OBEX File Transfer Profile (FTP)

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.

The remainder of this chapter has sections on the scope of this document, other documents applicable to this document, and a listing of acronyms and abbreviations. Chapter 2 is the API reference that contains a description of all programming interfaces for the Bluetooth OBEX File Transfer Profile Stack provided by Bluetopia. And, Chapter 3 contains the header file name list for the Bluetooth OBEX File Transfer 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

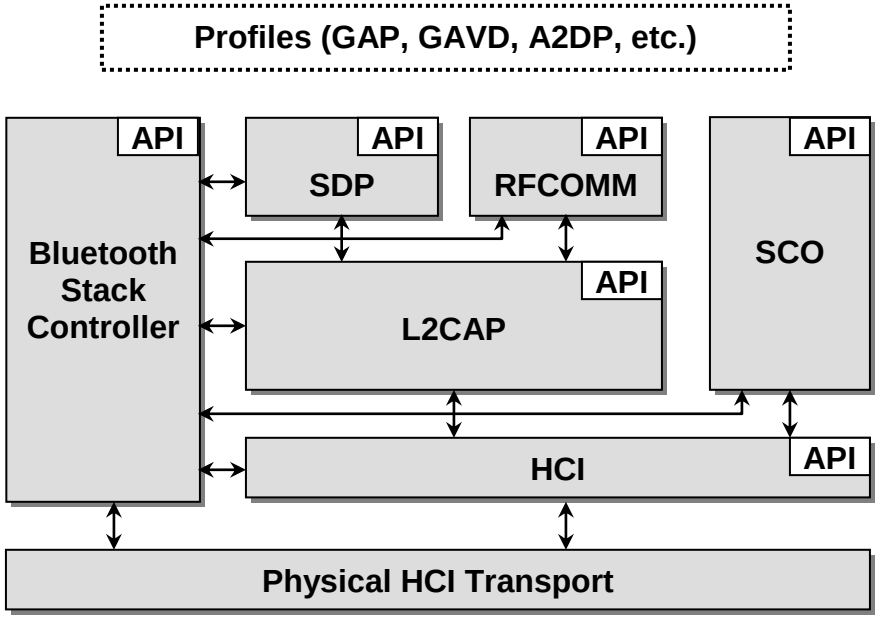

Figure 1-1 The Stonestreet One Bluetooth Protocol Stack

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® 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
API	Application Programming Interface
BD_ADDR	Bluetooth Device Address
BR	Basic Rate
BT	Bluetooth
EDR	Enhanced Data Rate
FTP	File Transfer Protocol

Term	Meaning
HCI	Host Controller Interface
HS	High Speed
L2CAP	Logical Link Control and Adaptation Protocol
LE	Low Energy
RFCOMM	Radio Frequency serial COMMunications – Serial cable emulation protocol based on ETSI TS 07.10
SCO link	Synchronous Connection-Oriented Link – Supports time-bounded information like voice. (Master to single slave)
SDP	Service Discovery Protocol
SPP	Serial Port Protocol

2. File Transfer Profile Programming Interfaces

The File Transfer Profile programming interface defines the protocols and procedures to be used to implement File Transfer capabilities. The File Transfer Profile commands are listed in section 2.1, the event callback prototypes are described in section 2.2, and the File Transfer Profile events are itemized in section 2.3. The actual prototypes and constants outlined in this section can be found in the **FTPAPI.H** header file in the Bluetopia distribution.

2.1 File Transfer Profile Commands

The available File Transfer Profile command functions are listed in the table below and are described in the text that follows.

Function	Description
FTP_Open_Server	Opens an FTP File Server.
FTP_Close_Server	Close an open FTP File Server.
FTP_Close_Server_Connection	Close a FTP connection to a local server.
FTP_Server_Connect_Request_Response	Respond to a connect request from the remote device.
FTP_Register_Server_SDP_Record	Add a generic SDP Service Record to the SDP database
FTP_Open_Remote_File_Server	Open a remote FTP File Server.
FTP_Close_Client	Close an FTP connection to a remote server.
FTP_Get_Directory	Request a listing of the current directory or a sub-directory off of the current directory
FTP_Set_Directory	Change the current working directory on the remote FTP Server.
FTP_Set_Root_Directory	Change the current working directory on the remote FTP Server to the Root Directory.
FTP_Create_Directory	Create a directory on the Remote FTP Server
FTP_Delete_Directory	Delete a directory on the Remote FTP Server
FTP_Create_File	Create a file on the Remote FTP Server.
FTP_Delete_File	Delete a file on the Remote FTP Server.
FTP_Get_File	Retrieves the specified remote file from the Remote FTP Server
FTP_Put_File	Sends the specified local file to the specified Remote FTP Server

FTP_Abort	Aborts any current outstanding FTP Client requests.
FTP_Get_Server_Mode	Query the current FTP Server Mode.
FTP_Set_Server_Mode	Change the current FTP Server Mode.

