IVT BTSDK

API Reference

(Part II)

This document describes the definitions, structures, and APIs of IVT BTSDK used by BlueSoleil™ on Windows, WinCE and Linux platforms.

Revision History

Version	Date	Comments
2008.10.18	Oct. 10th, 2008	Initial version.
2009.06.03	Jun. 3th, 2009	Add PIM APIs
2009.06.23	Jun. 23th, 2009	Add PIM APIs

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

IVT BTSDK API is the interface exported by IVT BTSDK (Bluetooth Software Development Kit). It is used to access the Bluetooth profiles from the application level software. It allows for:

- Standardized access to Bluetooth links.
- Supports applications that implement different Bluetooth profiles.
- Write portable applications to be used on different hardware and operating system platforms.
- Future expansions or hardware changes will not affect applications that use this interface.

To use the BTSDK API only a limited knowledge of Bluetooth basic principles and profile specifications is necessary. Therefore this document is not intended to be a Bluetooth profile tutorial.

This interface is divided into two categories, General and Profile Specific.

The General part interface provides basic Bluetooth functions defined in General Access Profile and Service Discovery Application Profile as well as:

- Local service registry.
- Remote device management.
- Security Management.
- Connection Management.

The Profile Specific interface provides functions defined in different Bluetooth profiles except for General Access Profile and Service Discovery Application Profile.

This document describes the Profile Specific interface of IVT BTSDK API. General part interface is discussed in a separate document.

1.2 Overview of IVT BTSDK

The intention of BTSDK is to relieve the Application from managing the Bluetooth related components and make the Application light load.

The general structure of IVT BTSDK is shown in Figure 1. BTSDK is between the Application and profile/stack. It wraps the various APIs of Bluetooth profiles and protocol stack and provides the Application with clean APIs. The key component is a core manager and a profile manager with the following tasks:

- Store Bluetooth device information, including security-related information on devices.
- Store Bluetooth service information, including security-related information on devices.
- Store active connection information.

- Provide access to different Bluetooth profiles.

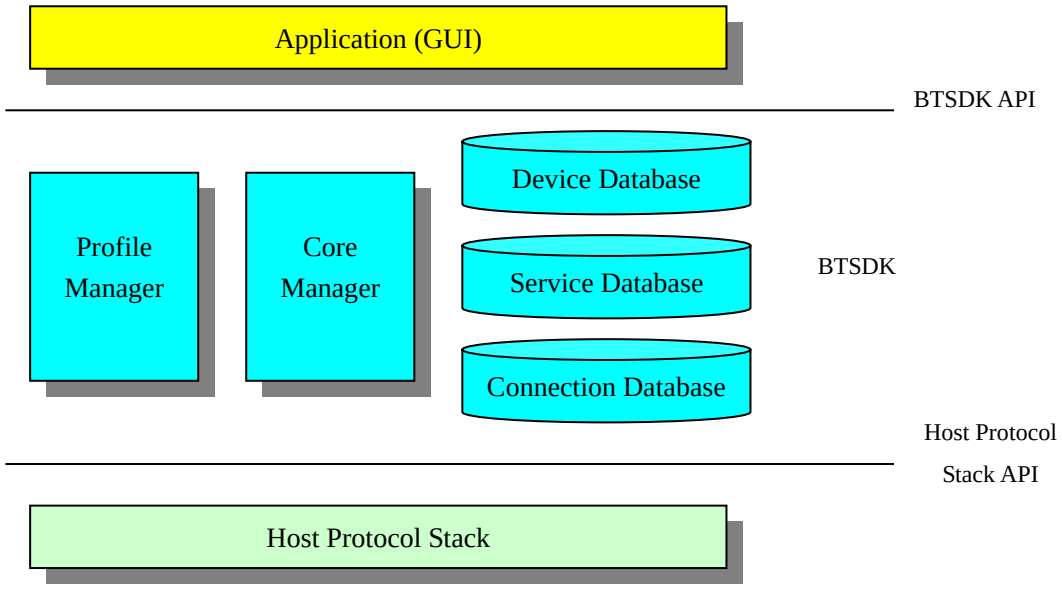

Figure 1: IVT BTSDK Structure

BTSDK maintains a list of remote devices, local services, remote services and active connections. Application can access these objects through a unique handle. BTSDK can automatically store and recover information of these objects and security settings.

BTSDK provides an abstraction of Bluetooth profiles that is independent of the underlying host stack used to provide Bluetooth services. Future expansions or hardware changes will not affect applications use BTSDK API.

1.3 Overview of PIM

Now, we support PIM including PBAP, AT PBAP and AT SMS. ALL the APIs begin with PIM_.
1）For the PBAP is better than AT PBAP, we will try to connect PBAP first. The circuit shows bellow:

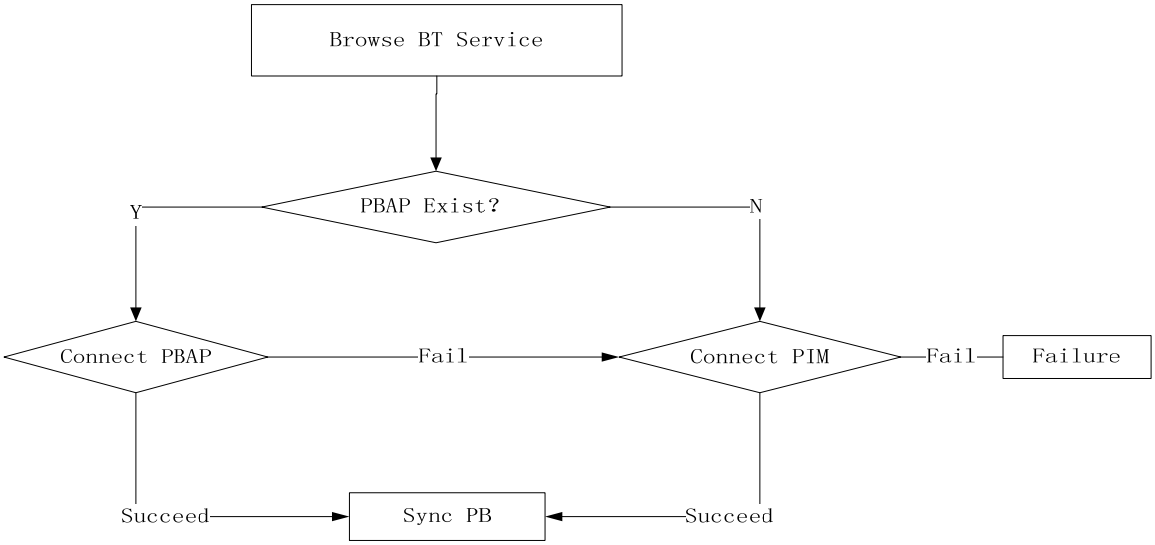

Figure 2: The circuit of the PIM PBAP

2）For the SMS, we just support AT SMS, and will add MAP later.

3）The Circuit of PIM:

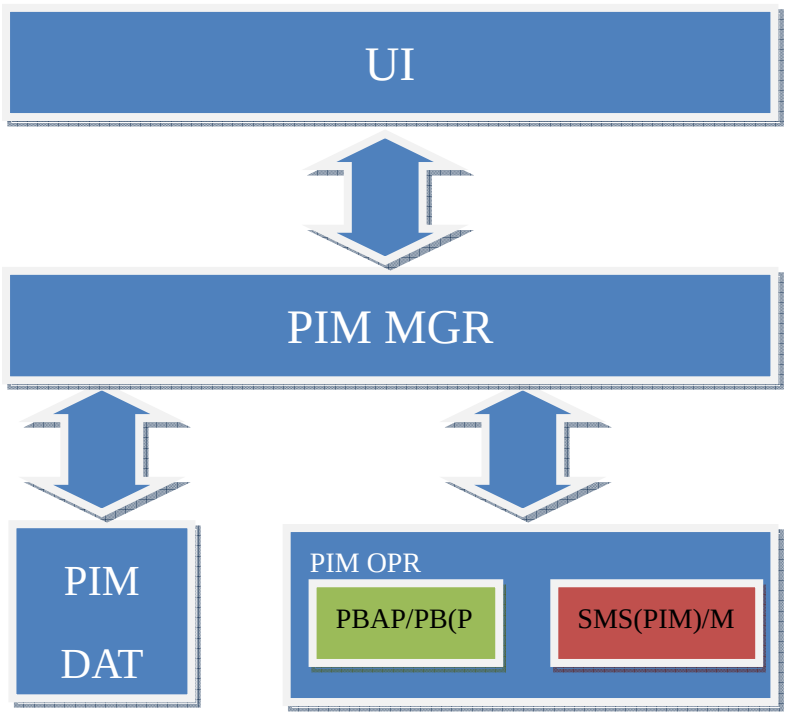

Figure 3: The PIM Model

1.4 Definitions, Acronyms and Abbreviations

AG	Audio Gateway
API	Application Programming Interface
AVRCP	Audio/Video Remote Control Profile
BIP	Basic Imaging Profile
CTP	Cordless Telephony Profile
DUN	Dial-up Networking Profile
EC	Echo canceling
FTP	File Transfer Profile
HEP	Headset Profile
HF	Hands-Free Unit
HFP	Hands-Free Profile
HID	Human Interface Device
HS	Headset
ICP	Intercom Profile
LAP	LAN Access Profile
NR	Noise Reduction
OPP	Object Push Profile
PAN	Personal Area Networking Profile
REF	BIP Referenced Objects
SA	Service Attribute
SDAP	Service Discovery Application Profile
SDM	Service Database Management
SDP	Service Discovery Protocol
SNK	Audio Sink
SPP	Serial Port Profile
SRC	Audio Source
SS	Service Search
SSA	Service Search Attribute
UUID	Universally Unique Identifier
PIM	Personal Information Manager

2. Develop Notes

When use the APIs to develop the application software, please pay attention to the notes bellow:

- Please don't use the BTSDK APIs in callback event function.
- Don't. do anything `Time-consuming in` callback event function.

3. Constant Reference

3.1 Error Codes

The following table provides a list of profile specific error codes. They are returned by many BTSDK functions when they fail.

Name	Value	Description
BTSDK_ER_CTP_GW_EXIST	0X0500	CTP gateway instance exists already. Current version SDK only supports one CTP gateway at a time.
BTSDK_ER_CTP_GW_NONEXIST	0X0501	There is no CTP gateway instance.
BTSDK_ER_USER_HANGUP	0X0502	The call is hung up by the user.
BTSDK_ER_REMOTE_HANGUP	0X0503	The call is hung up by the remote part.
BTSDK_ER_CONTINUE	0X0690	OBEX response code “Continue (0x90)” is received.
BTSDK_ER_SUCCESS	0X06A0	OBEX response code “OK, Success (0xA0)” is received.
BTSDK_ER_CREATED	0X06A1	OBEX response code “Created (0xA1)” is received.
BTSDK_ER_ACCEPTED	0X06A2	OBEX response code “Accepted (0XA2” is received
BTSDK_ER_NON_AUTH_INFO	0X06A3	OBEX response code “Non-Authoritative Information (0XA3)” is received.
BTSDK_ER_NO_CONTENT	0X06A4	OBEX response code “No Content (0xA4)” is received.
BTSDK_ER_RESET_CONTENT	0X06A5	OBEX response code “Reset Content (0XA5)” is received.
BTSDK_ER_PARTIAL_CONTENT	0X06A6	OBEX response code “Partial Content (0XA6)” is received.
BTSDK_ER_MULT_CHOICES	0X06B0	OBEX response code “Multiple Choices (0XB0)” is received.
BTSDK_ER_MOVE_PERM	0X06B1	OBEX response code “Moved Permanently (0XB1)” is received.
BTSDK_ER_MOVE_TEMP	0X06B2	OBEX response code “Moved Temporarily” is received.
BTSDK_ER_SEE_OTHER	0X06B3	OBEX response code “See Other (0XB3)” is received.
BTSDK_ER_NOT_MODIFIED	0X06B4	OBEX response code “Not Modified (0XB4)” is received.
BTSDK_ER_USE_PROXY	0X06B5	OBEX response code “Use Proxy” is received.
BTSDK_ER_BAD_REQUEST	0X06C0	OBEX response code “Bad Request – server couldn’t understand request (0XC0)” is received.

BTSDK_ER_UNAUTHORIZED	0X06C1	OBEX response code “Unauthorized (0XC1)” is received.
BTSDK_ER_PAY_REQ	0X06C2	OBEX response code “Payment Required (0XC2)” is received.
BTSDK_ER_FORBIDDEN	0X06C3	OBEX response code “Forbidden – operation is understood but refused (0XC3)” is received.
BTSDK_ER_NOTFOUND	0X06C4	OBEX response code “Not Found (0XC4)” is received.
BTSDK_ER_METHOD_NOT_ALLOWED	0X06C5	OBEX response code “Method not allowed (0XC5)” is received.
BTSDK_ER_NOT_ACCEPTABLE	0X06C6	OBEX response code “Not Acceptable (0XC6)” is received.
BTSDK_ER_PROXY_AUTH_REQ	0X06C7	OBEX response code “Proxy Authentication required” is received.
BTSDK_ER_REQUEST_TIMEOUT	0X06C8	OBEX response code “Request Timeout (0xC8)” is received.
BTSDK_ER_CONFLICT	0X06C9	OBEX response code “Conflict (0XC7)” is received.
BTSDK_ER_GONE	0X06CA	OBEX response code “Gone (0xCA)” is received.
BTSDK_ER_LEN_REQ	0X06CB	OBEX response code “Length Required (0XCB)” is received.
BTSDK_ER_PREC_FAIL	0X06CC	OBEX response code “Precondition failed (0XCC)” is received.
BTSDK_ER_REQ_ENTITY_TOO_LARGE	0X06CD	OBEX response code “Requested entity too large (0XCD)” is received.
BTSDK_ER_URL_TOO_LARGE	0X06CE	OBEX response code “Request URL too large (0XCE)” is received.
BTSDK_ER_UNSUPPORTED_MEDIA_TYPE	0X06CF	OBEX response code “Unsupported media type (0XCF)” is received.
BTSDK_ER_SVR_ERR	0X06D0	OBEX response code “Internal server error (0XD0)” is received.
BTSDK_ER_NOTIMPLEMENTED	0X06D1	OBEX response code “Not Implemented (0XD1)” is received.
BTSDK_ER_BAD_GATEWAY	0X06D2	OBEX response code “Bad Gateway (0XD2)” is received.
BTSDK_ER_SERVICE_UNAVAILABLE	0X06D3	OBEX response code “Service Unavailable (0XD3)” is received.
BTSDK_ER_GATEWAY_TIMEOUT	0X06D4	OBEX response code “Gateway timeout (0XD4)” is received.
BTSDK_ER_HTTP_NOTSUPPORT	0X06D5	OBEX response code “HTTP version not supported (0XD5)” is received.
BTSDK_ER_DATABASE_FULL	0X06E0	OBEX response code “Database Full (0XE0)” is received.
BTSDK_ER_DATABASE_LOCK	0X06E1	OBEX response code “Database Locked (0XE1)” is received.

Table 1: Profile Specific Error Codes.

Regarding other error definitions, please refer to the General Part API document.

4. Data Structures

4.1 Service Registry Parameters

4.1.1 BtSdkLocalSPPServerAttrStru

Definition	typedef struct _BtSdkLocalSPPServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT8 com_index; } BtSdkLocalSPPServerAttrStru, *PBtSdkLocalSPPServerAttrStru;	
Description	The structure BtSdkLocalSPPServerAttrStru contains additional features of a local SPP server.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set.
	com_index	Integer that specifies the serial port on which the connection to this SPP server is connected. For example, in the Windows OS, set *com_index* to 5 when the connection to this SPP server is connected on the COM5.

The ***mask*** member can be one or more of these values.

Value	Description
BTSDK_LSPPSAM_COMINDEX	If this flag is set, the *com_index* member is used to specify the serial port index. Otherwise, this member is ignored.

Remarks

Currently, both SPP client and server connections are combined with Bluetooth virtual serial ports pre-installed in the OS. After SPP connection is created, the application can use the standard OS serial port I/O functions to transfer data over the SPP connection.

