dotstack API Reference

Generated by Doxygen 1.8.9.1

Tue Sep 8 2015 17:42:45

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

Module Documentation

3.1 System Functions

Functions in this module provide interface to DotStack functionality that is common to all protocols and profiles.

Typedefs

- typedef void(∗ bt_sys_callback_fp) (bt_bool success, void ∗param)

  System start callback.

Functions

- void bt_sys_init (void)

  Initialize the Bluetooth system.
- void bt_sys_init_ex (bt_byte default_link_policy)

  Initialize the Bluetooth system.
- void bt_sys_start (bt_bool discoverable, bt_bool connectable, const bt_byte ∗sdp_db, bt_uint sdp_db_len, bt_sys_callback_fp callback, void ∗callback_param)

  Start the Bluetooth system.
- bt_l2cap_mgr_t ∗ bt_sys_get_l2cap_manager (void)

  Get the L2CAP manager.
- const bt_byte ∗ bt_sys_get_version (void)

  Get the version of the dotstack library.

3.1.1 Detailed Description

Functions in this module provide interface to DotStack functionality that is common to all protocols and profiles.

3.1.2 Typedef Documentation

3.1.2.1 typedef void(∗ bt_sys_callback_fp) (bt_bool success, void ∗param)

System start callback.

This callback function is called when system start initiated by bt_sys_start() has completed.

Parameters

success	Success of the operation: `BT_TRUE` if successfull, `BT_FALSE` otherwise.
param	Callback parameter that was specified when bt_sys_start() was called.

3.1.3 Function Documentation

3.1.3.1 bt_l2cap_mgr_t∗ bt_sys_get_l2cap_manager (void)

Get the L2CAP manager.

This function returns the L2CAP manager. The L2CAP manager is created as part of the start up sequence.

Returns

The L2CAP manager.

3.1.3.2 const bt_byte∗ bt_sys_get_version (void)

Get the version of the dotstack library.

Returns

The version of the dotstack library.

3.1.3.3 void bt_sys_init (void)

Initialize the Bluetooth system.

This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init().

This function essentially calls bt_sys_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.

3.1.3.4 void bt_sys_init_ex (bt_byte *default_link_policy*)

Initialize the Bluetooth system.

This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt_spp_init() must be called right after calling bt_sys_init().

Also, the caller must provide an SDP database.

Parameters

default_link_↩ policy	default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: • HCI_LINK_POLICY_ENABLE_ROLE_SWITCH • HCI_LINK_POLICY_ENABLE_HOLD_MODE • HCI_LINK_POLICY_ENABLE_SNIFF_MODE • HCI_LINK_POLICY_ENABLE_PARK_STATE To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

3.1.3.5 void bt_sys_start (bt_bool *discoverable,* bt_bool *connectable,* const bt_byte ∗ *sdp_db,* bt_uint *sdp_db_len,* bt_sys_callback_fp *callback,* void ∗ *callback_param*)

Start the Bluetooth system.

After all modules used by the application have been initialized this function should be called to start the Bluetooth system operation. During the start up sequence it will reset and initialize the HCI controller and then create the L2CAP manager. The application will be notified when the start up sequence completes by calling the provided callback function.

Also, the caller must provide an SDP database.

Parameters

discoverable	defines whether the device is discoverable after reset.
connectable	defines whether the device is connectable after reset.
sdp_db	SDP database data.
sdp_db_len	Length of SDP database data.
callback	A callback function that will be called when the start up sequence is complete.
callback_param	An arbitrary pointer that will be passed to the callback function.

3.2 HCI

The Host Controller Interface (HCI) provides a uniform interface method of accessing a Bluetooth Controller's capabilities.

Macros

- #define bt_hci_add_param_uint(cmd, value) bt_hci_add_param_int(cmd, (bt_int)(value))

  Add unsigned int parameter to an HCI command.
- #define bt_hci_add_param_ulong(cmd, value) bt_hci_add_param_long(cmd, (bt_long)(value))

  Add unsigned long parameter to an HCI command.
- #define bt_hci_add_param_hconn(pcmd, value) bt_hci_add_param_int(pcmd, value)

  Add connection handle parameter to an HCI command.
- #define bt_hci_get_param_hconn(pcmd, pvalue, poffset) bt_hci_get_param_int(pcmd, pvalue, poffset)

  Get connection handle parameter from HCI command.
- #define bt_hci_get_evt_param_hconn(pevt, pvalue, poffset) bt_hci_get_evt_param_int(pevt, pvalue, poffset)

  Get connection handle parameter from HCI event.