FTP_Open_Server

This function opens an FTP File Server on a specified Bluetooth SPP Serial Port.

Prototype:

int BTPSAPI **FTP_Open_Server**(unsigned int BluetoothStackID, unsigned int ServerPort, char *RootDirectory, unsigned long PermissionMask, FTP_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
RootDirectory	Pointer to a NULL terminated ASCII string that specifies the local directory path of the root directory of the File Server.
PermissionMask	File Server Permissions to use when responding to client requests.
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 if successful. The return value will be the FTPID for the server port that was successfully opened. *This* is the value that should be used in all subsequent function calls that require this parameter.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_INVALID_ROOT_DIRECTORY
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Server_Connect_Request_Indication
etFTP_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.

FTP_Close_Server

This function closes an FTP File Server that was opened by a successful call to the FTP_Open_Server( ) function. Note, this function does NOT delete any SDP Service Record Handles (i.e., added via an FTP_Register_SDP_Record( ) function call).

Prototype:

int BTPSAPI **FTP_Close_Server**(unsigned int BluetoothStackID,
unsigned int FTPID)

Parameters:

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

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

FTP_Close_Server_Connection

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

Prototype:

int BTPSAPI **FTP_Close_Server_Connection**(unsigned int BluetoothStackID,
unsigned int FTPID)

Parameters:

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

FTPID	The FTP server port to close the connection on. This is the value that was returned from the FTP_Open_Server ( ) function.

Return:

Zero if successful.

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

BTFTP_ERROR_NOT_INITIALIZED
BTFTP_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.

FTP_Server_Connect_Request_Response

This function is responsible for responding to requests to connect to a FTP 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 **FTP_Server_Connect_Request_Response**(unsigned int BluetoothStackID, unsigned int FTPID, Boolean_t AcceptConnection)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP ID this command applies to. This is the value that was returned from the FTP_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:

BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_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.

FTP_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 FTPID that was returned from the FTP_Open_Server( ) function. This function should **never** be used with the FTPID returned from the FTP_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 Database. This Macro maps FTP_Un_Register_SDP_Record**( )** to SDP_Delete_Service_Record(), and is defined as follows:

FTP_Un_Register_SDP_Record(__BluetoothStackID, __FTPID, __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 **FTP_Register_Server_SDP_Record**(unsigned int BluetoothStackID, unsigned int FTPID, char *ServiceName, DWord_t *SDPServiceRecordHandle)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTPID this command applies to. This is the value that was returned from the FTP_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:

BTFTP_ERROR_NOT_INITIALIZED

BTFTP_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.

FTP_Open_Remote_File_Server

This function opens a remote FTP File Server. The caller of this function becomes the FTP Client.

Prototype:

int BTPSAPI **FTP_Open_Remote_File_Server**(unsigned int BluetoothStackID, BD_ADDR_t BD_ADDR, unsigned int ServerPort, FTP_Client_Event_Callback_t Event_Callback, unsigned long CallbackParameter)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
BD_ADDR	Address of the Bluetooth device to connect with.
ServerPort	The remote device's FTP 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 if successful. The return value will be the Client FTP ID for the port that was successfully opened. *This* is the value that should be used in all subsequent function calls.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:
etFTP_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.

FTP_Close_Client

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

Prototype:

int BTPSAPI **FTP_Close_Client**(unsigned int BluetoothStackID, unsigned int FTPID)

Parameters:

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

Return:

Zero if successful.

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

BTFTP_ERROR_NOT_INITIALIZED
BTFTP_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.

FTP_Get_Directory

This function allows an FTP Client to request a listing of the current directory or a sub-directory off of the current directory.

Notes:

1. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.

2. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Get_Directory**(unsigned int BluetoothStackID, unsigned int FTPID, char *RemoteDirectoryName);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.
RemoteDirectoryName	NULL if the current directory is to be retrieved or a pointer to a NULL terminated ASIC string of the subdirectory to be retrieved.

Return:

Zero if successful.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_Directory_Request_Confirmation
etFTP_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.

FTP_Set_Directory

This function allows an FTP Client to change the current working directory on the remote FTP Server.

Notes:

1. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.

2. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Set_Directory**(unsigned int BluetoothStackID, unsigned int FTPID, char *RemoteDirectoryName);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.
RemoteDirectoryName	NULL to instruct the Remote Server to backup into the parent directory or a pointer to a NULL terminated ASIC string to indicate the directory to change to. This cannot indicate a path to change to, only the sub-directory of the current directory to change to.

Return:

Zero if successful.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_Change_Directory_Confirmation
etFTP_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.

FTP_Set_Root_Directory

This function an FTP Client to change the current working directory on the remote FTP Server to the Root Directory.

