IVT BTSDK

API Reference

(Part I)

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

Revision History

Version	Date	Comments
2008.10.18	Oct. 10th, 2008	Initial version.
2009.06.03	Jun. 3th, 2009	Add APIs for Mutil Clients support. Add C/S new features description.

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

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

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

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

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

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

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

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

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

1.2 Overview of IVT BTSDK

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

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

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

- Provide access to different Bluetooth profiles.

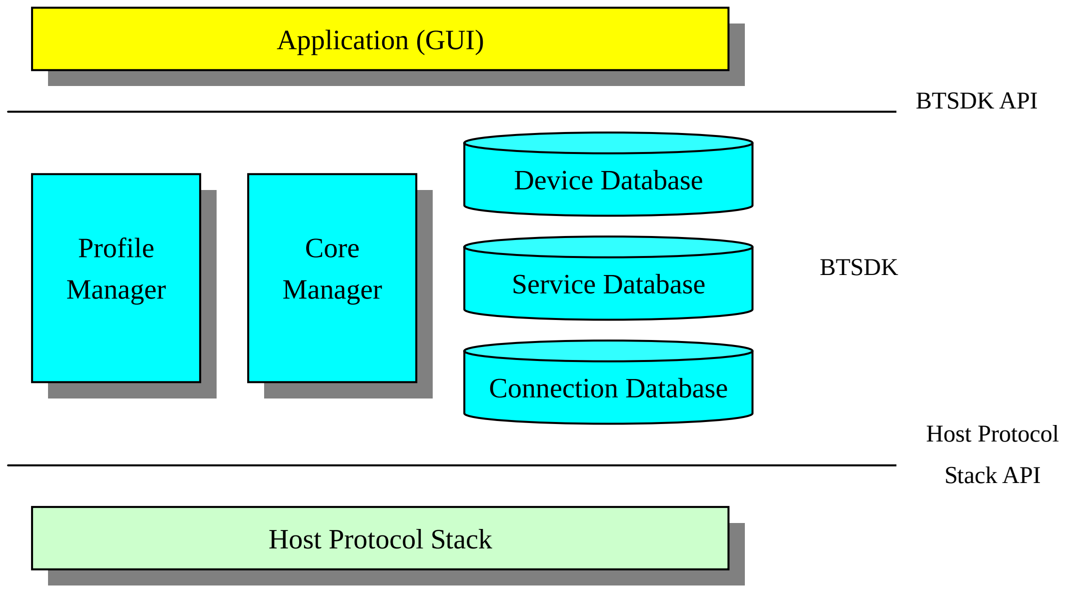

Figure 1: IVT BTSDK Structure

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

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

2. Develop Notes

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

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

3. BTSDK Data Types

The data types supported by IVT BTSDK 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.

4. Constant Reference

4.1 Error Codes

The following table provides a list of error codes. They are returned by many BTSDK 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 SDK database.
BTSDK_ER_OPERATION_FAILURE	0X0302	The operation fails for an undefined reason.
BTSDK_ER_SDK_UNINIT	0X0303	BTSDK 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 BTSDK.
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: BTSDK Error Codes.

Regarding other error definitions, please refer to the corresponding Profile Specific API document.

4.2 Service Class Identifier

The following table provides a list of class identifiers of services supported by current version BTSDK. 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_INTERCOM	0X1110	Intercom service.
BTSDK_CLS_FAX	0X1111	Fax service.
BTSDK_CLS_HEADSET_AG	0X1112	Headset Audio Gateway service.
BTSDK_CLS_PAN_PANU	0X1115	PANU service.
BTSDK_CLS_PAN_NAP	0X1116	NAP service.
BTSDK_CLS_PAN_GN	0X1117	GN 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_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_PNP_INFO	0X1200	Bluetooth Device Identification.

Table 2: BTSDK Service Class Identifiers.

4.3 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
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_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: BTSDK 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_POSITION	0x010000	Positioning (Location Identification).
BTSDK_SRVCLS_NETWORK	0x000100	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: BTSDK 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.

4.4 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 discoverability mode.
BTSDK_LIMITED_DISCOVERABLE	Sets the device into limited discoverable mode. If this value is specified, BTSDK_GENERAL_DISCOVERABLE mode value is ignored by BTSDK.
BTSDK_DISCOVERABLE	Makes the device discoverable. This is equivalent to BTSDK_GENERAL_DISCOVERABLE.
BTSDK_CONNECTABLE	Makes the device connectable. This is the default connectability mode.
BTSDK_PAIRABLE	Makes the device pairable. This is the default pairable mode.

Table 5: Bluetooth Device Modes

4.5 Bluetooth Security

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

Name	Description
BTSDK_SECURITY_LOW	Bluetooth security mode 1, that is, non-secure mode.
BTSDK_SECURITY_MEDIUM	Bluetooth security mode 2, that is, service level enforced security. This is the default security mode adopted by IVT BTSDK.
BTSDK_SECURITY_HIGH	Bluetooth security mode 3, that is, link level enforced security.
BTSDK_SECURITY_ENCRYPT_MODE1	Bluetooth security mode 3 along with encryption required for all connections.

Table 6: Bluetooth Security Modes

The following table provides a list of flags that specify the service security levels.

Name	Description
BTSDK_SSL_NO_SECURITY	The service requires no security protection.
BTSDK_SSL_AUTHENTICATION	The service requires authentication.
BTSDK_SSL_AUTHORIZATION	The service requires authorization.
BTSDK_SSL_ENCRYPTION	The service requires encryption.

Table 7: Bluetooth Service Security Levels

The following table provides a list of flags that specify how IVT BTSDK processes authorization request for an un-trusted device.

Name	Description
BTSDK_AUTHORIZATION_ACCEPT	Accept the authorization request always.
BTSDK_AUTHORIZATION_REJECT	Reject the authorization request always.
BTSDK_AUTHORIZATION_PROMPT	Report a BTSDK_AUTHORIZATION_IND message to the application and let the application make the decision.

Table 8: Bluetooth Authorization Method

4.6 Messages from BTSDK to the Application

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

Message Name	Callback Function Type	Description
BTSDK_PIN_CODE_REQ_IND	Btsdk_Pin_Req_Ind_Func	This message indicates the application to input PIN code for the specified device.
BTSDK_LINK_KEY_REQ_IND	Btsdk_Link_Key_Req_Ind_Func	This message indicates the application to input link key for the specified device.
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.
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_AUTHORIZATION_IND	Btsdk_Authorization_Req_Ind_Func	This message indicates that a remote device is trying to access a local service.
BTSDK_AUTHORIZATION_ABORT_IND	Btsdk_Author_Abort_Ind_Func	This message indicates that the authorization request is aborted due to link lost.
BTSDK_CONNECTION_EVENT_IND	Btsdk_Connection_Event_Ind_Func	This message indicates that a high-level protocol connection is created or disconnected.
BTSDK_VENDOR_EVENT_IND	Btsdk_Vendor_Event_Ind_Func	This message indicates that a vendor specific

		notification is received.