Typedefs

- typedef void(∗ bt_hci_start_callback_fp) (bt_bool success, void ∗param)

  HCI initialization callback.
- typedef void(∗ bt_hci_stop_callback_fp) (void ∗param)

  HCI stop callback.
- typedef void(∗ bt_hci_connect_callback_fp) (bt_byte status, bt_hci_conn_state_t ∗pconn, void ∗param)

  HCI connect callback.
- typedef void(∗ bt_hci_disconnect_callback_fp) (bt_byte status, bt_byte reason, bt_hci_conn_state_t ∗pconn, void ∗param)

  HCI disconnect callback.

Functions

- bt_hci_command_p bt_hci_alloc_command (bt_int opcode, bt_hci_cmd_callback_fp callback)

  Allocate and initialize an HCI command structure.
- void bt_hci_free_command (bt_hci_command_p cmd)

  Free HCI command.
- bt_hci_command_p bt_hci_alloc_canned_command (const bt_byte ∗canned_command, bt_hci_cmd_↩ callback_fp callback)

  Allocate and initialize an HCI command structure for a canned (pre-formatted) command.
- bt_bool bt_hci_add_param_byte (bt_hci_command_p pcmd, bt_byte value)

  Add byte parameter to an HCI command.
- bt_bool bt_hci_add_param_int (bt_hci_command_p pcmd, bt_int value)

  Add int parameter to an HCI command.
- bt_bool bt_hci_add_param_long (bt_hci_command_p pcmd, bt_long value)

  Add long parameter to an HCI command.
- bt_bool bt_hci_add_param_bdaddr (bt_hci_command_p pcmd, const bt_bdaddr_t ∗pbdaddr)

  Add BD address parameter to an HCI command.
- bt_bool bt_hci_add_param_string (bt_hci_command_p pcmd, const char ∗ps, bt_int len)

  Add string parameter to an HCI command.
- bt_bool bt_hci_add_param_cod (bt_hci_command_p pcmd, bt_long value)

Add class of device parameter to an HCI command.

- bt_bool bt_hci_add_param_linkkey (bt_hci_command_p pcmd, const bt_linkkey_t ∗linkkey)

  Add link key parameter to an HCI command.
- bt_bool bt_hci_get_param_byte (bt_hci_command_p pcmd, bt_byte ∗pvalue, bt_int ∗offset)

  Get byte parameter from HCI command.
- bt_bool bt_hci_get_param_int (bt_hci_command_p pcmd, bt_int ∗pvalue, bt_int ∗poffset)

  Get int parameter from HCI command.
- bt_bool bt_hci_get_param_long (bt_hci_command_p pcmd, bt_long ∗pvalue, bt_int ∗poffset)

  Get long parameter from HCI command.
- bt_bool bt_hci_get_param_bdaddr (bt_hci_command_p pcm, bt_bdaddr_p pvalue, bt_int_p poffset)

  Get BD address parameter from HCI command.
- bt_bool bt_hci_get_param_linkkey (bt_hci_command_p pcmd, bt_byte_p pvalue, bt_int_p poffset)

  Get link key parameter from HCI command.
- bt_bool bt_hci_get_evt_param_byte (bt_hci_event_p pevt, bt_byte ∗pvalue, bt_int ∗poffset)

  Get byte parameter from HCI event.
- bt_bool bt_hci_get_evt_param_int (bt_hci_event_p pevt, bt_int ∗pvalue, bt_int ∗poffset)

  Get int parameter from HCI event.
- bt_bool bt_hci_get_evt_param_long (bt_hci_event_p pevt, bt_long ∗pvalue, bt_int ∗poffset)

  Get long parameter from HCI event.
- bt_bool bt_hci_get_evt_param_bdaddr (bt_hci_event_p pevt, bt_bdaddr_p pvalue, bt_int_p poffset)

  Get bd address parameter from HCI event.
- bt_bool bt_hci_get_evt_param_devclass (bt_hci_event_p pevt, bt_long_p pvalue, bt_int_p poffset)