Notes:

1. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.
2. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Set_Root_Directory**(unsigned int BluetoothStackID,
unsigned int FTPID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.

Return:

Zero if successful.

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

BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_Change_Directory_Confirmation
etFTP_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.

FTP_Create_Directory

This function allows an FTP Client to create a directory on the Remote FTP Server.

Notes:

1. The Bluetooth File Transfer Profile (using OBEX) specifies that when a remote directory is successfully created that the new working directory is this newly created directory.

2. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.
3. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Create_Directory**(unsigned int BluetoothStackID, unsigned int FTPID, char *RemoteDirectoryName);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.
RemoteDirectoryName	Pointer to a NULL terminated ASCII string that specifies the directory name of the remote directory to create. This directory name must be specified and cannot contain path information. This directory is created in the current working directory on the remote file server.

Return:

Zero if successful.

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

BTPS_ERROR_INVALID_PARAMETER
BTPS_ERROR_INVALID_BLUETOOTH_STACK_ID
BTPS_ERROR_RFCOMM_NOT_INITIALIZED
BTPS_ERROR_GOEP_NOT_INITIALIZED

Possible Events:

etFTP_Client_Directory_Create_Confirmation
etFTP_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.

FTP_Delete_Directory

This function allows an FTP Client to delete a directory on the Remote FTP Server.

Notes:

1. Due to an OBEX FTP limitation, when a delete operation is specified, the client cannot be guaranteed whether the object deleted is a file or a directory. In other words, if this function is called, and a remote file exists in the current remote working directory, the file will be deleted, even though it is not a directory. This is due to OBEX FTP, which does not allow the specification of what type of object to delete, only the name of the object.
2. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.
3. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Delete_Directory**(unsigned int BluetoothStackID, unsigned int FTPID, char *RemoteDirectoryName);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function. */
RemoteDirectoryName	Pointer to a NULL terminated ASCII string that specifies the directory name of the remote directory to delete. This directory name must be specified and cannot contain path information. This directory is deleted in the current working directory on the remote file server.

Return:

Zero if successful.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_Directory_Delete_Confirmation
etFTP_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.

FTP_Create_File

This function allows an FTP Client to create a file on the Remote FTP Server.

Notes:

1. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.
2. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Create_File**(unsigned int BluetoothStackID, unsigned int FTPID, char *RemoteFileName);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.
RemoteFileName	Pointer to a NULL terminated ASCII string that specifies the file name of the remote file to create. This file name must be specified and cannot contain any path information. This file is created in the current working directory on the remote file server.

Return:

Zero if successful.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_File_Create_Confirmation
etFTP_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.

FTP_Delete_File

This function allows an FTP Client to delete a file on the Remote FTP Server.

Notes:

1. Due to an OBEX FTP limitation, when a delete operation is specified, the client cannot be guaranteed whether the object deleted is a file or a directory. In other words, if this function is called, and a remote directory exists in the current remote working directory, the directory will be deleted, even though it is not a file. This is due to OBEX FTP, which does not allow the specification of what type of object to delete, only the name of the object.
2. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.
3. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Delete_File**(unsigned int BluetoothStackID, unsigned int FTPID, char *RemoteFileName);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.
RemoteFileName	Pointer to a NULL terminated ASCII string that specifies the file name of the remote file to delete. This file name must be specified and cannot contain any path information. This file is deleted in the current working directory on the remote file server.

Return:

Zero if successful.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_File_Delete_Confirmation
etFTP_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.

FTP_Get_File

This function retrieves the specified remote file from the Remote FTP Server.

Notes:

1. The remote file length parameter exists simply to allow a total length value to be passed to the caller on file get confirmations. This value can be any value the caller would like it to be because the File Transfer Profile ignores this value and simply passes this value back to the caller in the File Get Confirmation Event.
2. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.
3. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Get_File**(unsigned int BluetoothStackID, unsigned int FTPID, char *RemoteFileName, unsigned int RemoteFileLength, char *LocalPath, char *LocalFileName);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.
RemoteFileName	Pointer to a NULL terminated ASCII string that specifies the file name of the remote file to retrieve. This file name must be specified and cannot contain any path information. This file is

	retrieved from the current working directory on the remote file server.
RemoteFileLength	Length of the file (in bytes) of the Remote File that is to be retrieved. This parameter only exists so that the total length of the file can be passed to the caller in the **etFTP_Client_File_Get_Confirmation** events. This allows the caller easy access to the total length of the file that is being retrieved from the Remote File Server. This information is passed as is in the Event Callback and its value is ignored by the FTP Module (other than passing it back to the caller).
LocalPath	Pointer to a NULL terminated ASCII string that specifies the Local Path that the Remote File is to be stored on the Local Machine. This parameter can be NULL which means that the **LocalFileName** parameter (last parameter) specifies the Local File Name AND Local Path Information (Local Path Information may or may not be present in which case the file will be stored in the current working directory).
LocalFileName	Pointer to a NULL terminated ASCII string that specifies the Local File Name that the Remote File is to stored on the Local Machine. This parameter cannot contain any path information if LocalPath is non-NULL. If LocalPath is NULL, then this parameter can specify both the Local Path and File Name (Local Path Information may or may not be present in which case the file will be stored in the current working directory).

