IVT BlueSoleil™ Software Development Kit

Developer Guide

Version 2.0.5

IVT Corporation

4/F, Fazhan Plaza,
NO. 12, Xinxi Road,
Haidian District,
Beijing, 100085
P.R. China

Tel: +86 10 82898230
Fax: +86 10 62963059

www.ivtcorporation.com
www.bluesoleil.com

Revision History

Version	Date	Comments
1.0 Release	Jan. 18th, 2008	Initial version.
1.0.1	Jan. 24th, 2008	Added HFP/HSP Audio Gateway APIs and structures.
1.1.0 Alpha	Jan. 31st, 2008	Added Hands-free Unit/Headset APIs and structures.
1.1.1	Mar. 24th, 2008	Added HFP/HSP Audio Gateway sample. Added HFP/HSP Device sample. Added HSP/HFP specific event.
1.1.2	Apr. 2nd, 2008	Added call back functions of pairing and authentication. Added call back relevant macros.
1.1.3	Apr. 7th, 2008	Added pairing APIs and relevant macros.
1.1.4	Apr. 15th, 2008	Added call back functions of pairing and authentication. Added releavant call back events. Added Bluesoleil activating API. Added sample code of pairing and authentication.
1.1.5	May. 14th, 2008	Added Hands-free Unit/Headset APIs about wavein/waveout device configuration.
1.1.6	May. 23rd, 2008	Added Btsdk_AGAP_SetDialHandlerFlag function. Modified the description of the callback event BTSDK_APP_EV_AGAP_HF_LASTNUM_REDIAL_IND.
1.1.7	Jun. 5th, 2008	Added offline activation APIs for BlueSoleil 6.x.
2.0.0	Oct. 15th, 2008	Updated Hands-free/Headset profile from Version 1.1 to Version 1.5. Added APIs about setting and getting fixed pincode of local device. Modified the description of the API: Btsdk_GetAvailableExtSPPCOMPort
2.0.1	Feb. 26th, 2009	Added SPP Profile sample application code.
2.0.2	Apr. 10th, 2009	Added A2DP Profile APIs.
2.0.4	Sept. 22 nd, 2009	Modified the API: Btsdk_GetRemoteRSSI
2.0.5	Nov. 27 th, 2009	Added HID Profile structures and APIs.

Contents

1. Introduction

1.1 Purpose

This document is a developer guide to IVT BlueSoleil™ Software Development Kit, which describes the detailed information of IVT BlueSoleil™ APIs on Microsoft Windows platforms.

1.2 Audience

This document is intended for general VC++ developers involved in the production of VC++ applications for the IVT BlueSoleil™.

1.3 Reference

Reference	Link
Bluetooth Official	http://www.bluetooth.org
IVT BlueSoleil	http://www.ivtcorporation.com http://www.Bluesoleil.com
IVT BlueSoleil SDK Free Download	http://www.bluesoleil.com/products/index.asp?topic=bluesoleilapi

1.4 Abbreviations and Acronyms

Acronyms	Description
ACL	Asynchronous Connectionl-Less
AG	Audio Gateway
API	Application Program Interface
AVRCP	Audio/Video Remote Control Profile
BIP	Basic Imaging Profile
BPP	Basic Printing Profile
CTP	Cordless Telephony Profile
DUN	Dial-up Networking Profile
EC	Echo Canceling
FTP	File Transfer Profile
HSP	Headset Profile
HF	Hands-Free Unit
HFP	Hands-free Profile
HID	Human Interface Device
HS	Head Set
ICP	Intercom Profile
LAP	LAN Access Profile
LMP	Link Management Protocol

NR	Noise Reduction
OBEX	Object Exchange
OPP	Object Push Profile
OS	Operation System
PAN	Personal Area Networking Profile
REF	BIP Referenced Objects
RFCOMM	Radio Frequency Communication Protocol
SA	Service Atrribute
SCO	Synchronous Connection-Oriented
SDAP	Service Discovery Application Profile
SDK	Software Development Kit
SDM	Service Database Management
SDP	Service Discovery Profile
SNK	Audio Sink
SPP	Serial Port Profile
SRC	Audio Source
SS	Service Search
SSA	Service Search Attribute
UUID	Universally Unique Identifier

2. Claim

The contents contained in this developer guide are provided “AS IS”, except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, are made in relation to the accuracy, reliability, or contents of this guide. In no event, shall IVT be liable for direct, indirect, incidental and consequential damages, including but not limited to, losses of profits and/or data, in connection with or rising out of the use of SDK API, even if IVT has been advised of the possibility of such damages.

3. Overview

3.1 Supported Protocols and Profiles

IVT BlueSoleil™ SDK provides Bluetooth application developers with APIs allowing direct access to the protocols and profiles listed as follows:

- GAP
- SDP
- SPP
- OBEX
- FTP
- OPP
- DUN
- FAX
- PAN
- AVRCP
- A2DP
- HFP/HSP
- HID

3.2 SDK Components

The SDK consists of:

- C++ header files
- Library files
- Source and project files for sample applications
- Document of SDK developer guide (this document)
- Document of SDK sample application instruction

3.3 System Requirements

The operational context for the SDK is a standard Bluetooth PC platform on which IVT BlueSoleil is installed.

Recommended Hardware Requirements

- CPU: 600MHz or above
- RAM: 128M or above
- Free hard disk space: At least 20MB

Software Requirements

- OS: Windows2000 / Windows XP / Windows Vista
- IDE: Microsoft Visual C++ 6.0 / Visual Studio 6.0 / Visual Studio .net 2003 / Visual Studio .net 2005

Correlative Requirements

- To use this SDK IVT BlueSoleil version 6.4 or above is required
- Bluetooth radio device (Integrated or Bluetooth Dongle)

4. API Abstract

IVT BlueSoleil™ API is the interface exported by IVT BlueSoleil™. The intention of this SDK is to relieve the Application from managing the Bluetooth related components and make the Application light load. It is used to access the Bluetooth profiles from the application level software. It allows for:

- Standardized access to Bluetooth links.
- Supporting applications that implement different Bluetooth profiles.
- Writing 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 BlueSoleil™ 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.

The general structure of BlueSoleil SDK is shown in **Figure 1**. BlueSoleil SDK is between the Application and profile/stack. It wraps the various APIs of Bluetooth profiles 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.

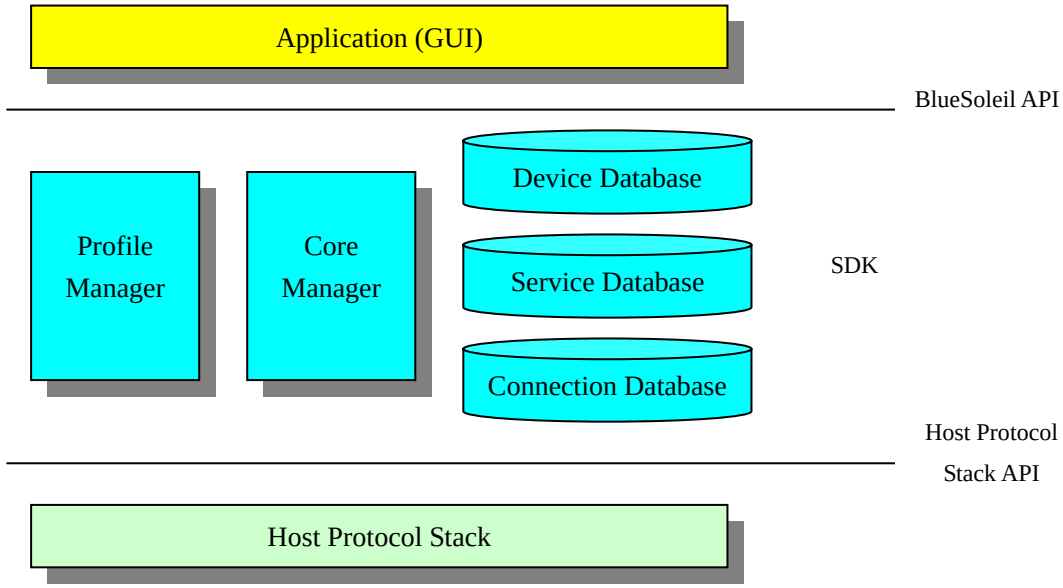

Figure 1: IVT BlueSoleil SDK Structure

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

The SDK 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 using SDK API.

The BlueSoleil SDK APIs are divided into two categories, General and Profile Specific.

The General part interface provides basic Bluetooth functions defined in General Access Profile (GAP) and Service Discovery Application Profile (SDAP). It also provides:

- 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.

These two categories of API are separately introduced in **Chapter 5** and **Chapter 6**.

5. General API Reference

5.1 BlueSoleil Data Types

The data types supported by IVT BlueSoleil are used to define function return values, function and message parameters, and structure members. They define the size and meaning of these elements.

Type	Definition
BTINT8	8-bit ANSI character.
BTUINT8	8-bit unsigned integer.
BTBOOL	Boolean variable (Should be BTSDK_TRUE or BTSDK_FALSE)
BTINT16	16-bit signed integer.
BTUINT16	16-bit unsigned integer.
BTINT32	32-bit signed integer.
BTUINT32	32-bit unsigned integer.
BTLPVOID	Pointer to any type.
BTDEVHDL	Handle to a device object.
BTSVCHDL	Handle to a service object.
BTCONNHDL	Handle to a connection object.
BTSHCHDL	Handle to a shortcut object.
BTSDKHANDLE	Handle to any object.

5.2 Constant Reference

5.2.1 Error Codes

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

Name	Value	Description
BTSDK_OK	0X0000	The operation completed successfully.
BTSDK_ER_SERVER_IS_ACTIVE	0X00C0	Local service is still active. When the application tries to remove or activate an active service, this error code is returned.
BTSDK_ER_NO_SERVICE	0X00C1	No service record with the specified search pattern is found on the remote device.
BTSDK_ER_SERVICE_RECORD_NOT_EXIST	0X00C2	The specified service record does not exist on the remote device.
BTSDK_ER_HANDLE_NOT_EXIST	0X0301	The object specified by the handle does not exist in local BlueSoleil SDK database.
BTSDK_ER_OPERATION_FAILURE	0X0302	The operation fails for an undefined reason.
BTSDK_ER_SDK_UNINIT	0X0303	BlueSoleil SDK has not been initialized.
BTSDK_ER_INVALID_PARAMETER	0X0304	The parameter value is invalid.
BTSDK_ER_NULL_POINTER	0X0305	The pointer value is NULL.
BTSDK_ER_NO_MEMORY	0X0306	Not enough storage is available to process this function.
BTSDK_ER_BUFFER_NOT_ENOUGH	0X0307	The specified buffer size is too small to hold the required information.
BTSDK_ER_FUNCTION_NOTSUPPORT	0X0308	The specified function is not supported by the BlueSoleil.
BTSDK_ER_NO_FIXED_PIN_CODE	0X0309	No fixed PIN code is available.
BTSDK_ER_CONNECTION_EXIST	0X030A	The specified service has been connected already.
BTSDK_ER_OPERATION_CONFLICT	0X030B	The request can’t be processed since a same request is being processed.
BTSDK_ER_NO_MORE_CONNECTION_ALLOWED	0X030C	The limit of connection number is reached.
BTSDK_ER_ITEM_EXIST	0X030D	An object with the specified attribute exists.

BTSDK_ER_ITEM_INUSE	0X030E	The specified object is accessed by other process. It can't be removed or modified.
BTSDK_ER_DEVICE_UNPAIRED	0X030F	The specified remote device is not paired.
BTSDK_ER_UNKNOWN_HCI_COMMAND	0X0401	HCI error "Unknown HCI Command (0X01)" is received.
BTSDK_ER_NO_CONNECTION	0X0402	HCI error "Unknown Connection Identifier (0X02)" is received.
BTSDK_ER_HARDWARE_FAILURE	0X0403	HCI error "Hardware Failure (0X03)" is received.
BTSDK_ER_PAGE_TIMEOUT	0X0404	HCI error "Page Timeout (0X04)" is received.
BTSDK_ER_AUTHENTICATION_FAILURE	0X0405	HCI error "Authentication Failure (0X05)" is received.
BTSDK_ER_KEY_MISSING	0X0406	HCI error "PIN or Key Missing (0X06)" is received.
BTSDK_ER_MEMORY_FULL	0X0407	HCI error "Memory Capacity Exceeded (0X07)" is received.
BTSDK_ER_CONNECTION_TIMEOUT	0X0408	HCI error "Connection Timeout (0X08)" is received.
BTSDK_ER_MAX_NUMBER_OF_CONNECTIONS	0X0409	HCI error "Connection Limit Exceeded (0X09)" is received.
BTSDK_ER_MAX_NUMBER_OF_SCO_CONNECTIONS	0X040A	HCI error "Synchronous Connection Limit to a Device Exceeded (0X0A)" is received.
BTSDK_ER_ACL_CONNECTION_ALREADY_EXISTS	0X040B	HCI error "ACL Connection Already Exists (0X0B)" is received.
BTSDK_ER_COMMAND_DISALLOWED	0X040C	HCI error "Command Disallowed (0X0C)" is received.
BTSDK_ER_HOST_REJECTED_LIMITED_RESOURCES	0X040D	HCI error "Connection Rejected due to Limited Resources (0X0D)" is received.
BTSDK_ER_HOST_REJECTED_SECURITY_REASONS	0X040E	HCI error "Connection Rejected due to Security Reasons (0X0E)" is received.
BTSDK_ER_HOST_REJECTED_PERSONAL_DEVICE	0X040F	HCI error "Connection Rejected due to Unacceptable BD_ADDR (0X0F)" is received.
BTSDK_ER_HOST_TIMEOUT	0X0410	HCI error "Connection Accept Timeout Exceeded (0X10)" is received.
BTSDK_ER_UNSUPPORTED_FEATURE	0X0411	HCI error "Unsupported Feature or

		Parameter Value (0X11)" is received.
BTSDK_ER_INVALID_HCI_COMMAND_PARAMETERS	0X0412	HCI error "Invalid HCI Command parameters (0X12)" is received.
BTSDK_ER_PEER_DISCONNECTION_USER_END	0X0413	HCI error "Remote User Terminated Connection (0X13)" is received.
BTSDK_ER_PEER_DISCONNECTION_LOW_RESOURCES	0X0414	HCI error "Remote Device Terminated Connection due to Low Resources (0X14)" is received.
BTSDK_ER_PEER_DISCONNECTION_TO_POWER_OFF	0X0415	HCI error "Remote Device Terminated Connection due to Power Off (0X15)" is received.
BTSDK_ER_LOCAL_DISCONNECTION	0X0416	HCI error "Connection Terminated by Local Host (0X16)" is received.
BTSDK_ER_REPEATED_ATTEMPTS	0X0417	HCI error "Repeated Attempts (0X17)" is received.
BTSDK_ER_PAIRING_NOT_ALLOWED	0X0418	HCI error "Pairing Not Allowed (0X18)" is received.
BTSDK_ER_UNKNOWN_LMP_PDU	0X0419	HCI error "Unknown LMP PDU (0X19)" is received.
BTSDK_ER_UNSUPPORTED_REMOTE_FEATURE	0X041A	HCI error "Unsupported Remote Feature / Unsupported LMP Feature (0X1A)" is received.
BTSDK_ER_SCO_OFFSET_REJECTED	0X041B	HCI error "SCO Offset Rejected (0X1B)" is received.
BTSDK_ER_SCO_INTERVAL_REJECTED	0X041C	HCI error "SCO Interval Rejected (0X1C)" is received.
BTSDK_ER_SCO_AIR_MODE_REJECTED	0X041D	HCI error "SCO Air Mode Rejected (0X1D)" is received.
BTSDK_ER_INVALID_LMP_PARAMETERS	0X041E	HCI error "Invalid LMP Parameters (0X1E)" is received.
BTSDK_ER_UNSPECIFIED_ERROR	0X041F	HCI error "Unspecified Error (0X1F)" is received.
BTSDK_ER_UNSUPPORTED_LMP_PARAMETER_VALUE	0X0420	HCI error "Unsupported LMP Parameter Value (0X20)" is received.
BTSDK_ER_ROLE_CHANGE_NOT_ALLOWED	0X0421	HCI error "Role Change Not Allowed (0X21)" is received.
BTSDK_ER_LMP_RESPONSE_TIMEOUT	0X0422	HCI error "LMP Response Timeout (0X22)" is received.
BTSDK_ER_LMP_ERROR_TRANSACTION_COLLISION	0X0423	HCI error "LMP Error Transaction Collision (0X23)" is received.
BTSDK_ER_LMP_PDU_NOT_ALLOWED	0X0424	HCI error "LMP PDU Not Allowed (0X24)" is received.
BTSDK_ER_ENCRYPTION_MODE_NOT_ACCEPTABLE	0X0425	HCI error "Encryption Mode Not Acceptable (0X25)" is received.

BTSDK_ER_UNIT_KEY_USED	0X0426	HCI error “Link Key Can not be Changed (0X26)” is received.
BTSDK_ER_QOS_IS_NOT_SUPPORTED	0X0427	HCI error “Requested QOS Not Supported (0X27)” is received.
BTSDK_ER_INSTANT_PASSED	0X0428	HCI error “Instant Passed (0X28)” is received.
BTSDK_ER_PAIRING_WITH_UNIT_KEY_NOT_SUPPORTED	0X0429	HCI error “Pairing with Unit Key Not Supported (0X29)” is received.
BTSDK_ER_DIFFERENT_TRANSACTION_COLLISION	0X042A	HCI error “Different Transaction Collision (0X2A)” is received.
BTSDK_ER_QOS_UNACCEPTABLE_PARAMETER	0X042C	HCI error “QOS Unacceptable Parameter (0X2C)” is received.
BTSDK_ER_QOS_REJECTED	0X042D	HCI error “QOS Rejected (0X2D)” is received.
BTSDK_ER_CHANNEL_CLASS_NOT_SUPPORTED	0X042E	HCI error “Channel Classification Not Supported (0X2E)” is received.
BTSDK_ER_INSUFFICIENT_SECURITY	0X042F	HCI error “Insufficient Security (0X2F)” is received.
BTSDK_ER_PARAMETER_OUT_OF_RANGE	0X0430	HCI error “Parameter Out of Mandatory Range (0X30)” is received.
BTSDK_ER_ROLE_SWITCH_PENDING	0X0432	HCI error “Role Switch Pending (0X32)” is received.
BTSDK_ER_RESERVED_SLOT_VIOLATION	0X0434	HCI error “Reserved Slot Violation (0X34)” is received.
BTSDK_ER_ROLE_SWITCH_FAILED	0X0435	HCI error “Role Switch Failed (0X35)” is received.

Table 1: BlueSoleil Error Codes.

5.2.2 Service Class Identifier

The following table provides a list of class identifiers of services supported by current version BlueSoleil. These service class identifiers are specified as 16-bit UUID. These values will be used when the service class is required as a parameter.

Name	UUID	Description
BTSDK_CLS_SERIAL_PORT	0X1101	Serial Port service.
BTSDK_CLS_LAN_ACCESS	0X1102	LAN Access service.
BTSDK_CLS_DIALUP_NET	0X1103	Dial-up Networking service.
BTSDK_CLS_IRMC_SYNC	0X1104	Synchronization service.
BTSDK_CLS_OBEX_OBJ_PUSH	0X1105	Object Push service.
BTSDK_CLS_OBEX_FILE_TRANS	0X1106	File Transfer service.
BTSDK_CLS_IRMC_SYNC_CMD	0X1107	IrMC Sync Command service.
BTSDK_CLS_HEADSET	0X1108	Headset service.
BTSDK_CLS_CORDLESS_TELE	0X1109	Cordless Telephony service.
BTSDK_CLS_AUDIO_SOURCE	0X110A	Audio Source service.
BTSDK_CLS_AUDIO_SINK	0X110B	Audio Sink service.
BTSDK_CLS_AVRCP_TG	0X110C	A/V Remote Control Target service.
BTSDK_CLS_ADV_AUDIO_DISTRIB	0X110D	Advanced Audio Distribution service.
BTSDK_CLS_AVRCP_CT	0X110E	A/V Remote Control service.
BTSDK_CLS_VIDEO_CONFERENCE	0X110F	Video conference service.
BTSDK_CLS_INTERCOM	0X1110	Intercom service.
BTSDK_CLS_FAX	0X1111	Fax service.
BTSDK_CLS_HEADSET_AG	0X1112	Headset Audio Gateway service.
BTSDK_CLS_WAP	0X1113	WAP service.
BTSDK_CLS_WAP_CLIENT	0X1114	WAP client service.
BTSDK_CLS_PAN_PANU	0X1115	PANU service.
BTSDK_CLS_PAN_NAP	0X1116	NAP service.
BTSDK_CLS_PAN_GN	0X1117	GN service.
BTSDK_CLS_DIRECT_PRINT	0X1118	Direct Print service.
BTSDK_CLS_REF_PRINT	0X1119	Referenced Print service.
BTSDK_CLS_IMAGING	0X111A	Imaging service.
BTSDK_CLS_IMAG_RESPONDER	0X111B	Imaging Responder service.
BTSDK_CLS_IMAG_AUTO_ARCH	0X111C	Imaging Automatic Archive service.
BTSDK_CLS_IMAG_REF_OBJ	0X111D	Imaging Referenced Objects service.
BTSDK_CLS_HANDSFREE	0X111E	Hands-free service.
BTSDK_CLS_HANDSFREE_AG	0X111F	Hands-free Audio Gateway service.
BTSDK_CLS_DPS_REF_OBJ	0X1120	DPS Referenced Objects service.
BTSDK_CLS_REFLECTED_UI	0X1121	Reflected UI service
BTSDK_CLS_BASIC_PRINT	0X1122	Basic Print service.
BTSDK_CLS_PRINT_STATUS	0X1123	Print Status service.

BTSDK_CLS_HID	0X1124	Human Interface Device service.
BTSDK_CLS_HCRP	0X1125	Hardcopy Cable Replacement service.
BTSDK_CLS_HCR_PRINT	0X1126	HCRP Print service.
BTSDK_CLS_HCR_SCAN	0X1127	HCRP Scan service.
BTSDK_CLS_SIM_ACCESS	0X112D	SIM Card Access service
BTSDK_CLS_PBAP_PCE	0X112E	PBAP Phonebook Client Equipment service.
BTSDK_CLS_PBAP_PSE	0X112F	PBAP Phonebook Server Equipment service.
BTSDK_CLS_PHONEBOOK_ACCESS	0X1130	Phonebook Access service.
BTSDK_CLS_PNP_INFO	0X1200	Bluetooth Device Identification.

Table 2: IVT BlueSoleil Service Class Identifiers.

5.2.3 Class of Device/Service Field

The following table provides a list of device class identifiers categorized by major device class. These device class identifiers are mapped to the device class field of the Class of Device/Service field (first format type).

Name	Value	Description
DEVICE_CLASS_MASK	0x1FFC	Mask of device class
BTSDK_DEVCLS_COMPUTER	0x000100	Computer major device class.
BTSDK_COMPCLS_UNCLASSIFIED	0x000100	Uncategorized computer, code for device not assigned.
BTSDK_COMPCLS_DESKTOP	0X000104	Desktop workstation.
BTSDK_COMPCLS_SERVER	0X000108	Server-class computer.
BTSDK_COMPCLS_LAPTOP	0X00010C	Laptop computer.
BTSDK_COMPCLS_HANDHELD	0X000110	Handheld PC/PDA (clam shell).
BTSDK_COMPCLS_PALMSIZED	0X000114	Palm sized PC/PDA.
BTSDK_COMPCLS_WEARABLE	0X000118	Wearable computer (Watch sized).
BTSDK_DEVCLS_PHONE	0X000200	Phone major device class.
BTSDK_PHONECLS_UNCLASSIFIED	0X000200	Uncategorized phone, code for device not assigned.
BTSDK_PHONECLS_CELLULAR	0X000204	Cellular phone.
BTSDK_PHONECLS_CORDLESS	0X000208	Cordless phone.
BTSDK_PHONECLS_SMARTPHONE	0X00020C	Smart phone.
BTSDK_PHONECLS_WIREDMODEM	0X000210	Wired modem or voice gateway.
BTSDK_PHONECLS_COMMONISDNACCESS	0X000214	Common ISDN Access.
BTSDK_PHONECLS_SIMCARDREADER	0X000218	SIM card reader
BTSDK_DEVCLS_LAP	0X000300	LAN / Network Access Point major device class.

