. This function can only be called by the Object Push Server. A successful call to this function terminates the Object Push connection. This function does NOT Un-Register an Object Push Server from the system, it ONLY disconnects any connection that is currently active on the Server. The OBJP_Close_Server() function can be used to Un-Register an Object Push Server.

Prototype:

int BTPSAPI **OBJP_Close_Server_Connection**(unsigned int BluetoothStackID,
unsigned int OBJPID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The Object Push server port to close the connection on. This is the value that was returned from the OBJP_Open_Server ( ) function.

Return:

Zero if successful.

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

BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_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.

OBJP_Server_Connect_Request_Response

This function is responsible for responding to requests to connect to an Object Push 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 **OBJP_Server_Connect_Request_Response**(unsigned int BluetoothStackID,
unsigned int OBJPID, Boolean_t AcceptConnection)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The OBJP ID this command applies to. This is the value that was returned from the OBJP_Open_Server( ) 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:

BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_ERROR_INVALID_PARAMETER

Possible Events:

etOBJP_Server_Connect_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.

OBJP_Register_Server_SDP_Record

This function adds a generic SDP Service Record to the SDP Database.

Notes:

1. This function should only be called with the OBJP ID that was returned from the OBJP_Open_Server() function. This function should NEVER be used with OBJP ID returned from the OBJP_Open_Remote_Server() 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 Data Base. This MACRO maps the OBJP_Un_Register_SDP_Record() to the SDP_Delete_Service_Record() function and is defined as follows:

OBJP_Un_Register_SDP_Record(__BluetoothStackID, __OBJPID, __SDPRecordHandle)
(SDP_Delete_Service_Record(__BluetoothStackID, __SDPRecordHandle))

3. 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 **OBJP_Register_Server_SDP_Record**(unsigned int BluetoothStackID,
unsigned int OBJPID, char *ServiceName, DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The OBJPID this command applies to. This value must be the value that was returned from the OBJP_Open_Server( ) function.
ServiceName	Name to appear in the SDP Database for this service.
SDPServiceRecordHandle	Returned handle to the SDP Database entry that 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:

BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_ERROR_INVALID_PARAMETER
BTOBJP_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.

OBJP_Open_Remote_Object_Server

This function opens a remote Object Push Server. The caller of this function becomes the Object Push Client. Once a remote server is opened, it can only be closed via a call to the OBJP_Close_Client() function (passing the return value from this function).

Prototype:

int BTPSAPI **OBJP_Open_Remote_Object_Server**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, unsigned int ServerPort, OBJP_Client_Event_Callback_t EventCallback, 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 Object Push Server port ID to connect with. This must fall in the range defined by the following constants: SPP_PORT_NUMBER_MINIMUM SPP_PORT_NUMBER_MAXIMUM
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 value if successful or a negative return error code if an error occurs. A successful return code will be an Object Push ID that can be used to reference the Opened Object Push Server in ALL other Object Push Client functions.

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

BTOBJP_ERROR_INSUFFICIENT_RESOURCES
BTOBJP_ERROR_INVALID_DEFAULT_BUSINESS_CARD
BTOBJP_ERROR_INVALID_ROOT_DIRECTORY
BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_ERROR_INVALID_PARAMETER

Possible Events:

etOBJP_Client_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.

OBJP_Close_Client

This function closes an Object Push Client connection to a remote server that was previously opened with the OBJP_Open_Remote_File_Server( ) function. This function can only be called by the Object Push Client. A successful call to this function terminates the remote Object Push Server connection.

Prototype:

int BTPSAPI **OBJP_Close_Client**(unsigned int BluetoothStackID, unsigned int OBJPID)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The Object Push remote server port to close. This is the value that was returned from the OBJP_Open_Remote_File_Server( ) function.

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

BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_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.

OBJP_Get_Default_Object

This function is responsible for retrieving the default object from the specified Remote Object Push Server.

Notes:

1. Issuing this command successfully does not mean that the Remote Object Push Server successfully issued the command. The caller needs to check the confirmation result to determine if the Remote Object Push Server successfully executed the Request.
2. Due to an OBEX Object Push limitation, there can only be one outstanding Object Push Client request active at any one time. Because of this, another Object Push Client request cannot be issued until either the current request is aborted (by calling the OBJP_Abort() function) or the current request is complete (this is signified by receiving a Confirmation Event in the Object Push Client Event Callback that was registered when the Object Push Client was opened).