Return:

Zero if successful.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_File_Get_Confirmation
etFTP_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.

FTP_Put_File

This function sends the specified local file to the specified Remote FTP Server.

Notes:

1. Issuing this command successfully does not mean that the Remote FTP Server successfully processed the command. The caller needs to check the confirmation result to determine if the Remote FTP Server successfully executed the request.
2. Due to an OBEX FTP limitation, there can only be one outstanding FTP client request active at once. Because of this, another FTP client request cannot be issued until either the current request is aborted using the FPT_Abort() function or the current request is complete through the reception of a Confirmation Event in the FTP client event callback function.

Prototype:

int BTPSAPI **FTP_Put_File**(unsigned int BluetoothStackID, unsigned int FTPID, char *LocalPath, char *LocalFileName, char *RemoteFileName);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.
LocalPath	Pointer to a NULL terminated ASCII string that specifies the local path name of file to. This path name can be NULL to indicate the current working directory.
LocalFileName	Pointer to a NULL terminated ASCII string that specifies the file name of the local file to put. This file name must be specified and cannot contain any path information.
RemoteFileName	Pointer to a NULL terminated ASCII string that specifies the file name to use for the file on the Remote FTP Server. This file name must be specified and cannot contain any path information. This file is created in the current working directory on the Remote FTP Server.

Return:

Zero if successful.

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

BTFTP_ERROR_INSUFFICIENT_RESOURCES
BTFTP_ERROR_UNABLE_TO_CREATE_LOCAL_FILE
BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_File_Put_Confirmation
etFTP_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.

FTP_Abort

This function aborts any current outstanding FTP Client requests.

Notes:

1. This FTP Client request is no different than the other FTP Client request functions in that a response to this request must be received before any other FTP Client Request can be issued.
2. Because of transmission latencies, it may be possible that an FTP 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 FTP 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 FTP 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 **FTP_Abort**(unsigned int BluetoothStackID, unsigned int FTPID);

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP Client ID returned from a successful call to the FTP_Open_Remote_File_Server() function.

Return:

Zero if successful.

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

BTFTP_ERROR_REQUEST_ALREADY_OUTSTANDING
BTFTP_ERROR_NOT_INITIALIZED
BTFTP_ERROR_NOT_ALLOWED_WHILE_NOT_CONNECTED
BTFTP_ERROR_INVALID_PARAMETER

Possible Events:

etFTP_Client_File_Abort_Confirmation
etFTP_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.

FTP_Get_Server_Mode

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

Prototype:

int BTPSAPI **FTP_Get_Server_Mode**(unsigned int BluetoothStackID, unsigned int FTPID, unsigned long *ServerModeMask)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP ID this command applies to. This is the value that was returned from the FTP_Open_Server( ) function.
ServerModeMask	Pointer to a variable to receive the current Server Mode Mask.

Return:

Zero if successful.

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

BTFTP_ERROR_NOT_INITIALIZED
BTFTP_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.

FTP_Set_Server_Mode

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

Prototype:

int BTPSAPI **FTP_Set_Server_Mode**(unsigned int BluetoothStackID, unsigned int FTPID, unsigned long ServerModeMask)

Parameters:

BluetoothStackID[1]	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
FTPID	The FTP ID this command applies to. This is the value that was returned from the FTP_Open_Server( ) function.
ServerModeMask	The new Server Mode being set.

Return:

Zero if successful.

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

BTFTP_ERROR_NOT_INITIALIZED
BTFTP_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 File Transfer Profile Event Callback Prototypes

2.2.1 Server Event Callback

The event callback function mentioned in the File_Open_Server command accepts the callback function described by the following prototype.

FTP_Server_Event_Callback_t

Prototype of callback function passed in the File_Open_Server command.