BTSDK_LAP_FULLY	0X000300	Fully available.
BTSDK_LAP_17	0X000320	1 - 17% utilized.
BTSDK_LAP_33	0X000340	17- 33% utilized.
BTSDK_LAP_50	0X000360	33 - 50% utilized.
BTSDK_LAP_67	0X000380	50 - 67% utilized.
BTSDK_LAP_83	0X0003A0	67 - 83% utilized.
BTSDK_LAP_99	0X0003C0	83 – 99% utilized.
BTSDK_LAP_NOSRV	0X0003E0	No service available.
BTSDK_DEVCLS_AUDIO	0X000400	Audio/Video major device class.
BTSDK_AV_UNCLASSIFIED	0X000400	Uncategorized A/V device, code for device not assigned.
BTSDK_AV_HEADSET	0X000404	Wearable headset device.
BTSDK_AV_HANDSFREE	0X000408	Hands-free device.
BTSDK_AV_MICROPHONE	0X000410	Microphone.
BTSDK_AV_LOUDSPEAKER	0X000414	Loudspeaker.
BTSDK_AV_HEADPHONES	0X000418	Headphones.
BTSDK_AV_PORTABLEAUDIO	0X00041C	Portable Audio.
BTSDK_AV_CARAUDIO	0X000420	Car Audio.
BTSDK_AV_SETTOPBOX	0X000424	Set-top box.
BTSDK_AV_HIFIAUDIO	0X000428	HiFi Audio device.
BTSDK_AV_VCR	0X00042C	Videocassette recorder
BTSDK_AV_VIDEOCAMERA	0X000430	Video camera
BTSDK_AV_CAMCORDER	0X000434	Camcorder
BTSDK_AV_VIDEOMONITOR	0X000438	Video monitor.
BTSDK_AV_VIDEODISPANDLOUDSPK	0X00043C	Video display and loudspeaker.
BTSDK_AV_VIDEOCONFERENCE	0X000440	Video conferencing.
BTSDK_AV_GAMEORTOY	0X000448	Gaming/Toy
BTSDK_DEVCLS_PERIPHERAL	0X000500	Peripheral major device class
BTSDK_PERIPHERAL_UNCLASSIFIED	0X000500	Uncategorized peripheral device, code for device not assigned.
BTSDK_PERIPHERAL_KEYBOARD	0X000540	Keyboard.
BTSDK_PERIPHERAL_POINT	0X000580	Pointing device.
BTSDK_PERIPHERAL_KEYORPOINT	0X0005C0	Combo keyboard/pointing device.
BTSDK_DEVCLS_IMAGE	0X000600	Imaging major device class.
BTSDK_IMAGE_DISPLAY	0X000610	Display.
BTSDK_IMAGE_CAMERA	0X000620	Camera.
BTSDK_IMAGE_SCANNER	0X000640	Scanner.
BTSDK_IMAGE_PRINTER	0X000680	Printer.
BTSDK_DEVCLS_WEARABLE	0x000700	Wearable major device class.
BTSDK_WERABLE_WATCH	0x000704	Wristwatch.
BTSDK_WERABLE_PAGER	0x000708	Pager.
BTSDK_WERABLE_JACKET	0x00070C	Jacket

BTSDK_WERABLE_HELMET	0x000710	Helmet.
BTSDK_WERABLE_GLASSES	0x000714	Glasses.

Table 3: BlueSoleil Device Class Filed Identifiers

The following table provides a list of major service class identifiers that are mapped to the service class field of the Class of Device/Service field (first format type).

Name	Value	Description
BTSDK_SRVCLS_LDM	0x002000	Limited Discoveralbe Mode
BTSDK_SRVCLS_POSITION	0x010000	Positioning (Location Identification).
BTSDK_SRVCLS_NETWORK	0x020000	Networking (LAN, AD hoc, …).
BTSDK_SRVCLS_RENDER	0x040000	Rendering (Printing, Speaker, …).
BTSDK_SRVCLS_CAPTURE	0x080000	Capturing (Scanner, Microphone, …).
BTSDK_SRVCLS_OBJECT	0x100000	Object Transfer (v-Inbox, v-Folder, …).
BTSDK_SRVCLS_AUDIO	0x200000	Audio (Speaker, Microphone, Headset service, …).
BTSDK_SRVCLS_TELEPHONE	0x400000	Telephony (Cordless telephony, Modem, Headset service, …).
BTSDK_SRVCLS_INFOR	0x800000	Information (WEB-server, WAP-server, …).

Table 4: IVT BlueSoleil Major Service Class Identifiers

A complete Class of Device/Service field (first format type) can be the combination of one device class identifier and multiple major service class identifiers.

5.2.4 Bluetooth Device Modes

The following table provides a list of flags that specify the Bluetooth device modes.

Name	Description
BTSDK_GENERAL_DISCOVERABLE	Sets the device into general discoverable mode. This is the default discoverable mode.
BTSDK_LIMITED_DISCOVERABLE	Sets the device into limited discoverable mode. If this value is specified, BTSDK_GENERAL_DISCOVERABLE mode value is ignored by BlueSoleil.
BTSDK_DISCOVERABLE	Makes the device discoverable. This is equivalent to BTSDK_GENERAL_DISCOVERABLE.
BTSDK_CONNECTABLE	Makes the device connectable. This is the default connectable mode.
BTSDK_PAIRABLE	Makes the device pairable. This is the default pairable mode.

Table 5: Bluetooth Device Modes

5.2.5 Messages from BlueSoleil to the Application

The following table provides a list of messages transferred from BlueSoleil to the application and the type of the callback functions to process these messages.

Message Name	Callback Function Type	Description
BTSDK_INQUIRY_RESULT_IND	Btsdk_Inquiry_Result_Ind_Func	This message indicates that a Bluetooth device has responded so far during the current inquiry process.
BTSDK_INQUIRY_COMPLETE_IND	Btsdk_Inquiry_Complete_Ind_Func	This message indicates that the inquiry is finished.
BTSDK_CONNECTION_EVENT_IND	Btsdk_Connection_Event_Ind_Func	This message indicates that a high-level protocol connection is created or disconnected.
BTSDK_PIN_CODE_IND	Btsdk_UserHandle_Pin_Req_Ind_Func	This message indicates the application to input PIN code for the specified device.
BTSDK_AUTHORIZATION_IND	Btsdk_UserHandle_Authorization_Req_Ind_Func	This message indicates that a remote device is trying to access a local service.
BTSDK_LINK_KEY_NOTIF_IND	Btsdk_Link_Key_Notif_Ind_Func	This message indicates that a new link key has been created for the specified device.
BTSDK_AUTHENTICATION_FAIL_IND	Btsdk_Authentication_Fail_Ind_Func	This message indicates that an error occurs when performing authentication with the specified device.

Table 6: Messages from BlueSoleil to the Application

5.3 Data Structures

BtSdkCallbackStru

Definition	typedef struct _BtSdkCallBackStru { BTUINT16 type; PVOID func } BtSdkCallbackStru, *PBtSdkCallbackStru;	
Description	The structure BtSdkCallbackStru contains information about a callback function.	
Members	*Type*	Specifies the message of the callback function to process. It also specifies the prototype of the callback function. It can be one of the values listed in Table 6.
	Func	Pointer to the callback function. If *func* is NULL, BlueSoleil will remove the callback.

Remarks

Detail about each callback function is discussed in the following section.

BtSdkLocalLMPInfoStru

Definition	typedef struct _BtSdkLocalLMPInfoStru { BTUINT8 lmp_feature[8]; BTUINT16 manuf_name; BTUINT16 lmp_subversion; BTUINT8 lmp_version; BTUINT8 hci_version; BTUINT16 hci_revision; BTUINT8 country_code; } BtSdkLocalLMPInfoStru, *PBtSdkLocalLMPInfoStru;	
Description	The structure BtSdkLocalLMPInfoStru contains information about local host controller.	
Members	*lmp_feature*	List of supported features for the local device.
	manuf_name	Integer specifies the manufacturer of the local device.
	lmp_subversion	Subversion of the current LMP in the local device.
	lmp_version	Version of the current LMP in the local device.
	hci_version	Version of the current HCI in the local device.
	hci_revision	Revision of the current HCI in the local device.
	Country_code	Integer defines which range of frequency band of the ISM 2.4GHz band is used by the local device. This member is for backwards compatibility with a prior version HCI (1.1 and 1.0A).

BtSdkVendorCmdStru

Definition	typedef struct _BtSdkVendorCmdStru { BTUINT16 ocf; BTUINT8 param_len; BTUINT8 param[1]; } BtSdkVendorCmdStru, *PBtSdkVendorCmdStru;	
Description	The structure BtSdkVendorCmdStru contains information about a vendor specific command.	
Members	Ocf	Specifies the OpCode Command Field value of this vendor specific command.
	param_len	Specifies the size in bytes of the content in the buffer pointer by the param element.
	Param	Pointer to the buffer containing the command parameters.

Remarks

The *param* element of this structure is a variable length array of octets. Contents in the buffer pointed to by the *param* element are copied to the final HCI command packet's parameter field directly. The core Bluetooth stack determines the number of octets to be copied by examining the value of the *param_len* element. The application must ensure the correctness and integrity of the parameters.

Example

```
/* This sample demonstrates how to set BtSdkVendorCmdStru for the vendor command:
        {0xFC, 0x01, 0x04, 0x00, 0x10, 0x3A, 0x33}. */
void AppVendorCommand (void)
{
        BTUINT8 param[] = {0x00, 0x10, 0x3A, 0x33};
        PBtSdkVendorCmdStru pCmd = (PBtSdkVendorCmdStru)malloc(szieof(BtSdkVendorCmdStru)+sizeof(param));
        pCmd->ocf = 0x01;
        pCmd->param_len = sizeof(param);
        memcpy(pCmd->param, param, pCmd->param_len);
        /* To Do: Processing the command. */
        free(pCmd);
```

```
}
```

BtSdkEventParamStru

Definition	typedef struct _BtSdkEventParamStru { BTUINT8 ev_code; BTUINT8 param_len; BTUINT8 param[1]; } BtSdkEventParamStru, *PBtSdkEventParamStru;	
Description	The structure BtSdkEventParamStru contains information about a HCI event.	
Members	ev_code	Specifies the event code.
	param_len	On input, specifies the size in bytes of the param buffer. On output, receives the number of bytes required to receive the event parameters.
	Param	Pointer to the buffer receiving the raw event parameters copied from the HCI event packet's parameter field.

Remarks

BtSdkEventParamStru structure is usually used to receive the HCI event generated for a specific HCI command. The *param* element of this structure is a variable length array of octets. Contents in the buffer pointed to by the *param* element are copied from the HCI event packet's parameter field directly. The core Bluetooth stack determines the number of octets to be copied by examining the value of the *param_len* element and the actual size of the event parameter list.

The application shall allocate a buffer large enough to hold all the event parameters. Generally, if the buffer size specified by the *param_len* element is smaller than the number of bytes required, the BlueSoleil function call returns BTSDK_ER_BUFFER_NOT_ENOUGH and *param_len* is set to the actual size required by BlueSoleil.

A buffer of 257 bytes, which is the maximum length of an event packet, is suggested if the user doesn't know the actual size of the event parameter list.

Example

```
/* This sample demonstrates how to send a vendor specific command {0x01, 0xFC, 0x04, 0x00, 0x10, 0x3A, 0x33}
```

```
        and receive the created event {0x0E, 0x04, 0x01, 0x01, 0xFC, 0x02}.
        Command and event packet in this sample are used only for demonstration. Do NOT execute this sample function
on
        your platform unless you are sure they are really exported by the Bluetooth device you used.
*/
void AppVendorCommand (void)
{
        BTUINT8 param[] = {0x00, 0x10, 0x3A, 0x33};
        PBtSdkVendorCmdStru pCmd = (PBtSdkVendorCmdStru)malloc(szieof(BtSdkVendorCmdStru)+sizeof(param));
        PBtSdkEventParamStru pEv = (PBtSdkEventParamStru)malloc(257);

        pCmd->ocf = 0x01;
        pCmd->param_len = sizeof(param);
        memcpy(pCmd->param, param, pCmd->param_len);
        memset(pEv, 0, 257);
        pEv->param_len = 255;
        Btsdk_VendorCommand(0, pCmd, pEv);
        /* If the command is executed successfully, we shall find that:
                pEv->ev_code = 0x0E;       pEv->param_len = 0x04;
                pEv->param[0] ＝ 0x01; pEv->param[1] = 0x01; pEv->param[2] = 0xFC; pEv->param[3] = 0x02;
        */
        free(pCmd);
        free(pEv);
}
```

BtSdkRemoteLMPInfoStru

Definition	typedef struct _BtSdkRemoteLMPInfoStru { BTUINT8 lmp_feature[8]; BTUINT16 manuf_name; BTUINT16 lmp_subversion; BTUINT8 lmp_version; } BtSdkRemoteLMPInfoStru, *PBtSdkRemoteLMPInfoStru;	
Description	The structure BtSdkRemoteLMPInfoStru contains information about remote host controller.	
Members	*lmp_feature*	List of supported features for the remote device.
	manuf_name	Integer specifies the manufacturer of the local device.
	lmp_subversion	Subversion of the current LMP in the remote device.
	lmp_version	Version of the current LMP in the remote device.

BtSdkRemoteDevicePropertyStru

Definition	typedef struct _BtSdkRemoteDevicePropertyStru { BTUINT32 mask; BTDEVHDL dev_hdl; BTUINT8 bd_addr[BTSDK_BDADDR_LEN]; BTUINT8 name[BTSDK_DEVNAME_LEN]; BTUINT32 dev_class; BtSdkRemoteLMPInfoStru lmp_info; BTUINT8 link_key[BTSDK_LINKKEY_LEN]; } BtSdkRemoteDevicePropertyStru, *PBtSdkRemoteDevicePropertyStru;	
Description	The structure BtSdkRemoteDevicePropertyStru contains information about a remote device.	
Members	*mask*	Specifies which member is available.
	dev_hdl	Handle assigned to this device record.
	bd_addr	Bluetooth device address of this device record.
	name	User-friendly name of this device record. This string is coded in UTF-8 format.
	dev_class	The Class of Device/Service setting of this device record. It can be one of the device class identifiers listed in Table 3 combined with multiple major service class identifiers listed in Table 4.
	lmp_info	Information about the host controller of this device.
	link_key	Link key for this device.

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

Value	Description
BTSDK_RDPM_HANDLE	The value of the *dev_hdl* member is available.
BTSDK_RDPM_ADDRESS	The value of the *bd_addr* member is available.
BTSDK_RDPM_NAME	The value of the *name* member is available.
BTSDK_RDPM_CLASS	The value of the *dev_class* is available.
BTSDK_RDPM_LMPINFO	The value of the *lmp_info* is available.
BTSDK_RDPM_LINKKEY	The value of the *link_key* is available.

BtSdkHoldModeStru

Definition	typedef struct _BtSdkHoldModeStru { BTUINT16 conn_hdl; BTUINT16 max; BTUINT16 min; } BtSdkHoldModeStru, *PBtSdkHoldModeStru;	
Description	The structure BtSdkHoldModeStru contains hold mode parameters.	
Members	*conn_hdl*	Reserved for future extension. Set it to 0.
	max	Specifies the maximum acceptable number of Baseband slots (0.625msec) to wait in the Hold mode. Range: 0x0002 to 0xFFFE; only even values are valid.
	min	Specifies the minimum acceptable number of Baseband slots (0.625msec) to wait in the Hold mode. Range: 0x0002 to 0xFF00; only even values are valid.

BtSdkSniffModeStru

Definition	typedef struct _BtSdkSniffModeStru { BTUINT16 conn_hdl; BTUINT16 max; BTUINT16 min; BTUINT16 attempt; BTUINT16 timeout; } BtSdkSniffModeStru, *PBtSdkSniffModeStru;	
Description	The structure BtSdkSniffModeStru contains sniff mode parameters.	
Members	*conn_hdl*	Reserved for future extension. Set it to 0.
	max	Specifies the maximum acceptable periods, in number of Baseband slots (0.625msec), in the Sniff mode. Range: 0x0002 to 0xFFFE; only even values are valid.
	min	Specifies the minimum acceptable periods, in number of Baseband slots (0.625msec), in the Sniff mode. Range: 0x0002 to 0xFFFE; only even values are valid.
	attempt	Specifies the number of Baseband receive slots (0.625msec) for sniff attempt. Range: 0x0001 to 0x7FFF.
	timeout	Specifies the number of Baseband receive slots (0.625msec) for sniff timeout. Range: 0x0000 to 0x7FFF.

BtSdkParkModeStru

Definition	typedef struct _BtSdkParkModeStru { BTUINT16 conn_hdl; BTUINT16 max; BTUINT16 min; } BtSdkParkModeStru, *PBtSdkParkModeStru;	
Description	The structure BtSdkParkModeStru contains park mode parameters.	
Members	*conn_hdl*	Reserved for future extension. Set it to 0.
	max	Specifies the acceptable longest length of the interval, in number of Baseband slots (0.625msec), between beacons in the Park mode. Range: 0x000E to 0xFFFE; only even values are valid.
	min	Specifies the acceptable shortest length of the interval, in number of Baseband slots (0.625msec), between beacons in the Park mode. Range: 0x000E to 0xFFFE; only even values are valid.

BtSdkUUIDStru

Definition	typedef struct _BtSdkUUIDStru { BTUINT32 Data1; BTUINT16 Data2; BTUINT16 Data3; BTUINT8 Data4[8]; } BtSdkUUIDStru, *PBtSdkUUIDStru;	
Description	The structure BtSdkUUIDStru defines Universally Unique Identifier (UUID). UUID provides unique designations of service class.	
Members	*Data1*	Specifies the first 8 hexadecimal digits of the UUID.
	Data2	Specifies the first group of 4 hexadecimal digits of the UUID.
	Data3	Specifies the second group of 4 hexadecimal digits of the UUID.
	Data4	Specifies an array of eight elements. The first two elements contain the third group of 4 hexadecimal digits of the UUID. The remaining six elements contain the final 12 hexadecimal digits of the UUID.

Example

```
/*UUID value 0x00001234-0000-1000-8000-00805F9B34FB */
BtSdkUUIDStru uuid128 = {
                                                    0x00001234,
                                                    0x0000,
                                                    0x1000,
                                                    {0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B,
0x34, 0xFB}
                                                };      /* Use BtSdkUUIDStru to represent a 128bit UUID
*/
```

BtSdkSDPSearchPatternStru

Definition	typedef struct _BtSdkSDPSearchPatternStru { BTUINT32 mask; BtSdkUUIDStru uuid; } BtSdkSDPSearchPatternStru, *PBtSdkSDPSearchPatternStru;	
Description	The structure BtSdkSDPSearchPatternStru contains information about a SDP search pattern.	
Members	*mask*	A set of flags which specify the valid bytes of the *uuid* member.
	uuid	A BtSdkUUIDStru type variable specifies the search pattern. A search pattern can be a 16bit, 32bit or 128bit UUID value according to the *mask* value.

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

Value	Description
BTSDK_SSPM_UUID16	The *uuid* member specifies a 16bit UUID value. That is, *uuid.Data1* contains the 16bit UUID value.
BTSDK_SSPM_UUID32	The *uuid* member specifies a 32bit UUID value. That is, *uuid.Data1* contains the 32bit UUID value.
BTSDK_SSPM_UUID128	The *uuid* member specifies a 128bit UUID value.

Example

```
/*Search pattern with UUID values 0x1002, 0x00112233 and 0x00001234-0000-1000-8000-00805F9B34FB */
BtSdkSDPSearchPatternStru ptn16 = {0}, ptn32 = {0}, ptn128 = {0};
BtSdkUUIDStru uuid128 = {
                                                    0x00001234,
                                                    0x0000,
                                                    0x1000,
                                                    {0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
                                                };          /* Use BtSdkUUIDStru to represent a 128bit UUID */
Ptn16.mask = BTSDK_SSPM_UUID16;
Ptn16.uuid.Data1 = 0x1002;
Ptn32.mask = BTSDK_SSPM_UUID32;
Ptn32.Data1 = 0x00112233;
Ptn128.mask = BTSDK_SSPM_UUID128;
```

```
memcpy(&ptn128.uuid, &uuid128, sizeof(BtSdkUUIDStru uuid128));
```

BtSdkRemoteServiceAttrStru

Definition	typedef struct _BtSdkRemoteServiceAttrStru { BTUINT32 mask; BTUINT16 service_class; BTDEVHDL dev_hdl; BTUINT8 svc_name[BTSDK_SERVICENAME_MAXLENGTH]; BTLPVOID ext_attributes; BTUINT16 status; } BtSdkRemoteServiceAttrStru, *PBtSdkRemoteServiceAttrStru;	
Description	The structure BtSdkRemoteServiceAttrStru contains information about a remote service record.	
Members	*mask*	A set of flags which specify members to retrieve.
	service_class	Type of the service record. It can be one of the values listed in the Table 2.
	dev_hdl	Handle to the remote device that exports this service record.
	svc_name	User-friendly name of this service record. This string is coded in UTF-8 format. Set *mask* to BTSDK_RSAM_SERVICENAME to use *svc_name*.
	ext_attributes	Profile specific attributes. It must be cast to a pointer to a structure decided by the service type. See following table. Set *mask* to BTSDK_RSAM_EXTATTRIBUTES to use *ext_attributes*. Always set it to NULL when input.
	status	Current status of this service record.

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

Value	Description
BTSDK_RSAM_SERVICENAME	Retrieves the *svc_name* member.
BTSDK_RSAM_EXTATTRIBUTES	Retrieves the *ext_attributes* member.

The ***ext_attributes*** member can be a pointer to one of these structures.

Value of ***service_class***	Type of ***ext_attributes***
BTSDK_CLS_SERIAL_PORT	PBtSdkRmtSPPSvcExtAttrStru
BTSDK_CLS_HID	PBtSdkRmtHIDSvcExtAttrStru
BTSDK_CLS_PNP_INFO	PBtSdkRmtDISvcExtAttrStru

Detail of these structures is specified in separate profile API documents.

The *ext_attributes* member is ignored and is set to NULL for profiles not listed in the upper table.

BtSdkRmtSPPSvcExtAttrStru

Definition	typedef struct _BtSdkRmtSPPSvcExtAttrStru { BTUINT32 size; BTUINT8 server_channel; } BtSdkRmtSPPSvcExtAttrStru, *PBtSdkRmtSPPSvcExtAttrStru;	
Description	The structure **BtSdkRmtSPPSvcExtAttrStru** describes the server_chanel of remote 128bit SPP service.	
Members	*size*	Size of the structure, in bytes.
	server_channel	Server channel value of this SPP service record.

BtSdkConnectionPropertyStru

Definition	typedef struct _BtSdkConnectionPropertyStru { BTUINT32 role : 2; BTUINT32 result : 30; BTDEVHDL device_handle; BTSVCHDL service_handle; BTUINT16 service_class; BTUINT32 duration; BTUINT32 received_bytes; BTUINT32 sent_bytes; } BtSdkConnectionPropertyStru, *PBtSdkConnectionPropertyStru;	
Description	The structure BtSdkConnectionPropertyStru contains information about a high-level protocol connection.	
Members	*role*	Specifies the role that local BlueSoleil SDK performs in the connection. See following table.
	result	Result of the connecting procedure. It can be one of the values listed in the Table 1.
	device_handle	Handle to the remote device that is the peer side of this connection.
	service_handle	If the *role* is BTSDK_CONNROLE_INITIATOR, it specifies the handle to the remote service record that local device connects to. If the *role* is BTSDK_CONNROLE_ACCEPTOR, it specifies the local service record that the remote device connects to.
	service_class	Type of the service record specified by the *service_handle*. It can be one of the values listed in the Table 2.
	duration	Specifies the time in seconds elapsed since the connection is created.
	received_bytes	Specifies the number of bytes received on this connection since the connection is created.
	sent_bytes	Specifies the number of bytes sent on this connection since the connection is created.