Prototype:

int BTPSAPI **OBJP_Get_Default_Object**(unsigned int BluetoothStackID, unsigned int OBJPID, char *LocalPath, char *LocalObjectFileName)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The Object Push Client ID returned from a successful call to the OBJP_Open_Remote_Object_Server() function.
LocalPath	Pointer, if NULL, then the retrieved object is written to the current directory (on the local machine). If not a NULL pointer, it is a pointer to a NULL terminated ASCII string, and the retrieved object will be written to the directory specified by the string.
LocalObjectFileName	Pointer to a NULL terminated ASCII string and MUST be specified. The retrieved object will be written to this file name on the local machine.

Return:

Zero if successful.

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

BTOBJP_ERROR_INSUFFICIENT_RESOURCES
BTOBJP_ERROR_UNABLE_TO_CREATE_LOCAL_FILE
BTOBJP_ERROR_NOT_INITIALIZED

	BTOBJP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED BTOBJP_ERROR_INVALID_PARAMETER

Possible Events:

etOBJP_Client_Get_Confirmation

etOBJP_Client_Disconnect_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.

OBJP_Put_Object

This function is responsible for sending the specified object to the specified Remote Object Push Server.

Notes:

1. Issuing this command successfully does not mean that the Remote Object Push Server successfully issued the command. The caller needs to check the confirmation result to determine if the Remote Object Push Server successfully executed the Request.
2. Due to an OBEX Object Push limitation, there can only be one outstanding Object Push Client request active at any one time. Because of this, another Object Push Client request cannot be issued until either the current request is aborted (by calling the OBJP_Abort() function) or the current request is complete (this is signified by receiving a Confirmation Event in the Object Push Client Event Callback that was registered when the Object Push Client was opened).

Prototype:

int BTPSAPI **OBJP_Put_Object**(unsigned int BluetoothStackID, unsigned int OBJPID, OBJP_ObjectType_t ObjectType, char *LocalPath, char *LocalObjectFileName, char *RemoteObjectName)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The Object Push Client ID returned from a successful call to the OBJP_Open_Remote_Object_Server() function.
ObjectType	Enumerated data type that specifies the type of object that is being pushed to the Remote Object Push Server. The following enumerations specify the type of objects that are supported and are defined as follows:

```
typedef enum
{
  obtvCard,
  obtvCalendar,
  obtiCalendar,
  obtvNote,
  obtvMessage,
  obtUnknownObject
} OBJP_ObjectType_t;
```

LocalPath	Pointer, if NULL, then the retrieved object is written to the current directory (on the local machine). If not a NULL pointer, it is a pointer to a NULL terminated ASCII string, and the retrieved object will be written to the directory specified by the string.
LocalObjectFileName	Pointer to a NULL terminated ASCII string and MUST be specified. The retrieved object will be written to this file name on the local machine.
RemoteObjectName	Pointer to NULL terminated ASCII string that specifies the name of the object that is to be stored on the Remote Object Push Server. The Object that is sent to the Remote Object Push Server is stored in the INBOX of the Remote Object Push Server.

Return:

Zero if successful.

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

BTOBJP_ERROR_INSUFFICIENT_RESOURCES
BTOBJP_ERROR_UNABLE_TO_CREATE_LOCAL_FILE
BTOBJP_ERROR_INVALID_PARAMETER
BTOBJP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED

Possible Events:

etOBJP_Client_Put_Confirmation

etOBJP_Client_Disconnect_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.

OBJP_Abort

This function aborts any current outstanding Object Push Client requests.