Prototype:

void (BTPSAPI ***FTP_Server_Event_Callback_t**)(unsigned int BluetoothStackID, FTP_Server_Event_Data_t *FTP_Server_Event_Data, unsigned long CallbackParameter)

Parameters:

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

```
typedef struct
{
  FTP_Server_Event_Type_t     Event_Data_Type;
  Word_t                      Event_Data_Size;
  union
  {
    FTP_Server_Connect_Indication_Data_t         *FTP_Server_Connect_Indication_Data;
    FTP_Server_Disconnect_Indication_Data_t
            *FTP_Server_Disconnect_Indication_Data;
    FTP_Server_Directory_Request_Indication_Data_t
            *FTP_Server_Directory_Request_Indication_Data;
    FTP_Server_Change_Directory_Indication_Data_t
            *FTP_Server_Change_Directory_Indication_Data;
    FTP_Server_Directory_Delete_Indication_Data_t
            *FTP_Server_Directory_Delete_Indication_Data;
    FTP_Server_Directory_Create_Indication_Data_t
            *FTP_Server_Directory_Create_Indication_Data;
    FTP_Server_File_Delete_Indication_Data_t
            *FTP_Server_File_Delete_Indication_Data;
    FTP_Server_File_Create_Indication_Data_t
            *FTP_Server_File_Create_Indication_Data;
    FTP_Server_File_Put_Indication_Data_t        *FTP_Server_File_Put_Indication_Data;
    FTP_Server_File_Get_Indication_Data_t        *FTP_Server_File_Get_Indication_Data;
    FTP_Server_Connect_Request_Indication_Data_t
            *FTP_Server_Connect_Request_Indication_Data;
  } Event_Data;
} FTP_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.

2.2.2 Client Event Callback

The event callback function mentioned in the FTP_Open_Remote_File_Server command accepts the callback function described by the following prototype.

FTP_Client_Event_Callback_t

Prototype of callback function passed in the File_Open_Remote_File_Server command.

Prototype:

void (BTPSAPI ***FTP_Client_Event_Callback_t**)(unsigned int BluetoothStackID, FTP_Client_Event_Data_t *FTP_Client_Event_Data, unsigned long CallbackParameter)

Parameters:

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

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

```
typedef struct
{
  FTP_Client_Event_Type_t    Event_Data_Type;
  Word_t                     Event_Data_Size;
  union
  {
    FTP_Client_Connect_Confirmation_Data_t
          *FTP_Client_Connect_Confirmation_Data;
    FTP_Client_Disconnect_Indication_Data_t
          *FTP_Client_Disconnect_Indication_Data;
    FTP_Client_Abort_Confirmation_Data_t
          *FTP_Client_Abort_Confirmation_Data;
    FTP_Client_Directory_Request_Confirmation_Data_t
          *FTP_Client_Directory_Request_Confirmation_Data;
    FTP_Client_Change_Directory_Confirmation_Data_t
          *FTP_Client_Change_Directory_Confirmation_Data;
    FTP_Client_Directory_Delete_Confirmation_Data_t
          *FTP_Client_Directory_Delete_Confirmation_Data;
    FTP_Client_Directory_Create_Confirmation_Data_t
          *FTP_Client_Directory_Create_Confirmation_Data;
    FTP_Client_File_Delete_Confirmation_Data_t
          *FTP_Client_File_Delete_Confirmation_Data;
    FTP_Client_File_Create_Confirmation_Data_t
          *FTP_Client_File_Create_Confirmation_Data;
    FTP_Client_File_Put_Confirmation_Data_t
          *FTP_Client_File_Put_Confirmation_Data;
    FTP_Client_File_Get_Confirmation_Data_t
          *FTP_Client_File_Get_Confirmation_Data;
  } Event_Data;
} FTP_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 File Transfer Profile Events

The File Transfer Profile contains events that are received by the Server and events that are received by the Client. The following sections detail those events.

2.3.1 File Transfer Profile Server Events

The possible File Transfer Profile Server Events from the Bluetooth stack are listed in the table below and are described in the text which follows:

Event	Description
etFTP_Server_Connect_Indication	Dispatched when a FTP Client connects to a registered FTP Server.
etFTP_Server_Connect_Request_Indication	Dispatched when a FTP Client requests to connect to a registered FTP Server.
etFTP_Server_Disconnect_Indication	Dispatched when a FTP Client disconnects from a registered FTP Server.
etFTP_Server_Directory_Request_Indication	Dispatched when a FTP Client requests a directory from the registered FTP Server.
etFTP_Server_Change_Directory_Indication	Dispatched with a FTP Client changes the current directory on the registered FTP Server.
etFTP_Server_Directory_Delete_Indication	Dispatched when a FTP Client deletes the specified directory from the registered FTP Server.
etFTP_Server_Directory_Create_Indication	Dispatched when a FTP Client creates the specified directory on the registered FTP Server.
etFTP_Server_File_Delete_Indication	Dispatched when a FTP Client deletes the specified file from the registered FTP Server.
etFTP_Server_File_Create_Indication	Dispatched when a FTP Client creates a file on the registered FTP Server.
etFTP_Server_File_Put_Indication	Dispatched when a FTP Client puts a file on the registered FTP Server.
etFTP_Server_File_Get_Indication	Dispatched when a FTP Client retrieves a file from the registered FTP Server.