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

Value	Description
BTSDK_CONNROLE_INITIATOR	The local BlueSoleil SDK initiates the connection to the remote service.
BTSDK_CONNROLE_ACCEPTOR	The remote device initiates the connection to a local service.

5.4 API Functions

5.4.1 Initialization/Termination

Btsdk_Init

Prototype	void Btsdk_Init (void);	
Description	The **Btsdk_Init** function initializes context for subsequent BTSDK function calls.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

This function MUST be called and the return value MUST be BTSDK_OK before any other functions (except for *Btsdk_IsSDKInitialized*, and *Btsdk_IsBluetoothReady*) can be called.

This function initializes resources required to run the BlueSoleil. But it DOES NOT enable Bluetooth device. Function *Btsdk_StartBluetooth* must be called to enable Bluetooth device after initializing BlueSoleil successfully. This allows the application to implement a clear “Turn On Bluetooth” function.

After BlueSoleil is initialized successfully, the application can call any functions that require no communication with Bluetooth device. For example, the application can get a list of pre-configured paired devices.

Each successful call to *Btsdk_Init* must be balanced by a corresponding call to *Btsdk_Done* after subsequent BTSDK function calls are finished and BTSDK is no longer required.

This function is highly recommended to be called only once for successful initialization in an application.

Btsdk_Done

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

Remarks

An application must call *Btsdk_Done* once for each successful call it has made to *Btsdk_Init*.

This function releases all resources allocated by BlueSoleil functions and disables Bluetooth device finally. If the application wants to disable Bluetooth device only, it shall call *Btsdk_StopBluetooth* separately. This allows the application to implement a clear “Turn off Bluetooth” function.

Btsdk_IsSDKInitialized

Prototype	BTBOOL Btsdk_IsSDKInitialized (void);	
Description	The **Btsdk_IsSDKInitialized** function indicates whether a successful call to *Btsdk_Init* is made.	
Parameters		
Return:	If BTSDK is initialized successfully, the return value is BTSDK_TRUE. If BTSDK is not initialized, the return value is BTSDK_FALSE.	

Remarks

An application can call this function at any time to check the state of BlueSoleil.

Btsdk_IsServerConnected

Prototype	BTBOOL Btsdk_IsServerConnected();	
Description	The **Btsdk_IsServerConnected** function checks whether client application can call **BlueSoleil** Server APIs. When this fuction returns BTSDK_TRUE, client application can call APIs normally, versa versit.	
Parameters	*None*	
Return:	BTSDK_FALSE: Server isn't connected. BTSDK_TRUE: Server is connected.	

Remarks

Btsdk_RegisterCallback4ThirdParty

Prototype	BTINT32 Btsdk_RegisterCallback4ThirdParty (PbtSdkCallbackStru call_back);	
Description	The **Btsdk_RegisterCallback4ThirdParty** function registers an application-defined callback function.	
Parameters	*call_back*	[in] Pointer to a BtSdkCallbackStru structure that contains information about the callback function to be registered.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

This is an specific BlueSoleil SDK API.

A message from BlueSoleil is transferred to the application using a callback function. Only one callback function is allowed for one message. That is, if the application calls this Btsdk_RegisterCallback4ThirdParty twice to register different callback functions for the same message type, the second callback function will replace the first one.

If *call_back->func* is NULL, the call to Btsdk_RegisterCallback4ThirdParty will remove the callback for the specified message from BlueSoleil.

Table 6 lists the possible messages and callback function prototypes.

Example

```
/* This sample demonstrates how to register a callback to process inquiry result indication. */
void AppInquiryResultInd(BTDEVHDL dev_hdl)
{
    /* Process the Indication. */
}

void AppRegisterCallback(void)
{
    BtSdkCallbackStru cb;
    cb.type = BTSDK_INQUIRY_RESULT_IND;
    cb.func = (PVOID) AppInquiryResultInd;
```

```
    Btsdk_RegisterCallback4ThirdParty (&cb);
}
```

Btsdk_RegisterGetStatusInfoCB4ThirdParty

Prototype	BTINT32 Btsdk_RegisterGetStatusInfoCB4ThirdParty(Func_ReceiveBluetoothStatusInfo* statusCBK)	
Description	The **Btsdk_RegisterGetStatusInfoCB4ThirdParty** function registers a client process callback function to **BlueSoleil** to receive Bluetooth Status changing event.	
Parameters	statusCBK	[in] pointer to Func_ReceiveBluetoothStatusInfo, whose prototype is defined in bssdk_ui.h
Return:	BTSDK_OK for success, other for error code.	

Remarks

We use **Btsdk_RegisterGetStatusInfoCB4ThirdParty** to register a callback function to deal with status change of **BlueSoleil**. If a user doesn't want to deal with the callback events any more, he should use Btsdk_RegisterGetStatusInfoCB4ThirdParty (NULL) to un-register the callback function.

Btsdk_SetStatusInfoFlag

Prototype	BTINT32 Btsdk_SetStatusInfoFlag(USHORT usMsgType);	
Description	The **Btsdk_SetStatusInfoFlag** function is used to set the status changing callback types which the user wants to receive.	
Parameters	*usMsgType*	Message type which user wants to receive.
Return:	BTSDK_OK for success, other for error code.	

Remarks

A client process can just register one flag to BlueSoleil server, That is, if a client process calls this Btsdk_SetStatusInfoFlag twice to register different flags, the second flag will replace the first one.

usMsgType can be one of the following value or their combination:

BTSDK_NTSERVICE_STATUS_FLAG	The status change of **BlueSoleil** server event or OS message event.
BTSDK_BLUETOOTH_STATUS_FLAG	Message event of the change of Bluetooth status.
BTSDK_REFRESH_STATUS_FLAG	Refresh event.

Func_ReceiveBluetoothStatusInfo

Prototype	typedef void Func_ReceiveBluetoothStatusInfo(ULONG usMsgType, ULONG pulData, ULONG param, BTUINT8 *arg);	
Description	The function prototype of the function to deal with change of **BlueSoleil**'s status.	
Parameters	*usMsgType*	Message type
	pulData	Message event relative to usMsgType.
	param	Be different according to the difference of usMsgType and pulData.
	arg	Be different according to the difference of usMsgType and pulData.
Return:		

Remarks

All the messages of **BlueSoleil** are dealt with by using callback function. If a user wants to deal with status changes of **BlueSoleil** server, a callback function using **Btsdk_RegisterGetStatusInfoCB4ThirdParty** should be registered.
The following table indicates the relationship of usMsgType , pulData, param and Arg.

usMsgType	pulData	Description	Param	Arg
BTSDK_BLUETOOTH_STATUS_FLAG	BTSDK_BTSTATUS_TURNON	Bluetooth is turned on	Not used	Not used
	BTSDK_BTSTATUS_TURNOFF	Bluetooth is turned off		
	BTSDK_BTSTATUS_HWPLUGGED	Bluetooth hardware is plugged.		
	BTSDK_BTSTATUS_HWPULLED	Bluetooth hardware is pulled.		

Example

```
/* This sample demonstrates how to set the flag and register a callback to process status change event. */
void BsStatusCBKFuc(ULONG usMsgType, ULONG pucData, ULONG param, BTUINT8 *arg)
{
    switch(usMsgType)
    {
```

```
        case BTSDK_REFRESH_STATUS_FLAG:
                {
                    Switch(pucData)
                  {
                  case BTSDK_DEL_DEVICE:
                    //do something
                  break;
                  case BTSDK_UNPAIR_DEVICE:
                 //do something
                 break;
                 …..
                 }
        case BTSDK_BLUETOOTH_STATUS_FLAG :
                  {
                     Switch(pucData)
                   {
                     case BTSDK_BTSTATUS_TURNON;
                     // do something
                     break;
                       ……
                     }
                 }
          case ….
      …..
          }
}

Btsdk_SetStatusInfoFlag(BTSDK_NTSERVICE_STATUS_FLAG|
                        BTSDK_BLUETOOTH_STATUS_FLAG|
                        BTSDK_REFRESH_STATUS_FLAG);
Btsdk_RegisterGetStatusInfoCB4ThirdParty(BsStatusCBKFuc);
```

5.4.2 Memory Management

Btsdk_MallocMemory

Prototype	void* Btsdk_MallocMemory (BTUINT32 size;);	
Description	The **Btsdk_MallocMemory** function allocates memory block, which will be passed to the BlueSoleil through BlueSoleil API and released by BlueSoleil module finally, for the upper application.	
Parameters	*size*	[in] Bytes to allocate.
Return:	The pointer to the allocated space, or **NULL** if there is insufficient memory available.	

Btsdk_FreeMemory

Prototype	void Btsdk_FreeMemory (void *memblock;);	
Description	The **Btsdk_FreeMemory** function is used for the upper application to free the memory allocated by Btsdk_MallocMemory.	
Parameters	*memblock*	[in] Memory block to be freed.
Return:	None.	

5.4.3 Local Bluetooth Device Management

5.4.3.1 Device Initialization

Btsdk_StartBluetooth

Prototype	BTINT32 Btsdk_StartBluetooth (void);	
Description	The **Btsdk_StartBluetooth** function enables the local device and initializes the device settings to values configured recently. This function also reads device features required by BlueSoleil Host Protocol Stack.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

This function should be called and the return value expected to be BTSDK_OK before any other functions that require communication with Bluetooth device can be called.

Btsdk_StopBluetooth

Prototype	BTINT32 Btsdk_StopBluetooth (void);	
Description	The **Btsdk_StopBluetooth** function disables the local device.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

This function only disables the local device. It doesn't release the resources allocated by other BlueSoleil functions.

After the application makes a successful call to *Btsdk_Init*, it can call *Btsdk_StartBluetooth* and *Btsdk_StopBluetooth* functions repeatedly to implement "Turn on Bluetooth" and "Turn off Bluetooth" functions.

Btsdk_IsBluetoothReady

Prototype	BTBOOL Btsdk_IsBluetoothReady (void);	
Description	The **Btsdk_IsBluetoothReady** function checks whether the local Bluetooth device is working.	
Parameters		
Return:	If Bluetooth device is enabled, the return value is BTSDK_TRUE. If Bluetooth device is disabled, the return value is BTSDK_FALSE.	

Remarks

An application can call this function at any time to check the working state of the current local device.

Btsdk_IsBluetoothHardwareExisted

Prototype	BTBOOL Btsdk_IsBluetoothHardwareExisted();	
Description	The **Btsdk_IsBluetoothHardwareExisted** function checks whether Bluetooth hardware exists.	
Parameters		
Return:	BTSDK_TRUE: Bluetooth Hardware exists. BTSDK_FALSE: Bluetooth Hardware not exists.	

Remarks

5.4.3.2 Device Modes

Btsdk_SetDiscoveryMode

Prototype	BTINT32 Btsdk_SetDiscoveryMode (BTUINT16 mode);	
Description	The **Btsdk_SetDiscoveryMode** function sets the accessibility modes of the local device.	
Parameters	*mode*	[in] Specifies the modes to be set. It can be one or more of the values listed in Table 5.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

Before calling *Btsdk_SetDiscoveryMode*, the local device must be enabled by a previous successful call to *Btsdk_StartBluetooth*. By default, the local device is in general discoverable mode, connectable mode and pairable mode.

If the application wants to make local device non-discoverable, it must call *Btsdk_SetDiscoveryMode* with none of BTSDK_GENERAL_DISCOVERABLE, BTSDK_DISCOVERABLE and BTSDK_LIMITED_DISCOVERABLE specified in *mode* parameter.

If BTSDK_CONNECTABLE is not specified in *mode* parameter, local device is set to non-connectable mode. If BTSDK_PAIRABLE is not specified in *mode* parameter, local device is set to non-pairable mode.

Example

```
/* This sample demonstrates how to set local device mode. */
void AppChangeMode (void)
{
    /* Make local device discoverable, connectable and non-pairable. */
    BTUINT16 mode = BTSDK_DISCOVERABLE | BTSDK_CONNECTABLE;
    Btsdk_SetDiscoveryMode(mode);
    /* To do: Add other operation. */
    /* Make local device non-discoverable, connectable and pairable. */
    mode = BTSDK_CONNECTABLE | BTSDK_PAIRABLE.
    Btsdk_SetDiscoveryMode(mode);
    /* To do: Add other operation. */
}
```

Btsdk_GetDiscoveryMode

Prototype	BTINT32 Btsdk_GetDiscoveryMode (BTUINT16* pmode);	
Description	The **Btsdk_GetDiscoveryMode** function gets the accessibility modes of the local device.	
Parameters	*pmode*	[out] Pointer to a variable that receives the modes of the local device. The return value can be one or more of the values listed in Table 5.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

If none of BTSDK_GENERAL_DISCOVERABLE, BTSDK_DISCOVERABLE and BTSDK_LIMITED_DISCOVERABLE values are specified in **pmode* parameter, local device is in non-discoverable mode.

If BTSDK_CONNECTABLE value is not specified in **pmode* parameter, local device is in non-connectable mode.

If BTSDK_PAIRABLE value is not specified in **pmode* parameter, local device is in non-pairable mode.

5.4.3.3 Device Information

Btsdk_GetLocalDeviceAddress

Prototype	BTINT32 Btsdk_GetLocalDeviceAddress (BTUINT8* bd_addr,);	
Description	The **Btsdk_GetLocalDeviceAddress** function gets the Bluetooth device address of the local device.	
Parameters	*bd_addr*	[out] Pointer to the buffer that receives the device address. The size, in bytes, of this buffer must be large enough to hold the 6bytes address value.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_SetLocalName

Prototype	BTINT32 Btsdk_SetLocalName (BTUINT8* name, BTUINT16 len);	
Description	The **Btsdk_SetLocalName** function sets the name of the local device.	
Parameters	*name*	[in] Pointer to the buffer containing the string to be used as the device name. This string must be coded in UTF-8 format.
	len	[in] Specifies the size in bytes of the string pointed to by the *name* parameter. It must be no more than BTSDK_DEVNAME_LEN. The exceeding bytes are ignored by BTSDK.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_GetLocalName

Prototype	BTINT32 Btsdk_GetLocalName (BTUINT8* name, BTUINT16* plen);	
Description	The **Btsdk_GetLocalName** function gets the name of the local device.	
Parameters	*name*	[out] Pointer to the buffer that receives the device name. This parameter can be NULL.
	plen	[in/out] Pointer to a variable that, on input, specifies the size, in bytes, of the buffer pointed to by the *name* parameter, or it can be NULL if the buffer size is larger than BTSDK_DEVNAME_LEN. On output, This variable receives the number of bytes copied to the buffer pointed to by the *name* parameter. To determine the required buffer size, call this function with *name* set to NULL. This function returns the required buffer size in **plen*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

The device name is a UTF-8 character string.

Btsdk_SetLocalDeviceClass

Prototype	BTINT32 Btsdk_SetLocalDeviceClass (BTUINT32 device_class);	
Description	The **Btsdk_SetLocalDeviceClass** function sets the Class of Device/Service field of the local device.	
Parameters	*device_class*	[in] Specifies the Class of Device/Service value to be set. It can be one of the device class identifiers listed in Table 3 combined with multiple major service class identifiers listed in Table 4.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

The default Class of Device/Service value of the local device is un-specified. The application shall call this function at least once to specify a proper value according to the usage scenario.

Example

```
/* This sample demonstrates how to set Class of Device/Service value. */
void AppChangeCoD (void)
{
    /* Set local device as a desktop PC.
        Furthermore, specifies that services of Networking and Object Transfer type are available. */
    BTUINT32 dev_class = BTSDK_COMPCLS_DESKTOP | BTSDK_SRVCLS_NETWORK |
                         BTSDK_SRVCLS_OBJECT;
    Btsdk_SetLocalDeviceClass(dev_class);
}
```

Btsdk_GetLocalDeviceClass

Prototype	BTINT32 Btsdk_GetLocalDeviceClass (BTUINT32* pdevice_class);	
Description	The **Btsdk_GetLocalDeviceClass** function gets the Class of Device/Service field value of the local device.	
Parameters	*pdevice_class*	[out] Pointer to a variable that receives the Class of Device/Service value of the local device. The return value can be one of the device class identifiers listed in Table 3 combined with multiple major service class identifiers listed in Table 4.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_GetLocalLMPInfo

Prototype	BTINT32 Btsdk_GetLocalLMPInfo (PBtSdkLocalLMPInfoStru plmp_info);	
Description	The **Btsdk_GetLocalLMPInfo** function gets information about the HCI and LMP in the local device.	
Parameters	*plmp_info*	[out] Pointer to a BtSdkLocalLMPInfoStru structure that receives the information about the HCI and LMP in the local device.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_SetFixedPinCode

Prototype	BTINT32 Btsdk_SetFixedPincode (BTUINT8 *pin_code, BTUINT16 size);	
Description	The **Btsdk_SetFixedPinCode** function sets a fixed PIN code for the local device.	
Parameters	*pin_code*	[in] Pointer to the fixed PIN code.
	size	[in] The size of the fixed PIN code. If the size is bigger than BTSDK_PIN_CODE_LEN, the length of pin_code will be cut to BTSDK_PIN_CODE_LEN.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

Btsdk_GetFixedPinCode

Prototype	BTINT32 Btsdk_GetFixedPincode (BTUINT8 *pin_code, BTUINT16 *psize);	
Description	The **Btsdk_GetFixedPinCode** function gets a fixed PIN code of the local device.	
Parameters	*pin_code*	[out] Pointer to the fixed PIN code.
	psize	[in/out] The size of the fixed PIN code. If *psize* is not NULL, *psize shall specify the maximum length of the 'pin_code'. If psize is NULL, the length of 'pin_code' buffer should not be less than BTSDK_PIN_CODE_LEN. The varible *psize returns the actually copied number.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

5.4.3.4 Application Extension

Btsdk_VendorCommand

Prototype	BTINT32 Btsdk_VendorCommand (BTUINT32 ev_flag, PBtSdkVendorCmdStru in_cmd, PBtSdkEventParamStru out_ev);	
Description	The **Btsdk_VendorCommand** function is used to send a vendor specific HCI command to the local device and receives the corresponding event.	
Parameters	*ev_flag*	[in] Specifies the events generated for the specified command. It is reserved for future extension. Always set it to 0.
	in_cmd	[in] Pointer to a BtSdkVendorCmdStru structure specifies the vendor specific command to be sent to the local device.
	out_ev	[out] Pointer to a BtSdkEventParamStru structure to receive the event generated for the command specified by *in_cmd* parameter.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_VendorCommand can be used to issue a command that generates only a command complete event or a vendor specific event. If more than one event are generated for the specified command, the behavior of BlueSoleil is undefined currently.

The return value BTSDK_OK only confirms that the specified command has been sent to the Bluetooth device and, a command complete event for this command or a vendor specific event is generated. The application shall examine the output event for the actual result itself. For example, if the command generates a command complete event and a “Status” parameter in the return parameters specifying the result, the application shall check the value of “Status” parameter.

Btsdk_EnumAVDriver

Prototype	BTUINT32 Btsdk_EnumAVDriver ();	
Description	The **Btsdk_ EnumAVDriver** function enumerates the AV audio card installed on local machine.	
Parameters		
Return:	The return value is the number of the AV audio cards installed on local machine.	

Remarks

.

Btsdk_DeEnumAVDriver

Prototype	void Btsdk_DeEnumAVDriver();	
Description	The **Btsdk_DeEnumAVDriver** function unplugs the AV audio cards installed on local machine.	
Parameters		
Return:		

Remarks

Btsdk_ActivateEx

Prototype	BTINT32 Btsdk_ActivateEx (const BTINT8 *pszSN, BTINT32 iSnlen);	
Description	The **Btsdk_ActivateEx** function activates BlueSoleil by Serial Number for third party.	
Parameters	*pszSN*	[in] Pointer to the buffer that receives character string of the serial number.
	iSnlen	[in] Length of character string of serial number.
Return:	If BlueSoleil is successfully activated, the return value is BTSDK_OK. If the serial number is not inputted correctly, the return value is BTSDK_ER_INVALID_PARAMETER. Other return value indicates there is a network malfunction or SDK is not initialized.	

Remarks

It will take several seconds for this function to return its value. Consequently, call this function in another thread in order not to block the main thread.

5.4.4 Remote Bluetooth Device Management

This section describes the interface functions used to:

- Discover other nearby Bluetooth devices.
- Retrieve information about other Bluetooth devices.
- Pair or un-pair other Bluetooth devices.
- Manage the link with other Bluetooth devices.
- Manage the Remote device database.

5.4.4.1 Device Discovery

Btsdk_StartDeviceDiscovery

Prototype	BTINT32 Btsdk_StartDeviceDiscovery (BTUINT32 device_class, BTUINT16 max_num, BTUINT16 max_durations);	
Description	The **Btsdk_StartDeviceDiscovery** function makes the Bluetooth device start an inquiry procedure. This procedure is used to discover other nearby Bluetooth devices. A remote device that responds during the inquiry procedure is reported to the application through a BTSDK_INQUIRY_RESULT_IND message. The message BTSDK_INQUIRY_COMPLETE_IND is reported to the application when the inquiry procedure has completed.	
Parameters	*device_class*	[in] Specifies the Class of Device of interest. That is, only a device with the Class of Device specified by *device_class* parameter will be reported to the application. The application can specify one of the device class identifiers listed in Table 3. If this value is set to 0, BlueSoleil reports all devices discovered to the application.
	max_num	[in] Specifies the maximum number of responses during the inquiry procedure. Range of this value is from 0x00 to 0xFF. If this value is set to 0, the number of responses is unlimited.

	max_durations	[in] Specifies the maximum amount of time before the inquiry is halted. The actual duration in seconds is (max_durations * 1.28). Range of this value is from 0x01 to 0x30. If this value is set to 0, BTSDK adopts a default value of 10 instead.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

A device discovered during the inquiry procedure is automatically stored in the device database and marked as an “Inquired” device. The “Inquired” flag will be kept until the next time *Btsdk_StartDeviceDiscovery* or *Btsdk_Done* is called. The application can refer to all “Inquired” devices by calling *Btsdk_GetInquiredDevices* in the future.

The application shall register at least a callback function BlueSoleil to process BTSDK_INQUIRY_COMPLETE_IND message, which indicates that the inquiry procedure has completed. To refer to the devices discovered, the application can register a callback function to BlueSoleil to process BTSDK_INQUIRY_RESULT_IND message, or call *Btsdk_GetInquiredDevices* after the inquiry procedure terminates.

Btsdk_Inquiry_Result_Ind_Func

Prototype	typedef void (Btsdk_Inquiry_Result_Ind_Func) (BTDEVHDL device_handle);	
Description	The **Btsdk_Inquiry_Result_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_INQUIRY_RESULT_IND message.	
Parameters	*device_handle*	[in] Handle assigned to the remote device discovered during the inquiry procedure.
Return:		

Remarks

This callback function is called to report each device discovered separately.

All information of the device discovered is stored in the device database. Each device record in the database is represented by a unique 32bit unsigned integer named as device handle. The handle value is reported to the application through *device_handle* parameter. And the application can call functions *Btsdk_GetRemoteDeviceAddress*, *Btsdk_GetRemoteDeviceClass* and *Btsdk_GetRemoteDeviceName* to get device information from the device database in the future.

Device handle value returned by *device_handle* parameter is valid until the device record is removed by *Btsdk_DeleteRemoteDeviceByHandle*, *Btsdk_DeleteUnpairedDevicesByClass*, or until *Btsdk_Done* is called to terminate using the Bluesoleil.