  Get class of device parameter from HCI event.
- bt_bool bt_hci_get_evt_param_linkkey (bt_hci_event_p pevt, bt_linkkey_p pvalue, bt_int_p poffset)

  Get link key parameter from HCI event.
- void bt_hci_init (void)

  Initialize the HCI layer.
- void bt_hci_init_ex (bt_byte default_link_policy)

  Initialize the HCI layer.
- bt_bool bt_hci_start (bt_hci_start_callback_fp callback, void ∗callback_param, bt_byte enable_scan)

  Start HCI layer.
- void bt_hci_start_no_init (void)

  Start HCI layer without controller configuration.
- void bt_hci_stop (bt_hci_stop_callback_fp callback, void ∗callback_param)

  Stop HCI layer.
- bt_bool bt_hci_reset (bt_hci_cmd_callback_fp callback, void ∗callback_param)

  Reset controller.
- bt_bool bt_hci_connect (bt_bdaddr_p dest, bt_uint packet_type, bt_byte pg_scan_rpt_mode, bt_byte role_↩
switch, bt_uint acl_config, bt_hci_connect_callback_fp callback, void ∗param)

  Connect to a remote device.
- bt_bool bt_hci_listen (bt_hci_connect_callback_fp cb, void ∗param)

  Listen for incoming connections.
- bt_bool bt_hci_disconnect (bt_hci_conn_state_t ∗pconn)

  Abort connection.
- bt_bool bt_hci_write_local_name (const char ∗device_name, bt_hci_cmd_callback_fp cb)

  Write local device name.

Link control commands

The Link Control commands allow a Controller to control connections to other BR/EDR Controllers.

- #define **HCI_INQUIRY** HCI_OPCODE(OGF_LINK_CONTROL, 0x0001)
- #define **HCI_INQUIRY_CANCEL** HCI_OPCODE(OGF_LINK_CONTROL, 0x0002)
- #define **HCI_PERIODIC_INQUIRY_MODE** HCI_OPCODE(OGF_LINK_CONTROL, 0x0003)
- #define **HCI_EXIT_PERIODIC_INQUIRY_MODE** HCI_OPCODE(OGF_LINK_CONTROL, 0x0004)
- #define **HCI_CREATE_CONNECTION** HCI_OPCODE(OGF_LINK_CONTROL, 0x0005)
- #define **HCI_DISCONNECT** HCI_OPCODE(OGF_LINK_CONTROL, 0x0006)
- #define **HCI_CREATE_CONNECTION_CANCEL** HCI_OPCODE(OGF_LINK_CONTROL, 0x0008)
- #define **HCI_ACCEPT_CONNECTION_REQUEST** HCI_OPCODE(OGF_LINK_CONTROL, 0x0009)
- #define **HCI_REJECT_CONNECTION_REQUEST** HCI_OPCODE(OGF_LINK_CONTROL, 0x000A)
- #define **HCI_LINK_KEY_REQUEST_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x000B)
- #define **HCI_LINK_KEY_REQUEST_NEGATIVE_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x000C)
- #define **HCI_PIN_CODE_REQUEST_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x000D)
- #define **HCI_PIN_CODE_REQUEST_NEGATIVE_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x000E)
- #define **HCI_CHANGE_CONNECTION_PACKET_TYPE** HCI_OPCODE(OGF_LINK_CONTROL, 0x000F)
- #define **HCI_AUTHENTICATION_REQUESTED** HCI_OPCODE(OGF_LINK_CONTROL, 0x0011)
- #define **HCI_SET_CONNECTION_ENCRYPTION** HCI_OPCODE(OGF_LINK_CONTROL, 0x0013)
- #define **HCI_CHANGE_CONNECTION_LINK_KEY** HCI_OPCODE(OGF_LINK_CONTROL, 0x0015)
- #define **HCI_MASTER_LINK_KEY** HCI_OPCODE(OGF_LINK_CONTROL, 0x0017)
- #define **HCI_REMOTE_NAME_REQUEST** HCI_OPCODE(OGF_LINK_CONTROL, 0x0019)
- #define **HCI_REMOTE_NAME_REQUEST_CANCEL** HCI_OPCODE(OGF_LINK_CONTROL, 0x001A)
- #define **HCI_READ_REMOTE_SUPPORTED_FEATURES** HCI_OPCODE(OGF_LINK_CONTRO↩ L, 0x001B)
- #define **HCI_READ_REMOTE_EXTENDED_FEATURES** HCI_OPCODE(OGF_LINK_CONTROL, 0x001C)
- #define **HCI_READ_REMOTE_VERSION_INFORMATION** HCI_OPCODE(OGF_LINK_CONTRO↩ L, 0x001D)
- #define **HCI_READ_CLOCK_OFFSET** HCI_OPCODE(OGF_LINK_CONTROL, 0x001F)
- #define **HCI_READ_LMP_HANDLE** HCI_OPCODE(OGF_LINK_CONTROL, 0x0020)
- #define **HCI_SETUP_SYNCHRONOUS_CONNECTION** HCI_OPCODE(OGF_LINK_CONTROL, 0x0028)
- #define **HCI_ACCEPT_SYNCH_CONNECTION_REQUEST** HCI_OPCODE(OGF_LINK_CONTRO↩ L, 0x0029)
- #define **HCI_REJECT_SYNCH_CONNECTION_REQUEST** HCI_OPCODE(OGF_LINK_CONTRO↩ L, 0x002A)
- #define **HCI_IO_CAPABILITY_REQUEST_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x002B)
- #define **HCI_USER_CONFIRMATION_REQUEST_REPLY** HCI_OPCODE(OGF_LINK_CONTRO↩ L, 0x002C)
- #define **HCI_USER_CONFIRMATION_REQ_NEGATIVE_REPLY** HCI_OPCODE(OGF_LINK_CONTRO↩ L, 0x002D)
- #define **HCI_USER_PASSKEY_REQUEST_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x002E)
- #define **HCI_USER_PASSKEY_REQUEST_NEGATIVE_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x002F)
- #define **HCI_REMOTE_OOB_DATA_REQUEST_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x0030)
- #define **HCI_REMOTE_OOB_DATA_REQUEST_NEGATIVE_REPLY** HCI_OPCODE(OGF_LINK_CONT↩ ROL, 0x0033)
- #define **HCI_IO_CAPABILITY_REQUEST_NEGATIVE_REPLY** HCI_OPCODE(OGF_LINK_CONTROL, 0x0034)