The com_index value takes effect only after the specified SPP server is activated successfully by calling *Btsdk_StartServer* function.

If the application doesn't know which Bluetooth virtual serial port is available, just set *pservice_attributes->ext_attributes* to NULL when it calls *Btsdk_AddServer* to add a SPP server. BTSDK will automatically select an idle COM port. The application can call *Btsdk_GetServerPort* to get the actual serial port assigned to the SPP server in the future.

Example

```
/* This sample demonstrates how to add a SPP server with and without a serial port specified. */
BTSVCHDL AppRegisterSPPService(BTUINT8 com_index)
{
    BtSdkLocalServerAttrStru  svc_attr = {0};
    BTSVCHDL svc_hdl = BTSDK_INVALID_HANDLE;
    BtSdkLocalSPPServerAttrStru spp_attr = {0};

    svc_attr.service_class = BTSDK_CLS_SERIAL_PORT;
    svc_attr.mask = BTSDK_LSAM_SERVICENAME;  /* All the other features use default value. */
    strcpy(svc_attr.svc_name, “IVT Serial Port”);
    if (com_index != 0)
    {
        spp_attr.size = sizeof(BtSdkLocalSPPServerAttrStru);
        spp_attr.mask = BTSDK_LSPPSAM_COMINDEX;
        spp_attr.com_index = com_index;
        svc_attr.mask |= BTSDK_LSAM_EXTATTRIBUTES;
        svc_attr.ext_attributes = &spp_attr;
    }
    svc_hdl = Btsdk_AddServer(&SvcAttr);
    if (svc_hdl != BTSDK_INVALID_HANDLE)
        Btsdk_StartServer(svc_hdl);
    return svc_hdl;
}
```

4.1.2 BtSdkLocalDUNServerAttrStru

Definition	typedef struct _BtSdkLocalDUNServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT8 com_index; } BtSdkLocalDUNServerAttrStru, *PBtSdkLocalDUNServerAttrStru;	
Description	The structure BtSdkLocalDUNServerAttrStru contains additional features of a local DUN server (Gateway).	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set. Currently, it is reserved and shall be set to 0.
	com_index	Integer that specifies the serial port to which the physical modem is connected. For example, in the Windows OS, set *com_index* to 1 when the physical modem is connected to the COM1. If it is set to 0, BTSDK uses the default value 1.

Remarks

A physical modem must be presented. After a connection to the DUN server (Gateway) is created, the DUN server (Gateway) will transfer data to and from the Internet through the physical modem.

If *pservice_attributes->ext_attributes* is set to NULL when the application calls *Btsdk_AddServer* to add a DUN server (Gateway). BTSDK assumes that the physical modem is connected to the COM1.

Example

```
/* This sample demonstrates how to add a DUN server with and without a serial port specified. */
BTSVCHDL AppRegisterDUNService(BTUINT8 com_index)
{
    BtSdkLocalServerAttrStru  svc_attr = {0};
    BTSVCHDL svc_hdl = BTSDK_INVALID_HANDLE;
    BtSdkLocalDUNServerAttrStru dun_attr = {0};

    svc_attr.service_class = BTSDK_CLS_DIALUP_NET;
    svc_attr.mask = BTSDK_LSAM_SERVICENAME; /* All the other features use default value. */
    strcpy(svc_attr.svc_name, "IVT Dialup Networking");
    if (com_index != 0)
    {
        dun_attr.size = sizeof(BtSdkLocalDUNServerAttrStru);
```

```
            dun_attr.com_index = com_index;
            svc_attr.mask |= BTSDK_LSAM_EXTATTRIBUTES;
            svc_attr.ext_attributes = &dun_attr;
        }
        svc_hdl = Btsdk_AddServer(&SvcAttr);
        if (svc_hdl != BTSDK_INVALID_HANDLE)
            Btsdk_StartServer(svc_hdl);
        return svc_hdl;
}
```

4.1.3 BtSdkLocalFAXServerAttrStru

Definition	typedef struct _BtSdkLocalFAXServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT8 com_index; } BtSdkLocalFAXServerAttrStru, *PBtSdkLocalFAXServerAttrStru;	
Description	The structure BtSdkLocalFAXServerAttrStru contains additional features of a local FAX server (Gateway).	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set. Currently, it is reserved and shall be set to 0.
	com_index	Integer that specifies the serial port to which the physical modem is connected. For example, in the Windows OS, set *com_index* to 1 when the physical modem is connected to the COM1. If it is set to 0, BTSDK uses the default value 1.

Remarks

A physical modem must be presented. After a connection to the FAX server (Gateway) is created, the FAX server (Gateway) will transfer data to and from the telephony network through the physical modem.

If *pservice_attributes->ext_attributes* is set to NULL when the application calls *Btsdk_AddServer* to add a FAX server (Gateway). BTSDK assumes that the physical modem is connected to the COM1.

Example

```
/* This sample demonstrates how to add a Fax server with and without a serial port specified. */
BTSVCHDL AppRegisterFAXService(BTUINT8 com_index)
{
    BtSdkLocalServerAttrStru  svc_attr = {0};
    BTSVCHDL svc_hdl = BTSDK_INVALID_HANDLE;
    BtSdkLocalFAXServerAttrStru fax_attr = {0};

    svc_attr.service_class = BTSDK_CLS_DIALUP_NET;
    svc_attr.mask = BTSDK_LSAM_SERVICENAME; /* All the other features use default value. */
    strcpy(svc_attr.svc_name, “IVT Fax gateway”);
    if (com_index != 0)
    {
        fax_attr.size = sizeof(BtSdkLocalFAXServerAttrStru);
```

```
            fax_attr.com_index = com_index;
            svc_attr.mask |= BTSDK_LSAM_EXTATTRIBUTES;
            svc_attr.ext_attributes = &fax_attr;
        }
        svc_hdl = Btsdk_AddServer(&SvcAttr);
        if (svc_hdl != BTSDK_INVALID_HANDLE)
            Btsdk_StartServer(svc_hdl);
        return svc_hdl;
}
```

4.1.4 BtSdkLocalFTPServerAttrStru

Definition	typedef struct _BtSdkLocalFTPServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT16 desired_access; BTUINT8 root_dir[BTSDK_PATH_MAXLENGTH]; } BtSdkLocalFTPServerAttrStru, *PBtSdkLocalFTPServerAttrStru;	
Description	The structure BtSdkLocalFTPServerAttrStru contains additional features of a local FTP server.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set.
	desired_access	Specifies how the folders and files of the FTP server can be shared to the FTP client
	root_dir	A null-terminated string that specifies the root directory of the FTP server. It must be a valid path recognized by the OS that running the application.

The ***mask*** member can be one or more of these values.

Value	Description
BTSDK_LFTPSAM_DESIREDACCESS	If this flag is set, the *desired_access* member is used to specify the share option. Otherwise, this member is ignored.
BTSDK_LFTPSAM_ROOTDIR	If this flag is set, the *root_dir* member is used to specify the root directory. Otherwise, this member is ignored.

The ***desired_access*** member can be one of these values.

Value	Description
BTSDK_FTPDA_NOACCESS	The folders and files of the FTP server cannot be accessed by the FTP client.
BTSDK_FTPDA_READWRITE	The folders and files of the FTP server are read only.
BTSDK_FTPDA_READONLY	The folders and files of the FTP server can be read as well as modified.

Remarks

The application must specify a root directory when it requests to add a FTP server.

The format of a path string depends on the target platform running the application. For example, the path string can be “C:\\Bluetooth” in the Windows PC OS, or “/usr/Bluetooth” in the Linux OS.

If not specified additionally in the release note, the path string uses the default code page of the

target platform.

Example

```
/* This sample demonstrates how to add a FTP service record in Window PC OS. */
void AppAddFTPService(void)
{
    BtSdkLocalServerAttrStru  svc_attr = {0};
    BtSdkLocalFTPServerAttrStru  ftp_attr = {0};
    BTSVCHDL svc_hdl = BTSDK_INVALID_HANDLE;

    ftp_attr.size = sizeof(BtSdkLocalFTPServerAttrStru);
    ftp_attr.mask = BTSDK_LFTPSAM_DESIREDACCESS | BTSDK_LFTPSAM_ROOTDIR;
    ftp_attr.desired_access = BTSDK_FTPDA_READONLY;
    strcpy((char*)ftp_attr.root_dir, "D:\\BT_FTP_Root\\");

    svc_attr.service_class = BTSDK_CLS_OBEX_FILE_TRANS;
    svc_attr.mask = BTSDK_LSAM_SECURITYLEVEL | BTSDK_LSAM_EXTATTRIBUTES;
    svc_attr.security_level = BTSDK_SSL_AUTHENTICATION | BTSDK_SSL_AUTHORIZATION |
                              BTSDK_SSL_ENCRYPTION;
    svc_attr.ext_attributes = &ftp_attr;
    svc_hdl = Btsdk_AddServer(&svc_attr);
    if (svc_hdl != BTSDK_INVALID_HANDLE)
        Btsdk_StartServer(svc_hdl);
}
```

4.1.5 BtSdkLocalOPPServerAttrStru

Definition	typedef struct _BtSdkLocalOPPServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT16 vcard_support; BTUINT16 vcal_support; BTUINT16 vnote_support; BTUINT16 vmessage_support; BTUINT8 inbox_path[BTSDK_PATH_MAXLENGTH]; BTUINT8 outbox_path[BTSDK_PATH_MAXLENGTH]; BTUINT8 own_card[BTSDK_CARDNAME_MAXLENGTH]; } BtSdkLocalOPPServerAttrStru, *PBtSdkLocalOPPServerAttrStru;	
Description	The structure BtSdkLocalOPPServerAttrStru contains additional features of a local OPP server.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set.

	vcard_support	A set of flags specifies the vCard content format supported and the operations allowed on the vCard object.
	vcal_support	A set of flags specifies the vCalendar content format supported and the operations allowed on the vCal object.
	vnote_support	A set of flags specifies the operations allowed on the vNote object.
	vmessage_support	A set of flags specifies the operations allowed on the vMessage object.
	inbox_path	A null-terminated string that specifies the directory used to receive files pushed to the OPP server. It must be a valid path recognized by the OS that running the application.
	outbox_path	A null-terminated string that specifies the directory used to store the files to be pulled from the OPP server. It must be a valid path recognized by the OS that running the application.
	own_card	A null-terminated string that specifies the vCard type (*.vcf) file contains the owner's information. It must be a valid path recognized by the OS that running the application. The OPP server will transfer this file when the OPP client request to pull business card from the OPP server.

The ***mask*** member can be one or more of these values.

Value	Description
BTSDK_LOPPSAM_VCARDSUPPORT	If this flag is set, the *vcard_support* member is used to specify the options for the vCard object. Otherwise, this member is ignored.
BTSDK_LOPPSAM_VCALSUPPORT	If this flag is set, the *vcal_support* member is used to specify the options for the vCard object. Otherwise, this member is ignored.
BTSDK_LOPPSAM_VNOTESUPPORT	If this flag is set, the *vnote_support* member is used to specify the options for the vNote object. Otherwise, this member is ignored.
BTSDK_LOPPSAM_VMESSAGESUPPORT	If this flag is set, the *vmessage_support* member is used to specify the options for the vMessage object. Otherwise, this member is ignored.
BTSDK_LOPPSAM_INBOXPATH	If this flag is set, the *inbox_path* member is used to specify inbox directory. Otherwise, this member is ignored.
BTSDK_LOPPSAM_OUTBOXPATH	If this flag is set, the *outbox_path* member is used to specify outbox directory. Otherwise, this member is ignored.
BTSDK_LOPPSAM_OWNCARD	If this flag is set, the *own_card* member is used to specify the default vCard file. Otherwise, this member is ignored.

The ***vcard_support*** member can be one or more of these values.

Value	Description
BTSDK_OPPVCARD_21	Supports vCard2.1 content format.
BTSDK_OPPVCARD_30	Supports vCard3.0 content format.
BTSDK_OBJ_ACCEPT_PUSH	Supports pushing vCard object.
BTSDK_OBJ_ACCEPT_PULL	Supports pulling vCard object.

The ***vcal_support*** member can be one or more of these values.

Value	Description
BTSDK_OPPVCAL_10	Supports vCalendar1.0 content format.
BTSDK_OPPICAL_20	Supports iCalendar2.0 content format.
BTSDK_OBJ_ACCEPT_PUSH	Supports pushing vCalendar object.

The ***vnote_support*** member can be one or more of these values.

Value	Description
BTSDK_OBJ_ACCEPT_PUSH	Supports pushing vNote object.

The ***vmessage_support*** member can be one or more of these values.

Value	Description
BTSDK_OBJ_ACCEPT_PUSH	Supports pushing vMessage object.

Remarks

The application must specify the *inbox_path* so as to support “Object Push” request form the OPP client.

The format of a path string depends on the target platform running the application. For example, the path string can be “C:\\Bluetooth” in the Windows PC OS, or “/usr/Bluetooth” in the Linux OS.

The application must specify the *outbox_path* and the *own_card* so as to support “Business Card Pull” request from the OPP client.

If not specified additionally in the release note, the path string uses the default code page of the target platform.

Example

```
/* This sample demonstrates how to add an OPP service record in Windows PC OS. */
void AppAddOPPService(void)
{
    BtSdkLocalServerAttrStru  svc_attr = {0};
    BtSdkLocalOPPServerAttrStru  opp_attr = {0};
    BTSVCHDL svc_hdl = BTSDK_INVALID_HANDLE;

    opp_attr.size = sizeof(BtSdkLocalOPPServerAttrStru);
    opp_attr.mask = BTSDK_LOPPSAM_VCARDSUPPORT | BTSDK_LOPPSAM_INBOXPATH |
                BTSDK_LOPPSAM_OUTBOXPATH | BTSDK_LOPPSAM_OWNCARD;
    opp_attr.vcard_support = BTSDK_OPPVCARD_21 | BTSDK_OPPVCARD_30 |
                BTSDK_OBJ_ACCEPT_PUSH | BTSDK_OBJ_ACCEPT_PULL;
    strcpy((char*)opp_attr.inbox_path, “D:\\BT_OPP_Inbox\\”);
    strcpy((char*)opp_attr.outbox_path, “D:\\BT_OPP_outbox\\”);
    strcpy((char*)opp_attr.own_card, “MyInfor.vcf”);

    svc_attr.service_class = BTSDK_CLS_OBEX_OBJ_PUSH;
    svc_attr.mask = BTSDK_LSAM_SECURITYLEVEL | BTSDK_LSAM_EXTATTRIBUTES;
    svc_attr.security_level = BTSDK_SSL_AUTHENTICATION | BTSDK_SSL_AUTHORIZATION |
                            BTSDK_SSL_ENCRYPTION;
    svc_attr.ext_attributes = &opp_attr;
    svc_hdl = Btsdk_AddServer(&svc_attr);
    if (svc_hdl != BTSDK_INVALID_HANDLE)
        Btsdk_StartServer(svc_hdl);
}
```

4.1.6 BtSdkLocalA2DPServerAttrStru

Definition	typedef struct _BtSdkLocalA2DPServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT16 dev_type; BTUINT16 trans_mask; BTUINT16 content_protect; BTUINT16 sep_type; BTUINT8 codec_num; BTUINT8 audio_card[BTSDK_A2DP_AUDIOCARD_NAME_LEN]; } BtSdkLocalA2DPServerAttrStru, *PBtSdkLocalA2DPServerAttrStru;	
Description	The structure BtSdkLocalA2DPServerAttrStru contains additional features of a local A2DP Sink service.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set.
	dev_type	Specifies the type of local device. Currently, it is reserved and shall be set to 0.
	trans_mask	A set of flags specifies the AVDTP transport capabilities supported by this A2DP service. Currently, it is reserved and shall be set to 0.
	content_protect	A set of flags specifies the AVDTP content protection capabilities supported by this A2DP service. Currently, it is reserved and shall be set to 0.
	sep_type	Specifies the SEP type of this A2DP service. Currently, it is reserved and shall be set to 0.
	codec_num	Specifies the number of codec supported by the application. Currently, it is reserved and shall be set to 0.
	audio_card	A null-terminated string that specifies the playback device used to play the audio stream received over the Bluetooth A2DP connection.