Table 9: Messages from BTSDK to the Application

5. Data Structures

5.1 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 the callback function to process. It also specifies the prototype of the callback function. It can be one of the values listed in Table 9.
	Func	Pointer to the callback function. If *func* is NULL, BTSDK will remove the callback.

Remarks

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

5.2 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).

5.3 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 to by the *param* element.
	Param	Pointer to the buffer contains 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:
   {0x01, 0xFC, 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);
}
```

5.4 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 receives 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 BTSDK function call returns BTSDK_ER_BUFFER_NOT_ENOUGH and *param_len* is set to the actual size required by BTSDK.

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);
}
```

5.5 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.

5.6 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.

5.7 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.

5.8 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.

5.9 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.

5.10 BtSdkLocalServerAttrStru

Definition	typedef struct _BtSdkLocalServerAttrStru
	{
	    BTUINT16 mask;
	    BTUINT16 service_class;
	    BTUINT8 svc_name[BTSDK_SERVICENAME_MAXLENGTH];
	    BTUINT16 security_level;
	    BTUINT16 author_method;
	    BTLPVOID ext_attributes;
	    BTUINT32 app_param;
	} BtSdkLocalServerAttrStru, *PBtSdkLocalServerAttrStru;	

Description	The structure BtSdkLocalServerAttrStru contains information about a local service record.	
Members	*mask*	A set of flags specifies members to retrieve or set.
	service_class	Type of the service record. It can be one of the values listed in the Table 2. This member must be specified when the application calls the function Btsdk_AddServer to add a new service record.
	svc_name	User-friendly name of this service record. This string is coded in UTF-8 format. Set *mask* to BTSDK_LSAM_SERVICENAME to use *svc_name*.
	security_level	A set of flags specifies the security requirements of this service record. It can be one or more of the values listed in Table 7. Set *mask* to BTSDK_LSAM_SECURITYLEVEL to use *security_level*.
	author_method	Specifies how IVT BTSDK processes authorization request for an un-trusted device when BTSDK_SSL_AUTHORIZATION is set to the *security_level* member. It can be one of the values listed in Table 8. Set *mask* to BTSDK_LSAM_AUTHORMETHOD to use *author_method*.
	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_LSAM_EXTATTRIBUTES to use *ext_attributes*. When this structure is used in a "Set" operation, e.g. Btsdk_AddServer, it can be set to NULL if it is not specified as mandatory in the following table; Otherwise it must be a valid structure pointer value. When this structure is used in a "Get" operation, e.g. Btsdk_GetServerAttributes, it must be set to NULL.

	app_param	Application defined value associated with the service record. Set *mask* to BTSDK_LSAM_APPPARAM to use *app_param*.

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

Value	Description
BTSDK_LSAM_SERVICENAME	Retrieves or sets the *svc_name* member.
BTSDK_LSAM_SECURITYLEVEL	Retrieves or sets the *security_level* member.
BTSDK_LSAM_AUTHORMETHOD	Retrieves or sets the *author_method* member.
BTSDK_LSAM_EXTATTRIBUTES	Retrieves or sets the *ext_attributes* member.
BTSDK_LSAM_APPPARAM	Retrieves or sets the app_param member.

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

Value of ***service_class***	Type of ***ext_attributes***	Mandatory
BTSDK_CLS_SERIAL_PORT	PBtSdkLocalSPPServerAttrStru	No
BTSDK_CLS_DIALUP_NET	PBtSdkLocalDUNServerAttrStru.	No
BTSDK_CLS_FAX	PBtSdkLocalFAXServerAttrStru	No
BTSDK_CLS_OBEX_OBJ_PUSH	PBtSdkLocalOPPServerAttrStru	Yes
BTSDK_CLS_OBEX_FILE_TRANS	PBtSdkLocalFTPServerAttrStru	Yes
BTSDK_CLS_AUDIO_SINK	PBtSdkLocalA2DPServerAttrStru	Yes
BTSDK_CLS_PAN_PANU	PBtSdkLocalPANUServerAttrStru	No
BTSDK_CLS_PAN_NAP	PBtSdkLocalNAPServerAttrStru	No
BTSDK_CLS_PAN_GN	PBtSdkLocalGNServerAttrStru	No

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

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

5.11 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 */
```

5.12 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 specifies 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));
```

5.13 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 specifies 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.

5.14 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 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 SDK initiates the connection to the remote service.
BTSDK_CONNROLE_ACCEPTOR	The remote device initiates the connection to a local service.

6. API Functions

6.1 MiddleWare Initialization / Termination

6.1.1 MiddleWareInitWithClientID

Prototype	long MiddleWareInitWithClientID(UINT Client_ID);	
Description	The MiddleWareInitWithClientID function initializes the communication channel between Client and Server with ClientID. It was useful when you want to start multi-clients.	
Parameters	*Client_ID*	[in] Specifies the identify of the client. Now it can be FIRST_CLIENT_ID or SECOND_CLIENT_ID 2 (you can find the define in header file)
Return:	If the function succeeds, the return value is BTSDK_OK.	

Remarks

IF use Client—Server Modle of IVT WindowsCE Solution, This function MUST be called and the return value MUST be BTSDK_OK before any other functions can be called.

This function initializes the communication channel between Client and Server, required to run the BTSDK.

Each successful call to MiddleWareInitWithClientID must be balanced by a corresponding call to MiddleWareUnInit 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.

6.1.2 MiddleWareInit

Prototype	long MiddleWareInit(void);	
Description	The MiddleWareInit function initializes the communication channel between Client and Server	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK.	

Remarks

IF use Client—Server Modle of IVT WindowsCE Solution, This function MUST be called and the return value MUST be BTSDK_OK before any other functions can be called.

This function initializes the communication channel between Client and Server, required to run the BTSDK.

Each successful call to MiddleWareInit must be balanced by a corresponding call to MiddleWareUnInit 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.

This function equals to function MiddleWareInitWithClientID with FIRST_CLIENT_ID parameter.

6.1.3 MiddleWareUnInite

Prototype	void `MiddleWareUnInit` (void);	
Description	The `MiddleWareUnInit` function releases the communication channel between Client and Server.	
Parameters		
Return:	If the function succeeds, the return value is BTSDK_OK..	

Remarks

IF use Client—Server Modle of IVT WindowsCE Solution, An application must call `MiddleWareUnInit` once for each successful call it has made to `MiddleWareInit`.