Link policy commands

The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet.

- #define **HCI_HOLD_MODE** HCI_OPCODE(OGF_LINK_POLICY, 0x0001)
- #define **HCI_SNIFF_MODE** HCI_OPCODE(OGF_LINK_POLICY, 0x0003)
- #define **HCI_EXIT_SNIFF_MODE** HCI_OPCODE(OGF_LINK_POLICY, 0x0004)
- #define **HCI_PARK_STATE** HCI_OPCODE(OGF_LINK_POLICY, 0x0005)
- #define **HCI_EXIT_PARK_STATE** HCI_OPCODE(OGF_LINK_POLICY, 0x0006)
- #define **HCI_QOS_SETUP** HCI_OPCODE(OGF_LINK_POLICY, 0x0007)
- #define **HCI_ROLE_DISCOVERY** HCI_OPCODE(OGF_LINK_POLICY, 0x0009)
- #define **HCI_SWITCH_ROLE** HCI_OPCODE(OGF_LINK_POLICY, 0x000B)
- #define **HCI_READ_LINK_POLICY_SETTINGS** HCI_OPCODE(OGF_LINK_POLICY, 0x000C)
- #define **HCI_WRITE_LINK_POLICY_SETTINGS** HCI_OPCODE(OGF_LINK_POLICY, 0x000D)
- #define **HCI_READ_DEFAULT_POLICY_SETTINGS** HCI_OPCODE(OGF_LINK_POLICY, 0x000E)
- #define **HCI_WRITE_DEFAULT_POLICY_SETTINGS** HCI_OPCODE(OGF_LINK_POLICY, 0x000F)
- #define **HCI_FLOW_SPECIFICATION** HCI_OPCODE(OGF_LINK_POLICY, 0x0010)
- #define **HCI_SNIFF_SUBRATING** HCI_OPCODE(OGF_LINK_POLICY, 0x0011)

Controller & Baseband commands

The Controller & Baseband Commands provide access and control to various capabilities of the Bluetooth hardware.