The ***mask*** member can be one or more of these values,

Value	Description
BTSDK_LA2DPSAM_DEVTYPE	Currently, this flag is ignored.
BTSDK_LA2DPSAM_CONTENTPROTECT	Currently, this flag is ignored.
BTSDK_LA2DPSAM_SEPTYPE	Currently, this flag is ignored.
BTSDK_LA2DPSAM_CODEC	Currently, this flag is ignored.
BTSDK_LA2DPSAM_AUDIOCARD	If this flag is set, the *audio_card* member is used to specify the playback device. Otherwise, this member is ignored.

Remarks

Currently, A2DP Sink service is combined with a playback device pre-installed in the OS. After

a connection with local A2DP Sink service is created, BTSDK will route audio stream received directly to the playback device specified by the *audio_card*.

If only one playback device is available in the target platform, *audio_card* can be set to a NULL string.

Example

```
/* This sample demonstrates how to add an A2DP Sink service record in Windows PC OS. */
void AppAddA2DPSNKService(BTINT8* audio_card)
{
    BtSdkLocalServerAttrStru  svc_attr = {0};
    BtSdkLocalA2DPServerAttrStru  snk_attr = {0};
    BTSVCHDL svc_hdl = BTSDK_INVALID_HANDLE;

    svc_attr.service_class = BTSDK_CLS_AUDIO_SINK;
    svc_attr.mask = BTSDK_LSAM_SECURITYLEVEL;
    svc_attr.security_level = BTSDK_SSL_AUTHENTICATION | BTSDK_SSL_AUTHORIZATION |
                              BTSDK_SSL_ENCRYPTION;

    if (audio_card != NULL && strlen(audio_card) < BTSDK_A2DP_AUDIOCARD_NAME_LEN)
    {
        snk_attr.size = sizeof(BtSdkLocalA2DPServerAttrStru);
        snk_attr.mask = BTSDK_LA2DPSAM_AUDIOCARD;
        strcpy((char*)snk_attr.audio_card, audio_card);
        svc_attr.mask  | = BTSDK_LSAM_EXTATTRIBUTES;
        svc_attr.ext_attributes = &snk_attr;
    }

    svc_hdl = Btsdk_AddServer(&svc_attr);
    if (svc_hdl != BTSDK_INVALID_HANDLE)
        Btsdk_StartServer(svc_hdl);
}
```

4.1.7 BtSdkLocalNAPServerAttrStru

Definition	typedef struct _BtSdkLocalNAPServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT16 security_description; BTUINT32 max_access_rate; BTUINT16 net_access_type; BTUINT16 packet_type_num; BTUINT16 packet_type_list[BTSDK_PACKETTYPE_MAXNUM]; } BtSdkLocalNAPServerAttrStru, *PBtSdkLocalNAPServerAttrStru;	
Description	The structure BtSdkLocalNAPServerAttrStru contains additional features of a local NAP service.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set.
	security_description	Specifies the security level of this service. It is the value of SecurityDescription attribute. The default value is BTSDK_PAN_SECU_SERVICE.
	max_access_rate	Specifies the maximum possible network access data rate supported by this NAP server. It is the value of MaxNetAccessRate attribute. The default value is (100 * 1024).
	net_access_type	Specifies the type of network access available. It is the value of NetAccessType attribute. The default value is BTSDK_PAN_NET_10ETH.
	packet_type_num	Number of supported network packet type. It specifies the number of valid items in the array *packet_type_list*.
	packet_type_list	List of network packet types supported by this NAP service.

The ***mask*** member can be one or more of these values,

Value	Description
BTSDK_LNAPSAM_SECURITYDESCRIPTION	If this flag is set, the *security_description* member is used to specify the security level of this service. Otherwise, this member is ignored.
BTSDK_LNAPSAM_MAXACCESSRATE	If this flag is set, the *max_access_rate* member is used to specify the maximum access data rate supported. Otherwise, this member is ignored.
BTSDK_LNAPSAM_NETACCESSTYPE	If this flag is set, the *net_access_type* member is used to specify the type of network access available. Otherwise, this member is ignored.
BTSDK_LNAPSAM_PACKETTYPE	If this flag is set, the *packet_type_num and packet_type_list* members are used to specify the network packet types supported. Otherwise, these members are ignored.

The ***security_description*** member can be one of these values,

Value	Description
BTSDK_PAN_SECU_NONE	No security is used.
BTSDK_PAN_SECU_SERVICE	Service level enforced security is used.
BTSDK_PAN_SECU_8021X	802.1X security mechanism is used.

The ***net_access_type*** member can be one of these values,

Value	Description
BTSDK_PAN_NET_PSTN	PSTN access is available.
BTSDK_PAN_NET_ISDN	ISDN access is available.
BTSDK_PAN_NET_DSL	DSL access is available.
BTSDK_PAN_NET_CABLEM	Cable Modem access is available.
BTSDK_PAN_NET_10ETH	10Mb Ethernet access is available.
BTSDK_PAN_NET_100ETH	100Mb Ethernet access is available.
BTSDK_PAN_NET_4TOKENR	4Mb Token Ring access is available.
BTSDK_PAN_NET_16TOKENR	16Mb Token Ring access is available.
BTSDK_PAN_NET_100TOKENR	100Mb Token Ring access is available.
BTSDK_PAN_NET_FDDI	FDDI access is available.
BTSDK_PAN_NET_GSM	GSM access is available.
BTSDK_PAN_NET_CDMA	CDMA access is available.
BTSDK_PAN_NET_GRPS	GPRS access is available.
BTSDK_PAN_NET_3GCELL	3G Cellular access is available.
BTSDK_PAN_NET_OTHER	Other network access is available.

Each entry in the ***packet_type_list*** member can be one of these values,

Value	Description
BTSDK_PAN_IPV4	IPv4 packet.
BTSDK_PAN_ARP	ARP packet.

The upper two packets are supported by default. More network packet types can be found in http://www.iana.org/assignments/ethernet-numbers.

Example

```
/* This sample demonstrates how to add a NAP service record in Windows PC OS. */
void AppAddNAPService(void)
{
    BtSdkLocalServerAttrStru  svc_attr = {0};
    BtSdkLocalOPPServerAttrStru  nap_attr = {0};
    BTSVCHDL svc_hdl = BTSDK_INVALID_HANDLE;

```

```
        nap_attr.size = sizeof(BtSdkLocalNAPServerAttrStru);
        nap_attr.mask = BTSDK_LNAPSAM_SECURITYDESCRIPTION | BTSDK_LNAPSAM_MAXACCESSRATE
                        | BTSDK_LNAPSAM_NETACCESSTYPE | BTSDK_LNAPSAM_PACKETTYPE;
        nap_attr.security_description = BTSDK_PAN_SECU_SERVICE;
        nap_attr.max_access_rate = 100 * 1024;
        nap_attr.net_access_type = BTSDK_PAN_NET_100ETH;
        nap_attr.packet_type_num = 2;
        nap_attr.packet_type_list[0] = BTSDK_PAN_IPV4;
        nap_attr.packet_type_list[1] = BTSDK_PAN_ARP;

        svc_attr.service_class = BTSDK_CLS_PAN_NAP;
        svc_attr.mask = BTSDK_LSAM_SECURITYLEVEL | BTSDK_LSAM_EXTATTRIBUTES;
        svc_attr.security_level = BTSDK_SSL_AUTHENTICATION | BTSDK_SSL_AUTHORIZATION |
                                  BTSDK_SSL_ENCRYPTION;
        svc_attr.ext_attributes = &nap_attr;
        svc_hdl = Btsdk_AddServer(&svc_attr);
        if (svc_hdl != BTSDK_INVALID_HANDLE)
            Btsdk_StartServer(svc_hdl);
}
```

4.1.8 BtSdkLocalGNServerAttrStru

Definition	typedef struct _BtSdkLocalGNServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT16 security_description; BTUINT16 packet_type_num; BTUINT16 packet_type_list[BTSDK_PACKETTYPE_MAXNUM]; } BtSdkLocalGNServerAttrStru, *PBtSdkLocalGNServerAttrStru;	
Description	The structure BtSdkLocalGNServerAttrStru contains additional features of a local GN service.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set.
	security_description	Specifies the security level of this service. It is the value of SecurityDescription attribute. The default value is BTSDK_PAN_SECU_SERVICE.
	packet_type_num	Number of supported network packet type. It specifies the number of valid items in the array *packet_type_list*.
	packet_type_list	List of network packet types supported by this NAP service.

The ***mask*** member can be one or more of these values.

Value	Description
BTSDK_LGNSAM_SECURITYDESCRIPTION	If this flag is set, the *security_description* member is used to specify the security level of this service. Otherwise, this member is ignored.
BTSDK_LGNSAM_PACKETTYPE	If this flag is set, the *packet_type_num and packet_type_list* members are used to specify the network packet types supported. Otherwise, these members are ignored.

The ***security_description*** member can be one of these values,

Value	Description
BTSDK_PAN_SECU_NONE	No security is used.
BTSDK_PAN_SECU_SERVICE	Service level enforced security is used.
BTSDK_PAN_SECU_8021X	802.1X security mechanism is used.

Each entry in the ***packet_type_list*** member can be one of these values,

Value	Description
BTSDK_PAN_IPV4	IPv4 packet.
BTSDK_PAN_ARP	ARP packet.

The upper two packets are supported by default. More network packet types can be found in http://www.iana.org/assignments/ethernet-numbers.

4.1.9 BtSdkLocalPANUServerAttrStru

Definition	typedef struct _BtSdkLocalPANUServerAttrStru { BTUINT32 size; BTUINT16 mask; BTUINT16 security_description; BTUINT16 packet_type_num; BTUINT16 packet_type_list[BTSDK_PACKETTYPE_MAXNUM]; } BtSdkLocalPANUServerAttrStru, *PBtSdkLocalPANUServerAttrStru;	
Description	The structure BtSdkLocalPANUServerAttrStru contains additional features of a local PANU service.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies members to retrieve or set.
	security_description	Specifies the security level of this service. It is the value of SecurityDescription attribute. The default value is BTSDK_PAN_SECU_SERVICE.
	packet_type_num	Number of supported network packet type. It specifies the number of valid items in the array *packet_type_list*.
	packet_type_list	If this flag is set, the *packet_type_num and packet_type_list* members are used to specify the network packet types supported. Otherwise, these members are ignored.

The ***mask*** member can be one or more of these values.

Value	Description
BTSDK_LPANUSAM_SECURITYDESCRIPTION	If this flag is set, the *security_description* member is used to specify the security level of this service. Otherwise, this member is ignored.
BTSDK_LPANUSAM_PACKETTYPE	Retrieves or sets the *packet_type_num and packet_type_list* member.

The ***security_description*** member can be one of these values,

Value	Description
BTSDK_PAN_SECU_NONE	No security is used.
BTSDK_PAN_SECU_SERVICE	Service level enforced security is used.
BTSDK_PAN_SECU_8021X	802.1X security mechanism is used.

Each entry in the ***packet_type_list*** member can be one of these values,

Value	Description
BTSDK_PAN_IPV4	IPv4 packet.
BTSDK_PAN_ARP	ARP packet.

The upper two packets are supported by default. More network packet types can be found in http://www.iana.org/assignments/ethernet-numbers.

4.1.10 BtSdkLocalDHCPServerAttrStru

Definition	typedef struct _BtSdkLocalDHCPServerAttrStru { BTUINT32 size; BTUINT16 flag; BTUINT16 mask; BTUINT32 start_ip; BTUINT32 end_ip; BTUINT32 local_ip; BTUINT32 LeaseTime; BTUINT32 RenewTime; BTUINT32 net_mask; } BtSdkLocalDHCPServerAttrStru, *PBtSdkLocalDHCPServerAttrStru;	
Description	The structure BtSdkLocalDHCPServerAttrStru contains information about dhcp server configuration. It is used in the windows PC enviroment only.	
Members	*size*	The length of the structure.
	flag	The status of dhcp
	mask	Specifies which member is available.
	start_ip	Start IP address of IP range dhcp provide
	end_ip	End IP address of IP range dhcp provide
	local_ip	Not used now
	LeaseTime	Lease time of IP
	RenewTime	Renew time of IP
	net_mask	The net mask of IP range.

The ***flag*** member can be one these values.

Value	Description
BTSDK_DHCPFLAG_NODHCP	If GN is started, no dhcp is started. Other members are not used.
BTSDK_DHCPFLAG_DEFAULTDHCP	If GN is started, start dhcp server, and the dhcp configuration is default. Other members are not used.
BTSDK_DHCPFLAG_DEFINEDDHCP	If GN is started, start dhcp server, and the dhcp configuration is defined by application.

The ***mask*** member can be one or more of these values.

Value	Description
BTSDK_LDHCPSAM_IPRANGE	The value of the *start_ip* and *end_ip* members is available.
BTSDK_LDHCPSAM_LOCALIP	The value of the *local_ip* member is available.

BTSDK_LDHCPSAM_TIME	The value of the *LeaseTime* and *RenewTime* members are available.
BTSDK_LDHCPSAM_NETMASK	The value of the *net_mask* member is available.
BTSDK_LDHCPSAM_ALL	The values of all members are available.

4.1.11 BtSdkAppExtSPPAttrStru

Definition	typedef struct _BtSdkAppExtSPPAttrStru { BTUINT32 size; BTUINT32 sdp_record_handle; BtSdkUUIDStru service_class_128; BTUINT8 svc_name[BTSDK_SERVICENAME_MAXLENGTH]; BTUINT16 rf_svr_chnl; BTUINT8 com_index; } BtSdkAppExtSPPAttrStru, *PBtSdkAppExtSPPAttrStru;	
Description	The structure BtSdkAppExtSPPAttrStru contains additional features of a application defined service based on SPP. This service has its own class identifier, but its behavior is the same as that of a SPP service.	
Members	*size*	Size of the structure, in bytes.
	sdp_record_handle	32bit interger specifies the SDP service record handle.
	service_class_128	128bit UUID specifies the service class of this service record
	svc_name	Name of the service record. This string must be coded in UTF-8 format.
	rf_svr_chnl	RFCOMM server channel assigned to this service record.
	com_index	Integer that specifies the serial port on which the connection is connected. For example, in the Windows OS, set *com_index* to 5 when the connection is connected on the COM5.

Remarks

Currently, both SPP client and server connections are combined with Bluetooth virtual serial ports pre-installed in the OS. After SPP connection is created, the application can use the standard OS serial port I/O functions to transfer data over the SPP connection.

4.2 Connection Establishment Parameters

4.2.1 BtSdkSPPConnParamStru

Definition	typedef struct _BtSdkSPPConnParamStru{ BTUINT32 size; BTUINT16 mask; BTUINT8 com_index; } BtSdkSPPConnParamStru, *PBtSdkSPPConnParamStru;	
Description	The structure BtSdkSPPConnParamStru contains additional parameters required to establish a SPP connection to a SPP server.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies connection options. Currently, it is reserved and shall be set to 0.
	com_index	Integer that specifies the serial port on which the SPP connection is connected. For example, in the Windows OS, set *com_index* to 5 when the SPP connection initiated by local application is connected on the COM5.

Remarks

In current version BTSDK, both SPP client and server connections are combined with Bluetooth virtual serial ports pre-installed in the OS. After SPP connection is created, the application can use the standard OS serial port I/O functions to transfer data over the SPP connection.

If the application doesn't know which Bluetooth virtual serial port is available, just set *lParam* to 0 when it calls *Btsdk_Connect or Btsdk_ConnectEx* to connect to a SPP server. BTSDK will automatically select an idle COM port. The application can call *Btsdk_GetClientPort* to get the actual serial port assigned to this SPP connection in the future.

4.2.2 BtSdkDUNConnParamStru

Definition	typedef struct _BtSdkDUNConnParamStru{ BTUINT32 size; BTUINT16 mask; BTUINT8 com_index; } BtSdkDUNConnParamStru, *PBtSdkDUNConnParamStru;	
Description	The structure BtSdkDUNConnParamStru contains additional parameters required to establish a DUN connection to a remote DUN gateway.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies connection options. Currently, it is reserved and shall be set to 0.
	com_index	Integer that specifies the serial port on which the DUN connection is connected. For example, in the Windows OS, set *com_index* to 5 when the DUN connection initiated by local application is connected on the COM5.