This function releases the communication channel between Client and Server.

6.2 Stack Version

6.2.1 Btsdk_GetVersionString

Prototype	BTUINT32 Btsdk_GetVersionString (BTUINT8 *pver_str, BTUINT32 length);	
Description	The **Btsdk_GetVersionString** function reads the version of IVT stack.	
Parameters	*pver_str*	[out] Pointer to the buffer to receive the version string. If this parameter is set to NULL, the function returns the number of bytes required for the buffer. (In this case, the *length* value is not used.)
	length	[in] Specifies the maximum number of bytes can be copied to the buffer pointed by the *pver_str* parameter. If this parameter is set to 0, the function returns the number of bytes required for the buffer. (In this case, the *pver_str* buffer is not used.)
Return:	If *pver_str* is not NULL and *length* is nonzero, the return value is the number of bytes copied to the buffer pointed to by *pver_str*. If *pver_str* is NULL or *length* is 0, the return value is the number of bytes required for the *pver_str* buffer.	

Remarks

An application can call this function at any time to check the version of BTSDK.

The version string has the same format as “6.2.0.20051216”, which reflects [major version].[minor version].[revision number].[build date (year/month/date)].

6.3 Initialization / Termination

6.3.1 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_GetVersionString*, *Btsdk_IsSDKInitialized*, and *Btsdk_IsBluetoothReady*) can be called.

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

After BTSDK 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.

6.3.2 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 BTSDK 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.

6.3.3 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 BTSDK.

6.3.4 Btsdk_RegisterCallback

Prototype	BTINT32 Btsdk_RegisterCallback (PbtSdkCallbackStru call_back);	
Description	The **Btsdk_RegisterCallback** 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

A message from BTSDK 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_RegisterCallback* 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_RegisterCallback* will remove the callback for the specified message from BTSDK.

Table 9 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_RegisterCallback(&cb);
}
```

6.4 Memory Management

6.4.1 Btsdk_MallocMemory

Prototype	void* Btsdk_MallocMemory (BTUINT32 size;);	
Description	The **Btsdk_MallocMemory** function allocates memory block, which will be passed to the BTSDK through BTSDK API and released by BTSDK 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.	

6.4.2 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.	

6.5 Local Bluetooth Device Management

6.5.1 Device Initialization

6.5.1.1 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 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 MUST be called and the return value MUST be BTSDK_OK before any other functions that require communication with Bluetooth device can be called.

6.5.1.2 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 DOES NOT release the resources allocated by other BTSDK 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.

6.5.1.3 Btsdk_IsBluetoothReady

Prototype	BTBOOL Btsdk_IsBluetoothReady (void);	
Description	The **Btsdk_IsBluetoothReady** function indicates whether the local 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 state of the local device.

6.5.2 Device Modes

6.5.2.1 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. */
}
```

6.5.2.2 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.

6.5.2.3 Btsdk_SetSecurityMode

Prototype	BTINT32 Btsdk_SetSecurityMode (BTUINT16 security_mode,);	
Description	The **Btsdk_SetSecurityMode** function changes the security mode of the local device.	
Parameters	*security_mode*	[in] Specifies the new security mode. It can be one of the values listed in Table 6.
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_SetSecurityMode*, the local device must be enabled by a previous successful call to *Btsdk_StartBluetooth*.

6.5.2.4 Btsdk_GetSecurityMode

Prototype	BTINT32 Btsdk_GetSecurityMode (BTUINT16* psecurity_mode,);	
Description	The **Btsdk_GetSecurityMode** function gets the current security mode setting of the local device.	
Parameters	*psecurity_mode*	[out] Pointer to a variable to receive the security mode. The return value can be one of the values listed in Table 6.
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_GetSecurityMode*, the local device must be enabled by a previous successful call to *Btsdk_StartBluetooth*.

6.5.2.5 Btsdk_SetAFHChannelClassification

Prototype	BTINT32 Btsdk_SetAFHChannelClassification (BTUINT8* afh_channels,);	
Description	The **Btsdk_SetAFHChannelClassification** function allows the Bluetooth host to specify a channel classification based on its "local information".	
Parameters	*afh_channels*	[in] Specify the AFH host channel classification. It shall be a buffer no smaller than 10 bytes. But only 79bits (from byte 0 to byte 9) are meaningful.
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_SetAFHChannelClassification*, the local device must be enabled by a previous successful call to *Btsdk_StartBluetooth*.

6.5.3 Device Information

6.5.3.1 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*.

6.5.3.2 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 contains 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*.

6.5.3.3 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.