- #define **HCI_SET_EVENT_MASK** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0001)
- #define **HCI_RESET** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0003)
- #define **HCI_SET_EVENT_FILTER** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0005)
- #define **HCI_FLUSH** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0008)
- #define **HCI_READ_PIN_TYPE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0009)
- #define **HCI_WRITE_PIN_TYPE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000A)
- #define **HCI_CREATE_NEW_UNIT_KEY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000B)
- #define **HCI_READ_STORED_LINK_KEY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x000D)
- #define **HCI_WRITE_STORED_LINK_KEY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0011)
- #define **HCI_DELETE_STORED_LINK_KEY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0012)
- #define **HCI_WRITE_LOCAL_NAME** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0013)
- #define **HCI_READ_LOCAL_NAME** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0014)
- #define **HCI_READ_CONNECTION_ACCEPT_TIMEOUT** HCI_OPCODE(OGF_CTRL_BASEBAN↩
D, 0x0015)
- #define **HCI_WRITE_CONNECTION_ACCEPT_TIMEOUT** HCI_OPCODE(OGF_CTRL_BASEBAN↩
D, 0x0016)
- #define **HCI_READ_PAGE_TIMEOUT** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0017)
- #define **HCI_WRITE_PAGE_TIMEOUT** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0018)
- #define **HCI_READ_SCAN_ENABLE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0019)
- #define **HCI_WRITE_SCAN_ENABLE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001A)
- #define **HCI_READ_PAGE_SCAN_ACTIVITY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001B)
- #define **HCI_WRITE_PAGE_SCAN_ACTIVITY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001C)
- #define **HCI_READ_INQUIRY_SCAN_ACTIVITY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001D)
- #define **HCI_WRITE_INQUIRY_SCAN_ACTIVITY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001E)
- #define **HCI_READ_AUTHENTICATION_ENABLE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x001F)
- #define **HCI_WRITE_AUTHENTICATION_ENABLE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0020)
- #define **HCI_READ_ENCRYPTION_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0021)
- #define **HCI_WRITE_ENCRYPTION_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0022)
- #define **HCI_READ_CLASS_OF_DEVICE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0023)
- #define **HCI_WRITE_CLASS_OF_DEVICE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0024)
- #define **HCI_READ_VOICE_SETTING** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0025)