Remarks

Currently, DUN Client (Data Terminal) connections are combined with a Bluetooth DUN modem pre-installed in the OS. Each Bluetooth DUN modem is connected to a pre-installed Bluetooth virtual serial port. After connection to a remote DUN gateway is created, the application can use the standard OS modem I/O functions to transfer data over the DUN connection.

If the application doesn't know which Bluetooth virtual serial port is available, just set *lParam* to 0 when it calls *Btsdk_Connect or Btsdk_ConnectEx* to connect to a DUN gateway. BTSDK will automatically select an idle COM port that is assigned to a Bluetooth DUN modem. The application can call *Btsdk_GetClientPort* to get the actual serial port assigned to this DUN connection in the future.

4.2.3 BtSdkFAXConnParamStru

Definition	typedef struct _BtSdkFAXConnParamStru{ BTUINT32 size; BTUINT16 mask; BTUINT8 com_index; } BtSdkFAXConnParamStru, *PBtSdkFAXConnParamStru;	
Description	The structure BtSdkFAXConnParamStru contains additional parameters required to establish a Fax connection to a remote Fax gateway.	
Members	*size*	Size of the structure, in bytes.
	mask	A set of flags specifies connection options. Currently, it is reserved and shall be set to 0.
	com_index	Integer that specifies the serial port on which the Fax connection is connected. For example, in the Windows OS, set *com_index* to 5 when the Fax connection initiated by local application is connected on the COM5.

Remarks

Currently, Fax Client (Data Terminal) connections are combined with a Bluetooth Fax modem pre-installed in the OS. Each Bluetooth Fax modem is connected to a pre-installed Bluetooth virtual serial port. After connection to a remote Fax gateway is created, the application can use the standard OS modem I/O functions to transfer data over the Fax connection.

If the application doesn't know which Bluetooth virtual serial port is available, just set *lParam* to 0 when it calls *Btsdk_Connect or Btsdk_ConnectEx* to connect to a Fax gateway. BTSDK will automatically select an idle COM port that is assigned to a Bluetooth Fax modem. The application can call *Btsdk_GetClientPort* to get the actual serial port assigned to this Fax connection in the future.

4.3 Message Parameters

4.3.1 Btsdk_AGAP_ConnInfo

Definition	struct Btsdk_AGAP_ConnInfo {
	BTUINT8 is_hsp;
	BTCONNHDL hdl;
	};	

Description	The structure Btsdk_AGAP_ConnInfo contains information of the newly created AG connection.	
Members	*is_hsp*	Specifies whether this connection is a Headset AG connection. Zero means it is a Hands-free AG connection. Any none zero value means it is a Headset AG connection.
	hdl	Handle to the new AG connection. The application can call *Btsdk_GetConnectionProperty* with this handle value to retrieve detail information of this connection.

4.3.2 Btsdk_HFAP_ConnInfo

Definition	struct Btsdk_HFAP_ConnInfo {
	BTUINT8 is_hsp_ag;
	BTCONNHDL hdl;
	};	

Description	The structure Btsdk_HFAP_ConnInfo contains information of the newly created AG connection.	
Members	*is_hsp_ag*	Specifies whether this connection is a Headset connection. Zero means it is a Hands-free connection. Any none zero value means it is a Headset connection.
	hdl	Handle to the new HF or HS connection. The application can call *Btsdk_GetConnectionProperty* with this handle value to retrieve detail information of this connection.

4.3.3 BtSdkFileTransferReqStru

Definition	typedef struct _BtSdkFileTransferReqStru { BTDEVHDL dev_hdl; BTUINT16 operation; BTUINT16 flag; BTUINT8 file_name[BTSDK_PATH_MAXLENGTH] } BtSdkFileTransferReqStru, *PBtSdkFileTransferReqStru;	
Description	The structure BtSdkFileTransferReqStru contains information of the file transfer mode selection initiated by the remote FTP or OPP client.	
Members	*dev_hdl*	Handle to the remote device tries to upload a file to or delete a file on the local FTP or OPP server.
	operation	Specifies the function selected.
	flag	Specifies the current status of the function selected.
	file_name	A NULL terminated string specifies the name of the file to be uploaded or deleted.

The ***operation*** parameter can be one of these values,

Value	Description
BTSDK_APP_EV_FTP_PUT	The remote FTP client request to upload a file.
BTSDK_APP_EV_FTP_DEL_FILE	The remote FTP client request to delete the file.
BTSDK_APP_EV_FTP_DEL_FOLDER	The remote FTP client request to delete the folder. In this case, the *file_name* specify the name of the folder to be deleted.
BTSDK_APP_EV_OPP_PUSH	The remote OPP client request to push a file.
BTSDK_APP_EV_OPP_PULL	The remote OPP client request to pull owner's business card.

The ***flag*** parameter can be one of these values,

Value	Description
BTSDK_ER_CONTINUE	Receives the file transfer request and the BTSDK is waiting for response from the application.
BTSDK_ER_SUCCESS	The file upload or delete operation is complete.
(Any other values)	Error code specifies the reason of the failure of file upload or delete operation.

4.4 Personal Information Manager

4.4.1 Phonebook

4.4.1.1 PBDATA

Definition	typedef struct _PBDATA{ DWORD dwSize; CHAR szContactID[MAX_CONTACT_ID_LENGTH]; PB_ContactNameItem * itemName; CHAR szContactBirthday[MAX_CONTACT_BIRTHDAY_LENGTH]; CHAR szContactAnniversary [MAX_CONTACT_BIRTHDAY_LENGTH]; CHAR szContactGroup [MAX_CONTACT_GROUP_LENGTH]; CHAR szContactMemo[MAX_CONTACT_MEMO_LENGTH]; PB_ContactOrgItem * itemOrg; PB_ContactPhotoItem *itemPhoto; DWORD* arrayCountTelephone; PB_ContactTelephoneItem* arrayContactTelephone; DWORD* arrayCountAddress; PB_ContactAddressItem* arrayContactAddress; DWORD* arrayCountEmail; PB_ContactEmailItem* arrayContactEmail; DWORD* arrayCountURL; PB_ContactURLItem* arrayContactURL; DWORD* arrayCountIM; PB_ContactIMItem* arrayContactIM; } PBDATA, *PPBDATA;	
Description	The structure **PBDATA** contains information of a contact got from remote cell phone.	
Members	*dwSize*	The length of the structure, in bytes
	szContactID	The contact ID by which it is stored in the phone, used for editing and deleting.
	itemName	Pointer to the structure PB_ContactNameItem
	szContactBirthday	Date of birth of the individual associated with the contact.
	szContactAnniversary	Anniversary of the individual associated with the contact.
	szContactGroup	Group of the individual associated with the contact.
	szContactMemo	Memo of the individual associated with the contact.
	itemOrg	Pointer to the structure PB_ContactOrgItem.
	itemPhoto	Pointer to the structure PB_ContactPhotoItem.
	arrayCountTelephone	Array count of PB_ContactTelephoneItem.
	arrayContactTelephone	Pointer to the structure PB_ContactTelephoneItem.
	arrayCountAddress	Array count of PB_ContactAddressItem.

	arrayContactAddress	Pointer to the structure PB_ContactAddressItem.
	arrayCountEmail	Array count of PB_ContactEmailItem.
	arrayContactEmail	Pointer to the structure PB_ContactEmailItem.
	arrayCountURL	Array count of PB_ContactURLItem.
	arrayContactURL	Pointer to the structure PB_ContactURLItem.
	arrayCountIM	Array count of PB_ContactIMItem.
	arrayContactIM	Pointer to the structure PB_ContactIMItem.

4.4.1.2 PB_ContactNameItem

Definition	typedef struct _PB_ContactNameItem{ DWORD dwSize; CHAR szContactFirstName[MAX_CONTACT_NAME_LENGTH]; CHAR szContactLastName[MAX_CONTACT_NAME_LENGTH]; CHAR szMiddleName[MAX_CONTACT_NAME_LENGTH]; CHAR szContactPrefixName[MAX_CONTACT_NAME_LENGTH]; CHAR szContactNickName[MAX_CONTACT_NAME_LENGTH]; } PB_ContactNameItem, *PPB_ContactNameItem;	
Description	The structure **PB_ContactNameItem** contains a structured representation of the name of the person, place or thing associated with the vCard object.	
Members	*dwSize*	Size of the structure, in bytes.
	szContactFirstName	First name of the person, place or thing associated with the vCard object
	szContactLastName	Last name of the person, place or thing associated with the vCard object
	szMiddleName	Middle name of the person, place or thing associated with the vCard object
	szContactPrefixName	Name prefix of the person, place or thing associated with the vCard object
	szContactNickName	Nick name of the person, place or thing associated with the vCard object

4.4.2 PB_ContactOrgItem

Definition	typedef struct _PB_ContactOrgItem{ DWORD dwSize; CHAR szOrgName [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szDepartmentName [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szRole [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szTitle [MAX_CONTACT_ADDRESS_LENGTH]; } PB_ContactOrgItem, *PPB_ContactOrgItem;	
Description	The structure **PB_ContactOrgItem** contains information associated with characteristics of the organizations or organizational unites associated with the vCard object.	
Members	*dwSize*	Size of the structure, in bytes.
	szOrgName	Name of the organization.
	szDepartmentName	Department name of the organization.
	szRole	Role of the vCard object within an organization.
	szTitle	Job title, functional position or function of the individual associated with the vCard object within an organization.

4.4.2.1 PB_ContactPhotoItem

Definition	typedef struct _PB_ContactPhotoItem{ INT nPhotoType; CHAR szPhotoURL[MAX_CONTACT_URL_LENGTH]; DWORD dwPhotoSize; BYTE* dataPhoto; } PB_ContactPhotoItem, *PPB_ContactPhotoItem;	
Description	The structure PB_ContactPhotoItem contains an image or photograph of the individual associated with the vCard.	
Members	*nPhotoType*	Graphics format for the Photo property value
	szPhotoURL	Network location of the graphic.
	dwPhotoSize	Size of image data.
	dataPhoto	Encoded image data.

4.4.2.2 PB_ContactTelephoneItem

Definition	typedef struct _PB_ContactTelephoneItem{ DWORD dwSize; BOOL bPreferred; INT nTelephoneType; CHAR szTelephone [MAX_CONTACT_TELEPHONE_LENGTH]; } PB_ContactTelephoneItem, *PPB_ContactTelephoneItem;	
Description	The structure **PB_ContactTelephoneItem** contains	
Members	*dwSize*	Size of the structure, in bytes.
	bPreferred	Preferred or not.
	nTelephoneType	Sub-type of telephone that is associated with the telephone number (e.g., Home, Work, Cellular, Facsimile, Video, Modem, Message Service, or Preferred).
	szTelephone	Telephone number

4.4.2.3 PB_ContactAddressItem

Definition	typedef struct _PB_ContactAddressItem{ DWORD dwSize; BOOL bPreferred; INT nAddressType; CHAR szNation [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szRegion [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szCity [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szStreet [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szPostBOX [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szPostolCODE [MAX_CONTACT_ADDRESS_LENGTH]; CHAR szAddressExtended [MAX_CONTACT_ADDRESS_LENGTH]; } PB_ContactAddressItem, *PPB_ContactAddressItem;	
Description	The structure **PB_ContactAddressItem** contains a structured representation of the physical delivery address for the vCard object.	
Members	*dwSize*	Size of the structure, in bytes.
	bPreferred	Preferred or not.
	nAddressType	Sub-type of the physical delivery address
	szNation	Nation of the physical delivery address
	szRegion	Region of the physical delivery address
	szCity	City of the physical delivery address
	szStreet	Street of the physical delivery address
	szPostBOX	Postbox of the physical delivery address
	szPostolCODE	Postal code of the physical delivery address
	szAddressExtended	Extension of address.

4.4.2.4 PB_ContactEmailItem

Definition	typedef struct _PB_ContactEmailItem{ DWORD dwSize; BOOL bPreferred; INT nEmailType; CHAR szEmail [MAX_CONTACT_TELEPHONE_LENGTH]; } PB_ContactEmailItem, *PPB_ContactEmailItem;	
Description	The structure **PB_ContactEmailItem** contains address for electronic mail communication with the vCard object.	
Members	*dwSize*	Size of the structure, in bytes.
	bPreferred	Preferred electronic mail address.
	nEmailType	Type of electronic mail address.
	szEmail	Address for electronic mail communication with the vCard object.

4.4.2.5 PB_ContactURLItem

Definition	typedef struct _PB_ContactURLItem{ DWORD dwSize; BOOL bPreferred; INT nURLType; CHAR szURL[MAX_CONTACT_URL_LENGTH]; } PB_ContactURLItem, *PPB_ContactURLItem;	
Description	The structure **PB_ContactURLItem** contains Network location of VCARD.	
Members	*dwSize*	Size of the structure, in bytes.
	bPreferred	Preferred or not.
	nURLType	Type of URL
	szURL	Network location of the VCARD

4.4.2.6 PB_ContactIMItem

Definition	typedef struct _PB_ContactIMItem{ DWORD dwSize; BOOL bPreferred; INT nIMType; CHAR szIMURI[MAX_CONTACT_IM_LENGTH]; } PB_ContactIMItem, *PPB_ContactIMItem;	
Description	The structure **PB_ContactIMItem** contains Information of instant messaging	
Members	*dwSize*	Size of the structure, in bytes.
	bPreferred	Preferred or not.
	nIMType	Type of IM.
	szIMURI	ID of instant messaging

4.4.2.7 PHONELIST

Definition	typedef struct _PHONELIST { TCHAR manu[16]; TCHAR model[64]; }PHONELIST , *PPHONELIST ;	
Description	The structure **PHONELIST** contains Information of a cell phone.	
Members	*manu*	Manufacture of the cell phone.
	model	Model of the cell phone

4.4.3 Short Message

4.4.3.1 SMSDATA

Definition	typedef struct _SMSDataInfo{ int ID; int index; int type; int readState; int locked; int position; Int wBoxNumber; TCHAR szSmsBody[PIM_SMS_BODY_LENGTH]; TCHAR szSmsCallerID[PIM_PHONE_NUMBER_LENGTH]; TCHAR timeStamp[PIM_TIME_STAMP_LENGTH]; }SMSDATA, *PSMSDATA;	
Description	The structure **SMSDATA** contains information of an SMS got from remote cell phone.	
Members	*ID*	ID of SMS in local database
	Index	ID of SMS in phone
	type	Type of SMS in phone: SIM or ME
	readState	State of SMS:SMS_UNREAD or SMS_READ
	locked	Locked message could not be deleted
	position	SIM or Phone
	wBoxNumber	State of box, 0:inbox, 1:outbox
	szSmsBody	Content of SMS
	szSmsCallerID	Phone number of caller
	timeStamp	Time of the SMS received/sent

5. 0API Functions

5.1 Serial Port Profile

5.1.1 SPP Server

5.1.1.1 Btsdk_RegisterSPPService

Prototype	BTSVCHDL Btsdk_RegisterSPPService (BTUINT16 index);	
Description	The **Btsdk_RegisterSPPService** function adds an SPP service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters	*index*	[in] Integer that specifies the serial port on which the connection to this SPP server is connected. For example, in the Windows OS, set *com_index* to 5 when the connection to this SPP server is connected on the COM5.
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

Before calling *Btsdk_RegisterSPPService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, both SPP client and server connections are combined with Bluetooth virtual serial ports pre-installed in the OS. After SPP connection is created, the application can use the standard OS serial port I/O functions to transfer data over the SPP connection.

5.1.1.2 Btsdk_UnregisterSPPService

Prototype	BTUINT32 Btsdk_UnregisterSPPService (BTSVCHDL service_handle);	
Description	The **Btsdk_UnregisterSPPService** function removes the specified SPP service record from the SDK service database. If a SPP client connects the SPP service, this function will release the connection first.	
Parameters	*service_handle*	[in] Handle to the SPP service record to be unregistered.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterSPPService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

The *service_handle* value is returned by a previous call to function *Btsdk_AddServer* or *Btsdk_RegisterSPPService*.