6.5.3.4 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);
}
```

6.5.3.5 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*.

6.5.3.6 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*.

6.5.4 Application Extension

6.5.4.1 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 BTSDK 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.

6.5.4.2 Btsdk_Vendor_Event_Ind_Func

Prototype	typedef void (Btsdk_Vendor_Event_Ind_Func) (BTUINT8 ev_code, BTUINT8 ev_param_size, BTUINT8* ev_param);	
Description	The **Btsdk_Vendor_Event_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_VENDOR_EVENT_IND message.	
Parameters	*ev_code*	[in] Specifies the event code.
	ev_param_size	[in] Specifies the size in bytes of the *ev_param* buffer.
	ev_param	[in] Pointer to the buffer receives the raw event parameters copied from the HCI event packet's parameter field.
Return:		

Remarks

Generally, vendor-specific event is received as confirm to the vendor-specific command. In this case, the event information is returned to the application through the output parameter of function *Btsdk_VendorCommand*.

But some Bluetooth chips make use of vendor-specific event to notify the application that their state or setting is changed aperiodically. When receiving such a vendor-specific event, the *Btsdk_Vendor_Event_Ind_Func* callback function will be called.

6.5.4.3 Btsdk_SendDataToHostController

Prototype	BTINT32 Btsdk_SendDataToHostController (BTUINT32 size, BTUINT8* data);	
Description	The **Btsdk_SendDataToHostController** function is used to transmit data to the host controller directly.	
Parameters	*size*	[in] Specifies the size, in bytes, of the buffer pointed to by the *data* parameter.
	data	[in] Pointed to the buffer contains the data to be transmitted to the host controller.
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_SendDataToHostController*, the local device must be enabled by a previous successful call to *Btsdk_StartBluetooth*.

It is usually used when the application need to transmit bytes arranged in a nonstandard HCI format.

6.6 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.

6.6.1 Device Discovery

6.6.1.1 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, BTSDK 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 to BTSDK 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 BTSDK to process BTSDK_INQUIRY_RESULT_IND message, or call *Btsdk_GetInquiredDevices* after the inquiry procedure terminates.

6.6.1.2 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 BTSDK.

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 BTSDK. 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 BTSDK doesn’t support pairing or connecting to a remote device before inquiry procedure is completed.

6.6.1.3 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 BTSDK. 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. 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.

6.6.1.4 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.

6.6.1.5 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.

6.6.1.6 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.

6.6.2 Device Pairing

6.6.2.1 Btsdk_IsDevicePairedExt

Prototype	BTBOOL Btsdk_IsDevicePairedExt (
	    BTDEVHDL device_handle,
	    BTUINT8* plink_key
	);	

Description	The **Btsdk_IsDevicePairedExt** function checks whether the specified remote device is paired, and acquires the link key for this device if it is available.	
Parameters	*device_handle*	[in] Handle to the device to get link key.
	plink_key	[out] Pointer to the buffer to receive the link key. The buffer size must be large enough to hold the 16-bytes link key. It can be set to NULL if the application doesn't care about the link key.
Return:	If the specified device is paired, the return value is BTSDK_TRUE. If the specified device is not paired, the return value is BTSDK_FALSE.	

Remarks

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

A paired device is the device that has been previously paired and the link key is stored in the device database.

6.6.2.2 Btsdk_PairDevice

Prototype	BTINT32 Btsdk_PairDevice (BTDEVHDL device_handle,);	
Description	The **Btsdk_PairDevice** function pairs the specified remote device. The BTSDK_PIN_CODE_REQ_IND message is reported to the application when the PIN code is required during the pairing procedure.	
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*.

The application shall register at least a callback function to BTSDK to process BTSDK_PIN_CODE_REQ_IND message, which requests the PIN code from the application. When the paring succeeds, the BTSDK_LINK_KEY_NOTIF_IND message is reported to the application. When the pairing fails, the BTSDK_AUTHENTICATION_FAIL_IND message is reported to the application.

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.

6.6.2.3 Btsdk_Pin_Req_Ind_Func

Prototype	typedef void (Btsdk_Pin_Req_Ind_Func) (BTDEVHDL device_handle);	
Description	The **Btsdk_Pin_Req_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_PIN_CODE_REQ_ IND message.	
Parameters	*device_handle*	[in] Handle to the remote device that a PIN code is required to create the new link key for.
Return:		

Remarks

This callback function is always called when the application is requested to provide a PIN code, no matter which side initiates the pairing procedure.

The application can call *Btsdk_PinCodeReply* to provide a PIN code or to reject the PIN code request directly. If the PIN code must be provided by the user, the application must create another thread to wait for the user input and return this callback function immediately.

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 BTSDK. DO NOT call inside this callback function any BTSDK functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

6.6.2.4 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 BTSDK. DO NOT call inside this callback function any BTSDK functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

6.6.2.5 Btsdk_Link_Key_Req_Ind_Func

Prototype	typedef void (Btsdk_Link_Key_Req_Ind_Func) (BTDEVHDL device_handle);	
Description	The **Btsdk_Link_Key_Req_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_LINK_KEY_REQ_IND message.	
Parameters	*device_handle*	[in] Handle to the remote device that is to be authenticated.
Return:		

Remarks

This callback function is always called when the application is requested to provide the link key, no matter which side initiates the authentication procedure.

As default, BTSDK stores link keys of all paired devices and reply the link key request directly if it finds the stored link key. If BTSDK can't find the stored link key and the application registers a callback function to process BTSDK_LINK_KEY_REQ_IND, BTSDK calls this callback function to request link key from the application. If the application hasn't registered a callback function to process BTSDK_LINK_KEY_REQ_IND, BTSDK rejects the link key request directly in case of finding no stored link key for the specified remote device.

The application can call *Btsdk_LinkKeyReply* to provide the link key or to reject the link key request directly.

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 BTSDK. DO NOT call inside this callback function any BTSDK functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

6.6.2.6 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 BTSDK. DO NOT call inside this callback function any BTSDK functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

6.6.2.7 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, BTSDK 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, BTSDK 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_REQ_IND message.

6.6.2.8 Btsdk_LinkKeyReply

Prototype	BTINT32 Btsdk_LinkKeyReply (BTDEVHDL device_handle, BTUINT8* link_key,);	
Description	The **Btsdk_LinkKeyReply** function is used to reply the link key request during the authentication procedure.	
Parameters	*device_handle*	[in] Handle to the remote device to be authenticated.
	link_key	[in] Pointer to the buffer contains the link key. The length of the link key must be 16-bytes. If the *link_key* parameter is set to NULL, BTSDK sends "HCI Link Key Request Negative Reply Command" and the authentication 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_LinkKeyReply* function to reply the link key request after it receives the BTSDK_LINK_KEY_REQ_IND message.

6.6.2.9 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.

6.6.3 Link Management

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

6.6.3.1 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*.

6.6.3.2 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.

6.6.3.3 Btsdk_SwitchRole

Prototype	BTINT32 Btsdk_SwitchRole (BTDEVHDL device_handle, BTUINT16 role);	
Description	The **Btsdk_SwitchRole** function changes the current role that local device is performing for the ACL link with the specified device.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the connection.
	role	[in] Specifies the new role that local device is performing for the ACL link with the device specified by the *device_handle* parameter. It 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_SwitchRole*, a connection between local device and the specified remote device must be created first.

6.6.3.4 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.

6.6.3.5 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 remote device used to specify the connection.
	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,* a connection between local device and the specified remote device must be created first.

6.6.3.6 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.

6.6.3.7 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.

6.6.3.8 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	[out] 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.

6.6.3.9 Btsdk_GetCurrentLinkMode

Prototype	BTINT32 Btsdk_GetCurrentLinkMode (BTDEVHDL device_handle, BTUINT8* plink_mode);	
Description	The **Btsdk_GetCurrentLinkMode** function gets the current power mode of the ACL link between local device and the specified remote device.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the ACL link.
	plink_mode	[out] Pointer to a variable to receive the current mode 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.	

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

Value	Description
BTSDK_LPM_ACTIVE_MODE	The specified ACL link is in the active mode.
BTSDK_LPM_HOLD_MODE	The specified ACL link is in the hold mode.
BTSDK_LPM_SNIFF_MODE	The specified ACL link is in the sniff mode.
BTSDK_LPM_PARK_MODE	The specified ACL link is in the park mode.

Remarks

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

6.6.3.10 Btsdk_ActivateACLLink

Prototype	BTINT32 Btsdk_ActivateACLLink (BTDEVHDL device_handle,);	
Description	The **Btsdk_ActivateACLLink** function switches the specified ACL link from the current mode to active mode.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the ACL link.
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_ActivateACLLink*, a connection between local device and the specified remote device must be created first.

The *Btsdk_ActivateACLLink* doesn’t support switch an ACL link from hold mode to active mode. It returns BTSDK_ER_COMMAND_DISALLOWED immediately if the specified connection is in the hold mode.

6.6.3.11 Btsdk_EnterHoldMode

Prototype	BTINT32 Btsdk_EnterHoldMode (BTDEVHDL device_handle, PBtSdkHoldModeStru param);	
Description	The **Btsdk_EnterHoldMode** function switches the specified ACL link from the current mode to hold mode.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the ACL link.
	param	[in] Pointer to a BtSdkHoldModeStru specifies the hold mode 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_EnterHoldMode*, a connection between local device and the specified remote device must be created first.

6.6.3.12 Btsdk_EnterSniffMode

Prototype	BTINT32 Btsdk_EnterSniffMode (BTDEVHDL device_handle, PBtSdkSniffModeStru param);	
Description	The **Btsdk_EnterSniffMode** function switches the specified ACL link from the current mode to sniff mode.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the ACL link.
	param	[in] Pointer to a BtSdkSniffModeStru specifies the sniff mode 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_EnterSniffMode*, a connection between local device and the specified remote device must be created first.

6.6.3.13 Btsdk_EnterParkMode

Prototype	BTINT32 Btsdk_EnterParkMode (BTDEVHDL device_handle, PBtSdkSniffModeStru param);	
Description	The **Btsdk_EnterParkMode** function switches the specified ACL link from the current mode to sniff mode.	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the ACL link.
	param	[in] Pointer to a BtSdkParkModeStru specifies the park mode 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_EnterParkMode*, a connection between local device and the specified remote device must be created first.

6.6.3.14 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 specifies 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	Do not use 2-DH1. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_3DH1	Do not use 3-DH1. 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	Do not use 2-DH3. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_3DH3	Do not use 3-DH3. 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	Do not use 2-DH5. Only supported by V2.0EDR Bluetooth device.
BTSDK_ACL_PKT_3DH5	Do not use 3-DH5. 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.

6.6.3.15 Btsdk_WriteLinkPolicy

Prototype	BTINT32 Btsdk_WriteLinkPolicy (BTDEVHDL device_handle, BTUINT16 policy);	
Description	The **Btsdk_WriteLinkPolicy** function changes the link policy setting for the connection between local device and the specified remote device	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the ACL link.
	policy	[in] A set of flags specifies the policies to be used. If it is set to 0, all the Link Manager modes are disabled.
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 ***policy*** parameter can be 0, or one or more of these values.

Value	Description
BTSDK_LP_ENABLE_ROLESWITCH	Enable Role Switch.
BTSDK_LP_ENABLE_HOLDMODE	Enable Hold Mode.
BTSDK_LP_ENABLE_SNIFFMODE	Enable Sniff Mode.
BTSDK_LP_ENABLE_PARKMODE	Enable Park Mode.

Remarks

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

6.6.3.16 Btsdk_ReadLinkPolicy

Prototype	BTINT32 Btsdk_ReadLinkPolicy (BTDEVHDL device_handle, BTUINT16 *policy);	
Description	The **Btsdk_ReadLinkPolicy** function reads the current link policy setting for the connection between local device and the specified remote device	
Parameters	*device_handle*	[in] Handle to the remote device used to specify the ACL link.
	policy	[out] Pointer to a 16bit integer to receive the current link policy setting.
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 returned **policy* value can be 0, or one or more of these values.

Value	Description
BTSDK_LP_ENABLE_ROLESWITCH	Role Switch is enabled.
BTSDK_LP_ENABLE_HOLDMODE	Hold Mode is enabled.
BTSDK_LP_ENABLE_SNIFFMODE	Sniff Mode is enabled.
BTSDK_LP_ENABLE_PARKMODE	Park Mode is enabled.

If the ***policy*** value is set to 0, all the Link Manager modes are disabled.

Remarks

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

6.6.4 Device Database Management

BTSDK 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 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.

6.6.4.1 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*.

6.6.4.2 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*.

6.6.4.3 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, BTSDK 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*.

6.6.4.4 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, BTSDK 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*.

6.6.4.5 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, BTSDK 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 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_GetStoredDevicesByClass*, the device database must be initialized by a previous successful call to *Btsdk_Init*.

6.6.4.6 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.

6.6.4.7 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, BTSDK 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.

6.6.4.8 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.

6.6.4.9 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	[in/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);
        }
}
```