DO NOT call inside this callback function any functions, e.g. function that waits for a semaphore or requires the user interference, which may block internal thread of BlueSoleil. DO NOT call inside this callback function any BTSDK functions that require communicating with a remote device, either, e.g. *Btsdk_PairDevice*, *Btsdk_Connect* and so on. Furthermore, current version BlueSoelil doesn't support pairing or connecting to a remote device before inquiry procedure is completed.

Btsdk_Inquiry_Complete_Ind_Func

Prototype	typedef void (Btsdk_Inquiry_Complete_Ind_Func) (void);	
Description	The **Btsdk_Inquiry_Complete_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_INQUIRY_COMPLETE_IND message.	
Parameters		
Return:		

Remarks

This callback function is called when the inquiry procedure has completed.

DO not call inside this callback function any functions, e.g. function that waits for a semaphore or requires the user interference, which may block internal thread of BlueSoleil. DO not call inside this callback function any BlueSoleil functions that require communicating with a remote device either, e.g. *Btsdk_PairDevice*, *Btsdk_Connect* and so on. If the application wants to pair or connect to remote device(s) soon after inquiry procedure finishes, it shall call related functions in another thread.

Btsdk_StopDeviceDiscovery

Prototype	BTINT32 Btsdk_StopDeviceDiscovery(void);	
Description	The **Btsdk_StopDeviceDiscovery** function stops the ongoing discovery procedure initiated by a previous call to *Btsdk_StartDeviceDiscovery* function.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

After the device discovery procedure is terminated by the *Btsdk_StopDeviceDiscovery* function, no BTSDK_INQUIRY_ COMPLETE_IND message will be reported to the application.

Btsdk_UpdateRemoteDeviceName

Prototype	BTINT32 Btsdk_UpdateRemoteDeviceName (BTDEVHDL device_handle, BTUINT8* name, BTUINT16* plen);	
Description	The **Btsdk_UpdateRemoteDeviceName** function gets the current user-friendly name of the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device object.
	name	[out] Pointer to the buffer that receives the device name. This parameter can be NULL.
	plen	[in/out] Pointer to a variable that, on input, specifies the size, in bytes, of the buffer pointed to by the *name* parameter, or it can be NULL if the buffer size is larger than BTSDK_DEVNAME_LEN. On output, This variable receives the number of bytes copied to the buffer pointed to by the *name* parameter. To determine the required buffer size, call this function with *name* set to NULL. This function returns the required buffer size in **plen*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

Before calling *Btsdk_UpdateRemoteDeviceName*, the device database must be initialized by a previous successful call to *Btsdk_StartBluetooth*.

The user-friendly device name is a UTF-8 character string. The device name acquired by this command is stored automatically in the device database.

Btsdk_CancelUpdateRemoteDeviceName

Prototype	BTINT32 Btsdk_CancelUpdateRemoteDeviceName (BTDEVHDL device_handle,);	
Description	The **Btsdk_CancelUpdateRemoteDeviceName** function cancels ongoing remote device name update process initiated by the *Btsdk_UpdateRemoteDeviceName* function.	
Parameters	*device_handle*	[in] Handle to the remote device object. It must be the same value as that of *device_handle* parameter of *Btsdk_UpdateRemoteDeviceName*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

Before calling *Btsdk_CancelUpdateRemoteDeviceName*, the device database must be initialized by a previous successful call to *Btsdk_StartBluetooth*.

If the cancellation is successful, Btsdk_UpdateRemoteDeviceName returns error code BTSDK_ER_NO_CONNECTION immediately.

The Btsdk_CancelUpdateRemoteDeviceName function returns error code BTSDK_ER_UNKNOWN_COMMAND immediately, if the local device does not support the cancellation of remote device name request process.

5.4.4.2 Device Pairing

Btsdk_IsDevicePaired

Prototype	BTINT32 Btsdk_IsDevicePaired (BTDEVHDL dev_hdl, BTBOOL *pis_paired);	
Description	The **Btsdk_IsDevicePaired** function checks if the remote device is paired or not.	
Parameters	*dev_hdl*	[in] Handle of the remote device.
	pis_paired	[out] Pointer to the variable of the condition, BTSDK_TRUE or BTSDK_FALSE.
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_IsDevicePaired*, the device database must be initialized by a previous successful call to *Btsdk_Init*.

Btsdk_PairDevice

Prototype	BTINT32 Btsdk_PairDevice (BTDEVHDL device_handle,);	
Description	The **Btsdk_PairDevice** function pairs the specified remote device.	
Parameters	*device_handle*	[in] Handle to the device to be paired.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

After a successful pairing, the new link key is stored automatically in the device database, and the remote device is marked as a “Paired” device. The link key and the “Paired” flag will be kept until the next time *Btsdk_PairDevice* or *Btsdk_UnPairDevice* function is called, or the authentication process with this remote device fails for some reasons (e.g., the remote device deletes the link key.). The application can refer to all “Paired” devices by calling *Btsdk_GetPairedDevices* in the future.

Do not call Btsdk_PairDevice inside a window’s SendMessage handler function, which may block message-processing thread and cause PINCODE dialog cannot pop up properly.

Btsdk_UnPairDevice

Prototype	BTINT32 Btsdk_UnPairDevice (BTDEVHDL device_handle,);	
Description	The **Btsdk_UnPairDevice** function removes the link key and the "Paired" flag of the specified device from the device database.	
Parameters	*device_handle*	[in] Handle to the device to be unpaired.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

After the application calls Btsdk_UnPairDevice to abolish the pair relation with a remote device, the remote device itself may still think of local device as a "Paired" device.

Btsdk_RegisterCallbackEx

Prototype	BTINT32 Btsdk_RegisterCallbackEx (BtSdkCallBackStru* call_back, DWORD priority);	
Description	The **Btsdk_RegisterCallbackEx** function is a extension callback function processing pairing and authentication events of the third party.	
Parameters	*call_back*	[in] Pointer to a BtSdkCallbackStru structure that contains information about the callback function to be registered.
	priority	[in] Specifies the priority of pairing proccessing.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

The default processing priority set by BlueSoleil is “low”. If the processing priority is handled by this call back function, it will not be handled by BlueSoleil. While if the processing priority is not handled by this call back function, it will be handled by BlueSoleil as “low”.

The **priority** parameter can be one of these value

Value	Description
BTSDK_CLIENTCBK_PRIORITY_HIGH	Indicates the priority is “high”
BTSDK_CLIENTCBK_PRIORITY_MEDIUM	Indicates the priority is “medium”

Btsdk_UserHandle_Pin_Req_Ind_Func

Prototype	typedef BTUINT8 (Btsdk_UserHandle_Pin_Req_Ind_Func) (BTDEVHDL dev_hdl);	
Description	The **Btsdk_UserHandle_Pin_Req_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_PIN_CODE_IND message	
Parameters	*dev_hdl*	[in] Handle to the remote device that a PIN code is required to create the new link key for.
Return:	Refer to the table below.	

Remarks

This callback function should return immediately, and the pairing should be handled through another thread. Otherwise BlueSoleil will be blocked.

The return value can be one of these:

Value	Description
BTSDK_CLIENTCBK_HANDLED	It indicates that the client callback is handled.
BTSDK_CLIENTCBK_NOTHANDLED	It indicates that the client callback is not handled.

Btsdk_UserHandle_Authorization_Req_Ind_Func

Prototype	typedef BTUINT8 (Btsdk_UserHandle_Authorization_Req_Ind_Func) (BTSVCHDL svc_hdl, BTDEVHDL dev_hdl);	
Description	The **Btsdk_UserHandle_Authorization_Req_Ind_Fun** function prototype is the prototype of application defined callback function used to process BTSDK_AUTHORIZATION_IND message	
Parameters	*svc_hdl*	[in] Handle to the local service record that the remote device specified by the *device_handle* tries to connect to.
	dev_hdl	[in] Handle to the remote device that tries to connect to the local service record specified by the *service_handle*.
Return:	Refer to the table below.	

Remarks

This callback function should return immediately, and the pairing processing should be handled through another thread. Otherwise BlueSoleil will be blocked.

The return value can be one of these:

Value	Description
BTSDK_CLIENTCBK_HANDLED	It indicates that the client callback is handled.
BTSDK_CLIENTCBK_NOTHANDLED	It indicates that the client callback is not handled.

Btsdk_PinCodeReply

Prototype	BTINT32 Btsdk_PinCodeReply (BTDEVHDL device_handle, BTUINT8* pin_code, BTUINT16 pin_len);	
Description	The **Btsdk_PinCodeReply** function is used to reply the PIN code request during the pair procedure.	
Parameters	*device_handle*	[in] Handle to the remote device to be paired.
	pin_code	[in] Pointer to the buffer contains the PIN code. If the *pin_code* parameter is set to NULL, BlueSoleil sends "HCI PIN Code Request Negative Reply Command" and the pair request fails.
	pin_len	[in] Specifies the length, in bytes, of the PIN code to be used. If the *pin_len* parameter is set to 0, BlueSoleil sends "HCI PIN Code Request Negative Reply Command" and the pair request fails.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

The application shall call the *Btsdk_PinCodeReply* function to reply the PIN code request after it receives the BTSDK_PIN_CODE_IND message.

Btsdk_AuthorizationResponse

Prototype	BTUINT32 Btsdk_AuthorizationResponse (BTSVCHDL service_handle, BTDEVHDL device_handle, BTUINT16 author_response);	
Description	The **Btsdk_AuthorizationResponse** function accepts or rejects the authorization request.	
Parameters	*service_handle*	[in] Handle to the local service record that the remote device specified by the *device_handle* tries to connect to.
	device_handle	[in] Handle to the remote device that tries to connect to the local service record specified by the *service_handle*.
	author_response	[in] BTSDK_AUTHORIZATION_GRANT to accept the authorization request, or BTSDK_AUTHORIZATION_DENY otherwise.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

The application shall call the *Btsdk_AuthorizationResponse* function to reply the authorization request after it receives the BTSDK_AUTHORIZATION_IND message.

Btsdk_Link_Key_Notif_Ind_Func

Prototype	typedef void (Btsdk_Link_Key_Notif_Ind_Func) (BTDEVHDL device_handle, BTUINT8* link_key);	
Description	The **Btsdk_Link_Key_Notif_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_LINK_KEY_NOTIF_IND message.	
Parameters	*device_handle*	[in] Handle to the remote device that a new link key is created for.
	link_key	[in] Pointer to the buffer contains the new link key created.
Return:		

Remarks

This callback function is always called when the pairing succeeds, no matter which side initiates the pairing procedure.

DO NOT call inside this callback function any functions, e.g. function that waits for a semaphore or requires the user interference, which may block internal thread of BlueSoleil. DO NOT call inside this callback function any BlueSoleil functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

Btsdk_Authentication_Fail_Ind_Func

Prototype	typedef void (Btsdk_Authentication_Fail_Ind_Func) (BTDEVHDL device_handle,);	
Description	The **Btsdk_Authentication_Fail_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_AUTHENTICATION_FAIL_IND message.	
Parameters	*device_handle*	[in] Handle to the remote device with which the pairing or authentication fails.
Return:		

Remarks

This callback function is always called when the pairing or authentication fails, no matter which side initiates the pairing or authentication procedure.

DO NOT call inside this callback function any functions, e.g. function that waits for a semaphore or requires the user interference, which may block internal thread of BlueSoleil. DO NOT call inside this callback function any BlueSoleil functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

5.4.4.3 Link Management

This section describes the interface functions used to acquire and modify the status of the ACL link.

Btsdk_IsDeviceConnected

Prototype	BTBOOL Btsdk_IsDeviceConnected (BTDEVHDL device_handle,);	
Description	The **Btsdk_IsDeviceConnected** function checks whether there exist connection between local device and the specified remote device.	
Parameters	*device_handle*	[in] Handle to the device to check role.
Return:	If a connection exists, the return value is BTSDK_TRUE. If no connection exists, the return value is BTSDK_FALSE.	

Remarks

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

Btsdk_GetRemoteDeviceRole

Prototype	BTINT32 Btsdk_GetRemoteDeviceRole (BTDEVHDL device_handle, BTUINT16* prole);	
Description	The **Btsdk_GetRemoteDeviceRole** function gets the current role that the specified device is performing for the ACL link with local device.	
Parameters	*device_handle*	[in] Handle to the device to check role.
	prole	[out] Pointer to a variable to receive the current role. The possible role value can be one of BTSDK_MASTER_ROLE (master role) and BTSDK_SLAVE_ROLE (slave role).
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_GetRemoteLMPInfo

Prototype	BTINT32 Btsdk_GetRemoteLMPInfo (BTDEVHDL device_handle, PBtSdkRemoteLMPInfoStru lmp_info);	
Description	The **Btsdk_GetRemoteLMPInfo** function gets information about the LMP in the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the connection.
	lmp_info	[out] Pointer to a BtSdkRemoteLMPInfoStru structure that receives the information about the LMP in the specified remote device.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_GetRemoteRSSI

Prototype	BTINT32 Btsdk_GetRemoteRSSI (BTDEVHDL device_handle, BTINT8* prssi);	
Description	The **Btsdk_GetRemoteRSSI** function gets the RSSI value of the specified remote device.	
Parameters	*device_handle*	[in] Handle to the specified remote device.
	prssi	[out] Pointer to a variable to receive the RSSI value. Range: -128 to 127 (dB).
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

Before calling *Btsdk_GetRemoteRSSI*, the specified remote device must be inquired or a connection between local device and the remote device must be created.

Btsdk_GetRemoteLinkQuality

Prototype	BTINT32 Btsdk_GetRemoteLinkQuality (BTDEVHDL device_handle, BTUINT16* plink_quality);	
Description	The **Btsdk_GetRemoteLinkQuality** function gets the current link quality value of the connection between local device and the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the connection.
	plink_quality	[out] Pointer to a variable to receive the current link quality value. The higher the value, the better the link quality is. Range: 0 to 0xFF.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_GetSupervisionTimeout

Prototype	BTINT32 Btsdk_GetSupervisionTimeout (BTDEVHDL device_handle, BTUINT16* ptimeout);	
Description	The **Btsdk_GetSupervisionTimeout** function gets the Link Supervision Timeout value for the connection between local device and the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the connection.
	ptimeout	[out] Pointer to a variable to receive the timeout value. The timeout value is measured in number of Bluetooth Baseband slots (0.625msec).
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_SetSupervisionTimeout

Prototype	BTINT32 Btsdk_SetSupervisionTimeout (BTDEVHDL device_handle, BTUINT16 timeout);	
Description	The **Btsdk_SetSupervisionTimeout** function sets the Link Supervision Timeout value for the connection between local device and the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the connection.
	timeout	[in] Specifies the timeout value to be set. The timeout value is measured in number of Bluetooth Baseband slots (0.625msec).
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

Before calling Btsdk_SetSupervisionTimeout, a connection between local device and the specified Remote device must be created first.

Btsdk_ChangeConnectionPacketType

Prototype	BTINT32 Btsdk_ChangeConnectionPacketType (BTDEVHDL device_handle, BTUINT16 packet_type);	
Description	The **Btsdk_ChangeConnectionPacketType** function changes the packet types that can be used for the connection that is currently established with the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the ACL link.
	packet_type	[in] A set of flags which specify the packet types to be used.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

The ***packet_type*** parameter can be one or more of these values.

Value	Description
BTSDK_ACL_PKT_2DH1	2-DH1 is requested. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_3DH1	3-DH1 is requested. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_DM1	DM1 is requested
BTSDK_ACL_PKT_DH1	DH1 is requested.
BTSDK_ACL_PKT_2DH3	2-DH3 is requested. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_3DH3	3-DH3 is requested. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_DM3	DM3 is requested
BTSDK_ACL_PKT_DH3	DH3 is requested.
BTSDK_ACL_PKT_2DH5	2-DH5 is requested. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_3DH5	3-DH5 is requested. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_DM5	DM5 is requested.
BTSDK_ACL_PKT_DH5	DH5 is requested.

Remarks

Before calling *Btsdk_ChangeConnectionPacketType*, a connection between local device and

the specified remote device must be created first.

5.4.4.4 Device Database Management

BlueSoleil stores all the remote devices discovered from the first time run in the device database. At run time, each device record in the database is represented by a unique 32bit unsigned integer named as device handle. The handle value can be used in any function that requires a handle to a remote device.

Btsdk_Init initializes the device database and recovers device records from backup file to the device database. *Btsdk_Done* releases the device database finally. A device handle is created automatically for each record added to the database. The device handle is closed when the device record is removed from the database or when *Btsdk_Done* is called.

The information of a device is added to the database automatically when it responds during the inquiry procedure or when it connects to the BlueSoleil local Bluetooth Host Stack. The application can also add a device record to the database by calling function *Btsdk_GetRemoteDeviceHandle*.

Currently, there is no limit on the number of device records stored in the device database. The application is responsible for determining which device is to be stored or removed.

Btsdk_GetRemoteDeviceHandle

Prototype	BTDEVHDL Btsdk_GetRemoteDeviceHandle (BTUINT8* bd_addr,);	
Description	The **Btsdk_GetRemoteDeviceHandle** function gets the handle to the remote device with the specified Bluetooth device address. If no device record matched the device address is found in the database, this function returns BTSDK_INVALID_HANDLE immediately.	
Parameters	*bd_addr*	[in] Pointer to the buffer contains the Bluetooth device address.
Return:	If the function succeeds, the return value is the handle to the specified remote device. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

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

Btsdk_AddRemoteDevice

Prototype	BTDEVHDL Btsdk_AddRemoteDevice (BTUINT8* bd_addr,);	
Description	The **Btsdk_AddRemoteDevice** function Adds a device record with the specified device address to the database. If a device record matched the device address is found in the database, this function returns the device handle directly.	
Parameters	*bd_addr*	[in] Pointer to the buffer contains the Bluetooth device address.
Return:	If the function succeeds, the return value is the handle to the specified remote device. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

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

Btsdk_DeleteRemoteDeviceByHandle

Prototype	BTINT32 Btsdk_DeleteRemoteDeviceByHandle (BTDEVHDL device_handle,);	
Description	The **Btsdk_DeleteRemoteDeviceByHandle** function removes a specified device record from the database. If a connection between the local device and the specified device exists, BlueSoleil returns the error code BTSDK_ER_ITEM_INUSE and the specified device record isn't removed from the database.	
Parameters	*device_handle*	[in] Device handle specified the device record to be removed from the database.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_DeleteUnpairedDevicesByClass

Prototype	BTINT32 Btsdk_DeleteUnpairedDevicesByClass (BTUINT32 device_class,);	
Description	The **Btsdk_DeleteUnpairedDevicesByClass** function removes all unpaired devices with the specified Class of Device from the device database. If a connection exists between the local device and one of the devices that match the condition, this device record isn't removed from the database.	
Parameters	*device_class*	[in] Specifies the Class of Device of interest. That is, only unpaired devices with the Class of Device specified by *device_class* parameter will be removed from the database. The application can specify one of the device class identifiers listed in Table 3. If this value is set to 0, BlueSoleil removes all unpaired devices from the database.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_GetStoredDevicesByClass

Prototype	BTUINT32 Btsdk_GetStoredDevicesByClass (BTUINT32 device_class, BTDEVHDL* pdevice_handles, BTUINT32 max_dev_num);	
Description	The **Btsdk_GetStoredDevicesByClass** function gets a list of handles to the device records with the specified Class of Device from the device database.	
Parameters	*device_class*	[in] Specifies the Class of Device of interest. That is, only devices with the Class of Device specified by *device_class* parameter will be reported to the application. The application can specify one of the device class identifiers listed in Table 3. If this value is set to 0, BlueSoleil reports all devices stored in the database to the application.
	pdevice_handles	[out] Pointer to the buffer to receive the device handles. If this parameter is set to NULL, the total number of available handles is returned.
	max_dev_num	[in] Specifies the maximum number of handles can be copied to the buffer pointed to by the *pdevice_handles* parameter. If *pdevice_handle* is set to NULL, the value of *max_dev_num* parameter is ignored.
Return:	If *pdevice_handle* is not NULL and *max_dev_num* is non-zero, the return value is the number of handles copied to the buffer pointed to by *pdevice_handles*. If pdevice_handle is NULL, the return value is the total number of available handles.	

Remarks

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

Btsdk_GetInquiredDevices

Prototype	BTUINT32 Btsdk_GetInquiredDevices (BTDEVHDL* pdevice_handles, BTUINT32 max_dev_num);	
Description	The **Btsdk_GetInquiredDevices** function gets a list of handles to the device records that are marked as "Inquired" devices.	
Parameters	*pdevice_handles*	[out] Pointer to the buffer to receive the device handles. If this parameter is set to NULL, the total number of available handles is returned.
	max_dev_num	[in] Specifies the maximum number of handles can be copied to the buffer pointed to by the *pdevice_handles* parameter. If *pdevice_handles* is set to NULL, the value of *max_dev_num* parameter is ignored.
Return:	If *pdevice_handle* is not NULL and *max_dev_num* is nonzero, the return value is the number of handles copied to the buffer pointed to by *pdevice_handles*. If pdevice_handle is NULL, the return value is the total number of available handles.	

Remarks

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

A device discovered during the inquiry procedure is marked as an "Inquired" device. The "Inquired" flag will be kept until the next time *Btsdk_StartDeviceDiscovery* or *Btsdk_Done* is called.

Btsdk_GetPairedDevices

Prototype	BTUINT32 Btsdk_GetPairedDevices (BTDEVHDL* pdevice_handles, BTUINT32 max_dev_num);	
Description	The **Btsdk_GetPairedDevices** function gets a list of handles to the device records that are marked as "Paired" devices.	
Parameters	*pdevice_handles*	[out] Pointer to the buffer to receive the device handles. If this parameter is set to NULL, the total number of available handles is returned.
	max_dev_num	[in] Specifies the maximum number of handles can be copied to the buffer pointed to by the *pdevice_handles* parameter. If *pdevice_handles* is set to NULL, the value of *max_dev_num* parameter is ignored.
Return:	If *pdevice_handles* is not NULL and *max_dev_num* is nonzero, the return value is the number of handles copied to the buffer pointed to by *pdevice_handles*. If pdevice_handles is NULL, the return value is the total number of available handles.	

Remarks

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

Both the local device and the other device may initiate a pairing procedure between them. After the pairing procedure with a remote device finishes successfully, BlueSoleil stores the link key in the device database and marks this remote device as a "Paired" device. The "Paired" flag of a remote device will be kept until *Btsdk_UnPairDevice* is called or an unsuccessful authentication procedure with this remote device occurs.

Btsdk_StartEnumRemoteDevice

Prototype	BTSDKHANDLE Btsdk_StartEnumRemoteDevice (BTUINT32 flag, BTUINT32 device_class);	
Description	The **Btsdk_StartEnumRemoteDevice** function starts to search the device database for devices that match the specified attributes.	
Parameters	*flag*	[in] Specified the attributes to be used in the search.
	device_class	[in] Specifies the Class of Device of interest. That is, only devices with the Class of Device specified by *device_class* parameter will be reported to the application. The application can specify one of the device class identifiers listed in Table 3. The *device_class* parameter is used only when the BTSDK_ERD_FLAG_DEVCLASS value is set in the *flag* parameter.
Return:	If the function succeeds, the return value is a search handle used in a subsequent call to *Btsdk_EnumRemoteDevice* and *Btsdk_EndEnumRemoteDevice*. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

The ***flag*** parameter can be one or more of these values.

Value	Description
BTSDK_ERD_FLAG_NOLIMIT	Search for all devices stored in the database. This value must be used separately.
BTSDK_ERD_FLAG_PAIRED	Search for devices marked as “Paired” devices.
BTSDK_ERD_FLAG_CONNECTED	Search for devices that are connecting with local device currently.
BTSDK_ERD_FLAG_INQUIRED	Search for devices marked as “Inquired” devices.
BTSDK_ERD_FLAG_TRUSTED	Search for devices marked as “Trusted” devices.
BTSDK_ERD_FLAG_DEVCLASS	Search for devices with the Class of Device specified by the *device_class* parameter.