5.1.1.3 Btsdk_RegisterAppExtSPPService

Prototype	BTSVCHDL Btsdk_RegisterAppExtSPPService (PBtSdkAppExtSPPAttrStru psvc, BTUINT32 *result);	
Description	The **Btsdk_RegisterAppExtSPPService** function adds an application defined SPP-based service record to SDK service database and then activates it.	
Parameters	*psvc*	[in/out] Pointer to a BtSdkAppExtSPPAttrStru structure. On input, it must specify the value of service_class_128 and svc_name, and may specify the value of com_index and sdp_record_handle. If these two members are set to 0, SDK will assign idle values to them. On output, rf_svr_chnl is set to the value assigned. If com_index and sdp_record_handle provided by the application are 0, SDK will set them to the values assigned internally.
	result	[out] Pointer to an interger to return the extended error information. If this function succeeds, the return value is BTSDK_OK. Otherwise, it is an error code.
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

Before calling *Btsdk_RegisterAppExtSPPService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, both SPP client and server connections are combined with Bluetooth virtual serial ports pre-installed in the OS. After SPP connection is created, the application can use the standard OS serial port I/O functions to transfer data over the SPP connection.Btsdk_UnregisterAppExtSPPService

Prototype	BTUINT32 Btsdk_UnregisterAppExtSPPService (BTSVCHDL service_handle);	
Description	The **Btsdk_UnregisterAppExtSPPService** function removes the specified application defined service record from the SDK service database. If a client connects this service, this function will release the connection first.	
Parameters	*service_handle*	[in] Handle to the service record to be unregistered.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterAppExtSPPService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

The *service_handle* value is returned by a previous call to *Btsdk_RegisterAppExtSPPService* function.

5.1.2 SPP Client

5.1.2.1 Btsdk_ConnectAppExtSPPService

Prototype	BTUINT32 Btsdk_ConnectAppExtSPPService (BTDEVHDL dev_hdl, PBtSdkAppExtSPPAttrStru psvc, BTCONNHDL *conn_hdl);	
Description	The **Btsdk_ConnectAppExtSPPService** function connects to an application defined SPP-based service record.	
Parameters	*dev_hdl*	[in] Handle to the remote device to connect.
	psvc	[in/out] Pointer to a BtSdkAppExtSPPAttrStru structure. On input, it must specify the value of service_class_128, and may specify the value of com_index. If com_index is set to 0, SDK will assign an idle value to it. On output, rf_svr_chnl, svc_name and sdp_record_handle are set to the values retrieved during SDP transaction. If com_index provided by the application is 0, SDK will set it to the value assigned internally.
	conn_hdl	[out] Pointer to a BTCONNHDL variable. If connection created successfully, it will be set to the handle to the connection. Otherwise, it will be set to BTSDK_INVALID_HANDLE.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_ConnectAppExtSPPService*, the local device must be enabled by a previous successful call to *Btsdk_StartBluetooth*.

Currently, both SPP client and server connections are combined with Bluetooth virtual serial ports pre-installed in the OS. After SPP connection is created, the application can use the standard OS serial port I/O functions to transfer data over the SPP connection.

5.1.2.2 Btsdk_SearchAppExtSPPService

Prototype	BTUINT32 Btsdk_SearchAppExtSPPService (BTDEVHDL dev_hdl, PBtSdkAppExtSPPAttrStru psvc,);	
Description	The **Btsdk_SearchAppExtSPPService** function searches a remote device for the application-defined service.	
Parameters	*dev_hdl*	[in] Handle to the remote device to search for the specified service.
	psvc	[in/out] Pointer to a BtSdkAppExtSPPAttrStru structure. On input, it must specify the value of service_class_128. On output, rf_svr_chnl, svc_name and sdp_record_handle are set to the values retrieved during SDP transaction. com_index is ignored by this function.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_SearchAppExtSPPService*, the local device must be enabled by a previous successful call to *Btsdk_StartBluetooth*.

5.2 Dial-up Networking Profile

5.2.1 DUN Gateway (GW)

5.2.1.1 Btsdk_RegisterDUNService

Prototype	BTSVCHDL Btsdk_RegisterDUNService (BTUINT16 index);	
Description	The **Btsdk_RegisterDUNService** function adds a DUN GW service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters	*index*	[in] Integer that specifies the serial port to which the physical modem is connected. For example, in the Windows OS, set *index* to 1 when the physical modem is connected to the COM1. If it is set to 0, BTSDK uses the default value 1.
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

Before calling *Btsdk_RegisterDUNService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, only one DUN GW service record is allowed at a time. That is, if the application calls the *Btsdk_RegisterDUNService* function twice, the second call will first remove the first DUN GW service record and than add a new DUN GW service record.

Currently, A physical modem must be presented. After a connection to the DUN GW is created, the DUN GW will transfer data to and from the Internet through the physical modem.

5.2.1.2 Btsdk_UnregisterDUNService

Prototype	BTUINT32 Btsdk_UnregisterDUNService (void);	
Description	The **Btsdk_UnregisterDUNService** function removes the current DUN GW service record from the SDK service database. If a DUN client connects the DUN GW service, this function will release the connection first.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterDUNService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This DUN GW service record is added to the service database by a previous call to the function *Btsdk_RegisterDUNService function*.

5.3 FAX Profile

5.3.1 FAX Gateway (GW)

5.3.1.1 Btsdk_RegisterFAXService

Prototype	BTSVCHDL Btsdk_RegisterFAXService (BTUINT16 index);	
Description	The **Btsdk_RegisterFAXService** function adds a FAX GW service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters	*index*	[in] Integer that specifies the serial port to which the physical modem is connected. For example, in the Windows OS, set *index* to 1 when the physical modem is connected to the COM1. If it is set to 0, BTSDK uses the default value 1.
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

Before calling *Btsdk_RegisterFAXService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, only one FAX GW service record is allowed at a time. That is, if the application calls the *Btsdk_RegisterFAXService* function twice, the second call will first remove the first FAX GW service record and than add a new FAX GW service record.

Currently, A physical modem must be presented. After a connection to the FAX GW is created, the FAX GW will transfer data to and from the Internet through the physical modem.

5.3.1.2 Btsdk_UnregisterFAXService

Prototype	BTUINT32 Btsdk_UnregisterFAXService (void);	
Description	The **Btsdk_UnregisterFAXService** function removes the current FAX service record from the SDK service database. If a FAX client connects the FAX service, this function will release the connection first.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterFAXService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This FAX service record is added to the service database by a previous call to the function *Btsdk_RegisterFAXService function*.

5.4 Hands-free and Headset Profile

BTSDK provides the same APIs for these two profiles.

5.4.1 Hands-free Unit/Headset (HF/HS)

5.4.1.1 Btsdk_HFAP_Init

Prototype	BTUINT32 Btsdk_HFAP_Init (BTUINT32 features, BTUINT8 sco_pkt_type, BTUINT16 *call_state);	
Description	The **Btsdk_HFAP_Init** function initializes the resources to run Hands-free Unit (HF) and Headset (HS) services.	
Parameters	*features*	[in] A set of flags specifies the features supported by the local HF service.
	sco_pkt_type	[in] A set of flags specifies the SCO packet types to be used for the SCO connection to be established. If can be set to 0, and BTSDK will use all possible packet types.
	call_state	[out] Returns an internal 16bit integer that stores the current call state of the HF/HS service. It can be set to NULL.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

The ***features*** parameter can be one or more of these values,

Value	Description
BTSDK_HFAP_FEA_NREC	Local HF application supports the EC and NR functions.
BTSDK_HFAP_FEA_3WAY_CALLING	Local HF application supports call waiting and three way calling.
BTSDK_HFAP_FEA_CALLING_LINE_NUM	Local HF application supports the CLI presentation capability.
BTSDK_HFAP_FEA_VOICE_RECOG	Local HF application supports voice recognition activation.
BTSDK_HFAP_FEA_RMT_VOL_CTRL	Local HF application supports remote volume control.

The ***sco_pkt_type*** parameter can be one or more of these values,

Value	Description
BTSDK_HFAP_SCO_PKT_HV1	HV1 packet type may be used.
BTSDK_HFAP_SCO_PKT_HV2	HV2 packet type may be used.

BTSDK_HFAP_SCO_PKT_HV3	HV3 packet type may be used.

Remarks

Before calling *Btsdk_HFAP_Init*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This function MUST be called and the return value MUST be BTSDK_OK before any other HF/HS functions can be called.

This function will enable both HF and HS services at the same time. But only one HF/HS connection is allowed at a time, no matter which side (local or remote application) initiates the connection. For example, if a connection between the local HF and a remote AG is created, no more connections with other AGs can be created until the previous connection is released.

A successful call to *Btsdk_HFAP_Init* must be balanced by a corresponding call to *Btsdk_HFAP_Done* after subsequent HF/HS function calls are finished and HF and HS services are no longer required. The application shall not call *Btsdk_HFAP_Init* once again before it calls *Btsdk_HFAP_Done*.

5.4.1.2 Btsdk_HFAP_Done

Prototype	void Btsdk_HFAP_Done (void);	
Description	The **Btsdk_HFAP_Done** function releases the context created by *Btsdk_HFAP_Init*.	
Parameters		
Return:		

Remarks

An application must call *Btsdk_HFAP_Done* once for each successful call it has made to *Btsdk_HFAP_Init.*

This function releases all resources allocated by HF/HS functions and disables HF and HS services finally.

5.4.1.3 Btsdk_HFAP_APPRegCbk

Prototype	void Btsdk_HFAP_APPRegCbk (Btsdk_HFAP_Event_Ind_Func *pfunc);	
Description	The **Btsdk_HFAP_APPRegCbk** function registers an application-defined callback function used to process HF/HS messages created by the BTSDK.	
Parameters	*pfunc*	[in] Pointer to the callback function of Btsdk_HFAP_Event_Ind_Func type. If p*func* is NULL, BTSDK will remove the callback information registered before.
Return:		

Remarks

All messages of both HF and HS from BTSDK are transferred to the application using the same callback function. That is, if the application calls *Btsdk_HFAP_APPRegCbk* twice to register different callback functions, the second callback function will replace the first one.

5.4.1.4 Btsdk_HFAP_Event_Ind_Func

Prototype	typedef void (Btsdk_HFAP_Event_Ind_Func) (BTUINT16 msgid, BTUINT8* pArg, BTUINT32 dwArg);	
Description	The **Btsdk_HFAP_Event_Ind_Func** function prototype is the prototype of application defined callback function used to process HF/HS messages.	
Parameters	*msgid*	[in] Message identifier.
	pArg	[in] First message parameters. It is usually a pointer to a message specific variable.
	dwArg	[in] Second message parameter. It specifies the size of the buffer pointed to by the *pArg* parameter in bytes.
Return:		

The ***msgid*** parameter can be one of these values,

Value	Description
BTSDK_APP_EV_HFAP_RINGING_IND *	An incoming call is alerting. *pArg* is a pointer to a BTUINT8 variable that specifies the connection type. If *pArg is 0, the alerting message is transferred over a Hands-free connection. Otherwise, the message is transferred over a Headset connection.
BTSDK_APP_EV_HFAP_HANGUP_IND *	The incoming call or outgoing call or ongoing call is canceled by either side.
BTSDK_APP_EV_HFAP_OUTGOINGCALL_IND	The AG has initiated outgoing call setup successfully.
BTSDK_APP_EV_HFAP_ONGOINGCALL_IND *	The incoming call or outgoing call is answered.
BTSDK_APP_EV_HFAP_AG_AVAILABLE_IND *	A connection between the local HF or HS module and a remote AG is established. For a HF connection, this message is reported after the service level connection is initialized. For a HS connection, this message is reported after the RFCOMM connection is established. *pArg* is a pointer to the Btsdk_HFAP_ConnInfo structure contains the connection information.

BTSDK_APP_EV_HFAP_AG_UNAVAILABLE_IND *	This message is reported after the RFCOMM connection between the local HF or HS module and a remote AG is released.
BTSDK_APP_EV_HFAP_CLI_IND	"+CLID :" result code is received. pArg is a pointer to a buffer contains the phone number.
BTSDK_APP_EV_HFAP_SPKVOL_CHANGED_IND *	The remote AG requests the local HF/HS application to change its speaker volume. pArg is a pointer to a BTUINT8 variable contains the new speaker volume level.
BTSDK_APP_EV_HFAP_VOICETAG_PHONE_NUM_RSP	The remote AG provides a phone number to be attached to the current voice tag in the local HF application. pArg is a pointer to a buffer contains the phone number.
BTSDK_APP_EV_HFAP_VOICE_RECOGN_ACTIVATED_IND	Voice recognition is enabled in the remote AG.
BTSDK_APP_EV_HFAP_VOICE_RECOGN_DEACTIVATED_IND	Voice recognition is disabled in the remote AG.
BTSDK_APP_EV_HFAP_TERMINATE_LOCAL_RINGTONE_IND	The BTSDK requests the HF/HS application to stop alerting the incoming call.
BTSDK_APP_EV_HFAP_CALL_WAITING_NOTIF	The remote AG reports the presence of new local waiting. pArg is a pointer to a buffer contains the phone number.
BTSDK_APP_EV_HFAP_EXTEND_CMD_IND *	BTSDK receives an extended command from the remote AG. pArg is a pointer to a buffer contains the extended command received.
BTSDK_APP_EV_HFAP_NETWORK_UNAVAILABLE_IND	The remote AG reports that the network service is unavailable currently.
BTSDK_APP_EV_HFAP_NETWORK_AVAILABLE_IND	The remote AG reports that the network service is available now.
BTSDK_APP_EV_HFAP_SCO_CONN_IND *	A SCO/eSCO connection between the local AG module and the remote device is established. *pArg* is a pointer to a BTUINT16 variable contains the SCO/eSCO connection handle.
BTSDK_APP_EV_HFAP_SCO_DISC_IND *	The SCO/eSCO connection between the local AG module and the remote device is released.

5.4.1.5 Btsdk_HFAP_AnswerCall

Prototype	BTUINT32 Btsdk_HFAP_AnswerCall (void);	
Description	The **Btsdk_HFAP_AnswerCall** function informs the AG that the HF has answered the incoming call.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.6 Btsdk_HFAP_CancelCall

Prototype	BTUINT32 Btsdk_HFAP_CancelCall (void);	
Description	The **Btsdk_HFAP_CancelCall** function informs the AG that the HF has cancelled a call. (HF may reject an incoming call or terminate an outgoing call or release an ongoing call.)	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.7 Btsdk_HFAP_LastNumRedial

Prototype	BTUINT32 Btsdk_HFAP_LastNumRedial (void);	
Description	The **Btsdk_HFAP_LastNumRedial** function instructs the AG to redial the last dialed number.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.8 Btsdk_HFAP_MemNumDial

Prototype	BTUINT32 Btsdk_HFAP_MemNumDial (void* mem_location, BTUINT16 len);	
Description	The **Btsdk_HFAP_MemNumDial** function instructs the AG to dial the phone number stored in the AG memory location given by a specific index.	
Parameters	*mem_location*	[in] Pointer to a buffer contains the index string that specifies the AG memory location.
	len	[in] Length of the string, not including the terminated null character.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.9 Btsdk_HFAP_Dial

Prototype	BTUINT32 Btsdk_HFAP_Dial (void* phone_num, BTUINT16 len);	
Description	The **Btsdk_HFAP_Dial** function instructs the AG to dial the supplied phone number.	
Parameters	*phone_num*	[in] Pointer to a buffer contains the phone number string.
	len	[in] Length of the string, not including the terminated null character.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.10 Btsdk_HFAP_VoiceRecognitionReq

Prototype	BTUINT32 Btsdk_HFAP_VoiceRecognitionReq (void);	
Description	The **Btsdk_HFAP_VoiceRecognitionReq** function requests the AG to activate or deactivate the voice recognition procedure.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.11 Btsdk_HFAP_3WayCallingHandler

Prototype	BTUINT32 Btsdk_HFAP_3WayCallingHandler (BTUINT8 op_code);	
Description	The **Btsdk_HFAP_3WayCallingHandler** function changes the status of the held and active calls.	
Parameters	*op_code*	[in] It is the n value used in the “AT+CHLD=<n> “ command. It can be one of the characters ‘0’, ‘1’, ‘2’, ‘3’ and ‘4’.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.12 Btsdk_HFAP_AudioConnTrans

Prototype	BTUINT32 Btsdk_HFAP_AudioConnTrans (void);	
Description	The **Btsdk_HFAP_AudioConnTrans** function transfers the audio path of the ongoing call from or towards the AG.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

If there is no audio connection established between the HF and the AG, this function transfers the audio path of the ongoing call from the AG towards the HF. If the audio connection already exists, this function transfers the audio path of the ongoing call from the HF towards the AG.