6.6.4.10 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*.

6.6.4.11 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*.

6.6.4.12 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.

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

6.6.4.13 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*.

6.6.4.14 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	[in/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.

6.6.5 Application Extension

6.6.5.1 Btsdk_SetRemoteDeviceParam

Prototype	BTINT32 Btsdk_SetRemoteDeviceParam (BTDEVHDL device_handle, BTUINT32 app_param);	
Description	The **Btsdk_SetRemoteDeviceParam** function attaches an application specific value to a remote device record.	
Parameters	*device_handle*	[in] Handle to the device 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_SetRemoteDeviceParam*, 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*.

6.6.5.2 Btsdk_GetRemoteDeviceParam

Prototype	BTINT32 Btsdk_GetRemoteDeviceParam (BTDEVHDL device_handle, BTUINT32* papp_param);	
Description	The **Btsdk_GetRemoteDeviceParam** function gets the application specific value attached to a remote device record.	
Parameters	*device_handle*	[in] Handle to the device that the value is attached to.
	papp_param	[out] Pointer to a variable to receive the application specific value 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_GetRemoteDeviceParam*, the device database must be initialized by a previous successful call to *Btsdk_Init*.

6.7 Service Database Management

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

6.7.1 Service Registry

6.7.1.1 Btsdk_AddServer

Prototype	BTSVCHDL Btsdk_AddServer (PBtSdkLocalServerAttrStru pservice_attributes);	
Description	The **Btsdk_AddServer** function adds a service record to SDK service database.	
Parameters	*pservice_attributes*	[in] Pointer to a BtSdkLocalServerAttrStru structure containing the information about the new service record to be added.
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_AddServer*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

The newly added service record is in the idle state and is not accessible to the remote device until is activated by a successful call to the function *Btsdk_StartServer*.

The type of the new service record must be provided through the *service_class* member. It determines the content of the *ext_attributes* member. The ext_attributes can be set to NULL for some service types, e.g. COM index for a SPP service record, and the default values are adopted by BTSDK. But, if the feature is totally implementation dependent and no default value is defined, the ext_attributes member must specify a valid value. Or the service record may work improperly. For example, the application must specify the root directory for a FTP service record to receive files.

The *svc_name* member is optional. If it is not set, the default service name reflected the service type is used. For example, “Dialup Networking Profile” is the default service name for the service record of BTSDK_CLS_DIALUP_NET type.