Notes:

1. This Object Push Client request is no different than the other Object Push Client request functions in that a response to this request must be received before any other Object Push Client Request can be issued.
2. Because of transmission latencies, it may be possible that an Object Push Client request that is to be aborted may have completed before the server was able to abort the request. In either case, the caller will be notified via Object Push Client callback of the status of the previous request.
3. Due to the nature of only one outstanding OBEX Command when an Abort is issued, it may be queued (for transmission when the response to the currently outstanding OBEX Command is received. A problem can occur if the Remote OBEX Server does not respond to the original request because the queued Abort Packet will never be sent. This is a problem because no new OBEX commands can be issued because the OBEX layer on the local machine thinks a Request is outstanding and will not issue another request. To aid in error recovery, this function forces an Abort Request out (and the clearing of the current OBEX Command Request) if this function is called twice. An application can call this function a second time to force a local cleanup if a response on the first Abort Packet is never received (via the Object Push Client Callback). It should be noted that under normal circumstances (i.e. the Remote Server is functioning properly) this function will NEVER have to be called twice.

Prototype:

int BTPSAPI **OBJP_Abort**(unsigned int BluetoothStackID, unsigned int OBJPID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The Object Push Client ID returned from a successful call to the OBJP_Open_Remote_File_Server() function.

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

BTOBJP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTOBJP_ERROR_INVALID_PARAMETER

Possible Events:

etOBJP_Client_Abort_Confirmation

etOBJP_Client_Disconnect_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.

OBJP_Change_Reject_Request_Mask

This function is responsible for changing the currently active Reject Request Mask. This function DOES NOT abort any current Object Push/Pull operations that are currently in progress. After this function completes successfully, only FUTURE Object Requests will be affected by a successful call to this function.

Prototype:

int BTPSAPI **OBJP_Change_Reject_Request_Mask**(unsigned int BluetoothStackID, unsigned int OBJPID, unsigned long NewRejectRequestsMask)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The Object Push Client ID returned from a successful call to the OBJP_Open_Remote_File_Server() function.
NewRejectRequestsMask	Specifies the NEW Reject Request Mask. This mask will replace the currently active Reject Request Mask that was set either by a prior call to this function OR when the Server was opened via a call to the OBJP_Open_Server() function.

Return:

Zero if successful.

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

BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTOBJP_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.

OBJP_Get_Server_Mode

This function is responsible for allowing a mechanism to query the Object Push Server Mode.

Prototype:

int BTPSAPI **OBJP_Get_Server_Mode**(unsigned int BluetoothStackID, unsigned int OBJPID, unsigned long *ServerModeMask)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The OBJP ID this command applies to. This is the value that was returned from the OBJP_Open_Server( ) function.
ServerModeMask	Pointer to a variable to receive the current Server Mode Mask. See ServerModeMask parameter of OBJP_Set_Server_Mode() below for a list of the possible values.

Return:

Zero if successful.

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

BTOBJP_ERROR_NOT_INITIALIZED
BTOBJP_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.

OBJP_Set_Server_Mode

This function is responsible for allowing a mechanism to change the Object Push Server Mode.

Prototype:

int BTPSAPI **OBJP_Set_Server_Mode**(unsigned int BluetoothStackID, unsigned int OBJPID, unsigned long ServerModeMask)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
OBJPID	The OBJP ID this command applies to. This is the value that was returned from the OBJP_Open_Server( ) function.
ServerModeMask	The new Server Mode being set. This may be one of the following:

OBJP_SERVER_MODE_AUTOMATIC_ACCEPT_CONNECTION
OBJP_SERVER_MODE_MANUAL_ACCEPT_CONNECTION

Return:

Zero if successful.

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

	BTOBJP_ERROR_NOT_INITIALIZED BTOBJP_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.

2.2 OBEX Object Push Profile Event Callback Prototypes

The event callback functions mentioned in the Object Push Profile Open commands accept the callback functions described by the following prototypes. OBJP_Open_Server() accepts the OBJP_Server_Event_Callback_t and OBJP_Open_Remote_Object_Server() accepts the OBJP_Client_Event_Callback_t.

OBJP_Server_Event_Callback_t

Prototype of callback function passed in OBJP_Open_Server(). This function will be called whenever an Object Push Event occurs that is associated with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the Object Push Server Event Data that occurred and the Object Push Server Event Callback Parameter that was specified when this Callback was installed. The caller is free to use the contents of the Object Push Server 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 Object Push Server Event 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 Object Push Server Event Packets. A Deadlock WILL occur because NO Object Push Server Event Callbacks will be issued while this function is currently outstanding.