Remarks

Before calling *Btsdk_StartEnumRemoteDevice*, the device database must be initialized by a

previous successful call to *Btsdk_Init*.

The *Btsdk_StartEnumRemoteDevice* function only opens a search handle. After the search handle has been established, use the *Btsdk_EnumRemoteDevice* function to search for device records that match the specified attributes.

Btsdk_EnumRemoteDevice

Prototype	BTDEVHDL Btsdk_EnumRemoteDevice (BTSDKHANDLE enum_handle, PBtSdkRemoteDevicePropertyStru rmt_dev_prop);	
Description	The **Btsdk_EnumRemoteDevice** function continues to search the device database for a device matches the specified attributes. The attributes are specified by a previous call to the *Btsdk_StartEnumRemoteDevice* function.	
Parameters	*enum_handle*	[in] Search handle returned by a previous call to the *Btsdk_StartEnumRemoteDevice* function.
	rmt_dev_prop	[out] Pointer to the BtSdkRemoteDevicePropertyStru structure that receives information about the found device record.
Return:	If the function succeeds, the return value is the handle specifies the found device. If no matching device can be found, the return value is BTSDK_INVALID_HANDLE.	

Remarks

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

Example

```
/* This sample demonstrates how to obtain the collection of paired devices. */
void AppGetPairedDevices(void)
{
    BtSdkRemoteDevicePropertyStru DevProp = {0};
    BTSDKHANDLE hEnumDev = BTSDK_INVALID_HANDLE;
    BTDEVHDL hDevFound = BTSDK_INVALID_HANDLE;

    hEnumDev = Btsdk_StartEnumRemoteDevice(BTSDK_ERD_FLAG_PAIRED, 0);
    if (hEnumDev != BTSDK_INVALID_HANDLE)
    {
        while ((hDevFound = Btsdk_EnumRemoteDevice(hEnumDev, &DevProp)) != BTSDK_INVALID_HANDLE)
        {
            /*To Do: Add additional processing here. */
        }
```

```
            Btsdk_EndEnumRemoteDevice(hEnumDev);
        }
}
```

Btsdk_EndEnumRemoteDevice

Prototype	BTINT32 Btsdk_EndEnumRemoteDevice (BTSDKHANDLE enum_handle,);	
Description	The **Btsdk_EndEnumRemoteDevice** function closes the specified search handle.	
Parameters	*enum_handle*	[in] Search handle returned by a previous call to the *Btsdk_StartEnumRemoteDevice* function.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

When *Btsdk_EnumRemoteDevice* returns BTSDK_INVALID_HANDLE, the application must close the search handle by calling the function *Btsdk_EndEnumRemoteDevice*.

Btsdk_GetRemoteDeviceAddress

Prototype	BTINT32 Btsdk_GetRemoteDeviceAddress (BTDEVHDL device_handle, BTUINT8* bd_addr,);	
Description	The **Btsdk_GetRemoteDeviceAddress** function gets the Bluetooth device address of the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device object.
	bd_addr	[out] Pointer to the buffer to receive the Bluetooth device address. The buffer must be large enough to receive 6 bytes device address.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_GetRemoteDeviceName

Prototype	BTINT32 Btsdk_GetRemoteDeviceName (BTDEVHDL device_handle, BTUINT8* name, BTUINT16* plen);	
Description	The **Btsdk_GetRemoteDeviceName** function gets the user-friendly name of the specified remote device from the device database.	
Parameters	*device_handle*	[in] Handle to the remote device object.
	name	[out] Pointer to the buffer that receives the device name. This parameter can be NULL.
	plen	[in/out] Pointer to a variable that, on input, specifies the size, in bytes, of the buffer pointed to by the *name* parameter, or it can be NULL if the buffer size is larger than BTSDK_DEVNAME_LEN. On output, This variable receives the number of bytes copied to the buffer pointed to by the *name* parameter. To determine the required buffer size, call this function with *name* set to NULL. This function returns the required buffer size in **plen*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

The user-friendly device name is a UTF-8 character string. The *Btsdk_GetRemoteDeviceName* function returns BTSDK_OPERATION_FAILURE immediately if the device name doesn't exist in the database. In this case, the application shall call *Btsdk_UpdateRemoteDeviceName* to acquire the name information directly from the remote device.

BlueSoleil will automatically update the device name when the local device connects to the specified remote device.

Btsdk_GetRemoteDeviceClass

Prototype	BTINT32 Btsdk_GetRemoteDeviceClass (BTDEVHDL device_handle, BTUINT32* pdevice_class,);	
Description	The **Btsdk_GetRemoteDeviceClass** function gets the Class of Device/Service field value of the specified remote device from the device database.	
Parameters	*device_handle*	[in] Handle to the remote device object.
	pdevice_class	[out] Pointer to a variable that receives the Class of Device/Service value of the local device. The return value can be one of the device class identifiers listed in Table 3 combined with multiple major service class identifiers listed in Table 4.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_GetRemoteDeviceProperty

Prototype	BTINT32 Btsdk_GetRemoteDeviceProperty (BTDEVHDL device_handle, PBtSdkRemoteDevicePropertyStru rmt_dev_prop);	
Description	The **Btsdk_GetRemoteDeviceProperty** function gets the information about the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device object.
	rmt_dev_prop	[out] Pointer to the BtSdkRemoteDevicePropertyStru structure that receives information about the specified device.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

The *rmt_dev_prop->bd_addr*, *rmt_dev_prop->dev_class* and *rmt_dev_prop->link_key* values are read from the device database directly.

If the local device doesn't connect to the remote device, the *rmt_dev_prop->name* value is read from the device database. Otherwise, the *rmt_dev_prop->name* value is read from the remote device.

The value of rmt_dev_prop->lmp_info is available only when the local device connects to the specified remote device.

	device_handle	[in] Handle to the remote device to set the trust relation.
	bIsTrusted	[in] BTSDK_TRUE if the specified remote device is trusted to the specified local service record or BTSDK_FALSE otherwise.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Btsdk_RemoteDeviceFlowStatistic

Prototype	BTINT32 Btsdk_RemoteDeviceFlowStatistic (BTDEVHDL dev_hdl, BTUINT32* rx_bytes, BTUINT32* tx_bytes);	
Description	The **Btsdk_RemoteDeviceFlowStatistic** function gets the statistic of data sent to and received from the remote device.	
Parameters	*dev_hdl*	[in] Handle of the remote device. If dev_hdl is set to BTSDK_INVALID_HANDLE, the statistic of data sent and received by the local device is returned.
	rx_bytes	[in] Pointer to the 32bit integer to store how many bytes received.
	tx_bytes	[in] Pointer to the 32bit integer to store how many bytes sent.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

5.4.5 Connection Management

When “connection” is said in this section, it means a synchronized high-level protocol connection defined in the related profile specification.

5.4.5.1 Service Discovery

At run time, each remote service record in the device database is represented by a unique 32bit unsigned integer named as remote service handle. The handle value can be used in any function that requires a handle to a remote service record.

The **service handle** specified here has nothing to do with the service record handle defined in the SDP specification. To differentiate these two concepts, we use **SDP record handle** in this document to represent the service record handle defined in the SDP specification.

Btsdk_BrowseRemoteServicesEx

Prototype	BTINT32 Btsdk_BrowseRemoteServicesEx (BTDEVHDL device_handle, PBtSdkSDPSearchPatternStru psch_ptn, BTUINT32 ptn_number, BTSVCHDL* pservice_handles, BTUINT32* phandle_number);	
Description	The **Btsdk_BrowseRemoteServicesEx** function discovers the available service records, which matches the specified search patterns, on the remote device and queries each service record for its attributes.	
Parameters	*device_handle*	[in] Handle to the remote device to browse service.
	psch_ptn	[in] Pointer to an array of *BtSdkSDPSearchPatternStru* structures that contains *ptn_number* elements. If the *psch_ptn* is a NULL pointer, BTSDK uses the 16bit UUID value 0x0100 as the default search pattern.

	ptn_number	[in] Specifies the number of elements present in the array *psch_ptn*. This value must be less than BTSDK_MAX_SEARCH_PATTERNS, or the exceeding elements are ignored. If the *ptn_number* value is 0, BlueSoleil uses the 16bit UUID value 0x0100 as the default search pattern.
	pservice_handles	[out] Pointer to the buffer to receive the remote service handles. This parameter can be NULL.
	phandle_number	[in/out] Pointer to a variable that, on input, specifies the number of handles can be copied to the *pservice_handles* buffer. On output, This variable receives the number of handles copied to the *pservice_handles* buffer. To determine the required buffer size, call this function with *pservice_handles* set to NULL. This function returns the total number of available handles in **phandle_number*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

All the service records discovered are stored in local SDK device database until *Btsdk_Done* is called. You can access them later by calling *Btsdk_GetRemoteServicesEx* or *Btsdk_GetRemoteServices*.

Btsdk_BrowseRemoteServices

Prototype	BTINT32 Btsdk_BrowseRemoteServices (BTDEVHDL device_handle, BTSVCHDL* pservice_handles, BTUINT32* phandle_number);	
Description	The **Btsdk_BrowseRemoteServices** function discovers all the service records available on the remote device and queries each service record for its attributes.	
Parameters	*device_handle*	[in] Handle to the remote device to browse service.
	pservice_handles	[out] Pointer to the buffer to receive the remote service handles. This parameter can be NULL.
	phandle_number	[in/out] Pointer to a variable that, on input, specifies the number of handles can be copied to the *pservice_handles* buffer. On output, This variable receives the number of handles copied to the *pservice_handles* buffer. To determine the required buffer size, call this function with *pservice_handles* set to NULL. This function returns the total number of available handles in **phandle_number*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

This function uses the 16bit UUID value 0x0100 as the search pattern.

All the service records discovered are stored in local SDK device database until *Btsdk_Done* is called. You can access them later by calling *Btsdk_GetRemoteServicesEx* or *Btsdk_GetRemoteServices*.

Btsdk_RefreshRemoteServiceAttributes

Prototype	BTINT32 Btsdk_RefreshRemoteServiceAttributes (BTSVCHDL service_handle, PBtSdkRemoteServiceAttrStru pservice_attributes);	
Description	The **Btsdk_RefreshRemoteServiceAttributes** function retrieves all the attribute values of a specified remote service record and returns the most useful attribute values to the application.	
Parameters	*service_handle*	[in] Handle to the remote service record.
	pservice_attributes	[out] Pointer to a BtSdkRemoteServiceAttrStru structure to receive the attribute values about the specified service record. This parameter can be NULL.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Use the *mask* member of the *pservice_attributes* parameter to specify the attributes to be retrieved. If *pservice_attributes->mask* includes BTSDK_RSAM_EXTATTRIBUTES, the function allocates a buffer using the *Btsdk_MallocMemory* function, and returns the pointer to the buffer through *pservice_attributes->ext_attributes*. The application should use the *Btsdk_FreeMemory* function to free the buffer when it is no longer needed.

All the attribute values retrieved are stored in local SDK device database. You can access them later by calling *Btsdk_GetRemoteServiceAttributes*.

Btsdk_GetRemoteServicesEx

Prototype	BTINT32 Btsdk_GetRemoteServicesEx (BTDEVHDL device_handle, PBtSdkSDPSearchPatternStru psch_ptn, BTUINT32 ptn_number, BTSVCHDL* pservice_handles, BTUINT32* phandle_number);	
Description	The **Btsdk_GetRemoteServicesEx** function gets the available service records, which matches the specified search patterns, from the device database.	
Parameters	*device_handle*	[in] Handle to the remote device to browse service.
	psch_ptn	[in] Pointer to an array of *BtSdkSDPSearchPatternStru* structures that contains *ptn_number* elements. If the *psch_ptn* is a NULL pointer, BlueSoleil uses the 16bit UUID value 0x0100 as the default search pattern.
	ptn_number	[in] Specifies the number of elements present in the array *psch_ptn*. This value must be less than BTSDK_MAX_SEARCH_PATTERNS, or the exceeding elements are ignored. If the *ptn_number* value is 0, BlueSoleil uses the 16bit UUID value 0x0100 as the default search pattern.
	pservice_handles	[out] Pointer to the buffer to receive the remote service handles. This parameter can be NULL.

	phandle_number	[in/out] Pointer to a variable that, on input, specifies the number of handles can be copied to the *pservice_handles* buffer. On output, This variable receives the number of handles copied to the *pservice_handles* buffer. To determine the required buffer size, call this function with *pservice_handles* set to NULL. This function returns the total number of available handles in **phandle_number*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

The *Btsdk_GetRemoteServicesEx* function won't initiate any SDP transactions. The application shall call *Btsdk_BrowseRemoteServicesEx* first to find out how many service records are available on the remote device and create a service list in local device database. Then call this function to get the list.

Btsdk_GetRemoteServices

Prototype	BTINT32 Btsdk_BrowseRemoteServices (BTDEVHDL device_handle, BTSVCHDL* pservice_handles, BTUINT32* phandle_number);	
Description	The **Btsdk_GetRemoteServices** function gets all the service records available on the remote device from the device database.	
Parameters	*device_handle*	[in] Handle to the remote device to browse service.
	pservice_handles	[out] Pointer to the buffer to receive the remote service handles. This parameter can be NULL.
	phandle_number	[in/out] Pointer to a variable that, on input, specifies the number of handles can be copied to the *pservice_handles* buffer. On output, This variable receives the number of handles copied to the *pservice_handles* buffer. To determine the required buffer size, call this function with *pservice_handles* set to NULL. This function returns the total number of available handles in **phandle_number*.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

The *Btsdk_GetRemoteServices* function won't initiate any SDP transactions. The application shall call *Btsdk_BrowseRemoteServicesEx* or *Btsdk_BrowseRemoteServices* first to find out how many service records are available on the remote device and create a service list in local device database. Then call this function to get the list.

Btsdk_GetRemoteServiceAttributes

Prototype	BTINT32 Btsdk_GetRemoteServiceAttributes (BTSVCHDL service_handle, PBtSdkRemoteServiceAttrStru pattributes);	
Description	The **Btsdk_GetRemoteServiceAttributes** function reads attribute values of a specified remote service record from BlueSoleil local SDK device database.	
Parameters	*service_handle*	[in] Handle to the remote service record.
	pattributes	[out] Pointer to a BtSdkRemoteServiceAttrStru structure to receive the attribute values about the specified service record. This parameter can't be NULL.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Use the *mask* member of the *pservice_attributes* parameter to specify the attributes to be retrieved. If *pservice_attributes->mask* includes BTSDK_RSAM_EXTATTRIBUTES, the function allocates a buffer using the *Btsdk_MallocMemory* function, and returns the pointer to the buffer through *pservice_attributes->ext_attributes*. The application should use the *Btsdk_FreeMemory* function to free the buffer when it is no longer needed.

The *Btsdk_GetRemoteServiceAttributes* function won't initiate any SDP transactions. The application shall call *Btsdk_RefreshRemoteServiceAttributes* first to retrieve attribute values from the remote device and stored the values in local device database. Then call this function to read the values.

Btsdk_StartEnumRemoteService

Prototype	BTSDKHANDLE Btsdk_StartEnumRemoteService (void);	
Description	The **Btsdk_StartEnumRemoteService** function starts to search the device database for all service records available on the specified remote device.	
Parameters		
Return:	If the function succeeds, the return value is a search handle used in a subsequent call to *Btsdk_EnumRemoteService* and *Btsdk_EndEnumRemoteService*. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

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

The *Btsdk_StartEnumRemoteService* function won't initiate any SDP transactions. The application shall call *Btsdk_BrowseRemoteServicesEx* first to find out how many service records are available on the remote device and create a service list in local device database. Then call this function to enumerate the list.

The *Btsdk_StartEnumRemoteService* function only opens a search handle. After the search handle has been established, use the *Btsdk_EnumRemoteService* function to search for available service records.

Btsdk_EnumRemoteService

Prototype	BTSVCHDL Btsdk_EnumRemoteService (BTSDKHANDLE enum_handle, PBtSdkRemoteServiceAttrStru pservice_attributes);	
Description	The **Btsdk_EnumRemoteService** function continues to search the device database for an available service record of a previous specified remote device.	
Parameters	*enum_handle*	[in] Search handle returned by a previous call to the *Btsdk_StartEnumRemoteService* function.
	pservice_attributes	[out] Pointer to the BtSdkRemoteServiceAttrStru structure that receives information about the found service record.
Return:	If the function succeeds, the return value is the handle specifies the found service record. If no more service can be found, the return value is BTSDK_INVALID_HANDLE.	

Remarks

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

Use the *mask* member of the *pservice_attributes* parameter to specify the attributes to be retrieved. If *pservice_attributes->mask* includes BTSDK_RSAM_EXTATTRIBUTES, the function allocates a buffer using the *Btsdk_MallocMemory* function, and returns the pointer to the buffer through *pservice_attributes->ext_attributes*. The application should use the *Btsdk_FreeMemory* function to free the buffer when it is no longer needed.

Example

```
/* This sample demonstrates how to obtain the collection of service records. */
void AppGetRemoteServices(void)
{
    BtSdkRemoteServerAttrStru SvcAttr = {0};
    BTSDKHANDLE hEnumSvc = BTSDK_INVALID_HANDLE;
    BTSVCHDL hSvcFound = BTSDK_INVALID_HANDLE;

    hEnumSvc = Btsdk_StartEnumRemoteService();
```

```
        if (hEnumSvc != BTSDK_INVALID_HANDLE)
        {
                SvcAttr.mask = BTSDK_RSAM_SERVICENAME | BTSDK_RSAM_EXTATTRIBUTES;
                while ((hSvcFound = Btsdk_EnumRemoteService(hEnumSvc, &SvcAttr)) != BTSDK_INVALID_HANDLE)
                {
                        // To Do: Process the service attribute values:
                        // …
                        // Free the buffer
                        Btsdk_FreeMemory(SvcAttr.ext_attributes);
                }
                Btsdk_EndEnumRemoteService(hEnumSvc);
        }
}
```

Btsdk_EndEnumRemoteService

Prototype	BTINT32 Btsdk_EndEnumRemoteService (BTSDKHANDLE enum_handle,);	
Description	The **Btsdk_EndEnumRemoteService** function closes the specified search handle.	
Parameters	*enum_handle*	[in] Search handle returned by a previous call to the *Btsdk_StartEnumRemoteService* function.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

When *Btsdk_EnumRemoteService* returns BTSDK_INVALID_HANDLE, the application must close the search handle by calling the function *Btsdk_EndEnumLocalServer*.

5.4.5.2 Application Extension

Btsdk_SetRemoteServiceParam

Prototype	BTINT32 Btsdk_SetRemoteServiceParam (BTSVCHDL service_handle, BTUINT32 app_param);	
Description	The **Btsdk_SetRemoteServiceParam** function attaches an application specific value to a remote service record.	
Parameters	*service_handle*	[in] Handle to the service that the value is attached to.
	app_param	[in] Parameter value to be attached to the remote device record.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

In current version, SDK stores this application specific value until *Btsdk_Done* is called. The application shall recover this value itself next time after it calls *Btsdk_Init*.

Btsdk_GetRemoteServiceParam

Prototype	BTINT32 Btsdk_GetRemoteServiceParam (BTDEVHDL service_handle, BTUINT32* papp_param);	
Description	The **Btsdk_GetRemoteServiceParam** function gets the application specific value attached to a remote device record.	
Parameters	*service_handle*	[in] Handle to the service that the value is attached to.
	papp_param	[out] Pointer to a variable to receive the application specific value attached to the remote service record.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

5.4.5.3 Connection Establishment

At run time, each connection in the connection database is represented by a unique 32bit unsigned integer named as connection handle. The handle value can be used in any function that requires a handle to an existing connection.

Btsdk_Connect

Prototype	BTINT32 Btsdk_Connect (BTSVCHDL service_handle, BTUINT32 lParam, BTCONNHDL* pconnection_handle,);	
Description	The **Btsdk_Connect** function establishes a connection to the specified remote service record.	
Parameters	*service_handle*	[in] Handle to the remote service record to connect.
	lParam	[in] Profile specific parameter. If "Mandatory" is not specified in this document, it can be set to 0.
	pconnection_handle	[out] Pointer to a buffer to receive the handle specified the new connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

The ***lParam*** member can be a pointer to one of these structures.

Type of remote service	Type of ***lParam***	Mandatory
BTSDK_CLS_SERIAL_PORT	PBtSdkSPPConnParamStru	No
BTSDK_CLS_DIALUP_NET	PBtSdkDUNConnParamStru.	No
BTSDK_CLS_FAX	PBtSdkFAXConnParamStru	No

Detail of these structures is specified in separate profile API documents.

The *lParam* member is ignored and shall be set to 0 for profiles not listed in the upper table.

Btsdk_ConnectEx

Prototype	BTINT32 Btsdk_ConnectEx (BTDEVHDL device_handle, BTUINT16 service_class, BTUINT32 lParam, BTCONNHDL* pconnection_handle,);	
Description	The **Btsdk_ConnectEx** function establishes a connection to a service record of the specified type on the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device to connect.
	service_class	[in] Type of the service record to connect. It can be one of the values listed in the Table 2.
	lParam	[in] Profile specific parameter. If "Mandatory" is not specified in this document, it can be set to 0.
	pconnection_handle	[out] Pointer to a buffer to receive the handle specified the new connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

The ***lParam*** member can be a pointer to one of these structures.

Value of ***service_class***	Type of ***lParam***	Mandatory
BTSDK_CLS_SERIAL_PORT	PBtSdkSPPConnParamStru	No
BTSDK_CLS_DIALUP_NET	PBtSdkDUNConnParamStru.	No
BTSDK_CLS_FAX	PBtSdkFAXConnParamStru	No

Detail of these structures is specified in separate profile API documents.

The *lParam* member is ignored and shall be set to 0 for profiles not listed in the upper table.

Remarks

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

If multiple service records of the specified type exist on the remote device, BlueSoleil SDK

will automatically select the first accessible record to connect.

Btsdk_Connection_Event_Ind_Func

Prototype	typedef void (Btsdk_Connection_Event_Ind_Func) (BTCONNHDL connection_handle, BTUINT16 event, BTUINT8* arg);	
Description	The **Btsdk_Connection_Event_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_CONNECTION_EVENT_IND message.	
Parameters	*connection_handle*	[in] Handle to the new connection created or to the connection lost.
	event	[in] Specifies the event type. See following table.
	arg	[in] Event specific parameter. If not specified additionally, it is a pointer to the BtSdkConnectionPropertyStru structure contains the details about the connection.
Return:		

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

Value	Description
BTSDK_APP_EV_CONN_IND	A remote device connects to a local service record.
BTSDK_APP_EV_DISC_IND	The remote device disconnects the connection, or the connection is lost due to radio communication problems, e.g. the remote device is out of communication range.
BTSDK_APP_EV_CONN_CFM	A local device connects to a remote service record.
BTSDK_APP_EV_DISC_CFM	The local device disconnects the connection from remote service.

Remarks

This callback function is called when a service level connection is created or lost.

DO NOT call inside this callback function any functions, e.g. function that waits for a semaphore or requires the user interference, which may block internal thread of BlueSoleil. DO NOT call inside this callback function any BlueSoleil functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

5.4.5.4 Connection Database Management

Btsdk_GetConnectionProperty

Prototype	BTINT32 Btsdk_GetConnectionProperty (BTCONNHDL connection_handle, PBtSdkConnectionPropertyStru pproperty,);	
Description	The **Btsdk_GetConnectionProperty** function gets information about the specified connection.	
Parameters	*connection_handle*	[in] Handle to the connection to be queried.
	pproperty	[out] Pointer to the BtSdkConnectionPropertyStru structure that receives information about the specified connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

Btsdk_StartEnumConnection

Prototype	BTSDKHANDLE Btsdk_StartEnumConnection (void);	
Description	The **Btsdk_StartEnumConnection** function starts to search the connection database for all connections available.	
Parameters		
Return:	If the function succeeds, the return value is a search handle used in a subsequent call to *Btsdk_EnumConnection* and *Btsdk_EndEnumConnection*. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

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

The *Btsdk_StartEnumConnection* function only opens a search handle. After the search handle has been established, use the *Btsdk_EnumConnection* function to search for available connections.