etFTP_Server_Connect_Indication

Dispatched when a FTP Client connects to a registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int      FTPID;
  BD_ADDR_t         BD_ADDR;
} FTP_Server_Connect_Indication_Data_t;
```

Event Parameters:

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

etFTP_Server_Disconnect_Indication

Dispatched when a FTP Client disconnects from a registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int    FTPID;
} FTP_Server_Disconnect_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server from which the Remote Client has disconnected.

etFTP_Server_Directory_Request_Indication

Dispatched when a FTP Client requests a directory from the registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int   FTPID;
  char           *DirectoryName;
} FTP_Server_Directory_Request_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server that the Remote Client has requested.
DirectoryName	Specifies the directory name requested.

etFTP_Server_Change_Directory_Indication

Dispatched with a FTP Client changes the current directory on the registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int   FTPID;
  char           *DirectoryName;
} FTP_Server_Change_Directory_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server of which the Remote Client has changed the directory.
DirectoryName	Specifies the directory path that the Remote Client has changed to.

etFTP_Server_Directory_Delete_Indication

Dispatched when a FTP Client deletes the specified directory from the registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int    FTPID;
  char            *DeletedDirectoryName;
  char            *DirectoryName;
} FTP_Server_Directory_Delete_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server from which the Remote Client has deleted the directory.
DeletedDirectoryName	Specifies the name of the directory that was deleted.
DirectoryName	Specifies the name of the directory from which the specified directory was deleted.

etFTP_Server_Directory_Create_Indication

Dispatched when a FTP Client creates the specified directory on the registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int    FTPID;
  char           *CreatedDirectoryName;
  char           *DirectoryName;
} FTP_Server_Directory_Create_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server on which the Remote Client created the directory.
CreatedDirectoryName	Specifies the name of the directory that was created.
DirectoryName	Specifies the name of the directory in which the specified directory was created.

etFTP_Server_File_Delete_Indication

Dispatched when a FTP Client deletes the specified file from the registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int    FTPID;
  char           *DeletedFileName;
  char           *DirectoryName;
} FTP_Server_File_Delete_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server from which the Remote Client has deleted a file.
DeletedFileName	Specifies the name of the file that was deleted.
DirectoryName	Specifies the name of the directory from which the specified file was deleted.

etFTP_Server_File_Create_Indication

Dispatched when a FTP Client creates a file on the registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int    FTPID;
  char           *CreatedFileName;
  char           *DirectoryName;
} FTP_Server_File_Create_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server on which the Remote Client has created the file.
CreatedFileName	Specifies the name of the file that was created.
DirectoryName	Specifies the name of the directory in which the specified file was created.

etFTP_Server_File_Put_Indication

Dispatched when a FTP Client puts a file on the registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int    FTPID;
  char           *FileName;
  char           *DirectoryName;
  Boolean_t       TransferComplete;
  unsigned int    TransferredLength;
  unsigned int    TotalLength;
} FTP_Server_File_Put_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server to which the Remote Client is putting the file.
FileName	Specifies the name of the file that is being put on the FTP Server.
DirectoryName	Specifies the name of the directory in which the specified file is being put.
TransferComplete	Specifies whether or not the file transfer has been completed. TRUE when completed; FALSE when in progress.
TransferredLength	Specifies how many bytes have been transferred so far.
TotalLength	Specifies how many bytes will be transferred in total.

etFTP_Server_File_Get_Indication

Dispatched when a FTP Client retrieves a file from the registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  char            *FileName;
  char            *DirectoryName;
  Boolean_t        TransferComplete;
  unsigned int     TransferredLength;
  unsigned int     TotalLength;
} FTP_Server_File_Get_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Server from which the Remote Client is retrieving the file.
FileName	Specifies the name of the file that is being retrieved on the FTP Server.
DirectoryName	Specifies the name of the directory from which the specified file is being retrieved.
TransferComplete	Specifies whether or not the file transfer has been completed. TRUE when completed; FALSE when in progress.
TransferredLength	Specifies how many bytes have been transferred so far.
TotalLength	Specifies how many bytes will be transferred in total.

etFTP_Server_Connect_Request_Indication

Dispatched when a FTP Client requests to connect to a registered FTP 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       FTPID;
  BD_ADDR_t          BD_ADDR;
} FTP_Server_Connect_Request_Indication_Data_t;
```

Event Parameters:

FTPID	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.