Prototype:

void (BTPSAPI ***OBJP_Server_Event_Callback_t**)(unsigned int BluetoothStackID,
    OBJP_Server_Event_Data_t *OBJP_Server_Event_Data,
    unsigned long CallbackParameter)

Parameters:

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

OBJP_Server_Event_Data	Data describing the event for which the callback function is called. This is defined by the following structure:

```
typedef struct
{
  OBJP_Server_Event_Type_t   Event_Data_Type;
  Word_t                     Event_Data_Size;
  union
  {
    OBJP_Server_Connect_Indication_Data_t      *OBJP_Server_Connect_Indication_Data;
    OBJP_Server_Disconnect_Indication_Data_t *OBJP_Server_Disconnect_Indication_Data;
    OBJP_Server_Object_Put_Indication_Data_t *OBJP_Server_Object_Put_Indication_Data;
    OBJP_Server_Object_Get_Indication_Data_t *OBJP_Server_Object_Get_Indication_Data;
    OBJP_Server_Connect_Request_Indication_Data_t
              OBJP_Server_Connect_Request_Indication_Data;
  } Event_Data;
} OBJP_Server_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.

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.

OBJP_Client_Event_Callback_t

Prototype of callback function passed in OBJP_Open_Remote_Object_Server(). This function will be called whenever an Object Push Client Event occurs that is associated with the specified Bluetooth Stack ID. This function passes to the caller the Bluetooth Stack ID, the Object Push Client Event Data that occurred and the Object Push Client Event Callback Parameter that was specified when this Callback was installed. The caller is free to use the contents of the Object Push Client 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 Object Push Client Event 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 Object Push Client Events. A Deadlock WILL occur because NO Object Push Client Event Callbacks will be issued while this function is currently outstanding.

Prototype:

void (BTPSAPI ***OBJP_Client_Event_Callback_t**)(unsigned int BluetoothStackID,
OBJP_Client_Event_Data_t *OBJP_Client_Event_Data,
unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
OBJP_Client_Event_Data	Data describing the event for which the callback function is called. This is defined by the following structure:

```
typedef struct
{
  OBJP_Client_Event_Type_t   Event_Data_Type;
  Word_t                     Event_Data_Size;
  union
  {
    OBJP_Client_Connect_Confirmation_Data_t
             *OBJP_Client_Connect_Confirmation_Data;
    OBJP_Client_Disconnect_Indication_Data_t   *OBJP_Client_Disconnect_Indication_Data;
    OBJP_Client_Abort_Confirmation_Data_t      *OBJP_Client_Abort_Confirmation_Data;
    OBJP_Client_Object_Put_Confirmation_Data_t
             *OBJP_Client_Object_Put_Confirmation_Data;
    OBJP_Client_Object_Get_Confirmation_Data_t
             *OBJP_Client_Object_Get_Confirmation_Data;
  } Event_Data;
} OBJP_Client_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.

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.

2.3 Object Push Profile Events

The possible Object Push Profile events from the Bluetooth stack are listed in the table below and are described in the text which follows:

Event	Description
etOBJP_Server_Connect_Indication	Indicates that an Object Push Client connected to a registered Object Push Server.
etOBJP_Server_Disconnect_Indication	Dispatched when an Object Push Client disconnects from a registered Object Push Server.
etOBJP_Server_Object_Put_Indication	Dispatched when an Object Push Client puts an Object on the registered Object Push Server.
etOBJP_Server_Object_Get_Indication	Dispatched when an Object Push Client retrieves an Object from the registered Object Push Server.
etOBJP_Server_Connect_Request_Indication	Dispatched when an Object Push Client requests to connect to a registered Object Push Server.
etOBJP_Client_Connect_Confirmation	Dispatched when an Object Client receives the Connection Response from a Remote Object Push Server that was previously attempted to be connected to.
etOBJP_Client_Disconnect_Indication	Dispatched when an Object Push Client disconnects from a registered Object Push Server.
etOBJP_Client_Abort_Confirmation	Dispatched when an Object Push Client Abort Response is received from the Remote Object Push Server.

etOBJP_Client_Object_Put_Confirmation	Dispatched when an Object Push Client receives an Object Put Confirmation from the registered Object Push Server.
etOBJP_Client_Object_Get_Confirmation	Dispatched when an Object Push Client receives an Object from the registered Object Push Server.

etOBJP_Server_Connect_Indication

Dispatched when an Object Push Client Connects to a registered Object Push Server. The OBJP ID member specifies the Local Server that has been connected to and the BD_ADDR member specifies the Client Bluetooth Device that has connected to the specified Server.