The security_level member is optional. If it is not set, the default value is (BTSDK_SSL_AUTHORIZATION | BTSDK_SSL_AUTHENTICATION). The security level setting takes effect only when the local device is in security mode 2. The application can call *Btsdk_SetSecurityMode* with BTSDK_SECURITY_MEDIUM to change the local device to security mode 2.

The *author_method* member is optional. If it is not set, the default value is BTSDK_AUTHORIZATION_PROMPT.

The *app_param* member is optional. It is only meaningful to the application.

Example

```
/* This sample demonstrates how to add a SPP service record and a FTP service record. */
void AppAddSPPService(void)
{
        BtSdkLocalServerAttrStru  SvcAttr = {0};
        SvcAttr.service_class = BTSDK_CLS_SERIAL_PORT;
        SvcAttr.mask = BTSDK_LSAM_SERVICENAME;  /* All the other features use default value. */
        strcpy(SvcAttr.svc_name, “IVT Serial Port”);
        Btsdk_AddServer(&SvcAttr);
}

void AppAddFTPService(void)
{
        BtSdkLocalServerAttrStru  SvcAttr = {0};
        BtSdkLocalFTPServerAttrStru  FtpAttr = {0};

        FtpAttr.size = sizeof(BtSdkLocalFTPServerAttrStru);
        FtpAttr.mask = BTSDK_LFTPSAM_DESIREDACCESS | BTSDK_LFTPSAM_ROOTDIR;
        FtpAttr.desired_access = BTSDK_FTPDA_READONLY;
        strcpy((char*)FtpAttr.root_dir, “D:\\Bluetooth FTP Root\\”);

        SvcAttr.service_class = BTSDK_CLS_OBEX_FILE_TRANS;
        SvcAttr.mask = BTSDK_LSAM_SECURITYLEVEL | BTSDK_LSAM_EXTATTRIBUTES;
        SvcAttr.security_level = BTSDK_SSL_AUTHENTICATION | BTSDK_SSL_AUTHORIZATION |
                                 BTSDK_SSL_ENCRYPTION;
        SvcAttr.ext_attributes = &FtpAttr;
        Btsdk_AddServer(&SvcAttr);
}
```

6.7.1.2 Btsdk_RemoveServer

Prototype	BTINT32 Btsdk_RemoveServer (BTSVCHDL service_handle);	
Description	The **Btsdk_RemoveServer** function removes the specified service record from the local service database.	
Parameters	*service_handle*	[in] Handle to the service record to be removed.
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_RemoveServer*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

If the specified service record is in the active state, this function returns BTSDK_ER_SERVER_IS_ACTIVE.

6.7.1.3 Btsdk_UpdateServerAttributes

Prototype	BTINT32 Btsdk_UpdateServerAttributes (BTSVCHDL service_handle, PBtSdkLocalServerAttrStru pservice_attributes);	
Description	The **Btsdk_UpdateServerAttributes** function modifies the attributes of the specified service record.	
Parameters	*service_handle*	[in] Handle to the service record to be modified.
	pservice_attributes	[in] Pointer to a BtSdkLocalServerAttrStru structure containing the new attribute values about the specified 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_UpdateServerAttributes*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

If the specified service record is in the active state, this function will restart the service to make the new attribute values take effect. In such case, all the connections to the specified service record are broken.

Use the *mask* member of the *pservice_attributes* parameter to specify the attributes to be modified.

Example

```
/* This sample demonstrates how to modify a FTP service record. */
void AppModifyFTPService(BTSVCHDL hFtp)
{
        BtSdkLocalServerAttrStru  SvcAttr = {0};
        BtSdkLocalFTPServerAttrStru  FtpAttr = {0};

        FtpAttr.size = sizeof(BtSdkLocalFTPServerAttrStru);
        /* Modify security level and FTP specific attribute – Desired Access. */
        FtpAttr.mask = BTSDK_LFTPSAM_DESIREDACCESS;
        FtpAttr.desired_access = BTSDK_FTPDA_READWRITE;

        SvcAttr.mask = BTSDK_LSAM_SECURITYLEVEL | BTSDK_LSAM_EXTATTRIBUTES;
        SvcAttr.security_level = BTSDK_SSL_AUTHENTICATION | BTSDK_SSL_AUTHORIZATION;
        SvcAttr.ext_attributes = &FtpAttr;
        Btsdk_UpdateServerAttributes(hFtp, &SvcAttr);
}
```

6.7.1.4 Btsdk_StartServer

Prototype	BTINT32 Btsdk_StartServer (BTSVCHDL service_handle);	
Description	The **Btsdk_StartServer** function activates a service record so that a remote client can access it.	
Parameters	*service_handle*	[in] Handle to the service record to be activated.
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_StartServer*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

6.7.1.5 Btsdk_StopServer

Prototype	BTINT32 Btsdk_StopServer (BTSVCHDL service_handle);	
Description	The **Btsdk_StopServer** function deactivates a service record so that a remote client cannot access it.	
Parameters	*service_handle*	[in] Handle to the service record to be deactivated.
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_StopServer*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

If a remote device connects to this service record, the *Btsdk_StopServer* function will disconnect the connection.

6.7.1.6 Btsdk_GetServerStatus

Prototype	BTINT32 Btsdk_GetServerStatus (BTSVCHDL service_handle, BTUINT16* pstatus);	
Description	The **Btsdk_GetServerStatus** function gets the current status of a service record.	
Parameters	*service_handle*	[in] Handle to the service record to be deactivated.
	pstatus	[out] Pointer to a variable to receive the status 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.	

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

Value	Description
BTSDK_SERVER_STOPPED	The service record is in the idle state. A remote client cannot create a connection to a service in the idle state.
BTSDK_SERVER_STARTED	The service record is in the active state. A remote client can create a connection to a service in the active state.
BTSDK_SERVER_CONNECTED	One or more remote clients connect to the service.