5.4.1.13 Btsdk_HFAP_TxDTMF

Prototype	BTUINT32 Btsdk_HFAP_TxDTMF (BTUINT8 chr);	
Description	The **Btsdk_HFAP_TxDTMF** function changes the status of the held and active calls.	
Parameters	*chr*	[in] The DTMF character.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.14 Btsdk_HFAP_SetSpkVol

Prototype	BTUINT32 Btsdk_HFAP_SetSpkVol (BTUINT8 spk_vol);	
Description	The **Btsdk_HFAP_SetSpkVol** function informs the remote AG that the speaker volume of the HF has been changed.	
Parameters	*spk_vol*	[in] The speaker volume level. Range from 0 to 15. 0 = minimum gain; 15 = maximum gain.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

5.4.1.15 Btsdk_HFAP_VoiceTagPhoneNumReq

Prototype	BTUINT32 Btsdk_HFAP_VoiceTagPhoneNumReq (void);	
Description	The **Btsdk_HFAP_VoiceTagPhoneNumReq** function requests from the remote AG a phone number that will be attached to a voice tag in the local HF application.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

The phone number provided by the remote AG will be sent to the HF application through the BTSDK_APP_EV_HFAP_VOICETAG_PHONE_NUM_RSP message.

5.4.1.16 Btsdk_HFAP_ExtendCmd

Prototype	BTUINT32 Btsdk_HFAP_ExtendCmd (void *cmd);	
Description	The **Btsdk_HFAP_ExtendCmd** function transfers a user defined AT command to the remote AG.	
Parameters	*cmd*	[in] A pointer to the buffer that contains the AT command string. This string must be NULL terminated.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

The hands-free profile uses a subset of AT commands and result codes from existing standards. The application may require transferring more AT commands and result codes. This function provides the application this kind of ability.

5.5 File Transfer Profile

The format of a path string depends on the target platform running the application. For example, the path string can be “C:\\Bluetooth” in the Windows PC OS, or “/usr/Bluetooth” in the Linux OS.

Currently, if not specified additionally in the release note, the path string and the file name parameters use the default code page of the target platform.

5.5.1 General

5.5.1.1 Btsdk_FTPRegisterStatusCallback

Prototype	void Btsdk_FTPRegisterStatusCallback(BTCONNHDL conn_hdl, Btsdk_FTP_STATUS_INFO_CB* func);	
Description	The **Btsdk_FTPRegisterStatusCallback** function registers an application-defined callback function used to deal with FTP tranfer file status information.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection. For a FTP client connection, this handle value is returned by a previous successful call to functions *Btsdk_Connect* or *Btsdk_ConnectEx*. For a FTP server connection, this handle value is returned by the BTSDK_CONNECTION_EVENT_IND callback function.
	func	[in] Pointer to the callback function of Btsdk_FTP_STATUS_INFO_CB type.
Return:		

Remarks

This function registers callback function of FTP transfer file status information for the specified FTP connection. Only one callback function of Btsdk_FTP_STATUS_INFO_CB type is allowed for the same *conn_hdl* value. That is, if the application calls *Btsdk_FTPRegisterStatusCallback* twice to register different callback functions for the same connection handle, the second callback function will replace the first one.

If *func* is NULL, the call to *Btsdk_FTPRegisterStatusCallback* will remove the callback for the specified connection handle.

5.5.1.2 Btsdk_FTP_STATUS_INFO_CB

Prototype	typedef void (Btsdk_FTP_STATUS_INFO_CB)(BTUINT8 first, BTUINT8 last, BTUINT8* filename, BTUINT32 filesize, BTUINT32 cursize);	
Description	The **Btsdk_FTP_STATUS_INFO_CB** function prototype is the prototype of application defined callback function used to deal with file transfer status.	
Parameters	*first*	[in] Flag specifies whether it is the first call to this function. Any none zero (TRUE) value means it is the fist call. Otherwise, it is a continuous call.
	last	[in] Flag specifies whether it is the last call to this function. Any none zero (TRUE) value means it is the last call. Otherwise, it is not a last call.
	filename	[in] Pointer to the buffer contains the file name. It is valid only when first flag is not zero.
	filesize	[in] Specifies full size of the file to be transferred in bytes, only valid when first flag is not zero.
	cursize	[in] Specifies current transferred size in bytes.
Return:		

Remarks

This callback function needs to be registered using *Btsdk_FTPRegisterStatusCallback* function. It is always called when the device sends/receives an OBEX package over the specified FTP connection.

5.5.2 FTP Server

5.5.2.1 Btsdk_RegisterFTPService

Prototype	BTSVCHDL Btsdk_RegisterFTPService (BTUINT16 desired_access, BTUINT8 * root_dir);	
Description	The **Btsdk_RegisterFTPService** adds an FTP service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters	*desired_access*	[in] Specifies how the folders and files of the FTP server can be shared to the FTP client
	root_dir	[in] A null-terminated string that specifies the root directory of the FTP server. It must be a valid path recognized by the OS that running the application.
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

The ***desired_access*** member can be one of these values.

Value	Description
BTSDK_FTPDA_NOACCESS	The folders and files of the FTP server cannot be accessed by the FTP client.
BTSDK_FTPDA_READWRITE	The folders and files of the FTP server are read only.
BTSDK_FTPDA_READONLY	The folders and files of the FTP server can be read as well as modified.

Remarks

Before calling *Btsdk_RegisterFTPService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, only one FTP service record is allowed at a time. That is, if the application calls the *Btsdk_RegisterFTPService* function twice, the second call will first remove the first FTP service record and than add a new FTP service record.

5.5.2.2 Btsdk_UnregisterFTPService

Prototype	BTUINT32 Btsdk_UnregisterFTPService (void);	
Description	The **Btsdk_UnregisterFTPService** function removes the current FTP service record from the SDK service database. If a FTP client connects the FTP service, this function will release the connection first.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterFTPService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This FTP service record is added to the service database by a previous call to the function *Btsdk_RegisterFTPService function*.

5.5.2.3 Btsdk_FTPRegisterDealReceiveFileCB

Prototype	Void Btsdk_FTPRegisterDealReceiveFileCB (BTSDK_FTP_UIDealReceiveFile* func);	
Description	The **Btsdk_FTPRegisterDealReceiveFileCB** function registers an application-defined callback function used to process file transfer mode selection requests from the remote FTP client.	
Parameters	*func*	[in] Pointer to the callback function of BTSDK_FTP_UIDealReceiveFile type.
Return:		

Remarks

If the application wants to intervene in the file transfer procedure, e.g. to allow the user to determine whether to accept the file uploading request, it shall register a callback function after the local FTP service is enabled.

5.5.2.4 BTSDK_FTP_UIDealReceiveFile

Prototype	typedef BTBOOL (BTSDK_FTP_UIDealReceiveFile)(PBtSdkFileTransferReqStru pFileInfo);	
Description	The **BTSDK_FTP_UIDealReceiveFile** function prototype is the prototype of application defined callback function used to deal with file transfer requests from the remote FTP client.	
Parameters	*pFileInfo*	[in/out] Pointer to a BtSdkFileTransferReqStru structure specifies the information of the file transfer request.
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is an error code listed in FALSE.	

Remarks

On input, if *pFileInfo->flag* is set to BTSDK_ER_CONTINUE, following operation is allowed:

(1) If the application wants to save the file using a different name, copy the new file name to *pFileInfo->file_name*.

(2) If the application wants to reject the file upload or delete request, change the *pFileInfo->flag* to one of OBEX error code except for BTSDK_ER_CONTINUE and BTSDK_ER_SUCCESS.

(3) If the application allows saving the file, just keep *pFileInfo->flag* unchanged.

5.5.3 FTP Client

5.5.3.1 Btsdk_FTPBrowseFolder

Prototype	BTINT32 Btsdk_FTPBrowseFolder (BTCONNHDL conn_hdl, BTUINT8 * szPath, BTSDK_FTP_UIShowBrowseFile* pShowFunc, BTUINT8 op_type);	
Description	The **Btsdk_FTPBrowseFolder** function browses the remote device folder.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	szPath	[in] Specifies the remote path to be browsed. A NULL pointer is used to specify the root directory.
	pShowFunc	[in] Pointer to the callback function of BTSDK_FTP_UIShowBrowseFile type.
	op_type	[in] Specifies the operation type.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

The ***op_type*** member can be one of these values.

Value	Description
FTP_OP_REFRESH	Refresh the current directory. The *szPath* shall contain the name of the current directory.
FTP_OP_UPDIR	Up one level directory. The *szPath* is ignored.
FTP_OP_NEXT	Change the current directory to *szPath* and show the content of the directory. The *szPath* shall be the name of a sub-folder of the current directory.

Remarks

Before calling *Btsdk_FTPBrowseFolder*, a FTP connection between local device and the target device must be created first.

The *Btsdk_FTPBrowseFolder* function will go through the specified folder and report information of each file or sub-folder to the application through the callback function *pShowFunc*.

5.5.3.2 BTSDK_FTP_UIShowBrowseFile

Prototype	typedef void (BTSDK_FTP_UIShowBrowseFile) (BTUINT8* SYS_FIND_DATA);	
Description	The **BTSDK_FTP_UIShowBrowseFile** function prototype is the prototype of application defined callback function used to show file or folder information on the remote device.	
Parameters	*SYS_FIND_DATA*	[in] Pointer to an OS dependent structure describes the file found. The application should use the *Btsdk_FreeMemory* function to free the buffer pointed to by the *SYS_FIND_DATA* when it is no longer needed
Return:		

Remarks

Refers to the porting guide for detail information of the structure type of *SYS_FIND_DATA*.

Currently, the SYS_FIND_DATA shall be converted to a pointer of *WIN32_FIND_DATA* type if the application runs in the Windows OS (98/2000/XP/CE).

5.5.3.3 Btsdk_FTPSetRmtDir

Prototype	BTINT32 Btsdk_FTPSetRmtDir (BTCONNHDL conn_hdl, BTUINT8 * szDir);	
Description	The **Btsdk_FTPSetRmtDir** function sets the current directory of the remote device.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	szDir	[in] Pointer to a buffer that contains the current directory to be set. It must be a relative path start with '\', which means the root directory, e.g. "\dir1\dir2". If szDir is NULL, root directory will be set. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPSetRmtDir*, a FTP connection between local device and the specified remote device must be created first.

After calling this function successfully, the application can call *Btsdk_FTPGetRmtDir* to get the current directory, call *Btsdk_FTPBrowseFolder* to browse the contents or call *Btsdk_FTPBackDir* to go up one level directory.

5.5.3.4 Btsdk_FTPGetRmtDir

Prototype	BTINT32 Btsdk_FTPGetRmtDir (BTCONNHDL conn_hdl, BTUINT8 * szDir);	
Description	The **Btsdk_FTPGetRmtDir** function gets the current directory of the remote device.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	szDir	[out] Pointer to a buffer used to receive the current directory. The size of this buffer shall be larger than BTSDK_PATH_MAXLENGTH in bytes.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPGetRmtDir*, a FTP connection between local device and the specified remote device must be created first.

The application can call *Btsdk_FTPSetRmtDir* to set the current directory of the remote device first. If the application does not call *Btsdk_FTPSetRmtDir* before, calling *Btsdk_FTPGetRmtDir* may get the root directory of the remote device.

After calling this function, the application can call *Btsdk_FTPBrowseFolder* to browse the contents of the current directory on the remote device.

5.5.3.5 Btsdk_FTPCreateDir

Prototype	BTINT32 Btsdk_FTPCreateDir (BTCONNHDL conn_hdl, BTUINT8 * szDir);	
Description	The **Btsdk_FTPCreateDir** function creates a new folder on the remote FTP server.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	szDir	[in] Pointer to a buffer contains the name of the new folder to be created.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPCreateDir*, a FTP connection between local device and the specified remote device must be created first.

After calling this function successfully, the application can call *Btsdk_FTPDeleteDir* to delete the directory or call *Btsdk_FTPSetRmtDir* to set it as the current directory.

5.5.3.6 Btsdk_FTPDeleteDir

Prototype	BTINT32 Btsdk_FTPDeleteDir (BTCONNHDL conn_hdl, BTUINT8 * szDir);	
Description	The **Btsdk_FTPDeleteDir** function deletes a folder on the remote FTP server.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	szDir	[in] Pointer to a buffer contains the name of the folder to be deleted.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPDeleteDir*, a FTP connection between local device and the specified remote device must be created first.

5.5.3.7 Btsdk_FTPDeleteFile

Prototype	BTINT32 Btsdk_FTPDeleteFile (BTCONNHDL conn_hdl, BTUINT8 * szFile);	
Description	The **Btsdk_FTPDeleteFile** function deletes a file on the remote FTP server.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	szFile	[in] Pointer to a buffer contains the name of the file to be deleted.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPDeleteFile*, a FTP connection between local device and the specified remote device must be created first.

5.5.3.8 Btsdk_FTPCancelTransfer

Prototype	BTINT32 Btsdk_FTPCancelTransfer (BTCONNHDL conn_hdl,);	
Description	The **Btsdk_FTPCancelTransfer** function terminates the file transfer procedure.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

This function only terminates the ongoing file transfer procedures over the specified connection. It DOES NOT release the specified connection.

5.5.3.9 Btsdk_FTPPutDir

Prototype	BTINT32 Btsdk_FTPPutDir (BTCONNHDL conn_hdl, BTUINT8 * loc_dir, BTUINT8* new_dir);	
Description	The **Btsdk_FTPPutDir** function uploads all contents under the specified directory to the remote FTP server.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	loc_dir	[in] Pointer to a buffer contains the full path of the local directory to be uploaded. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
	new_dir	[in] Pointer to a buffer contains the name of the destination folder on the remote FTP server.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPPutDir*, a FTP connection between local device and the specified remote device must be created first.

The application can call *Btsdk_FTPCancelTransfer* function to terminate the transfer procedure.

5.5.3.10 Btsdk_FTPPutFile

Prototype	BTINT32 Btsdk_FTPPutFile (BTCONNHDL conn_hdl, BTUINT8 * loc_file, BTUINT8* new_file);	
Description	The **Btsdk_FTPPutFile** function uploads all contents under the specified directory to the remote FTP server.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	loc_file	[in] Pointer to a buffer contains the full path of the local file to be uploaded. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
	new_file	[in] Pointer to a buffer contains the name of the destination file on the remote FTP server.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPPutFile*, a FTP connection between local device and the specified remote device must be created first.

The application can call *Btsdk_FTPCancelTransfer* function to terminate the transfer procedure.

5.5.3.11 Btsdk_FTPGetDir

Prototype	BTINT32 Btsdk_FTPGetDir (BTCONNHDL conn_hdl, BTUINT8 * rmt_dir, BTUINT8* new_dir);	
Description	The **Btsdk_FTPGetDir** function downloads all contents under the specified directory from the remote FTP server.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	rmt_dir	[in] Pointer to a buffer contains the name of the source folder on the remote FTP server.
	new_dir	[in] Pointer to a buffer contains the full path of the local directory to receive the downloaded contents. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPGetDir*, a FTP connection between local device and the specified remote device must be created first.

The application can call *Btsdk_FTPCancelTransfer* function to terminate the transfer procedure.

5.5.3.12 Btsdk_FTPGetFile

Prototype	BTINT32 Btsdk_FTPGetFile (BTCONNHDL conn_hdl, BTUINT8 * rmt_file, BTUINT8* new_file);	
Description	The **Btsdk_FTPGetFile** function downloads a file from the remote FTP server.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
	rmt_file	[in] Pointer to a buffer contains the name of the source file on the remote FTP server.
	new_file	[in] Pointer to a buffer contains the full path of the local file to store the downloaded content. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPGetFile*, a FTP connection between local device and the specified remote device must be created first.