Return Structure:

```
typedef struct
{
  unsigned int OBJPID;
  BD_ADDR_t    BD_ADDR;
} OBJP_Server_Connect_Indication_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.
BD_ADDR	Address of the Bluetooth Device making the request.

etOBJP_Server_Disconnect_Indication

Dispatched when an Object Push Client disconnects from a registered Object Push Server. The OBJP ID member specifies the Local Server that the Remote Client has disconnected from.

Return Structure:

```
typedef struct
{
  unsigned int OBJPID;
} OBJP_Server_Disconnect_Indication_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.

etOBJP_Server_Object_Put_Indication

Dispatched when an Object Push Client puts an Object on the registered Object Push Server. The OBJP ID member specifies the Local Server that the Remote Client has put the specified Object on. The Object Type member specifies the type of the Object that has been put on the specified server. The Object File Name member specifies the name of the Object that is being written to the INBOX Directory. The Transfer Complete flag specifies whether or not the Object Transfer has been completed (TRUE when completed, FALSE while in progress). The Transferred Length and Total Length members specify how many bytes have been transferred and how many bytes are to be transferred in total.

Return Structure:

```
typedef struct
{
  unsigned int          OBJPID;
  OBJP_ObjectType_t     ObjectType;
  char                 *ObjectFileName;
  Boolean_t             TransferComplete;
  unsigned int          TransferredLength;
  unsigned int          TotalLength;
} OBJP_Server_Object_Put_Indication_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.
ObjectType	Type of object put in INBOX.
ObjectFileName	Name of the object written to INBOX.
TransferComplete	Indicates status of transfer.
TransferredLength	Number bytes transferred.
TotalLength	Total number bytes to be transferred.

etOBJP_Server_Object_Get_Indication

Dispatched when an Object Push Client retrieves an Object from the registered Object Push Server. The OBJP ID member specifies the Local Server that the Remote Client is retrieving the specified Object from. The Object Type member specifies the type of the Object requested from the specified server. The Object File Name member specifies the name of the requested Object. The Transfer Complete flag specifies whether or not the Object Transfer has been completed (TRUE when completed, FALSE while in progress). The Transferred Length and Total Length members specify how many bytes have been transferred and how many bytes are to be transferred in total.

Return Structure:

```
typedef struct
{
   unsigned int          OBJPID;
   OBJP_ObjectType_t     ObjectType;
   char                 *ObjectFileName;
   Boolean_t             TransferComplete;
   unsigned int          TransferredLength;
   unsigned int          TotalLength;
} OBJP_Server_Object_Put_Indication_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.
ObjectType	Type of object requested.
ObjectFileName	Name of the object requested.
TransferComplete	Indicates status of transfer.
TransferredLength	Number bytes transferred.
TotalLength	Total number bytes to be transferred.

etOBJP_Server_Connect_Request_Indication

Dispatched when an Object Push Client requests to connect to a registered Object Push 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.

Return Structure:

```
typedef struct
{
   unsigned int     OBJPID;
   BD_ADDR_t        BD_ADDR;
} OBJP_Server_Connect_Request_Indication_Data_t;
```

Event Parameters:

OBJPID	Identifies the Local Server to which the Remote Client has requested to connected.
BD_ADDR	Specifies the address of the Client Bluetooth device that has requested to connect to the specified Server.

etOBJP_Client_Connect_Confirmation

Dispatched when an Object Client receives the Connection Response from a Remote Object Push Server that was previously attempted to be connected to. The OBJP ID member specifies the Local Server that has been connected to, the Object Push Open Status represents the Connection Status of the Request, and the BD_ADDR member specifies the Remote Bluetooth Device that the Remote Bluetooth Object Push Server resides on.