Btsdk_EnumConnection

Prototype	BTCONNHDL Btsdk_EnumConnection (BTSDKHANDLE enum_handle, PBtSdkConnectionPropertyStru pproperty);	
Description	The **Btsdk_EnumConnection** function continues to search the connection database for an available connection.	
Parameters	*enum_handle*	[in] Search handle returned by a previous call to the *Btsdk_StartEnumConnection* function.
	pproperty	[out] Pointer to the BtSdkConnectionPropertyStru structure that receives information about the found connection.
Return:	If the function succeeds, the return value is the handle specifies the found connection. If no more service can be found, the return value is BTSDK_INVALID_HANDLE.	

Remarks

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

Example

```
/* This sample demonstrates how to obtain the collection of connections. */
void AppGetConnections(void)
{
    BtSdkConnectionPropertyStru prop = {0};
    BTSDKHANDLE hEnumConn = BTSDK_INVALID_HANDLE;
    BTCONNHDL hConn = BTSDK_INVALID_HANDLE;

    hEnumConn = Btsdk_StartEnumConnection();
    if (hEnumConn != BTSDK_INVALID_HANDLE)
    {
        while ((hConn = Btsdk_EnumConn(hEnumConn, &prop)) != BTSDK_INVALID_HANDLE)
        {
            // To Do: Process the connection property:
            // …
        }
```

```
            Btsdk_EndEnumConnection(hEnumConn);
        }
}
```

Btsdk_EndEnumConnection

Prototype	BTINT32 Btsdk_EndEnumConnection (BTSDKHANDLE enum_handle,);	
Description	The **Btsdk_EndEnumConnection** function closes the specified search handle.	
Parameters	*enum_handle*	[in] Search handle returned by a previous call to the *Btsdk_StartEnumConnection* function.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

When *Btsdk_EnumConnection* returns BTSDK_INVALID_HANDLE, the application must close the search handle by calling the function *Btsdk_EndEnumConnection*.

5.4.5.5 Connection Release

Btsdk_Disconnect

Prototype	BTUINT32 Btsdk_Disconnect (BTCONNHDL connection_handle);	
Description	The **Btsdk_GetAllIncomingConnections** function disconnects a connection.	
Parameters	*connection_handle*	[in] Handle to the connection to disconnect.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code listed in Table 1.	

Remarks

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

5.4.6 BlueSoleil Extend APIs

Btsdk_VDIInstallDev

Prototype	BTUINT32 Btsdk_VDIInstallDev(BTINT8 *HardwareID, BTINT8 *COMName);	
Description	The **Btsdk_VDIInstallDev** function is used to install a device specified by HardwareID.	
Parameters	*HardwareID*	[in] hardware ID could be
	COMName	[in/out] [in]: name of COM Port to install. [out]: name of COM Port actually installed.
Return:	BTSDK_OK for success other for error code	

Hardware ID can be one of the following values:

HARDWAREID_MDMDUN	Argument for installation of DUN modem.
HARDWAREID_MDMFAX	Argument for installation of FAX modem.

Remarks

Btsdk_VDIDelModem

Prototype	BTUINT32 Btsdk_VDIDelModem(BTINT8 *COMName);	
Description	The **Btsdk_VDIDelModem** function deletes a modem which has been installed on the COM port specified by COMName.	
Parameters	*COMName*	[in] name of COM Port.
Return:	BTSDK_OK for success other for error code	

Remarks

Btsdk_GetActivationInformation

Prototype	BTUINT32 Btsdk_GetActivationInformation(BTINT8* SerialNumber, BTINT8* ActivateInformation, BTUINT32 ActiveInformationLen);	
Description	The **Btsdk_GetActivationInformation** function allows users to acquire the URL of activate information.	
Parameters	*SerialNumber*	[in] Pointer to the buffer contains the Serial Number for activation of BlueSoleil.
	ActivateInformation	[out] Pointer to the buffer contains URL for Serial Number.
	ActiveInformationLen	[in] Specifies the length, in bytes, of the URL information. The length should not be less than 500 bytes.
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 is used for offline activation when BlueSoleil 6.x is installed to a platform without accessing network and cannot be automatically activated.

Btsdk_EnterUnlockCode

Prototype	BTUINT32 Btsdk_EnterUnlockCode (BTINT8* UnlockCode);	
Description	The **Btsdk_EnterUnlockCode** function allows users to activate BlueSoleil 6.x without network service on local device. Users may get the activate information (unlock code) through another PC with network service, using the URL get from Btsdk_GetActivationInformation function. Store the unlock code in memory pointed by the *UnlockCode* parameter on local device. Then call this **Btsdk_EnterUnlockCode** function to activate BlueSoleil 6.x.	
Parameters	*UnlockCode*	[in] Pointer to the buffer contains the Serial Number.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

6. Profile Specific API Reference

6.1 Constant Reference

6.1.1 Error Codes

The following table provides a list of profile specific error codes. They are returned by many BlueSoleil 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 10: Profile Specific Error Codes.

6.2 Data Structures

6.2.1 Service Registry Parameters

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 about a request of file transferring through FTP.	
Members	*dev_hdl*	Specifies the handle of the remote device which tries to upload /delete the files.
	operation	Specifies the operation on the file.
	flag	Specifies the current status of uploading /deleting.
	file_name	Specifies the name of the file uploaded /deleted or to be uploaded /deleted.

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

FTP specific event	
Value	Description
BTSDK_APP_EV_FTP_PUT	The remote device request to upload the file.
BTSDK_APP_EV_FTP_GET	The remote device request to download the file.
BTSDK_APP_EV_FTP_DEL_FILE	The remote device request to delete the file.
BTSDK_APP_EV_FTP_DEL_FOLDER	The remote device request to delete the folder. In this case, file_name specify the name of the folder to be deleted.

OPP specific event	
Value	Description
BTSDK_APP_EV_OPP_PULL	The remote device request to pull the object.
BTSDK_APP_EV_OPP_PUSH	The remote device request to push the object.
BTSDK_APP_EV_OPP_PUSH_CARD	The remote device request to push the card.
BTSDK_APP_EV_OPP_EXCHG	The remote device request to exchange the objects with local server.

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

Value	Description
BTSDK_ER_CONTINUE	The remote device request to upload /delete the file.
BTSDK_ER_SUCCESS	The remote device uploads /deletes the file successfully.
Other value	Error code specifies the reason of uploading /deleting failure.

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.

6.2.2 Connection Establishment Parameters

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 which specify 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 BlueSoleil, 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. BlueSoleil 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.

BtSdkOPPConnParamStru

Definition	typedef struct _BtSdkOPPConnParamStru { BTUINT32 size; BTUINT8 inbox_path[BTSDK_PATH_MAXLENGTH]; BTUINT8 outbox_path[BTSDK_PATH_MAXLENGTH]; BTUINT8 own_card[BTSDK_CARDNAME_MAXLENGTH]; } BtSdkOPPConnParamStru, *PBtSdkOPPConnParamStru;	
Description	The structure BtSdkOPPConnParamStru contains additional parameters required to establish an OPP connection to a remote OPP gateway.	
Members	*size*	Size of the structure, in bytes.
	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.

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 which specify 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. BlueSoleil 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.

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 which specify 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. BlueSoleil 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.

6.2.3 Message Parameters

Btsdk_HFP_COPSInfoStru

Definition	struct Btsdk_HFP_COPSInfoStru {
	BTUINT8 mode;
	BTUINT8 format;
	BTUINT8 operator_len;
	BTINT8 operator_name[1];
	};	

Description	The structure Btsdk_HFP_COPSInfoStru contains the information of network operator.	
Members	*mode*	Current mode and provides no information with regard to the name of the operator.
	format	The format of the operator parameter string.
	operator_len	The length of the operator name.
	operator_name[1]	the string in alphanumeric format representing the name of the network operator

Btsdk_HFP_PhoneInfoStru

Definition	struct Btsdk_HFP_PhoneInfoStru {
	BTUINT8 type;
	BTUINT8 service;
	BTUINT8 num_len;
	BTINT8 number[32];
	BTUINT8 name_len;
	BTINT8 alpha_str[1];
	}	

Description	The structure Btsdk_HFP_PhoneInfoStru contains the information of subscriber.	
Members	*type*	The format of the phone number provided.
	service	This member indicates which service this phone number relates to. It shall be either 4 (voice) or 5 (fax).
	num_len	The length of the phone number provided
	number[32]	Subscriber number, the length shall be 32
	name_len	Length of sub-address.
	alpha_str[1]	String type sub-address of format specified by <cli_validity>

Btsdk_HFP_CLCCInfoStru

Definition	struct Btsdk_HFP_CLCCInfoStru{
	BTUINT8 idx;
	BTUINT8 dir;
	BTUINT8 status;
	BTUINT8 mode;
	BTUINT8 mpty;
	BTUINT8 type;
	BTUINT8 num_len;
	BTINT8 number[1];
	}	

Description	The structure Btsdk_HFP_CLCCInfoStru contains the information of current call.	
Members	*idx*	The numbering (start with 1) of the call given by the sequence of setting up or receiving the calls.
	dir	The direction of the call. 0 = outgoing, 1 = incoming
	status	The status of current call. 0=active, 1=held, 2 = dialing (outgoing), 3 = alerting (outgoing), 4 = incoming (incoming), 5 = waiting (incoming)
	mode	Current calling's mode. 0 = voice, 1 = data, 2 = fax
	mpty	The flag of multi-party calling. 0 = no multi-party, 1= multi-party.
	type	The format of the phone number provided.
	num_len	The length of the phone number provided.
	number[1]	Phone number.

Btsdk_HFP_CINDInfoStru

Definition	struct Btsdk_HFP_CINDInfoStru {
	BTUINT8 service;
	BTUINT8 call;
	BTUINT8 callsetup;
	BTUINT8 callheld;
	BTUINT8 signal;
	BTUINT8 roam;
	BTUINT8 battchg;
	};	

Description	The structure Btsdk_HFP_CINDInfoStru contains current state mask code for function BtSDK_AGAP_SetCurIndicatorVal.	
Members	*service*	Indicates the status of service. 0 = unavailable, 1 = available
	call	Indicates the status of active call. 0 = no active call, 1 = on an active call
	callsetup	Indicates the status of callsetup. 0 = no callsetup, 1 = incoming, 2 = outgoing, 3 = outalert
	callheld	Indicates the status of callheld. 0 = no callheld, 1 = active-hold, 2 = onhold
	signal	The strength of signal. 0~5
	roam	Indicates the status of roam. 0 = no roam, 1 = roam
	battchg	The strength of signal. The range is 0~5

Btsdk_HFP_ConnInfo

Definition	struct Btsdk_HFP_ConnInfo {
	BTUINT16 role;
	BTDEVHDL dev_hdl;
	}	

Description	The structure Btsdk_HFP_ConnInfo contains the information of HFP connection.	
Members	*role*	Specifies the role of the local device of the connection.
	dev_hdl	The handle of remote device.

Remarks

This structure is a parameter of the BTSDK_HFP_EV_SLC_ESTABLISHED_IND and BTSDK_HFP_EV_SLC_RELEASED_IND events.

The ***role*** parameter can be one of these values

Value	Description
BTSDK_CLS_HANDSFREE	Local device acts as a Hands-free device
BTSDK_CLS_HANDSFREE_AG	Local device acts as a Hands-free AG.
BTSDK_CLS_HEADSET	Local device acts as a Handset.
BTSDK_CLS_HEADSET_AG	Local device acts as a Headset AG.

Btsdk_HFP_ATCmdResult

Definition	struct Btsdk_HFP_ATCmdResult {
	BTUINT16 cmd_code;
	BTUINT8 result_code;
	}	

Description	The structure Btsdk_HFP_ATCmdResult contains the result of AT command.	
Members	*cmd_code*	Specify the AT command code.
	result_code	Result of the AT command *cmd_code*, it might be BTSDK_HFP_APPERR_TIMEOUT, CME Error Code or standard error result code.

Remarks

This structure is a parameter of the BTSDK_HFP_EV_ATCMD_RESULT events.

BtSdkHFPUIParam

Definition	struct BtSdkHFPUIParam {
	BTUINT32 size;
	BTUINT16 mask;
	BTUINT16 features;
	}	

Description	The structure BtSdkHFPUIParam contains the supported feature of local device.	
Members	*size*	The size of the structure BtSdkHFPUIParam
	mask	The mask is reserved and it should be set to 0.
	features	Supported features of local device.

Remarks

1) For HSP, it shall be 0.

2) For HFP-HF, it can be binary combination of the following values:

Value	Description
BTSDK_HF_BRSF_NREC	EC and/or NR function
BTSDK_HF_BRSF_3WAYCALL	Call waiting and 3-way calling
BTSDK_HF_BRSF_CLIP	CLI presentation capability
BTSDK_HF_BRSF_BVRA	Voice recognition activation
BTSDK_HF_BRSF_RMTVOLCTRL	Remote volume control
BTSDK_HF_BRSF_ENHANCED_CALLSTATUS	Enhanced call status
BTSDK_HF_BRSF_ENHANCED_CALLCONTROL	Enhanced call control

3) For HFP-AG, it can be binary combination of the following values:

Value	Description
BTSDK_AG_BRSF_3WAYCALL	Three-way calling
BTSDK_AG_BRSF_NREC	EC and/or NR function
BTSDK_AG_BRSF_BVRA	Voice recognition function
BTSDK_AG_BRSF_INBANDRING	In-band ring tone capability
BTSDK_AG_BRSF_BINP	Attach a number to a voice tag
BTSDK_AG_BRSF_REJECT_CALL	Ability to reject a call
BTSDK_AG_BRSF_ENHANCED_CALLSTATUS	Enhanced call status
BTSDK_AG_BRSF_ENHANCED_CALLCONTROL	Enhanced call control
BTSDK_AG_BRSF_EXTENDED_ERRORRESULT	Extended Error Result Codes

BtSdk_SDAP_PNPINFO

Definition	struct BtSdk_SDAP_PNPINFO{
	BTUINT16 size;
	BTUINT16 mask;
	BTUINT32 svc_hdl;
	BTUINT16 spec_id;
	BTUINT16 vendor_id;
	BTUINT16 product_id;
	BTUINT16 version_value;
	BTUINT16 vendor_id_src;
	};	

Description	The structure BtSdk_SDAP_PNPINFO contains the information of Plug and Play.	
Members	*size*	The size of the structure BtSdk_SDAP_PNPINFO.
	mask	Specify the optional or mandatory bool type attribute mask.
	svc_hdl	The service handle.
	spec_id	Specify the specification ID.
	vendor_id	Specify the vendor ID.
	product_id	Specify the product ID.
	version_value	Specify the version.
	vendor_id_src	Specify the vendor ID source.

Remarks

BtSdkRmtDISvcExtAttrStru

Definition	typedef struct BtSdkRmtDISvcExtAttrStru{
	BTUINT32 size;
	BTUINT16 mask;
	BTUINT16 spec_id;
	BTUINT16 vendor_id;
	BTUINT16 product_id;
	BTUINT16 version;
	BTBOOL primary_record;
	BTUINT16 vendor_id_source;
	BTUINT16 list_size;
	BTUINT8 str_url_list[1];
	};	

Description	The structure BtSdkRmtDISvcExtAttrStru contains the information of device ID.	
Members	*size*	The size of the structure BtSdkRmtDISvcExtAttrStru.
	mask	Specify whether an optional attribute value is available.
	spec_id	Specify the specification ID.
	vendor_id	Specify the vendor ID.
	product_id	Specify the product ID.
	version	Specify the version.
	primary_record	Specify the primary record.
	vendor_id_source	Specify the vendor ID source.
	list_size	The size of the text string list.
	str_url_list[1]	Specify the List of ClientExecutableURL, DocumentationURL and ServiceDescription attributes.

Remarks

6.3 API Functions

6.3.1 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.

6.3.1.1 General

Btsdk_FTPRegisterStatusCallback4ThirdParty

Prototype	void Btsdk_FTPRegisterStatusCallback4ThirdParty (BTCONNHDL conn_hdl, Btsdk_FTP_STATUS_INFO_CB* func);	
Description	The **Btsdk_FTPRegisterStatusCallback4ThirdParty** 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.

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

6.3.1.2 FTP Server

Btsdk_FTPRegisterDealReceiveFileCB4ThirdParty

Prototype	Void Btsdk_FTPRegisterDealReceiveFileCB4ThirdParty (BTSDK_FTP_UIDealReceiveFile* func);	
Description	The **Btsdk_FTPRegisterDealReceiveFileCB4ThirdParty** function registers an application-defined callback function used to process file transferring 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.

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 transfering 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.

6.3.1.3 FTP Client

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*.

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).

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.

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.

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.

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.

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.

Btsdk_FTPCancelTransfer

Prototype	BTINT32 Btsdk_FTPCancelTransfer (BTCONNHDL conn_hdl,);	
Description	The **Btsdk_FTPCancelTransfer** function terminates the file transferring 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.

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.

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.

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.

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.

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.

6.3.2 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.

6.3.2.1 General

Btsdk_OPPRegisterStatusCallback4ThirdParty

Prototype	void Btsdk_OPPRegisterStatusCallback4ThirdParty (BTCONNHDL conn_hdl, Btsdk_OPP_STATUS_INFO_CB* func);	
Description	The **Btsdk_OPPRegisterStatusCallback4ThirdParty** function registers an application-defined callback function used to deal with FTP tranferring 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_OPPRegisterStatusCallback4ThirdParty* 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_OPPRegisterStatusCallback4ThirdParty* will remove the callback for the specified connection handle.

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.

6.3.2.2 OPP Server

Btsdk_OPPRegisterDealReceiveFileCB4ThirdParty

Prototype	Void Btsdk_OPPRegisterDealReceiveFileCB4ThirdParty (BTSDK_OPP_UIDealReceiveFile* func);	
Description	The **Btsdk_OPPRegisterDealReceiveFileCB4ThirdParty** 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.

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.

6.3.2.3 OPP Client

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.

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.

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 OPP 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.

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.

6.3.3 Personal Area Networking Profile

6.3.3.1 General

Btsdk_PAN_RegIndCbk4ThirdParty

Prototype	void Btsdk_PAN_RegIndCbk4ThirdParty (Btsdk_PAN_Event_Ind_Func *pfunc);	
Description	The **Btsdk_PAN_RegIndCbk4ThirdParty** function registers an application-defined callback function used to deal with PAN callback messages.	
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

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.

6.3.4 Audio/Video Remote Control Profile

6.3.4.1 AVRCP Target (TG)

Btsdk_AVRCP_RegPassThrCmdCbk4ThirdParty

Prototype	void Btsdk_AVRCP_RegPassThrCmdCbk4ThirdParty (Btsdk_AVRCP_PassThr_Cmd_Func *pfunc);	
Description	The **Btsdk_AVRCP_RegPassThrCmdCbk4ThirdParty** 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, BlueSoleil 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.

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.

Btsdk_AVRCP_RegIndCbk4ThirdParty

Prototype	void Btsdk_AVRCP_RegIndCbk (Btsdk_AVRCP_Event_Ind_Func *pfunc);	
Description	The **Btsdk_AVRCP_RegIndCbk4ThirdParty** function is register client another callback function to Bssdk.dll. If one client call Btsdk_AVRCP_RegIndCbk and this function to register callback the same time, bssdk will call these two callback functions.	
Parameters	*pfunc*	[in] pointer to Btsdk_AVRCP_Event_Ind_Func, the detailed prototype definition information is shown in **BlueSoleil-**API(A2DP,AVRCP,PAN).doc, please refer to it for details.
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.

Two events:
BTSDK_APP_EV_AVRCP_IND_CONN and BTSDK_APP_EV_AVRCP_IND_DISCONN need to be processed.For example, the application needs implement something to control player.

The application should free the param.

Example

```
Btsdk_AVRCP_RegIndCbk4ThirdParty (AVRCP_Event_CbkFunc);

void AVRCP_Event_CbkFunc(BTUINT8 event, BTUINT8 *param)
{
    switch (event)
    {
        case BTSDK_APP_EV_AVRCP_IND_CONN:
            /*prepare to control player */
            break;
        case BTSDK_APP_EV_AVRCP_IND_DISCONN:
```

```
                                /*exit to control player*/
                                break;
                        default:
                                break;
                }
        if (NULL != param)
        {
                Btsdk_FreeMemory(param);
        }
}
```

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.

Remarks

This callback function is called local avrcp target connect with or disconnect from remote avrcp controller.

6.3.5 Serial Port Profile

Btsdk_InitCommObj

Prototype	BTINT32 Btsdk_InitCommObj (BTUINT8 com_idx, BTUINT16 svc_class);	
Description	The **Btsdk_InitCommObj** function initializes the COM port object.	
Parameters	*com_idx*	Integer that specifies the COM port to be initialized.
	svc_class	Type of the service record. It can be one of the values listed in the Table 2.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code of either BTSDK_ER_COM_INUSED or BTSDK_ER_INVALID_PARAMETER.	

Remarks

Btsdk_DeinitCommObj

Prototype	BTINT32 Btsdk_DeinitCommObj (BTUINT8 com_idx);	
Description	The **Btsdk_DeinitCommObj** function deletes the COM port designated by com_idx.	
Parameters	*com_idx*	Integer that specifies the COM port 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 of BTSDK_ER_COM_INUSED.	

Remarks

Btsdk_GetClientPort

Prototype	BTINT16 Btsdk_GetClientPort(BTCONNHDL conn_hdl);	
Description	The **Btsdk_GetClientPort** function gets the client COM port of the SPP, DUN, and LAP connection.	
Parameters	*conn_hdl*	[in] Handle of the connection.
Return:	If the function succeeds, the return value is the ID number of COM port. If an error, the return value is 0.	

Remarks

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

Btsdk_GetAvailableExtSPPCOMPort

Prototype	BTUINT8 Btsdk_GetAvailableExtSPPCOMPort (BTBOOL bIsForLocalSPPService);	
Description	The **Btsdk_GetAvailableExtSPPCOMPort** function gets available COM port used for 128 bit spp.	
Parameters	*bIsForLocalSPPService*	[in] Notify BlueSoleil the usage of this COM port. BTSDK_TRUE: This COM port is used to register a local defined SPP-based service record. BTSDK_FALSE: This COM port is used to connect to an application defined SPP-based service record.
Return:	If there is a COM port available, the return value is the ID number of the serial port. If there is no COM port available, the return value is 0.	

Remarks

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*.

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.

Btsdk_GetASerialNum

Prototype	BTUINT32 Btsdk_GetASerialNum();	
Description	The **Btsdk_GetASerialNum** function gets a currently available serial number of COM port.	
Parameters	*None*	
Return:	The return value is the currently available serial number of COM port.	

Remarks

Btsdk_PlugInVComm

Prototype	BOOL Btsdk_PlugInVComm (UINT serialNum, ULONG *comportNumber, UINT usageType, ULONG flag, DWORD dwTimeout);	
Description	The **Btsdk_PlugInVComm** function plugs in a currently available COM port.	
Parameters	*serialNum*	[in] Serial number of COM port which is return value of the function Btsdk_GetASerialNum
	comportNumber	[in/out] Pointer to buffer containing Com port number specified by OS
	usageType	[in] This parameter must be 1
	flag	[in] This parameter must be COMM_SET_RECORD|COMM_SET_USAGETYPE
	dwTimeout	[in] The timeouts of plugging in the serial port.
Return:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE	

Remarks

Before calling Btsdk_PlugInVComm, the Btsdk_GetASerialNum must be called to get the parameter of *serialNum*

After calling Btsdk_PlugInVComm, the Btsdk_InitCommObj should be called to initialize the COM port.

Flag description:

Value	Description
COMM_SET_RECORD	This macro indicates if the COM port is recorded by BlueSoleil.
COMM_SET_USAGETYPE	This macro is an identity of BlueSoleil designated by OS.