The application can call *Btsdk_FTPCancelTransfer* function to terminate the transfer procedure.

5.5.3.13 Btsdk_FTPBackDir

Prototype	BTINT32 Btsdk_FTPBackDir (BTCONNHDL conn_hdl,);	
Description	The **Btsdk_FTPBackDir** function changes the current directory on the remote FTP server to its parent directory.	
Parameters	*conn_hdl*	[in] Handle to the FTP connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_FTPBackDir*, a FTP connection between local device and the specified remote device must be created first.

The application can call this function to go up one step of the remote directory after calling *Btsdk_FTPSetRmtDir* successfully.

5.6 Object Push Profile

The format of a path string depends on the target platform running the application. For example, the path string can be “C:\\Bluetooth” in the Windows PC OS, or “/usr/Bluetooth” in the Linux OS.

Currently, if not specified additionally in the release note, the path string and the file name parameters use the default code page of the target platform.

5.6.1 General

5.6.1.1 Btsdk_OPPRegisterStatusCallback

Prototype	void Btsdk_OPPRegisterStatusCallback(BTCONNHDL conn_hdl, Btsdk_OPP_STATUS_INFO_CB* func);	
Description	The **Btsdk_OPPRegisterStatusCallback** function registers an application-defined callback function used to deal with FTP tranfer file status information.	
Parameters	*conn_hdl*	[in] Handle to the OPP connection. For an OPP client connection, this handle value is returned by a previous successful call to functions *Btsdk_Connect* or *Btsdk_ConnectEx*. For an OPP server connection, this handle value is returned by the BTSDK_CONNECTION_EVENT_IND callback function.
	func	[in] Pointer to the callback function of Btsdk_OPP_STATUS_INFO_CB type.
Return:		

Remarks

This function registers callback function of OPP transfer file status information for the specified OPP connection. Only one callback function of Btsdk_OPP_STATUS_INFO_CB type is allowed for the same *conn_hdl* value. That is, if the application calls *Btsdk_OPPRegisterStatusCallback* twice to register different callback functions for the same connection handle, the second callback function will replace the first one.

If *func* is NULL, the call to *Btsdk_OPPRegisterStatusCallback* will remove the callback for the specified connection handle.

5.6.1.2 Btsdk_OPP_STATUS_INFO_CB

Prototype	typedef void (Btsdk_OPP_STATUS_INFO_CB)(BTUINT8 first, BTUINT8 last, BTUINT8* filename, BTUINT32 filesize, BTUINT32 cursize);	
Description	The **Btsdk_FTP_STATUS_INFO_CB** function prototype is the prototype of application defined callback function used to deal with file transfer status.	
Parameters	*first*	[in] Flag specifies whether it is the first call to this function. Any none zero (TRUE) value means it is the fist call. Otherwise, it is a continuous call.
	last	[in] Flag specifies whether it is the last call to this function. Any none zero (TRUE) value means it is the last call. Otherwise, it is not a last call.
	filename	[in] Pointer to the buffer contains the file name. It is valid only when first flag is not zero.
	filesize	[in] Specifies full size of the file to be transferred in bytes, only valid when first flag is not zero.
	cursize	[in] Specifies current transferred size in bytes.
Return:		

Remarks

This callback function needs to be registered using *Btsdk_OPPRegisterStatusCallback* function. It is always called when the device sends/receives an OBEX package over the specified OPP connection.

5.6.2 OPP Server

5.6.2.1 Btsdk_RegisterOPPService

Prototype	BTSVCHDL Btsdk_RegisterOPPService (BTUINT8* inbox_path, BTUINT8* outbox_path, BTUINT8* own_card);	
Description	The **Btsdk_RegisterOPPService** adds an OPP service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters	*inbox_path*	[in] A null-terminated string that specifies the directory used to receive files pushed to the OPP server. It must be a valid path recognized by the OS that running the application.
	outbox_path	[in] A null-terminated string that specifies the directory used to store the files to be pulled from the OPP server. It must be a valid path recognized by the OS that running the application.
	own_card	[in] A null-terminated string that specifies the vCard type (*.vcf) file contains the owner's information. It must be a valid path recognized by the OS that running the application. The OPP server will transfer this file when the OPP client request to pull business card from the OPP server.
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

Before calling *Btsdk_RegisterOPPService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, only one OPP service record is allowed at a time. That is, if the application calls the *Btsdk_RegisterOPPService* function twice, the second call will first remove the first OPP service record and than add a new OPP service record.

The application must specify the *inbox_path* so as to support "Object Push" request form the OPP client. The application must specify the *outbox_path* and the *own_card* so as to support "Business Card Pull" request from the OPP client.

5.6.2.2 Btsdk_UnregisterOPPService

Prototype	BTUINT32 Btsdk_UnregisterOPPService (void);	
Description	The **Btsdk_UnregisterOPPService** function removes the current OPP service record from the SDK service database. If an OPP client connects the OPP service, this function will release the connection first.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterOPPService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This OPP service record is added to the service database by a previous call to the function *Btsdk_RegisterOPPService function*.

5.6.2.3 Btsdk_OPPRegisterDealReceiveFileCB

Prototype	Void Btsdk_OPPRegisterDealReceiveFileCB (BTSDK_OPP_UIDealReceiveFile* func);	
Description	The **Btsdk_OPPRegisterDealReceiveFileCB** function registers an application-defined callback function used to process file transfer mode selection requests from the remote OPP client.	
Parameters	*func*	[in] Pointer to the callback function of BTSDK_OPP_UIDealReceiveFile type.
Return:		

Remarks

If the application wants to intervene in the file transfer procedure, e.g. to allow the user to determine whether to accept the file uploading request, it shall register a callback function after the local OPP service is enabled.

5.6.2.4 BTSDK_OPP_UIDealReceiveFile

Prototype	typedef BTBOOL (BTSDK_OPP_UIDealReceiveFile) (PBtSdkFileTransferReqStru pFileInfo);	
Description	The **BTSDK_OPP_UIDealReceiveFile** function prototype is the prototype of application defined callback function used to deal with file transfer requests from the remote OPP client.	
Parameters	*pFileInfo*	[in/out] Pointer to a BtSdkFileTransferReqStru structure specifies the information of the file transfer request.
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is an error code listed in FALSE.	

Remarks

On input, if *pFileInfo->flag* is set to BTSDK_ER_CONTINUE, following operation is allowed:

(4) If the application wants to save the file using a different name, copy the new file name to *pFileInfo->file_name*.

(5) If the application wants to reject the file upload request, change the *pFileInfo->flag* to one of OBEX error code except for BTSDK_ER_CONTINUE and BTSDK_ER_SUCCESS.

(6) If the application allows saving the file, just keep *pFileInfo->flag* unchanged.

5.6.3 OPP Client

5.6.3.1 Btsdk_OPPCancelTransfer

Prototype	BTINT32 Btsdk_OPPCancelTransfer (BTCONNHDL conn_hdl,);	
Description	The **Btsdk_OPPCancelTransfer** function terminates the file transfer procedure.	
Parameters	*conn_hdl*	[in] Handle to the OPP connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

This function only terminates the ongoing file transfer procedures over the specified connection. It DOES NOT release the specified connection.

5.6.3.2 Btsdk_OPPPushObj

Prototype	BTINT32 Btsdk_OPPPushObj (BTCONNHDL conn_hdl, BTUINT8 * szPushFilePath);	
Description	The **Btsdk_OPPPushObj** function pushes an object to the remote OPP server. Currently, the object contents must be stored in a file.	
Parameters	*conn_hdl*	[in] Handle to the OPP connection.
	szPushFilePath	[in] Pointer to a buffer contains the full path of the local file containing the object contents to be pushed. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_OPPPushObj*, an OPP connection between local device and the specified remote device must be created first.

The application can call *Btsdk_OPPCancelTransfer* function to terminate the transfer procedure.

5.6.3.3 Btsdk_OPPPullObj

Prototype	BTINT32 Btsdk_OPPPullObj (BTCONNHDL conn_hdl, BTUINT8 * szPushFilePath);	
Description	The **Btsdk_OPPPullObj** function pulls the owner's business card form the remote OPP server.	
Parameters	*conn_hdl*	[in] Handle to the OPPP connection.
	szPushFilePath	[in] Pointer to a buffer contains the local path to store the business card file. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_OPPPullObj*, a FTP connection between local device and the specified remote device must be created first.

Currently, the received business card file is always named as "remote.vcf".

The application can call *Btsdk_OPPCancelTransfer* function to terminate the transfer procedure.

5.6.3.4 Btsdk_OPPExchangeObj

Prototype	BTINT32 Btsdk_OPPExchangeObj (BTCONNHDL conn_hdl, BTUINT8 * szPushFilePath, BTUINT8 * szPullFilePath, BTINT32 * npushError, BTINT32 * npullError);	
Description	The **Btsdk_OPPExchangeObj** function exchanges business card with the remote OPP server.	
Parameters	*conn_hdl*	[in] Handle to the OPP connection.
	szPushFilePath	[in] Pointer to a buffer contains the full path of the local file containing the object contents to be pushed. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
	szPullFilePath	[in] Pointer to a buffer contains the local path to store the business card file. The path size must be smaller than BTSDK_PATH_MAXLENGTH.
	nPushError	[out] Pointer to a buffer to receive the push operation result.
	nPullError	[out] Pointer to a buffer to receive the pull operation result.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code. Check *npushError and npullError result of push and pull operation separately.	

Remarks

Before calling *Btsdk_OPPExchangeObj*, an OPP connection between local device and the specified remote device must be created first.

Currently, the received business card file is always named as “remote.vcf”.

The application can call *Btsdk_OPPCancelTransfer* function to terminate the transfer procedure.

5.7 Personal Area Networking Profile

5.7.1 General

5.7.1.1 Btsdk_PAN_RegIndCbk

Prototype	void Btsdk_PAN_RegIndCbk (Btsdk_PAN_Event_Ind_Func *pfunc);	
Description	The **Btsdk_PAN_RegIndCbk** function registers an application-defined callback function used to deal with PAN messages created by the BTSDK.	
Parameters	*pfunc*	[in] Pointer to the callback function of Btsdk_PAN_Event_Ind_Func type.
Return:		

Remarks

Only one callback function of Btsdk_PAN_Event_Ind_Func type is allowed at a time. That is, if the application calls *Btsdk_PAN_RegIndCbk* twice to register different callback functions, the second callback function will replace the first one.

If *func* is NULL, the call to *Btsdk_PAN_RegIndCbk* will remove the callback function information.

5.7.1.2 Btsdk_PAN_Event_Ind_Func

Prototype	typedef void (Btsdk_PAN_Event_Ind_Func)(BTUINT16 event, BTUINT16 len, BTUINT8* param);	
Description	The **Btsdk_PAN_Event_Ind_Func** function prototype is the prototype of application defined callback function used to deal with PAN messages.	
Parameters	*event*	[in] Event identifier.
	len	[in] If *param* is not set to NULL, *len* specifies the size of the buffer pointed to by the *param* parameter in bytes. Otherwise, it is set to 0.
	param	[in] Event specific parameter.
Return:		

The ***event*** parameter can be one of these values,

Value	Description
BTSDK_PAN_EV_IP_CHANGE	The IP address of the Bluetooth network adapter is changed. The *param* parameter is a pointer to a 32bit integer contains the new IP address value.

5.7.1.3 Btsdk_RegisterPANService

Prototype	BTSVCHDL Btsdk_RegisterPANService (BTUINT16 svcUUID, BTUINT16 len, const BTINT8* param);	
Description	The **Btsdk_RegisterPANService** adds a PAN service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters	*svcUUID*	[in] 16bit UUID specifies the type of the PAN service.
	len	[in] If *param* is not set to NULL, *len* specifies the size of the buffer pointed to by the *param* parameter in bytes. Otherwise, it shall be set to 0.
	param	[in] Additional service information.
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

The ***svcUUID*** parameter can be one of these values,

Value	Description
BTSDK_CLS_PAN_GN	To add a PAN GN service record. The *param* parameter is a pointer to a BtSdkLocalDHCPServerAttrStru structure. If *param* is set to NULL, BTSDK will adopt a default DHCP setting.
BTSDK_CLS_PAN_NAP	To add a PAN NAP service record. The *param* parameter is a pointer to a UNINCODE string specifies the adapter link. The length of this string is limited to BTSDK_ADAPTERLINK_MAXNUM in wide characters. The *param* parameter is used only in the Windows PC environment. For other OS, it shall be set to NULL.
BTSDK_CLS_PAN_PANU	To add a PAN PANU service record. The *param* parameter shall be set to NULL.

Remarks

Before calling *Btsdk_RegisterPANService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, only one PAN service record is allowed at a time. That is, if the application calls the *Btsdk_RegisterPANService* function twice, the second call will first remove the first PAN service record and than add a new PAN service record.

5.7.1.4 Btsdk_UnregisterPANService

Prototype	BTUINT32 Btsdk_UnregisterPANService (void);	
Description	The **Btsdk_UnregisterPANService** function removes the current PAN service record from the SDK service database. If a PAN client connects the PAN service, this function will release the connection first.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterPANService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This PAN service record is added to the service database by a previous call to the function *Btsdk_RegisterPANService function*.

5.8 Advanced Audio Distributed Profile

5.8.1 A2DP Source

5.8.1.1 Btsdk_RegisterA2DPSRCService

Prototype	BTSVCHDL Btsdk_RegisterA2DPSRCService (void);	
Description	The **Btsdk_RegisterA2DPSRCService** function adds an A2DP SRC service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters		
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

Before calling *Btsdk_RegisterA2DPSRCService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, only one A2DP SRC service record is allowed at a time. That is, if the application calls the *Btsdk_RegisterA2DPSRCService* function twice, the second call will first remove the first A2DP SRC service record and than add a new A2DP SRC service record.

5.8.1.2 Btsdk_UnregisterA2DPSRCService

Prototype	BTSVCHDL Btsdk_UnregisterA2DPSRCService (void);	
Description	The **Btsdk_UnregisterA2DPSRCService** function removes the current A2DP SRC service record from the SDK service database. If an A2DP SNK connects the SRC service, this function will release the connection first.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterA2DPSRCService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This A2DP SRC service record is added to the service database by a previous call to the function *Btsdk_RegisterA2DPSRCService function.*

5.8.1.3 Btsdk_A2DPWritePCMData

Prototype	BTINT32 Btsdk_A2DPWritePCMData (BTCONNHDL conn_hdl, BTUINT8 *pcm_buf, BTUINT32 pcm_length);	
Description	The **Btsdk_A2DPWritePCMData** function transmits PCM raw stream through the specified A2DP connection.	
Parameters	*conn_hdl*	[in] Handle to the A2DP connection.
	pcm_buf	[in] Pointer to the buffer contains the PCM raw stream.
	pcm_length	[in] Specify the length, in bytes, of the PCM raw stream stored in the buffer pointed to by the *pcm_buf*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_A2DPWritePCMData*, an A2DP connection between local SRC and the specified remote SNK device must be created first.

5.8.2 A2DP Sink

5.8.2.1 Btsdk_RegisterA2DPSNKService

Prototype	BTSVCHDL Btsdk_RegisterA2DPSNKService(BTUINT16 len, const BTUINT8* audio_card);	
Description	The **Btsdk_RegisterA2DPSNKService** function adds an A2DP SNK service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters	*len*	[in] Specifies the size, in bytes, of the buffer pointed to by the *audio_card* parameter. It shall be smaller than BTSDK_A2DP_AUDIOCARD_NAME_LEN.
	audio_card	[in] A character string that specifies the playback device used to play the audio stream received over the Bluetooth A2DP connection.
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

Before calling *Btsdk_RegisterA2DPSNKService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, only one A2DP SNK service record is allowed at a time. That is, if the application calls the *Btsdk_RegisterA2DPSNKService* function twice, the second call will first remove the first A2DP SNK service record and than add a new A2DP SNK service record.

5.8.2.2 Btsdk_UnregisterA2DPSNKService

Prototype	BTSVCHDL Btsdk_UnregisterA2DPSNKService (void);	
Description	The **Btsdk_UnregisterA2DPSNKService** function removes the current A2DP SNK service record from the SDK service database. If an A2DP SRC connects the SNK service, this function will release the connection first.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterA2DPSNKService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This A2DP SNK service record is added to the service database by a previous call to the function *Btsdk_RegisterA2DPSNKService function*.