Return Structure:

```
typedef struct
{
  unsigned int     OBJPID;
  unsigned int     OBJPConnectStatus;
  BD_ADDR_t    BD_ADDR;
} OBJP_Client_Connect_Confirmation_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.
OBJPConnectStatus	Connection status of request.
BD_ADDR	Address of the Bluetooth Device making the request.

etOBJP_Client_Disconnect_Indication

Dispatched when an Object Push Client disconnects from a registered Object Push Server. The Object Push ID member specifies the Local Client that the Remote Server has disconnected from. This Event is NOT dispatched in response to a client requesting a disconnection. This Event is dispatched when the Remote Server terminates the connection (and/or Bluetooth Link).

Return Structure:

```
typedef struct
{
  unsigned int    OBJPID;
} OBJP_Client_Disconnect_Indication_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.

etOBJP_Client_Abort_Confirmation

Dispatched when an Object Push Client Abort Response is received from the Remote Object Push Server. The Object Push ID member specifies the Local Client that the Remote Server has responded to the Abort Request on.

Return Structure:

```
typedef struct
{
  unsigned int    OBJPID;
} OBJP_Client_Abort_Confirmation_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.

etOBJP_Client_Object_Put_Confirmation

Dispatched when an Object Push Client receives an Object Put Confirmation from the registered Object Push Server. The OBJP ID member specifies the Local Object Push Client that has requested the specified Object to be put on the Remote Server. The Object Type member specifies the type of the Object that has been put on the remote server. The Object File Name member specifies the name of the Object that is being written to the remote INBOX Directory. The Transfer Complete flag specifies whether or not the Object Transfer has been completed (TRUE when completed, FALSE while in progress). The Transferred Length and Total Length members specify how many bytes have been transferred and how many bytes are to be transferred in total. The Success member specifies whether or not the request has been completed successfully.

Return Structure:

```
typedef struct
{
  unsigned int            OBJPID;
  Boolean_t               Success;
  OBJP_ObjectType_t       ObjectType;
  char                   *ObjectFileName;
  Boolean_t               TransferComplete;
  unsigned int            TransferredLength;
  unsigned int            TotalLength;
} OBJP_Client_Object_Put_Confirmation_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.
Success	Specifies whether or not request completed successfully.
ObjectType	Type of object put in INBOX.
ObjectFileName	Name of the object written to INBOX.
TransferComplete	Indicates status of transfer.
TransferredLength	Number bytes transferred.
TotalLength	Total number bytes to be transferred.

etOBJP_Client_Object_Get_Confirmation

Dispatched when an Object Push Client receives an Object from the registered Object Push Server. The OBJP ID member specifies the Remote Server that the Local Client is retrieving the specified Object from. The Object Type member specifies the type of the Object that is being requested from the specified server. The Object File Name specifies the Object File Name that is currently being requested. The Transfer Complete flag specifies whether or not the Object Transfer has been completed (TRUE when completed, FALSE while in progress). The Transferred Length specifies how many bytes have been transferred. Note that there is no way to determine how many total bytes of the Remote Object will be transferred. This is an OBEX limitation. The Success member specifies whether or not the request has been completed successfully.

Return Structure:

```
typedef struct
{
   unsigned int          OBJPID;
   Boolean_t             Success;
   OBJP_ObjectType_t     ObjectType;
   char                 *ObjectFileName;
   Boolean_t             TransferComplete;
   unsigned int          TransferredLength;
} OBJP_Client_Object_Get_Confirmation_Data_t;
```

Event Parameters:

OBJPID	Identifier of the OBJP server connection.
Success	Specifies whether or not request completed successfully.
ObjectType	Type of object requested.
ObjectFileName	Name of the object requested.
TransferComplete	Indicates status of transfer.
TransferredLength	Number bytes transferred.

3. File Distributions

The header files that are distributed with the Bluetooth OBEX Object Push Profile Library are listed in the table below.

File	Contents/Description
OBJPAPI.h	Bluetooth OBEX Object Push Profile API definitions
SS1BTOBP.h	Bluetooth OBEX Object Push Profile Include file