Remarks

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

6.7.1.7 Btsdk_GetServerAttributes

Prototype	BTINT32 Btsdk_GetServerAttributes (BTSVCHDL service_handle, PBtSdkLocalServerAttrStru pservice_attributes);	
Description	The **Btsdk_GetServerAttributes** function gets the attributes of the specified service record.	
Parameters	*service_handle*	[in] Handle to the service record to be modified.
	pservice_attributes	[in/out] Pointer to a BtSdkLocalServerAttrStru structure to receive the attribute values about the specified 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_GetServerAttributes*, the service 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_LSAM_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 read attribute values of a service record. */
void AppGetServerAttributes(BTSVCHDL hService)
{
        BtSdkLocalServerAttrStru  SvcAttr = {0};

        SvcAttr.mask = BTSDK_LSAM_SERVICENAME | BTSDK_LSAM_EXTATTRIBUTES;
        Btsdk_GetServerAttributes(hService, &SvcAttr);
        // To Do: Process the service attribute values:
        // …
        // Free the buffer
        Btsdk_FreeMemory(SvcAttr.ext_attributes);
}
```

6.7.1.8 Btsdk_GetLocalServers

Prototype	BTINT32 Btsdk_GetLocalServers (BTDEVHDL* pdevice_handles, BTUINT32* pmax_svc_num);	
Description	The **Btsdk_GetLocalServers** function gets a list of handles to the service records from the service database.	
Parameters	*pservice_handles*	[out] Pointer to the buffer to receive the service handles. If this parameter is set to NULL, the total number of available handles is returned in **pmax_svc_num*.
	pmax_svc_num	[in/out] Pointer to a variable that, on input, specifies the maximum number of handles can be copied to the buffer pointed to by the pservice_handles parameter. On output, This variable receives the number of handles copied to the buffer pointed to by the *pservice_handles* parameter. To determine the total number of available handles, call this function with *pservice_handles* set to NULL. This function returns the total number of available handles in **pmax_svc_num*.
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_GetLocalServers*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

6.7.1.9 Btsdk_StartEnumLocalServer

Prototype	BTSDKHANDLE Btsdk_StartEnumLocalServer (void);	
Description	The **Btsdk_StartEnumLocalServer** function starts to search the service database for all service records available.	
Parameters		
Return:	If the function succeeds, the return value is a search handle used in a subsequent call to *Btsdk_EnumLocalServer* and *Btsdk_EndEnumLocalServer*. If the function fails, the return value is BTSDK_INVALID_HANDLE.	

Remarks

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

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

6.7.1.10 Btsdk_EnumLocalServer

Prototype	BTSVCHDL Btsdk_EnumLocalServer (BTSDKHANDLE enum_handle, PBtSdkLocalServerAttrStru pservice_attributes);	
Description	The **Btsdk_EnumLocalServer** function continues to search the service database for an available service record.	
Parameters	*enum_handle*	[in] Search handle returned by a previous call to the *Btsdk_StartEnumLocalServer* function.
	pservice_attributes	[in/out] Pointer to the BtSdkLocalServerAttrStru 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_EnumLocalServer*, the service 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_LSAM_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 AppGetLocalServices(void)
{
        BtSdkLocalServerAttrStru SvcAttr = {0};
        BTSDKHANDLE hEnumSvc = BTSDK_INVALID_HANDLE;
        BTSVCHDL hSvcFound = BTSDK_INVALID_HANDLE;

        hEnumSvc = Btsdk_StartEnumLocalServer();
        if (hEnumSvc != BTSDK_INVALID_HANDLE)
        {
```

```
		SvcAttr.mask = BTSDK_LSAM_SERVICENAME | BTSDK_LSAM_EXTATTRIBUTES;
		while ((hSvcFound = Btsdk_EnumLocalServer(hEnumSvc, &SvcAttr)) != BTSDK_INVALID_HANDLE)
		{
			// To Do: Process the service attribute values:
			// …
			// Free the buffer
			Btsdk_FreeMemory(SvcAttr.ext_attributes);
		}
		Btsdk_EndEnumLocalServer(hEnumSvc);
	}
}
```

6.7.1.11 Btsdk_EndEnumLocalServer

Prototype	BTINT32 Btsdk_EndEnumLocalServer (BTSDKHANDLE enum_handle,);	
Description	The **Btsdk_EndEnumLocalServer** function closes the specified search handle.	
Parameters	*enum_handle*	[in] Search handle returned by a previous call to the *Btsdk_StartEnumLocalServer* 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_EndEnumLocalServer*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

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

6.7.2 Security Requirements

6.7.2.1 Btsdk_SetServiceSecurityLevel

Prototype	BTINT32 Btsdk_SetServiceSecurityLevel (BTSVCHDL service_handle, BTUINT8 security_level,);	
Description	The **Btsdk_SetServiceSecurityLevel** function changes the security level of a service record.	
Parameters	*service_handle*	[in] Handle to the service record to be modified.
	security_level	[in] Specifies the new security level. It can be one or more of the values listed in Table 7.
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_SetServiceSecurityLevel*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

The security level setting takes effect only when the local device is in security mode 2. The application can call *Btsdk_SetSecurityMode* with BTSDK_SECURITY_MEDIUM to change the local device to security mode 2.

6.7.2.2 Btsdk_GetServiceSecurityLevel

Prototype	BTINT32 Btsdk_GetServiceSecurityLevel (BTSVCHDL service_handle, BTUINT8* psecurity_level,);	
Description	The **Btsdk_GetServiceSecurityLevel** function gets the security level of a service record.	
Parameters	*service_handle*	[in] Handle to the service record interested.
	psecurity_level	[in] Pointer to a variable to receive the security level. The return value can be one or more of the values listed in Table 7.
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_GetServiceSecurityLevel*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

6.7.2.3 Btsdk_SetAuthorizationMethod

Prototype	BTINT32 Btsdk_SetAuthorizationMethod (BTSVCHDL service_handle, BTUINT32 author_method,);	
Description	The **Btsdk_SetAuthorizationMethod** function changes the authorization method of a service record. The authorization method specifies how IVT BTSDK processes authorization request for an un-trusted device.	
Parameters	*service_handle*	[in] Handle to the service record to be modified.
	author_method	[in] Specifies the new authorization method. It can be one of the values listed in Table 8.
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_SetAuthorizationMethod*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

The authorization method setting takes effect only when the local device is in security mode 2 and the specified service record must request authorization.

6.7.2.4 Btsdk_GetAuthorizationMethod

Prototype	BTINT32 Btsdk_GetAuthorizationMethod (BTSVCHDL service_handle, BTUINT8* psecurity_level,);	
Description	The **Btsdk_GetAuthorizationMethod** function gets the authorization method of a service record.	
Parameters	*service_handle*	[in] Handle to the service record interested.
	psecurity_level	[in] Pointer to a variable to receive the security level. The return value can be one of the values listed in Table 8.
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_GetAuthorizationMethod*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

6.7.2.5 Btsdk_SetTrustedDevice

Prototype	BTINT32 Btsdk_SetTrustedDevice (BTSVCHDL service_handle, BTDEVHDL device_handle, BTBOOL bIsTrusted,);	
Description	The **Btsdk_SetTrustedDevice** function sets the trust relation between a remote device and a local service record.	
Parameters	*service_handle*	[in] Handle to the local service record to set the trust relation. This parameter can be BTSDK_INVALID_HANDLE. In this case, SDK changes the trust relations between the specified remote device and all local service records.
	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.	