5.9 Audio/Video Remote Control Profile

5.9.1 AVRCP Target (TG)

5.9.1.1 Btsdk_RegisterAVRCPTGService

Prototype	BTSVCHDL Btsdk_RegisterAVRCPTGService (void);	
Description	The **Btsdk_RegisterAVRCPTGService** function adds an AVRCP TG service record to SDK service database and then activates it. It has the same effect as calling function *Btsdk_AddServer* first and then *Btsdk_StartServer*.	
Parameters		
Return:	If the function succeeds, the return value is the handle to the new service record. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

Before calling *Btsdk_RegisterAVRCPTGService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

Currently, only one AVRCP TG service record is allowed at a time. That is, if the application calls the *Btsdk_RegisterAVRCPTGService* function twice, the second call will first remove the first AVRCP TG service record and than add a new AVRCP TG service record.

5.9.1.2 Btsdk_UnregisterAVRCPTGService

Prototype	BTSVCHDL Btsdk_UnregisterAVRCPTGService (void);	
Description	The **Btsdk_UnregisterAVRCPTGService** function removes the current AVRCP TG service record from the SDK service database. If an AVRCP Controller (CT) connects the TG service, this function will release the connection first.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_UnregisterAVRCPTGService*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This AVRCP TG service record is added to the service database by a previous call to the function *Btsdk_RegisterAVRCPTGService function*.

5.9.1.3 Btsdk_AVRCP_RegPassThrCmdCbk

Prototype	void Btsdk_AVRCP_RegPassThrCmdCbk (Btsdk_AVRCP_PassThr_Cmd_Func *pfunc);	
Description	The **Btsdk_AVRCP_RegPassThrCmdCbk** function registers an application-defined callback function used to deal with PASS THROUTH command from the Controller.	
Parameters	*pfunc*	[in] Pointer to the callback function of Btsdk_AVRCP_PassThr_Cmd_Func type. If p*func* is NULL, BTSDK will remove the callback information registered before.
Return:		

Remarks

Only one callback function of Btsdk_AVRCP_PassThr_Cmd_Func type is allowed at a time. That is, if the application calls *Btsdk_AVRCP_RegPassThrCmdCbk* twice to register different callback functions, the second callback function will replace the first one.

5.9.1.4 Btsdk_AVRCP_PassThr_Cmd_Func

Prototype	typedef void (Btsdk_AVRCP_PassThr_Cmd_Func) (BTUINT8 op_id, BTUINT8 state_flag,);	
Description	The **Btsdk_AVRCP_PassThr_Cmd_Func** function prototype is the prototype of application defined callback function used to deal with PASS THROUTH command from the Controller.	
Parameters	*op_id*	[in] Operation identifier specifies the command.
	statte_flag	[in] Button status.
Return:		

The ***op_id*** parameter can be one of these values,

Value	Description
BTSDK_AVRCP_OPID_AVC_PANEL_POWER	Power operation.
BTSDK_AVRCP_OPID_AVC_PANEL_VOLUME_UP	Volume Up operation.
BTSDK_AVRCP_OPID_AVC_PANEL_VOLUME_DOWN	Volume Down operation.
BTSDK_AVRCP_OPID_AVC_PANEL_MUTE	Mute operation.
BTSDK_AVRCP_OPID_AVC_PANEL_PLAY	Play operation.
BTSDK_AVRCP_OPID_AVC_PANEL_STOP	Stop operation.
BTSDK_AVRCP_OPID_AVC_PANEL_PAUSE	Pause operation.
BTSDK_AVRCP_OPID_AVC_PANEL_RECORD	Record operation.
BTSDK_AVRCP_OPID_AVC_PANEL_REWIND	Rewind operation.
BTSDK_AVRCP_OPID_AVC_PANEL_FAST_FORWARD	Fast Forward operation.
BTSDK_AVRCP_OPID_AVC_PANEL_EJECT	Reject operation.
BTSDK_AVRCP_OPID_AVC_PANEL_FORWARD	Forward operation.
BTSDK_AVRCP_OPID_AVC_PANEL_BACKWARD	Backward operation.

The ***state_flag*** parameter can be one of these values,

Value	Description
BTSDK_AVRCP_BUTTON_STATE_PRESSED	Button is pressed down.
BTSDK_AVRCP_BUTTON_STATE_RELEASED	Button is released.

Remarks

All operation requests from the remote Controller are transferred to the application using this callback function.

5.9.1.5 Btsdk_AVRCP_RegIndCbk

Prototype	void Btsdk_AVRCP_RegIndCbk (Btsdk_AVRCP_Event_Ind_Func *pfunc);	
Description	The **Btsdk_AVRCP_RegIndCbk** function registers an application-defined callback function used to deal with TG connection events.	
Parameters	*pfunc*	[in] Pointer to the callback function of Btsdk_AVRCP_Event_Ind_Func type. If p*func* is NULL, BTSDK will remove the callback information registered before.
Return:		

Remarks

Only one callback function of Btsdk_AVRCP_Event_Ind_Func type is allowed at a time. That is, if the application calls *Btsdk_AVRCP_RegIndCbk* twice to register different callback functions, the second callback function will replace the first one.

5.9.1.6 Btsdk_AVRCP_Event_Ind_Func

Prototype	typedef void (Btsdk_AVRCP_Event_Ind_Func) (BTUINT16 event, BTUINT8* param,);	
Description	The **Btsdk_AVRCP_Event_Ind_Func** function prototype is the prototype of application defined callback function used to deal with TG connection events.	
Parameters	*event*	[in] Event identifier.
	param	[in] Event specific parameter.
Return:		

The ***event*** parameter can be one of these values,

Value	Description
BTSDK_APP_EV_AVTG_ATTACHPLAYER_IND	A remote Controller connects to the local TG service. The application can now select a media player program to be controlled by the remote Controller. The *param* parameter is ignored.
BTSDK_APP_EV_AVRCP_DETACHPLAYER_IND	The connection from the remote Controller is released. The application can now release the control to the selected media player program. The *param* parameter is ignored.

5.10 Cordless Telephony Profile and Intercom Profile

BTSDK integrates CTP TL and ICP TL functions into one module.

5.10.1 CTP/ICP Terminal (TL)

5.10.1.1 Btsdk_CtpIcpInit

Prototype	BTUINT32 Btsdk_CtpIcpInit (void);	
Description	The **Btsdk_CtpIcpInit** function initializes the resources to run CTP/ICP TL application.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Before calling *Btsdk_CtpIcpInit*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

This function MUST be called and the return value MUST be BTSDK_OK before any other TL functions can be called.

A successful call to *Btsdk_CtpIcpInit* must be balanced by a corresponding call to *Btsdk_CtpIcpDone* after subsequent TL function calls are finished and TL services are no longer required. The application shall not call *Btsdk_CtpIcpInit* once again before it calls *Btsdk_CtpIcpDone*.

5.10.1.2 Btsdk_CtpIcpDone

Prototype	BTUINT32 Btsdk_CtpIcpDone (void);	
Description	The **Btsdk_CtpIcpDone** function releases the context created by *Btsdk_AGAP_Init*.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

An application must call *Btsdk_CtpIcpDone* once for each successful call it has made to *Btsdk_CtpIcpInit*.

This function releases all resources allocated by TL functions and disables TL services finally.

5.11 Personal Information Manager

5.11.1 General

5.11.1.1 PIM_MGR_Init

Prototype	BOOL PIM_MGR_Init(TCHAR *pszPIMWorkingPath, TCHAR *pszPBDBWorkingPath, TCHAR *pszSMSDBWorkingPath);	
Description	The **PIM_MGR_Init** function initializes the resources to run personal information manager.	
Parameters	pszPIMWorkingPath	[in] File path of the PIM resources
	pszPBDBWorkingPath	[in] File path of phonebook got from cell phone
	pszSMSDBWorkingPath	[in] File path of message got from cell phone
Return:	If the function succeeds, the return value is TRUE.. If the function fails, the return value is FALSE.	

Remarks

This function MUST be called and the return value MUST be TRUE before any other functions of PIM can be called.

5.11.1.2 PIM_MGR_Connect

Prototype	INT PIM_MGR_Connect(BTDEVHDL dvhdl, BTCONNHDL connhdl, TCHAR * pszManu, TCHAR * pszModel, INT SvcType);	
Description	The **PIM_MGR_Connect** function establishes a connection to the specified remote cell phone.	
Parameters	dvhdl	[in] Handle to the remote cell phone to connect.
	connhdl	[in] Handle to the remote service record to connect.
	pszManu	[in] Manufacturer of remote cell phone.
	pszModel	[in] Model of remote cell phone
	SvcType	[in] Type of PIM service to connect
Return:	If the function succeeds, the return value is PIM_CONNECT_OK. If the function fails, the return value is an error code.	

The **SvcType** member can be one of these values:

ST_PB	Phonebook only
ST_SMS	Short Message only
ST_PB_SMS	Phonebook and short message both

The **return value** can be one of these values:

PIM_CONNECT_OK	Connect ok
PIM_CONNECT_FAIL	Connect fail
PIM_CONNECT_NEEDPATCH	Should upload a patch and connect again
PIM_CONNECT_NEEDPHONEINFO	Should input the phone information manually and connect again.

Remarks

Before calling PIM_MGR_Connect, the local device resources must be initialized by a previous successful call to *PIM_MGR_Init.*

5.11.1.3 PIM_MGR_UpdatePatch

Prototype	BOOL PIM_MGR_UpdatePatch(void);	
Description	The **PIM_MGR_UpdatePatch** function updates a patch to remote cell phone.	
Parameters		
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

5.11.1.4 PIM_MGR_Disconnect

Prototype	BOOL PIM_MGR_Disconnect (void);	
Description	The **PIM_MGR_Disconnect** function disconnects the current connection.	
Parameters		
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

Remarks

Before setting up another connection, this function must be called to disconnect the current connection first.

5.11.1.5 PIM_MGR_Uninit

Prototype	BOOL PIM_MGR_Uninit (void);	
Description	The **PIM_MGR_Uninit** function releases all resources allocated by *PIM_MGR_Init*.	
Parameters		
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

5.11.1.6 PIM_MGR_GetPhoneList

Prototype	INT PIM_MGR_GetPhoneList(PPHONELIST pPhoneList);	
Description	The **PIM_MGR_GetPhoneList** gets information of all supported cell phones.	
Parameters	pPhoneList	[out] Pointer to the **PHONELIST** structure that receives Phone information from the local database. To determine the required buffer size, call this function with pPhoneList set to NULL. The return value of this function is the required buffer size.
Return:	>=0, the function succeeds, and the return value is the buffer size. <0, the function fails.	

5.11.2 Phonebook

5.11.2.1 PIM_MGR_SyncContacts

Prototype	INT PIM_MGR_SyncContacts (PPBDATA pPBdata);	
Description	The **PIM_MGR_SyncContacts** gets all contacts from the remote cell phone and save these as a file.	
Parameters	pPBdata	[out] Pointer to the **PBDATA** structure that receives contacts from the remote cell phone. To determine the required buffer size, call this function with pPBdata set to NULL. The return value of this function is the required buffer size.
Return:	>=0, the function succeeds, and the return value is the buffer size. <0, the function fails.	

Remarks

Before calling PIM_MGR_SyncContacts, PIM_MGR_Connect MUST be called and the return value MUST be PIM_CONNECT_OK.

5.11.2.2 INT PIM_MGR_GetContacts

Prototype	INT PIM_MGR_Getcontacts (PPBDATA pPBdata);	
Description	The **PIM_MGR_GetContacts** gets all contacts from the local database.	
Parameters	pPBdata	[out] Pointer to the **PBDATA** structure that receives contacts from the local database To determine the required buffer size, call this function with pPBdata set to NULL. The return value of this function is the required buffer size.
Return:	>=0, the function succeeds, and the return value is the buffer size. <0, the function fails.	

5.11.2.3 PIM_MGR_AddContacts

Prototype	BOOL PIM_MGR_AddContacts (int icount, PPBDATA pPBdata,);	
Description	The PIM_MGR_AddContacts adds contacts to local database.	
Parameters	icount	[in]Count of the contacts that add to local database.
	pPBdata	[in] Pointer to the **PBDATA** structure that adds to the local database.
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

5.11.2.4 PIM_MGR_ClearContacts

Prototype	BOOL PIM_MGR_ClearContacts (void);	
Description	The **PIM_MGR_ClearContacts** function deletes all contacts in local database.	
Parameters		
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

5.11.3 Short Message

5.11.3.1 PIM_MGR_SyncSMS

Prototype	INT PIM_MGR_SyncSMS (PSMSDATA pSMSdata);	
Description	The **PIM_MGR_SyncSMS** gets all Messages from the remote cell phone and save these as a file.	
Parameters	pSMSdata	[out] Pointer to the **SMSDATA** structure that receives SMS from the remote cell phone. To determine the required buffer size, call this function with pSMSdata set to NULL. The return value of this function is the required buffer size.
Return:	>=0, the function succeeds, and the return value is the buffer size. <0, the function fails.	

Remarks

Before calling PIM_MGR_SyncSMS, PIM_MGR_Connect MUST be called and the return value MUST be PIM_CONNECT_OK.

5.11.3.2 PIM_MGR_GetSMS

Prototype	INT PIM_MGR_GetSMS (PSMSDATA pSMSdata);	
Description	The **PIM_MGR_GetSMS** gets all contacts from the local database.	
Parameters	pSMSdata	[out] Pointer to the **SMSDATA** structure that receives SMS from the local database To determine the required buffer size, call this function with pSMSdata set to NULL. The return value of this function is the required buffer size.
Return:	>=0, the function succeeds, and the return value is the buffer size. <0, the function fails.	

5.11.3.3 PIM_MGR_AddSMS

Prototype	BOOL PIM_MGR_AddSMS (int icount, PSMSDATA pSMSData,);	
Description	The PIM_MGR_AddContacts adds short messages to local database.	
Parameters	icount	[in]Count of the SMS that add to local database.
	pSMSData	[in] Pointer to the **SMSDATA** structure that adds to the local database.
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

5.11.3.4 PIM_MGR_ClearSMS

Prototype	BOOL PIM_MGR_ClearSMS (void);	
Description	The **PIM_MGR_ClearSMS** function deletes all short messages in local database.	
Parameters		
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

5.11.3.5 PIM_MGR_SendSMS

Prototype	BOOL PIM_MGR_SendSMS (TCHAR *pszNumber, TCHAR *pszSMSbody,);	
Description	The PIM_MGR_SendSMS sends a SMS via the mobile phone, which is currently connected	
Parameters	pszNumber	[in]Phone number of the receiver
	pszSMSbody	[in]Contents of the short message.
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

Remarks

Before calling PIM_MGR_SendSMS, PIM_MGR_Connect MUST be called and the return value MUST be PIM_CONNECT_OK.

5.11.3.6 PIM_MGR_DelSMS

Prototype	BOOL PIM_MGR_DelSMS (PSMSDATA pSMSData);	
Description	The PIM_MGR_DelSMS function deletes an short message from both local device and remote cell phone.	
Parameters	pSMSData	[in] Short message want to delete.
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

Remarks

Before calling PIM_MGR_DelSMS, PIM_MGR_Connect MUST be called and the return value MUST be PIM_CONNECT_OK.

5.11.3.7 PIM_MGR_ClearSMS

Prototype	BOOL PIM_MGR_ClearSMS (void);	
Description	The **PIM_MGR_ClearSMS** function deletes all short messages in local database.	
Parameters		
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

5.11.3.8 PIM_MGR_SetAsyncSMSCB

Prototype	BOOL PIM_MGR_SetAsyncSMSCB (PPIMCB NewAsyncCallBack);	
Description	The **PIM_MGR_SetAsyncSMS** registers an application-defined callback function for asynchronous (new) message.	
Parameters	NewAsyncCallBack	[in] Pointer to the callback function that will be called when new message arrives.
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.	

Remarks

Prototype of the callback function is:

typedef void (***PPIMCB**)(PSMSDATA pSMSData);