Btsdk_CommNumToSerialNum

Prototype	BTUINT32 Btsdk_CommNumToSerialNum (int comportNum);	
Description	The **Btsdk_CommNumToSerialNum** gets the serial number of COM port from COM port number.	
Parameters	*comportNum*	[in] Com port number specified by OS
Return:	The return value is a serial number of COM port.	

Remarks

Btsdk_PlugOutVComm

Prototype	void Btsdk_PlugOutVComm (UINT serialNum, ULONG flag);	
Description	The **Btsdk_PlugInVComm** function plugs out a COM port.	
Parameters	*serialNum*	[in] Serial number of COM port which is return value of the function Btsdk_GetASerialNum
	flag	[in] This parameter must be COMM_SET_RECORD
Return:		

Remarks

After calling Btsdk_PlugOutVComm, the Btsdk_DeinitCommObj should be called to delete the COM port.

6.3.6 Hands-free and Headset Profile

BlueSoleil SDK provides the same APIs for these two profiles.

Btsdk_RegisterHFPService

Prototype	BTSVCHDL Btsdk_RegisterHFPService(BTUINT8 *svc_name, BTUINT16 svc_class, BTUINT16 features);	
Description	The **Btsdk_RegisterHFPService** function registers a HFP or HEP service.	
Parameters	svc_name	[in] User friendly name of the new service. It shall be a null-terminated UTF-8 string. It can't be NULL. Its length shall be limited within BTSDK_SERVICENAME_MAXLENGTH, including the terminated '\0'.
	svc_class	[in] 16bit UUID specifies the service type. It can be one of: BTSDK_CLS_HANDSFREE, BTSDK_CLS_HANDSFREE_AG, BTSDK_CLS_HEADSET, BTSDK_CLS_HEADSET_AG.
	features	[in] A set of flags specifies the BRSF features supported by the new Hands-free HF or AG service. Its value is ignored If the service is of Headset HS or AG type.
Return:	The handle of the service.	

The ***features*** parameter can be binary combination of the following values:

Value	Description
BTSDK_AG_BRSF_3WAYCALL	Three-way calling
BTSDK_AG_BRSF_NREC	EC and NR function
BTSDK_AG_BRSF_BVRA	Voice recognition function
BTSDK_AG_BRSF_INBANDRING	In-band ring tone capability
BTSDK_AG_BRSF_BINP	Attach a number to a voice tag
BTSDK_AG_BRSF_REJECT_CALL	Ability to reject a call
BTSDK_AG_BRSF_ENHANCED_CALLSTATUS	Enhanced call status
BTSDK_AG_BRSF_ENHANCED_CALLCONTROL	Enhanced call control

BTSDK_AG_BRSF_EXTENDED_ERRORRESULT	Extended Error Result Codes
BTSDK_AG_BRSF_ALL	Support all the upper features
BTSDK_HF_BRSF_NREC	EC and/or NR function
BTSDK_HF_BRSF_3WAYCALL	Call waiting and 3-way calling
BTSDK_HF_BRSF_CLIP	CLI presentation capability
BTSDK_HF_BRSF_BVRA	Voice recognition activation
BTSDK_HF_BRSF_RMTVOLCTRL	Remote volume control
BTSDK_HF_BRSF_ENHANCED_CALLSTATUS	Enhanced call status
BTSDK_HF_BRSF_ENHANCED_CALLCONTROL	Enhanced call control
BTSDK_HF_BRSF_ALL	Support all the upper features

Remarks

This function MUST be called and the return value MUST be BTSDK_OK before any other HFP functions is called.

This function will enable both Hands-free and Headset services at the same time. But only one connection is allowed every time, no matter which side (local or remote application) initiates the connection. For example, if a connection between the local Hands-free AG and a remote Hands-free Unit is created, no more connections with other Hands-free Units or Headsets can be created until the previous connection is released.

Btsdk_UnregisterHFPService

Prototype	BTUINT32 Btsdk_UnregisterHFPService(BTSVCHDL svc_hdl);	
Description	The **Btsdk_UnregisterHFPService** function unregisters HFP service.	
Parameters	*svc_hdl*	The handle of the service.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFP_Callback

Prototype	typedef void (Btsdk_HFP_Callback)(BTCONNHDL hdl, BTUINT16 event, BTUINT8 *param, BTUINT16 len);	
Description	The **Btsdk_HFP_Callback** function prototype is the prototype of application defined callback function used to process Hands-free/Headset events.	
Parameters	hdl	[in] Handle to the HFP connection with a remote HF that is to send the call answered indication.
	event	[in] Event identifier.
	param	[in] First event parameters. It is usually a pointer to an evevt specific variable.
	len	[in] Specify the length, in bytes, of the string pointed to by the param, not including the terminated NULL.
Return:		

The ***event*** and ***param***:

event	param
BTSDK_HFP_EV_SPP_ESTABLISHED_IND	Btsdk_HFP_ConnInfoStru
BTSDK_HFP_EV_SLC_ESTABLISHED_IND	Btsdk_HFP_ConnInfoStru
BTSDK_HFP_EV_SLC_RELEASED_IND	Btsdk_HFP_ConnInfoStru
BTSDK_HFP_EV_STANDBY_IND	NULL
BTSDK_HFP_EV_ONGOINGCALL_IND	NULL
BTSDK_HFP_EV_RINGING_IND	BTUINT8: Specify the type of ring tone. 0 – Local ring tone; 1 – In-band ring tone
BTSDK_HFP_EV_OUTGOINGCALL_IND	NULL
BTSDK_HFP_EV_CALLHELD_IND	NULL
BTSDK_ HFP_EV_CALL_WAITING_IND	Btsdk_HFP_PhoneInfoStru
BTSDK_HFP_EV_TBUSY_IND	NULL
BTSDK_HFP_EV_GENERATE_INBAND_RINGTONE_IND	NULL

BTSDK_HFP_EV_TERMINATE_LOCAL_RINGTONE_IND	NULL
BTSDK_HFP_EV_VOICE_RECOGN_ACTIVATED_IND	NULL
BTSDK_HFP_EV_VOICE_RECOGN_DEACTIVATED_IND	NULL
BTSDK_HFP_EV_NETWORK_AVAILABLE_IND	NULL
BTSDK_HFP_EV_NETWORK_UNAVAILABLE_IND	NULL
BTSDK_HFP_EV_ROAMING_RESET_IND	NULL
BTSDK_HFP_EV_ROAMING_ACTIVE_IND	NULL
BTSDK_HFP_EV_SIGNAL_STRENGTH_IND	BTUINT8: The signal strength value.
BTSDK_HFP_EV_BATTERY_CHARGE_IND	BTUINT8: Battery charge indicator value.
BTSDK_HFP_EV_CHLDHELD_ACTIVATED_IND	NULL
BTSDK_HFP_EV_CHLDHELD_RELEASED_IND	NULL
BTSDK_HFP_EV_MICVOL_CHANGED_IND	BTUINT8: The gain value of microphone.
BTSDK_HFP_EV_SPKVOL_CHANGED_IND	BTUINT8: The speaker gain value.
BTSDK_HFP_EV_CURRENT_CALLS_REQ	NULL
BTSDK_HFP_EV_NETWORK_OPERATOR_FORMAT_REQ	NULL
BTSDK_HFP_EV_NETWORK_OPERATOR_REQ	NULL
BTSDK_HFP_EV_SUBSCRIBER_NUMBER_REQ	NULL
BTSDK_HFP_EV_VOICETAG_PHONE_NUM_REQ	NULL
BTSDK_HFP_EV_CUR_INDICATOR_VAL_REQ	NULL
BTSDK_HFP_EV_HF_DIAL_REQ	For a HFP-AG connection, it points to a buffer containing the phone number to dial; For a HSP-AG connection, it is set to NULL.
BTSDK_HFP_EV_HF_MEM_DIAL_REQ	BTUINT8*: Points to a buffer containing the memory location index.
BTSDK_HFP_EV_HF_LASTNUM_REDIAL_REQ	NULL
BTSDK_HFP_EV_MANUFACTURER_REQ	NULL
BTSDK_HFP_EV_MODEL_REQ	NULL

BTSDK_HFP_EV_NREC_DISABLE_REQ	NULL
BTSDK_HFP_EV_DTMF_REQ	BTUINT8: The DTMF code.
BTSDK_HFP_EV_ANSWER_CALL_REQ	BTUINT8: Specifies the type of the call to answer. It can be one of BTSDK_HFP_TYPE_INCOMING_CALL, BTSDK_HFP_TYPE_HELDINCOMING_CALL.
BTSDK_HFP_EV_CANCEL_CALL_REQ	BTUINT8: Specifies the type of the call to release. It can be one of BTSDK_HFP_TYPE_ALL_CALLS, BTSDK_HFP_TYPE_INCOMING_CALL, BTSDK_HFP_TYPE_HELDINCOMING_CALL, BTSDK_HFP_TYPE_OUTGOING_CALL, BTSDK_HFP_TYPE_ONGOING_CALL.
BTSDK_HFP_EV_HOLD_CALL_REQ	NULL
BTSDK_HFP_EV_REJECTWAITINGCALL_REQ	NULL
BTSDK_HFP_EV_ACPTWAIT_RELEASEACTIVE_REQ	BTUINT8:The value of idx is specified by AT+CHLD=1<idx>
BTSDK_HFP_EV_HOLDACTIVECALL_REQ	BTUINT8: The value of idx is specified by AT+CHLD=2<idx>
BTSDK_HFP_EV_ADD_ONEHELDCALL_2ACTIVE_REQ	NULL
BTSDK_HFP_EV_LEAVE3WAYCALLING_REQ	NULL
BTSDK_HFP_EV_AUDIO_CONN_ESTABLISHED_IND	BTUINT16: the SCO connection handle.
BTSDK_HFP_EV_AUDIO_CONN_RELEASED_IND	BTUINT16: the SCO connection handle.
BTSDK_HFP_EV_EXTEND_CMD_IND	BTUINT8*: Points to the buffer contains the full extended AT command including the ending <cr>, or extended result code, including the starting and ending <cr><lf>.
BTSDK_HFP_EV_PRE_SCO_CONNECTION_IND	Btsdk_AGAP_PreSCOConnIndStru
BTSDK_HFP_EV_SIGNAL_STRENGTH_IND	BTUINT8: The signal strength value.
BTSDK_HFP_EV_BATTERY_CHARGE_IND	BTUINT8: Battery charge indicator value.
BTSDK_HFP_EV_CHLDHELD_ACTIVATED_IND	
BTSDK_HFP_EV_CHLDHELD_RELEASED_IND	
BTSDK_HFP_EV_MICVOL_CHANGED_IND	BTUINT8: The gain value of microphone.

BTSDK_HFP_EV_SPKVOL_CHANGED_IND	BTUINT8: The gain value of speaker.
BTSDK_HFP_EV_ATCMD_RESULT	Btsdk_HFP_ATCmdResultStru
BTSDK_HFP_EV_CLIP_IND	Btsdk_HFP_PhoneInfoStru
BTSDK_HFP_EV_CURRENT_CALLS_IND	Btsdk_HFP_CLCCInfoStru
BTSDK_HFP_EV_NETWORK_OPERATOR_IND	Btsdk_HFP_COPSInfoStru
BTSDK_HFP_EV_SUBSCRIBER_NUMBER_IND	Btsdk_HFP_PhoneInfoStru
BTSDK_HFP_EV_VOICETAG_PHONE_NUM_IND	Btsdk_HFP_PhoneInfoStru
BTSDK_HFP_EV_SIGNAL_STRENGTH_IND	BTUINT8: The signal strength value.
BTSDK_HFP_EV_BATTERY_CHARGE_IND	BTUINT8: Battery charge indicator value.
BTSDK_HFP_EV_HF_MANUFACTURERID_IND	BTUINT8* - Manufacturer ID of the AG device, a null-terminated ASCII string.
BTSDK_HFP_EV_HF_MODELID_IND	BTUINT8* - Model ID of the AG device, a null-terminated ASCII string.

Remarks

If not specified in the upper table, the event parameters shall be ignored.

Btsdk_HFP_ExtendCmd

Prototype	BTUINT32 Btsdk_HFP_ExtendCmd(BTCONNHDL hdl, void *cmd, BTUINT16 len, BTUINT32 timeout);	
Description	The **Btsdk_HFP_ExtendCmd** function is called to transmit the extended command to AG/HF device. If it is an AT command, the application will receive BTSDK_HFP_EV_ATCMD_RESULT event after the remote AG responds to the command or the specified time expires. If it is a result code, the application will receive no confirms.	
Parameters	*hdl*	[in] Handle to the HF connection to send the command.
	cmd	[in] Pointer to the AT command or result code to be transmitted. It shall be an AT command if local device acts as HF/HS in the specified connection, including the ending <cr>. E.g., "AT+CGMM\r".It can be any bytes stream if local device acts as AG in the specified connection.
	len	[in] Size of the content stored in the cmd buffer. If local device acts as HF/HS in the specified connection, the length of the AT command shall exclude the terminated null. E.g. strlen("AT+CGMM\r").
	timeout	[in] Specifies the maximum time, in seconds, the lower HF entity will wait for the response to this command. If the time expires before the remote AG response to the command, the command execution will be considered to have failed. If timeout is 0, a default time value will be adopted.
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.

6.3.6.1 Hands-free/Headset Audio Gateway (AG) Btsdk_AGAP_APPRegCbk4ThirdParty

Prototype	BTUINT32 Btsdk_AGAP_APPRegCbk4ThirdParty(Btsdk_HFP_Callback *pfunc);	
Description	The **Btsdk_AGAP_APPRegCbk4ThirdParty** function registers an application-defined callback function used to process Hands-free/Headset AG messages created by the BlueSoleil.	
Parameters	*pfunc*	[in] Pointer to the callback function of Btsdk_HFP_Callback type. If p*func* is NULL, BlueSoleil will remove the callback information registered before.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

All messages of both Hands-free AG and Headset AG from BlueSolei are transferred to the applications using the same callback function. That is, if the application calls *Btsdk_AGAP_APPRegCbk4ThirdParty* twice to register different callback functions, the second callback function will replace the first one.

Btsdk_AGAP_AnswerCall

Prototype	BTUINT32 Btsdk_AGAP_AnswerCall(BTCONNHDL hdl, BTUINT8 mode);	
Description	The **Btsdk_AGAP_AnswerCall** function informs the HF that the AG has answered the incoming call.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the call answered indication.
	mode	[in] Specify whether to setup SCO connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

The ***mode*** parameter can be one of these values

Value	Description
BTSDK_HFP_AG_PRIVATE_MODE	Do not setup SCO connection.
BTSDK_HFP_AG_HANDSFREE_MODE	Setup SCO connection.

Btsdk_AGAP_OriginateCall

Prototype	BTUINT32 Btsdk_AGAP_OriginateCall(BTCONNHDL hdl, BTUINT8 mode);	
Description	The **Btsdk_AGAP_OriginateCall** function informs the HF that the AG has originated a call. (This function can be called when the AG application starts to call the remote party after a successful voice recognition procedure.)	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the call answered indication.
	mode	[in] Specify whether to setup SCO connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

The ***mode*** parameter can be one of these values

Value	Description
BTSDK_HFP_AG_PRIVATE_MODE	Do not setup SCO connection.
BTSDK_HFP_AG_HANDSFREE_MODE	Setup SCO connection.

Btsdk_AGAP_CancelCall

Prototype	BTUINT32 Btsdk_AGAP_CancelCall(BTCONNHDL hdl, BTUINT8 type);	
Description	The **Btsdk_AGAP_CancelCall** function informs the HF that the AG has cancelled a call. (AG may reject an incoming call or terminate an outgoing call or release an ongoing call.)	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the call canceled indication.
	type	[in] Specifies the type of the call released.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

The ***type*** parameter can be one of these values

Value	Description
BTSDK_HFP_CANCELED_ALLCALL	AG has released all the existing calls (active, outgoing, waiting, holding).
BTSDK_HFP_CANCELED_CALLSETUP	AG has rejected a waiting call or terminated an outgoing call.
BTSDK_HFP_CANCELED_LASTCALL	AG has released the last active call.

Btsdk_AGAP_ChangeInbandRingSetting

Prototype	BTUINT32 Btsdk_AGAP_ChangeInbandRingSetting(BTCONNHDL hdl, BTUINT8 inband_ring);	
Description	The **Btsdk_AGAP_ChangeInbandRingSetting** function informs the HF device the new in-band ring tone setting.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the call cancelled indication.
	inband_ring	[in] Specify whether the AG will provide the in-band ring tones or not. 0 -- The AG won't provide the in-band ring tones. 1 -- The AG will provide the in-band ring tones.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_NetworkEvent

Prototype	BTUINT32 Btsdk_AGAP_NetworkEvent(BTCONNHDL hdl, BTUINT8 ev, void *param);	
Description	The **Btsdk_AGAP_NetworkEvent** function informs BlueSoleil that the AG application receives an event from the external network, e.g. a result code from the cellular network.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the network indication.
	ev	[in] Event identifier
	param	[in] event parameter.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

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

Value	Description
BTSDK_AGAP_NETWORK_RMT_IS_BUSY	The remote called party is already in communication. For example, the answer to the ATD command is BUSY.
BTSDK_AGAP_NETWORK_ALERTING_RMT	The remote called party is reached and being alerted. For example, the answer to the ATD command is “0” (OK).
BTSDK_AGAP_NETWORK_INCOMING_CALL	The AG application receives an incoming call from the network. For example, “RING” (may be followed by a “+CLIP<number>”) is received. *param* is a pointer to a buffer that contains a NULL terminated ASCII string that specifies the phone number if it is available. *param* shall be set to NULL if the phone number is unavailable.

BTSDK_AGAP_NETWORK_RMT_ANSWER_CALL	The AG application detects that the remote called party has answered the call. For example, the answer to the AT+CLCC command is “+CLCC: 0, 0, …”.
BTSDK_AGAP_NETWORK_LINK_NOT_ESTABLISHED	The remote called party can’t be reached or the remote called party hang-up the ongoing call. For example, the AG application receives NO ANSWER, NO CARRIER, or NO DIALTONE.
BTSDK_AGAP_NETWORK_SVC_UNAVAILABLE	The AG application detects that the network service is unavailable.
BTSDK_AGAP_NETWORK_SVC_AVAILABLE	The AG application detects that the network service is available.
BTSDK_AGAP_NETWORK_SIGNAL_STRENGTH	
BTSDK_AGAP_NETWORK_ROAMING_RESET	
BTSDK_AGAP_NETWORK_ROAMING_ACTIVE	

1. BTSDK_AGAP_NETWORK_RMT_IS_BUSY: Pointer to a BTUINT8 variable specifies which call is canceld by this busy event. Its value can be one of BTSDK_HFP_CANCELED_LASTCALL, BTSDK_HFP_CANCELED_CALLSETUP and BTSDK_HFP_CANCELED_CALLHELD).
The default value is BTSDK_HFP_CANCELED_LASTCALL if param is set to NULL.

2. BTSDK_AGAP_NETWORK_INCOMING_CALL: Pointer to a Btsdk_HFP_PhoneInfoStru structure contains the phone number.

3. BTSDK_AGAP_NETWORK_SIGNAL_STRENGTH: Pointer to a BTUINT8 variable specifies the signal strength. Its range is from 0 to 5.

4. For all the other events, the param is ignored and should be NULL.

Btsdk_AGAP_VoiceRecognitionReq

Prototype	BTUINT32 Btsdk_AGAP_VoiceRecognitionReq(BTCONNHDL hdl, BTUINT8 param);	
Description	The **Btsdk_AGAP_VoiceRecognitionReq** function informs the HF device that AG has activated or deactivated the voice recognition.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to attach the voice tag.
	param	[in] 1=enable, 0=disable.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_VoiceTagPhoneNumRsp

Prototype	BTUINT32 Btsdk_AGAP_VoiceTagPhoneNumRsp(BTCONNHDL hdl, void *phone_num, BTUINT8 len);	
Description	The **Btsdk_AGAP_VoiceTagPhoneNumRsp** function specifies a phone number to be attached to a voice tag in the HF side.	
Parameters	hdl	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	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.	

Remarks

Btsdk_AGAP_DialRsp

Prototype	BTUINT32 Btsdk_AGAP_DialRsp(BTCONNHDL hdl, BTUINT8 err_code);	
Description	The **Btsdk_AGAP_DialRsp** function responds the AG dialing status, which HF device requested in ATD, ATD> and AT+BLDN.	
Parameters	hdl	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	status	[in] Specifies the result of dialing operation. It shall be one of BTSDK_HFP_OK, CME error codes and standard error result codes. BTSDK_HFP_OK - The operation is successful. Otherwise, CME error codes or standard error result codes.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_HoldIncomingCall

Prototype	BTUINT32 Btsdk_AGAP_HoldIncomingCall(BTCONNHDL hdl);	
Description	The **Btsdk_AGAP_HoldIncomingCall** function informs HF device that AG has put the incoming call on hold.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_AcceptHeldIncomingCall

Prototype	BTUINT32 Btsdk_AGAP_AcceptHeldIncomingCall(BTCONNHDL hdl, BTUINT8 mode);	
Description	The **Btsdk_AGAP_AcceptHeldIncomingCall** function informs HF device that AG has accepted the held incoming call.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	mode	[in] Specify whether to setup SCO connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

The ***mode*** parameter can be one of these values

Value	Description
BTSDK_HFP_AG_PRIVATE_MODE	Do not setup SCO connection.
BTSDK_HFP_AG_HANDSFREE_MODE	Setup SCO connection.

Btsdk_AGAP_RejectHeldIncomingCall

Prototype	BTUINT32 Btsdk_AGAP_RejectHeldIncomingCall(BTCONNHDL hdl);	
Description	The **Btsdk_AGAP_RejectHeldIncomingCall** function informs HF device that AG has rejected the held incoming call.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_NetworkOperatorRsp

Prototype	BTUINT32 Btsdk_AGAP_NetworkOperatorRsp(BTCONNHDL hdl, PBtsdk_HFP_COPSInfoStru op_info);	
Description	The **Btsdk_AGAP_NetworkOperatorRsp** function is called to respond to the AT+COPS? command..	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	op_info	[in] Pointer to the Btsdk_HFP_COPSInfoStru structure contains the operator information. If the operator information is unavailable, op_info shall be NULL.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_SubscriberNumberRsp

Prototype	BTUINT32 Btsdk_AGAP_SubscriberNumberRsp(BTCONNHDL hdl, PBtsdk_HFP_PhoneInfoStru usr_info, BTUINT8 complete);	
Description	The **Btsdk_AGAP_SubscriberNumberRsp** function is called to respond to the AT+CNUM command. If there are multiple subscriber numbers available, this function shall only be called once for a number.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	usr_info	[in] Pointer to the Btsdk_HFP_PhoneInfoStru structure containing one subscriber number's information. The type, service, num_len and number member of this structure shall be set to the proper value. All the other members are ignored.
	complete	[in] Specify whether it is the last subscriber number. 0 - There are still more numbers to be sent. 1 - This is the last number. "\r\nOK\r\n" won't be sent to the HF device until complete is set to 1.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

If no subscriber number information is available, usr_info shall be set to NULL and complete shall be set to 1.

Btsdk_AGAP_CurrentCallRsp

Prototype	BTUINT32 Btsdk_AGAP_CurrentCallRsp(BTCONNHDL hdl, PBtsdk_HFP_CLCCInfoStru call_info, BTUINT8 complete);	
Description	The **Btsdk_AGAP_CurrentCallRsp** function is called to respond to the AT+CLCC command.If there are multiple concurrent calls, this function shall only be called once for a call.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	call_info	[in] Pointer to the Btsdk_HFP_CLCCInfoStru structure containing information of one of the current call.
	complete	[in] Specify whether it is the last available call. 0 - There are still more is called to be sent. 1 - This is the last call. "\r\nOK\r\n" won't be sent to the HF device until is_last is set to 1.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

If no calls are available, call_info shall be set to NULL and complete shall be set to 1.