Remarks

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

The device specified by the *device_handle* must be authenticated previously and the link key is stored in the device database. Or the *Btsdk_SetTrustedDevice* function returns error code BTSDK_ER_DEVICE_UNPAIRED.

A device trusted to a specified local service record is always allowed access to this service record. If a device tries to access a local service record to which it is untrusty, SDK may,

(1) Accept the request, if local security mode is not BTSDK_SECURITY_MEDIUM, or the service record doesn't require authorization, or the authorization method is BTSDK_AUTHORIZATION_ACCEPT.

(2) Reject the request, if local security mode is BTSDK_SECURITY_MEDIUM, and the service record requires authorization, and the authorization method is BTSDK_AUTHORIZATION_REJECT.

(3) Let the application make the decision, if local security mode is BTSDK_SECURITY_MEDIUM, and the service record requires authorization, and the

authorization method is BTSDK_AUTHORIZATION_PROMPT.

If the *service_handle* is SDK_INVALID_HANDLE and the *bIsTrusted* is BTSDK_TRUE, this device is marked as a "Trusted" device. A device marked as "Trusted" is allowed access to all service records.

SDK removes the "Trusted" flag when the device is set as untrusty to whichever service record.

6.7.2.6 Btsdk_GetTrustedDevices

Prototype	BTINT32 Btsdk_GetTrustedDevices (BTSVCHDL service_handle, BTDEVHDL *pdevice_handles, BTUINT32 *phandle_count,);	
Description	The **Btsdk_GetTrustedDevices** function gets a list of handles to the device records that are trusted to the specified service record.	
Parameters	*service_handle*	[in] Handle to the local service record to query. This parameter can be BTSDK_INVALID_HANDLE. In this case, this function returns handles to the device records that are marked as "Trusted".
	pdevice_handles	[out] Pointer to the buffer to receive the device handles. It can be NULL.
	phandle_count	[in/out] Pointer to a variable that, on input, specifies the maximum number of handles can be copied to the buffer pointed to by the *pdevice_handles* parameter. On output, This variable receives the number of handles copied to the buffer pointed to by the *pdevice_handles* parameter. To determine the number of available device handles, call this function with *pdevice_handles* set to NULL. This function returns the available handle number in **pdevice_handles*.
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_GetTrustedDevices*, the service database must be initialized by a previous successful call to *Btsdk_Init*.

6.8 Connection Management

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

6.8.1 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.

6.8.1.1 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, BTSDK 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*.

6.8.1.2 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*.

6.8.1.3 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	[in\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*.

6.8.1.4 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, 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, BTSDK 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.

6.8.1.5 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.

6.8.1.6 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 local SDK device database.	
Parameters	*service_handle*	[in] Handle to the remote service record.
	pattributes	[in/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.

6.8.1.7 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.

6.8.1.8 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	[in/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);
        }
}
```

6.8.1.9 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*.

6.8.2 Application Extension

6.8.2.1 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*.

6.8.2.2 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*.

6.8.3 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.

6.8.3.1 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.

6.8.3.2 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, SDK will automatically select the first accessible record to connect.

6.8.3.3 Btsdk_Authorization_Req_Ind_Func

Prototype	typedef void (Btsdk_Authorization_Req_Ind_Func) (BTSVCHDL service_handle, BTDEVHDL device_handle);	
Description	The **Btsdk_Authorization_Req_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_AUTHORIZATION_IND message.	
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*.
Return:		

Remarks

This callback function is called when the SDK requires the application to decide whether to grant the device specified by *device_handle* access to the service record specified by *service_handle*.

The application can call *Btsdk_AuthorizationResponse* to accept or reject the request directly. If the user interaction is required, the application must create another thread to wait for the user response and return this callback function immediately.

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 BTSDK. DO NOT call inside this callback function any BTSDK functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

6.8.3.4 Btsdk_Author_Abort_Ind_Func

Prototype	typedef void (Btsdk_Author_Abort_Ind_Func) (BTSVCHDL service_handle, BTDEVHDL device_handle);	
Description	The **Btsdk_Author_Abort_Ind_Func** function prototype is the prototype of application defined callback function used to process BTSDK_AUTHORIZATION_ABORT_IND message.	
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*.
Return:		

Remarks

This callback function is called when the remote device cancels the connection request before the application responds to the authorization request.

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 BTSDK. DO NOT call inside this callback function any BTSDK functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

6.8.3.5 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:		

Remarks

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

6.8.3.6 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.

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 BTSDK. DO NOT call inside this callback function any BTSDK functions that require communicating with a remote device either, e.g. *Btsdk_Connect* and so on.

6.8.4 Connection Database Management

6.8.4.1 Btsdk_GetAllIncomingConnections

Prototype	BTUINT32 Btsdk_GetAllIncomingConnections (BTCONNHDL* pconn_handles, BTUINT32 count,);	
Description	The **Btsdk_GetAllIncomingConnections** function gets a list of handles to the connections that are initiated by the remote devices.	
Parameters	*pconn_handles*	[out] Pointer to the buffer to receive the connection handles. If this parameter is set to NULL, the total number of available handles is returned.
	count	[in] Specifies the maximum number of handles can be copied to the buffer pointed to by the *pconn_handles* parameter. If *pconn_handles* is set to NULL, the value of *count* parameter is ignored.
Return:	If *pconn_handles* is not NULL and *count* is nonzero, the return value is the number of handles copied to the buffer pointed to by *pdevice_handles*. If *pconn_handles* is NULL, the return value is the total number of available handles.	

Remarks

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

6.8.4.2 Btsdk_GetAllOutgoingConnections

Prototype	BTUINT32 Btsdk_GetAllOutgoingConnections (BTCONNHDL* pconn_handles, BTUINT32 count,);	
Description	The **Btsdk_GetAllOutgoingConnections** function gets a list of handles to the connections that are initiated by the local device.	
Parameters	*pconn_handles*	[out] Pointer to the buffer to receive the connection handles. If this parameter is set to NULL, the total number of available handles is returned.
	count	[in] Specifies the maximum number of handles can be copied to the buffer pointed to by the *pconn_handles* parameter. If *pconn_handles* is set to NULL, the value of *count* parameter is ignored.
Return:	If *pconn_handles* is not NULL and *count* is nonzero, the return value is the number of handles copied to the buffer pointed to by *pdevice_handles*. If *pconn_handles* is NULL, the return value is the total number of available handles.	

Remarks

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

6.8.4.3 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*.

6.8.4.4 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.

6.8.4.5 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);
        }
}
```

6.8.4.6 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*.

6.8.5 Connection Release

6.8.5.1 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*.