2.3.2 File Transfer Profile Client Events

The possible File Transfer Profile Client Events from the Bluetooth stack are listed in the table below and are described in the text which follows:

Event	Description
etFTP_Client_Connect_Confirmation	Dispatched when a FTP Client receives the Connection Response from an FTP Server that was previously attempted to be connected to.
etFTP_Client_Disconnect_Indication	Dispatched when a FTP Client disconnects from a registered FTP Server.
etFTP_Client_Abort_Confirmation	Dispatched when a FTP Client Abort Response is received from the Remote FTP Server.
etFTP_Client_Directory_Request_Confirmation	Dispatched when a FTP Client receives a response from the FTP Server regarding a previous FTP Directory Request.
etFTP_Client_Change_Directory_Confirmation	Dispatched when a FTP Client receives a response from the FTP Server regarding a previous FTP Change Directory Request.
etFTP_Client_Directory_Delete_Confirmation	Dispatched when a FTP Client deletes the specified directory from a remote FTP Server and a response from the server is received.
etFTP_Client_Directory_Create_Confirmation	Dispatched when a FTP Client creates the specified directory on the remote FTP Server and a response from the server is received.
etFTP_Client_File_Delete_Confirmation	Dispatched when a FTP Client deletes the specified file from a remote FTP Server and a response from the server is received.
etFTP_Client_File_Create_Confirmation	Dispatched when a FTP Client creates the specified file on the remote FTP Server and a response from the server is received.
etFTP_Client_File_Put_Confirmation	Dispatched when a FTP Client puts a file on the remote FTP Server and the Remote Server has responded to the request.
etFTP_Client_File_Get_Confirmation	Dispatched when a FTP Client retrieves a file from a FTP Server and a response is received from the FTP Server.

etFTP_Client_Connect_Confirmation

Dispatched when a FTP Client receives the Connection Response from an FTP Server that was previously attempted to be connected to.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  unsigned int     FTPConnectStatus;
  BD_ADDR_t   BD_ADDR;
} FTP_Client_Connect_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client that has requested the connection.
FTPConnectStatus	Identifies the connection status of the request. One of the following: FTP_CLIENT_OPEN_STATUS_SUCCESS FTP_CLIENT_OPEN_STATUS_CONNECTION_TIMEOUT FTP_CLIENT_OPEN_STATUS_CONNECTION_REFUSED FTP_CLIENT_OPEN_STATUS_UNKNOWN_ERROR
BD_ADDR	Specifies the Remote Bluetooth Device on which the Remote FTP Server resides.

etFTP_Client_Disconnect_Indication

Dispatched when a FTP Client disconnects from a registered FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int FTPID;
} FTP_Client_Disconnect_Indication_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client that the Remote Server has disconnected from.

etFTP_Client_Abort_Confirmation

Dispatched when a FTP Client Abort Response is received from the Remote FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int FTPID;
} FTP_Client_Abort_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client that the Remote Server has responded to the Abort Request on.

etFTP_Client_Directory_Request_Confirmation

Dispatched when a FTP Client receives a response from the FTP Server regarding a previous FTP Directory Request.

Return Structure:

```
typedef struct
{
  unsigned int              FTPID;
  Boolean_t                 RequestCompleted;
  Boolean_t                 ParentDirectory;
  char                     *DirectoryName;
  unsigned int              NumberDirectoryEntries;
  FTP_Directory_Entry_t    *FTPDirectoryEntries;
} FTP_Client_Directory_Request_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client that has connected to the Remote FTP Server.
RequestCompleted	Specifies whether or not the original directory request has been completed.
ParentDirectory	Specifies whether or not a parent directory exists above the specified current directory.
DirectoryName	Specifies the directory that was requested. NULL if the current directory is requested.
NumberDirectoryEntries	Specifies how many FTP directory entries are pointed to by the FTPDirectoryEntries member.
FTPDirectoryEntries	Structure defined by the following:

```
typedef struct
{
   Boolean_t            DirectoryEntry;
   unsigned int         FieldMask;
   unsigned int         NameLength;
   char                *Name;
   unsigned int         Size;
   unsigned int         TypeLength;
   char                *Type;
   FTP_TimeDate_t       Modified;
   FTP_TimeDate_t       Created;
   FTP_TimeDate_t       Accessed;
   Word_t               Permission;
   unsigned int         OwnerLength;
   char                *Owner;
   unsigned int         GroupLength;
   char                *Group;
} FTP_Directory_Entry_t;
```

where, FieldMask is defined as:

FTP_OBJECT_INFO_MASK_CLEAR
FTP_OBJECT_INFO_MASK_NAME
FTP_OBJECT_INFO_MASK_SIZE
FTP_OBJECT_INFO_MASK_TYPE
FTP_OBJECT_INFO_MASK_MODIFIED
FTP_OBJECT_INFO_MASK_CREATED
FTP_OBJECT_INFO_MASK_ACCESSED
FTP_OBJECT_INFO_MASK_USER_PERMISSION
FTP_OBJECT_INFO_MASK_GROUP_PERMISSION
FTP_OBJECT_INFO_MASK_OTHER_PERMISSION
FTP_OBJECT_INFO_MASK_OWNER
FTP_OBJECT_INFO_MASK_GROUP

Permission is defined as:

FTP_USER_PERMISSION_READ
FTP_USER_PERMISSION_WRITE
FTP_USER_PERMISSION_DELETE
FTP_GROUP_PERMISSION_READ
FTP_GROUP_PERMISSION_WRITE
FTP_GROUP_PERMISSION_DELETE
FTP_OTHER_PERMISSION_READ
FTP_OTHER_PERMISSION_WRITE
FTP_OTHER_PERMISSION_DELETE

and, FTP_TimeDate_t is defined as:

```
typedef struct
{
  Word_t      Year;
  Word_t      Month;
  Word_t      Day;
  Word_t      Hour;
  Word_t      Minute;
  Word_t      Second;
  Boolean_t   UTC_Time;
} FTP_TimeDate_t;
```

etFTP_Client_Change_Directory_Confirmation

Dispatched when a FTP Client receives a response from the FTP Server regarding a previous FTP Change Directory Request.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  Boolean_t        Success;
  char            *ChangedDirectoryName;
} FTP_Client_Change_Directory_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client that has connected to the Remote FTP Server.
Success	Specifies whether or not the request to change the directory was successful or not.
ChangedDirectoryName	Specifies the directory that was changed to.

etFTP_Client_Directory_Delete_Confirmation

Dispatched when a FTP Client deletes the specified directory from a remote FTP Server and a response from the server is received.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  Boolean_t        Success;
  Boolean_t        DirectoryNotEmpty;
  char            *DeletedDirectoryName;
} FTP_Client_Directory_Delete_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client that the Remote Server has deleted the directory from.
Success	Specifies whether or not the request to delete the specified directory was successful.
DirectoryNotEmpty	Specifies if an unsuccessful attempt to delete a directory was caused by the directory not being empty.
DeletedDirectoryName	Specifies the name of the directory that was deleted.

etFTP_Client_Directory_Create_Confirmation

Dispatched when a FTP Client creates the specified directory on the remote FTP Server and a response from the server is received.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  Boolean_t        Success;
  char            *CreatedDirectoryName;
} FTP_Client_Directory_Create_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client that the Remote Server has created the directory on.
Success	Specifies whether or not the directory was successfully created.
CreatedDirectoryName	Specifies the name of the directory that was created.

etFTP_Client_File_Delete_Confirmation

Dispatched when a FTP Client deletes the specified file from a remote FTP Server and a response from the server is received.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  Boolean_t        Success;
  char            *DeletedFileName;
} FTP_Client_File_Delete_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client from which the Remote Server has deleted a file.
Success	Specifies whether or not the file was deleted successfully.
DeletedFileName	Specifies the name of the file that was deleted.

etFTP_Client_File_Create_Confirmation

Dispatched when a FTP Client creates the specified file on the remote FTP Server and a response from the server is received.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  Boolean_t        Success;
  char            *CreatedFileName;
} FTP_Client_File_Create_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client on which the Remote Server has created the file.
Success	Specifies whether or not the file was created successfully.
CreatedFileName	Specifies the name of the file that was created.

etFTP_Client_File_Put_Confirmation

Dispatched when a FTP Client puts a file on the remote FTP Server and the Remote Server has responded to the request.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  Boolean_t        Success;
  char            *FileName;
  Boolean_t        TransferComplete;
  unsigned int     TransferredLength;
  unsigned int     TotalLength;
} FTP_Client_File_Put_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client to which the Remote Server is putting the file.
Success	Specifies whether or not the file was put successfully.
FileName	Specifies the name of the file that is being put on the FTP Server.
TransferComplete	Specifies whether or not the file transfer has been completed. TRUE when completed; FALSE when in progress.
TransferredLength	Specifies how many bytes have been transferred so far.
TotalLength	Specifies how many bytes will be transferred in total.

etFTP_Client_File_Get_Confirmation

Dispatched when a FTP Client retrieves a file from a FTP Server and a response is received from the FTP Server.

Return Structure:

```
typedef struct
{
  unsigned int     FTPID;
  Booleant_t       Success;
  char            *FileName;
  Booleant_t       TransferComplete;
  unsigned int     TransferredLength;
  unsigned int     TotalLength;
} FTP_Client_File_Get_Confirmation_Data_t;
```

Event Parameters:

FTPID	Identifies the Local Client from which the Remote Server is retrieving the file.
Success	Specifies whether or not the file was retrieved successfully.
FileName	Specifies the name of the file that is being retrieved on the FTP Server.
TransferComplete	Specifies whether or not the file transfer has been completed. TRUE when completed; FALSE when in progress.
TransferredLength	Specifies how many bytes have been transferred so far.
TotalLength	Specifies how many bytes will be transferred in total.

3. File Distributions

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

File	Contents/Description
FTPAPI.h	Bluetooth OBEX File Transfer Profile API definitions
SS1BTFTP.h	Bluetooth OBEX File Transfer Profile Include file