Btsdk_AGAP_ManufacturerIDRsp

Prototype	BTUINT32 Btsdk_AGAP_ManufacturerIDRsp(BTCONNHDL hdl, BTINT8 *mid, BTUINT16 len);	
Description	The **Btsdk_AGAP_ManufacturerIDRsp** function is called to transmit response to AT+CGMI command.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	mid	[in] Pointer to the buffer containing the manufacturer identification. It shall be an ASCII text string.
	len	[in] Specify the length, in bytes, of the string pointed to by the mid, not including the terminated NULL.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_ ModelIDRsp

Prototype	BTUINT32 Btsdk_AGAP_ ModelIDRsp (BTCONNHDL hdl, BTINT8 *mid, BTUINT16 len);	
Description	The **Btsdk_AGAP_ ModelIDRsp** function is called to transmit response to AT+CGMM command.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	mid	[in] Pointer to the buffer contains the model identification.It shall be an ASCII text string.
	len	[in] Specify the length, in bytes, of the string pointed to by the mid, not included the terminated NULL.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_SendBatteryChargeIndicator

Prototype	BTUINT32 Btsdk_AGAP_SendBatteryChargeIndicator(BTCONNHDL hdl, BTUINT8 indicator);	
Description	The **Btsdk_AGAP_SendBatteryChargeIndicator** function is called to transmit current battery charge indicator.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	indicator	[in] Specify the current battery charge indicator value. Range: 0 - 5.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_ SendErrorMessage

Prototype	BTUINT32 Btsdk_AGAP_SendErrorMessage(BTCONNHDL hdl, BTUINT8 err_code);	
Description	The **Btsdk_AGAP_SendErrorMessage** function is called to transmit "+CME ERROR" result code to the HF.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	err_code	[in] Specify the error code. It shall be one of CME error codes.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_SetSpkVol

Prototype	BTUINT32 Btsdk_AGAP_SetSpkVol(BTCONNHDL hdl, BTUINT8 spk_vol);	
Description	The **Btsdk_AGAP_SetSpkVol** function is called to set the speaker volume of HF device..	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	spk_vol	[in] The speaker volume level. Range: 0 - 15
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_SetMicVol

Prototype	BTUINT32 BTUINT32 Btsdk_AGAP_SetMicVol(BTCONNHDL hdl, BTUINT8 mic_vol);	
Description	The **Btsdk_AGAP_SetMicVol** function is called to set the microphone volume of HF device.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	mic_vol	[in] The microphone volume level. Range: 0 - 15
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_SetCurIndicatorVal

Prototype	BTUINT32 Btsdk_AGAP_SetCurIndicatorVal(BTCONNHDL hdl, PBtsdk_HFP_CINDInfoStru indicators);	
Description	The **Btsdk_AGAP_SetCurIndicatorVal** function sets the current call/service indicator value in order to synchronize the state with the HF during the service level connection establishing procedure with the HF.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	indicators	[in] Pointer to the Btsdk_HFP_CINDInfoStru containing the current value of the HFP defined indicators.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_AudioConnTrans

Prototype	BTUINT32 Btsdk_AGAP_AudioConnTrans(BTCONNHDL hdl);	
Description	The **Btsdk_AGAP_AudioConnTrans** function transfers the audio path of the ongoing call from or towards the HF.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to transfer the audio connection.
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 AG and the HF, 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.

Btsdk_AGAP_GetAGState

Prototype	BTUINT32 Btsdk_AGAP_GetAGState(BTUINT16* agstate);	
Description	The **Btsdk_AGAP_GetAGState** function gets current AG's state.	
Parameters	*agstate*	[out] Pointer to the variable which indicates current AG's state.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

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

Value	Description
BTSDK_AGAP_ST_IDLE	Before service level connection is established.
BTSDK_AGAP_ST_STANDBY	Service level connection is established.
BTSDK_AGAP_ST_RINGING	Ringing state.
BTSDK_AGAP_ST_OUTGOINGCALL	Outgoing call state.
BTSDK_AGAP_ST_ONGOINGCALL	Ongoing call state.
BTSDK_AGAP_ST_BVRA	Voice recognition is ongoing.
BTSDK_AGAP_ST_VOVG	SCO link doesn't exist between AG and HF while a call is ongoing.
BTSDK_AGAP_ST_HELDINCOMINGCALL	the incoming call is held
BTSDK_AGAP_ST_THREEWAYCALLING	three way calling

Remarks

Btsdk_AGAP_CurrentCallSync

Prototype	BTUINT32 Btsdk_AGAP_CurrentCallSync(BTCONNHDL hdl, PBtsdk_HFP_CLCCInfoStru call_info, BTUINT8 complete);	
Description	The **Btsdk_AGAP_CurrentCallSync** function is used to tell the lower HFP AG module current existing phone calls. It is different from the Btsdk_AGAP_CurrentCallRsp that it would not send any result code to the remote HF device. If there are multiple concurrent calls, this function shall only be called once for a call. If no calls are available, *call_info* shall be set to NULL and the *complete* shall be set to 1.	
Parameters	*hdl*	[in] Handle to the local HFP AG entity.
	call_info	[in] Pointer to the Btsdk_HFP_CLCCInfoStru structure contains information of one of the current call.
	complete	[in] Specify whether it is the last available call. 0 - There are still more calls to be synchronize. 1 - This is the last call.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_3WayCallingHandler

Prototype	BTUINT32 Btsdk_AGAP_3WayCallingHandler(BTCONNHDL hdl, BTUINT16 op_code, BTUINT8 idx);	
Description	The **Btsdk_AGAP_3WayCallingHandler** function calls by the AG application to notify the HF current 3way-calling status. It is called after the AG application sends AT+CHLD=<n> command to the mobile network. The application should first call Btsdk_AGAP_CurrentCallSync to synchronize with the lower HFP AG module with the current available call list. Then it sends AT+CHLD=<n> command to the mobile network. After it got "OK" from the network, it calls Btsdk_HFAP_3WayCallingHandler.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to handle the 3way-calling.
	op_code	[in] Operation code for the 3way-calling. It can be one of the: BTSDK_HFP_CMD_CHLD_0, BTSDK_HFP_CMD_CHLD_1, BTSDK_HFP_CMD_CHLD_2, BTSDK_HFP_CMD_CHLD_3, BTSDK_HFP_CMD_CHLD_4.
	idx	[in] Specify the call to be handled separately. If op_code is one of BTSDK_HFP_CMD_CHLD_0, BTSDK_HFP_CMD_CHLD_3 and BTSDK_HFP_CMD_CHLD_4, idx is ignored and shall be set to 0. If op_code is BTSDK_HFP_CMD_CHLD_1, a none-zero idx specify the call to released; a zero idx force to release all active calls if any exist. If op_code is BTSDK_HFP_CMD_CHLD_2, a none-zero idx specify the call not to be placed on hold; a zero idx force to hold all active calls if any exist.
Return:	BTSDK_TRUE if the specified status is set. BTSDK_FALSE if the specified status is not set.	

Remarks

The op_code along with idx determines the AT command the application sends to the mobile network.

The ***op_code*** can be one of these values.

Value	idx	AT Command
BTSDK_HFP_CMD_CHLD_0	0	AT+CHLD=0
BTSDK_HFP_CMD_CHLD_1		AT+CHLD=1
BTSDK_HFP_CMD_CHLD_1	>0	AT+CHLD=1<idx>
BTSDK_HFP_CMD_CHLD_2	0	AT+CHLD=2
BTSDK_HFP_CMD_CHLD_2	>0	AT+CHLD=2<idx>
BTSDK_HFP_CMD_CHLD_3	0	AT+CHLD=3
BTSDK_HFP_CMD_CHLD_4	0	AT+CHLD=4

Btsdk_AGAP_IsAudioConnExisted

Prototype	BTUINT32 Btsdk_AGAP_IsAudioConnExisted(BTBOOL* audioconn);	
Description	The **Btsdk_AGAP_IsAudioConnExisted** function judges whether SCO connection is established.	
Parameters	*audioconn*	[out] status of SCO connection BTSDK_TRUE: SCO is established BTSDK_FALSE: SCO is released
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_AGAP_SetDialHandlerFlag

Prototype	BTBOOL Btsdk_AGAP_SetDialHandlerFlag (BTBOOL bFlag);	
Description	The **Btsdk_AGAP_SetDialHandlerFlag** function sets whether the HFAG dial indication is handled by user application or not.	
Parameters	*bFlag*	[in] status of handling HFAG dial indication BTSDK_TRUE: Applications need to handle HFAG dial indication. BTSDK_FALSE: Applications don't care about HFAG dial indication anymore.
Return:	If the function succeeds, the return value is BTSDK_TRUE. If the function fails, the return value is BTSDK_FALSE.	

Remarks

This function must be called immediately after the callback event BTSDK_APP_EV_AGAP_HF_AVAILABLE_IND is received by application, and it must be called every time after this callback event is received.

After calling this function, when the BTSDK_APP_EV_AGAP_HF_LASTNUM_REDIAL_IND, BTSDK_APP_EV_AGAP_HF_MEM_DIAL_IND or BTSDK_APP_EV_AGAP_HF_DIAL_IND is received, the function **Btsdk_DialRsp** has to be called to imform the dialing result to BlueSoleil.

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

Btsdk_HFAP_APPRegCbk4ThirdParty

Prototype	void Btsdk_HFAP_APPRegCbk4ThirdParty (Btsdk_HFP_Callback *pfunc);	
Description	The **Btsdk_HFAP_APPRegCbk4ThirdParty** function registers an application-defined callback function used to process HF/HS messages created by the BlueSoleil.	
Parameters	*pfunc*	[in] Pointer to the callback function of Btsdk_HFP_Callback type. If p*func* is NULL, BlueSoleil will remove the callback information registered before.
Return:		

Remarks

All messages of both HF and HS from BlueSoleil are transferred to the application using the same callback function. That is, if the application calls *Btsdk_HFAP_APPRegCbk4ThirdParty* twice to register different callback functions, the second callback function will replace the first one.

Btsdk_HFAP_AnswerCall

Prototype	BTUINT32 Btsdk_HFAP_AnswerCall(BTCONNHDL hdl);	
Description	The **Btsdk_HFAP_AnswerCall** function informs the AG that the HF has been answered the incoming call.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to answer the call.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_CancelCall

Prototype	BTUINT32 Btsdk_HFAP_CancelCall(BTCONNHDL hdl);	
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	hdl	[in] Handle to the HFP connection with a remote AG that is to cancel the call.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_LastNumRedial

Prototype	BTUINT32 Btsdk_HFAP_LastNumRedial(BTCONNHDL hdl);	
Description	The **Btsdk_HFAP_LastNumRedial** function instructs the AG to redial the last dialed number.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to dial the number.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_MemNumDial

Prototype	BTUINT32 Btsdk_HFAP_MemNumDial(BTCONNHDL hdl, 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	hdl	[in] Handle to the HFP connection with a remote AG that is to dial the number.
	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.	

Remarks

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 provided phone number.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote AG that is to dial the number.
	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.	

Remarks

Btsdk_HFAP_VoiceRecognitionReq

Prototype	BTUINT32 Btsdk_HFAP_VoiceRecognitionReq(BTCONNHDL hdl, BTUINT8 param);	
Description	The **Btsdk_HFAP_VoiceRecognitionReq** function requests the AG to activate or deactivate the voice recognition procedure.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote AG that is to activate or deactivate voice recognition.
	param	[in] 1=enable, 0=disable.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Btsdk_HFAP_3WayCallingHandler

Prototype	BTUINT32 Btsdk_HFAP_3WayCallingHandler(BTCONNHDL hdl, BTUINT16 op_code, BTUINT8 idx);	
Description	The **Btsdk_HFAP_3WayCallingHandler** function is called to handle the 3way-calling. It sends AT+CHLD=<n> command to the remote AG. <n> is determined by the values of op_code and idx.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote AG that is to handle the 3way-calling.
	op_code	[in] Specify the call to be handled separately. If op_code is one of BTSDK_HFP_CMD_CHLD_0, BTSDK_HFP_CMD_CHLD_3 and BTSDK_HFP_CMD_CHLD_4, idx is ignored and shall be set to 0. If op_code is BTSDK_HFP_CMD_CHLD_1, a none-zero idx specify the call to released; a zero idx force to release all active calls if any exist. If op_code is BTSDK_HFP_CMD_CHLD_2, a none-zero idx specify the call not to be placed on hold; a zero idx force to hold all active calls if any exist.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

The ***op_code*** of this function is the similar as the one of Btsdk_HFAP_3WayCallingHandler.

Btsdk_HFAP_DisableNREC

Prototype	BTUINT32 Btsdk_HFAP_DisableNREC(BTCONNHDL hdl);	
Description	The **Btsdk_HFAP_DisableNREC** function to request AG to disable NREC function.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote AG that is to disable NREC function.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_TxDTMF

Prototype	BTUINT32 Btsdk_HFAP_TxDTMF(BTCONNHDL hdl, BTUINT8 chr);	
Description	The **Btsdk_HFAP_TxDTMF** function is called to instruct AG to transmit the specific DTMF code.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote AG that is to transmit the DTMF code.
	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.	

Btsdk_HFAP_SetSpkVol

Prototype	BTUINT32 Btsdk_HFAP_SetSpkVol(BTCONNHDL hdl, BTUINT8 spk_vol);	
Description	The **Btsdk_HFAP_SetSpkVol** function informs the remote AG that the speaker volume of the HF has been changed.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to set the speaker volume.
	spk_vol	[in] The current 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.	

Btsdk_HFAP_SetMicVol

Prototype	BTUINT32 Btsdk_HFAP_SetMicVol(BTCONNHDL hdl, BTUINT8 mic_vol);	
Description	The **Btsdk_HFAP_SetMicVol** function is called to inform AG that the microphone volume of HF device has been changed.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to set the microphone volume.
	mic_vol	[in] The current micphone 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.	

Btsdk_HFAP_VoiceTagPhoneNumReq

Prototype	BTUINT32 Btsdk_HFAP_VoiceTagPhoneNumReq(BTCONNHDL hdl);	
Description	The **Btsdk_HFAP_VoiceTagPhoneNumReq** function is called to request AG to enter a phone number to be attached to the HF device's voice-tag, which is used for voice recognition. The phone number will be returned by the BTSDK_HFP_EV_VOICETAG_PHONE_NUM_IND event.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote AG that is to attach the voice tag.
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.

Btsdk_ HFAP_GetManufacturerID

Prototype	BTUINT32 Btsdk_HFAP_GetManufacturerID(BTCONNHDL hdl, BTUINT8 *manufacturer_id, BTUINT16 *id_len);	
Description	The **Btsdk_ HFAP_ManufacturerIDRsp** function is called to get the manufacturer ID information.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote AG that is to get the manufacturer ID.
	manufacturer_id	[out]: The buffer to store manufacture ID, If it is NULL, the *id_len should be set as 0; Otherwise, the *id_len specifies the manufacturer_id buffer size
	id_len	[in]: The size of the manufacturer_id buffer [out]: The real length of the manufacturer ID, including the terminated null character is returned.
Return:	If manufacturer ID is got, the return value is BTSDK_OK. If the return value is BTSDK_ER_FUNCTION_NOTSUPPORT, it indicates that the function failed to get the manufacturer ID of the AG.	

Remarks

Btsdk_ HFAP_GetModelID

Prototype	BTUINT32 Btsdk_HFAP_GetModelID(BTCONNHDL hdl, BTUINT8 *model_id, BTUINT16 *id_len);	
Description	The **Btsdk_HFAP_GetModelID** function is called to get the model nformation.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote HF that is to send the indication.
	model_id	[out] The buffer used to store model D, If it is NULL, the *id_len should be set as 0; Otherwise, the *id_len specifies the model_id buffer size.
	id_len	[in/out] The size of the model_id buffer (Input). On output, returns the real length of the model ID, including the terminated null charater.
Return:	If model ID is got, the return value is BTSDK_OK. If the return value is BTSDK_ER_FUNCTION_NOTSUPPORT, it indicates that the function failed to get the model ID of the AG.	

Remarks

Btsdk_HFAP_AudioConnTrans

Prototype	BTUINT32 Btsdk_HFAP_AudioConnTrans(BTCONNHDL hdl)	
Description	The **Btsdk_HFAP_AudioConnTrans** function is called to transfer the audio connection.	
Parameters	*hdl*	[in] Handle to the HFP connection with a remote AG that is to transfer the audio connection.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_NetworkOperatorReq

Prototype	BTUINT32 Btsdk_HFAP_NetworkOperatorReq(BTCONNHDL hdl);	
Description	The **Btsdk_HFAP_NetworkOperatorReq** function is called to request for the network operator name of the AG device.The operator name will be returned by the BTSDK_HFP_EV_NETWORK_OPERATOR_IND event.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to get the operator name.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_SetExtendedErrors

Prototype	BTUINT32 Btsdk_HFAP_SetExtendedErrors(BTCONNHDL hdl, BTUINT8 enable);	
Description	The **Btsdk_HFAP_SetExtendedError**s function is called to enable the Extended Audio Gateway Error Result Code in the AG.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to enable the extended error result code.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_GetResponseHoldStatus

Prototype	BTUINT32 Btsdk_HFAP_GetResponseHoldStatus(BTCONNHDL hdl)	
Description	The **Btsdk_HFAP_GetResponseHoldStatus** function is called to query the current Response and Hold status of the AG. If the AG responds with "+BTRH:0", the application will be informed by the BTSDK_HFP_EV_CALLHELD_IND event.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to query the Response and Hold status.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_HoldIncomingCall

Prototype	BTUINT32 Btsdk_HFAP_HoldIncomingCall(BTCONNHDL hdl)	
Description	The **Btsdk_HFAP_HoldIncomingCall** function is called to inform the AG to hold the incoming call.If the AG responds with "+BTRH:0", the application will be informed by the BTSDK_HFP_EV_CALLHELD_IND event.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to hold the incoming call.
Return:	BTSDK_OK if the request is sent to the AG. Other for error code. No events are generated in this case.	

Remarks

Btsdk_HFAP_AcceptHeldIncomingCall

Prototype	BTUINT32 Btsdk_HFAP_AcceptHeldIncomingCall(BTCONNHDL hdl)	
Description	The **Btsdk_HFAP_AcceptHeldIncomingCall** function is called to inform the AG to accept the held incoming call. If the AG responds with "+BTRH:1", the application will be informed by the BTSDK_HFP_EV_ONGOINGCALL_IND event.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to accept the held call.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_RejectHeldIncomingCall

Prototype	BTUINT32 Btsdk_HFAP_RejectHeldIncomingCall(BTCONNHDL hdl);	
Description	The **Btsdk_HFAP_RejectHeldIncomingCall** function is called to inform the AG to reject the held incoming call. If the AG responds with "+BTRH:2", the application will be informed by the BTSDK_HFP_EV_STANDBY_IND event.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to reject the held call.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_GetSubscriberNumber

Prototype	BTUINT32 Btsdk_HFAP_GetSubscriberNumber(BTCONNHDL hdl)	
Description	The **Btsdk_HFAP_GetSubscriberNumber** function is called to get the subscriber number information of AG. The subscriber number will be returned by the BTSDK_HFP_EV_SUBSCRIBER_NUMBER_IND event. If the AG responds with one of "OK", "ERROR" and "+CMER:" or the local timer expired before receiving the upper result code from the AG, the event BTSDK_HFP_EV_ATCMD_RESULT is reported to the application.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to get the number.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_GetCurrentCalls

Prototype	BTUINT32 Btsdk_HFAP_GetCurrentCalls(BTCONNHDL hdl)	
Description	The **Btsdk_HFAP_GetCurrentCalls** function is called to query the list of current calls. Information of each existing call will be returned by a BTSDK_HFP_EV_CURRENT_CALLS_IND event.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to get the call list.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_GetAGFeatures

Prototype	BTUINT32 Btsdk_HFAP_GetAGFeatures(BTCONNHDL hdl)	
Description	The **Btsdk_HFAP_GetAGFeatures** function is called to query the features of the remote AG.	
Parameters	hdl	[in] Handle to the HFP connection with a remote AG that is to get the features.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks

Btsdk_HFAP_GetCurrHFState

Prototype	BTUINT32 Btsdk_HFAP_GetCurrHFState (BTUINT16 *agstate);	
Description	The **Btsdk_HFAP_GetCurrHFState** function gets current state of Hands-free device.	
Parameters	*agstate*	[out] A pointer to the variable which indicates current Hands-free device's state.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

The ***agstate*** can be one of these values.

Value	Description
BTSDK_HFAP_ST_IDLE	Before service level connection is established.
BTSDK_HFAP_ST_STANDBY	Service level connection is established.
BTSDK_HFAP_ST_RINGING	Ringing state.
BTSDK_HFAP_ST_OUTGOINGCALL	Outgoing call state.
BTSDK_HFAP_ST_ONGOINGCALL	Ongoing call state.
BTSDK_HFAP_ST_BVRA	voice recognition is ongoing
BTSDK_HFAP_ST_VOVG	SCO link doesn't exist between AG and HF while a call is ongoing.
BTSDK_HFAP_ST_HELDINCOMINGCALL	the incoming call is held

Remarks

Btsdk_HFAP_SetWaveInDevice

Prototype	BTBOOL Btsdk_HFAP_SetWaveInDevice(BTUINT8* pWaveInDevice, BTUINT32 devNamelen);	
Description	The **Btsdk_HFAP_SetWaveInDevice** function sets wavein audio device for Handsfree application.	
Parameters	*pWaveInDevice*	[in] A pointer to the buffer that contains the wavein audio device name. If this parameter is NULL, the default audio device will be opened
	devNamelen	[in] Specifies the size in bytes of the string pointed to by the *pWaveInDevice* parameter.
Return:	If the function succeeds, the return value is BTSDK_TRUE. If the function fails, the return value is BTSDK_FALSE.	

Remarks

This function can be called to set wavein audio device in advance before establishing HF connection. It is also can be called to dynamically switch wavein audio device after establishing SCO link.

This funciton does not set the wavein audio device specified by pWaveInDevice as the default audio device.

Btsdk_HFAP_SetWaveOutDevice

Prototype	BTBOOL Btsdk_HFAP_SetWaveOutDevice(BTUINT8* pWaveOutDevice, BTUINT32 devNamelen);	
Description	The **Btsdk_HFAP_SetWaveInDevice** function sets waveout audio device for Handsfree application.	
Parameters	*pWaveOutDevice*	[in] A pointer to the buffer that contains the waveout audio device name. If this parameter is NULL, the default audio device will be opened.
	devNamelen	[in] Specifies the size in bytes of the string pointed to by the *pWaveOutDevice* parameter.
Return:	If the function succeeds, the return value is BTSDK_TRUE. If the function fails, the return value is BTSDK_FALSE.	

Remarks

This function can be called to set waveout audio device in advance before establishing HF connection. It is also can be called to dynamically switch waveout audio device after establishing SCO link.

This funciton does not set the wavein audio device specified by *pWaveOutDevice* as the default audio device.

6.3.7 Advancecd Audio Distribute Profile

6.3.7.1 A2DP Source

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.	
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 then add a new A2DP SRC service record.

Btsdk_UnregisterA2DPSRCService

Prototype	BTUINT32 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

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

6.3.7.2 A2DP Sink

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.	
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 null-terminated 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 then add a new A2DP SNK service record.

Btsdk_UnregisterA2DPSNKService

Prototype	BTUINT32 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

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

6.3.8 Human Interface Device Profile

Btsdk_Hid_ClntUnPluggedDev

Prototype	BTUINT32 Btsdk_Hid_ClntUnPluggedDev(BTUINT8 * bdaddr);	
Description	The **Btsdk_Hid_ClntUnPluggedDev** function unplugs hid device.	
Parameters	*bdaddr*	[in] Pointer to the remote Bluetooth device address.
Return:	If the function succeeds, the return value is BTSDK_OK. If the function fails, the return value is an error code.	

Remarks