- #define **HCI_WRITE_VOICE_SETTING** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0026)
- #define **HCI_READ_AUTOMATIC_FLASH_TIMEOUT** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0027)
- #define **HCI_WRITE_AUTOMATIC_FLASH_TIMEOUT** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0028)
- #define **HCI_READ_NUM_BROADCST_RETR** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0029)
- #define **HCI_WRITE_NUM_BROADCST_RETR** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002A)
- #define **HCI_READ_HOLD_MODE_ACTIVITY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002B)
- #define **HCI_WRITE_HOLD_MODE_ACTIVITY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002C)
- #define **HCI_READ_TRANSMIT_POWER_LEVEL** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x002D)
- #define **HCI_READ_SYNC_FLOW_CONTROL_ENABLE** HCI_OPCODE(OGF_CTRL_BASEBAN↩
D, 0x002E)
- #define **HCI_WRITE_SYNC_FLOW_CONTROL_ENABLE** HCI_OPCODE(OGF_CTRL_BASEBAN↩
D, 0x002F)
- #define **HCI_SET_CTRL_TO_HOST_FLOW_CONTROL** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0031)
- #define **HCI_HOST_BUFFER_SIZE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0033)
- #define **HCI_HOST_NUM_OF_COMPLETED_PACKETS** HCI_OPCODE(OGF_CTRL_BASEBAN↩
D, 0x0035)
- #define **HCI_READ_LINK_SUPERVISION_TIMEOUT** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0036)
- #define **HCI_WRITE_LINK_SUPERVISION_TIMEOUT** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0037)
- #define **HCI_READ_NUM_OF_SUPPORTED_IAC** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0038)
- #define **HCI_READ_CURRENT_IAC_LAP** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0039)
- #define **HCI_WRITE_CURRENT_IAC_LAP** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003A)
- #define **HCI_READ_PAGE_SCAN_PERIOD_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003B)
- #define **HCI_WRITE_PAGE_SCAN_PERIOD_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x003C)
- #define **HCI_SET_AFH_HOST_CHANNEL_CLASSIFICATION** HCI_OPCODE(OGF_CTRL_BASEBAND,
0x003F)
- #define **HCI_READ_INQUIRY_SCAN_TYPE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0042)
- #define **HCI_WRITE_INQUIRY_SCAN_TYPE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0043)
- #define **HCI_READ_INQUIRY_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0044)
- #define **HCI_WRITE_INQUIRY_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0045)
- #define **HCI_READ_PAGE_SCAN_TYPE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0046)
- #define **HCI_WRITE_PAGE_SCAN_TYPE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0047)
- #define **HCI_READ_AFH_CHANNEL_ASSESSMENT_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND,
0x0048)
- #define **HCI_WRITE_AFH_CHANNEL_ASSESSMENT_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND,
0x0049)
- #define **HCI_READ_EXTENDED_INQUIRY_RESPONSE** HCI_OPCODE(OGF_CTRL_BASEBAN↩
D, 0x0051)
- #define **HCI_WRITE_EXTENDED_INQUIRY_RESPONSE** HCI_OPCODE(OGF_CTRL_BASEBAN↩
D, 0x0052)
- #define **HCI_READ_REFRESH_ENCRYPTION_KEY** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0053)
- #define **HCI_READ_SIMPLE_PAIRING_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0055)
- #define **HCI_WRITE_SIMPLE_PAIRING_MODE** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0056)
- #define **HCI_READ_LOCAL_OOB_DATA** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0057)
- #define **HCI_READ_INQUIRY_RESPONSE_TX_POWER_LEVEL** HCI_OPCODE(OGF_CTRL_BASEBA↩
ND, 0x0058)
- #define **HCI_WRITE_INQUIRY_TX_POWER_LEVEL** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0059)
- #define **HCI_READ_DEFAULT_ERRONEOUS_DATA_REPORTING** HCI_OPCODE(OGF_CTRL_BASE↩
BAND, 0x005A)
- #define **HCI_WRITE_DEFAULT_ERRONEOUS_DATA_REPORTING** HCI_OPCODE(OGF_CTRL_BASE↩
BAND, 0x005B)
- #define **HCI_ENHANCED_FLUSH** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x005F)
- #define **HCI_SEND_KEY_PRESS_NOTIFICATION** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x0060)
- #define **HCI_READ_LE_HOST_SUPPORT** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x006C)
- #define **HCI_WRITE_LE_HOST_SUPPORT** HCI_OPCODE(OGF_CTRL_BASEBAND, 0x006D)

Informational Parameters

The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the BR/EDR Controller and the capabilities of the Link Manager and Baseband in the BR/EDR Controller and PAL in the AMP Controller. The host device cannot modify any of these parameters.

- #define **HCI_READ_LOCAL_VERSION_INFORMATION** HCI_OPCODE(OGF_INFORMATION, 0x0001)
- #define **HCI_READ_LOCAL_SUPPORTED_COMMANDS** HCI_OPCODE(OGF_INFORMATION, 0x0002)
- #define **HCI_READ_LOCAL_SUPPORTED_FEATURES** HCI_OPCODE(OGF_INFORMATION, 0x0003)
- #define **HCI_READ_LOCAL_EXTENDED_FEATURES** HCI_OPCODE(OGF_INFORMATION, 0x0004)
- #define **HCI_READ_BUFFER_SIZE** HCI_OPCODE(OGF_INFORMATION, 0x0005)
- #define **HCI_READ_BD_ADDR** HCI_OPCODE(OGF_INFORMATION, 0x0009)

Status Parameters

The Controller modifies all status parameters. These parameters provide information about the current state of the Link Manager and Baseband in the BR/EDR Controller and the PAL in an AMP Controller. The host device cannot modify any of these parameters other than to reset certain specific parameters.

- #define **HCI_READ_FAILED_CONTACT_COUNTER** HCI_OPCODE(OGF_STATUS, 0x0001)
- #define **HCI_RESET_FAILED_CONTACT_COUNTER** HCI_OPCODE(OGF_STATUS, 0x0002)
- #define **HCI_READ_LINK_QUALITY** HCI_OPCODE(OGF_STATUS, 0x0003)
- #define **HCI_READ_RSSI** HCI_OPCODE(OGF_STATUS, 0x0005)
- #define **HCI_READ_AFH_CHANNEL_MAP** HCI_OPCODE(OGF_STATUS, 0x0006)
- #define **HCI_READ_CLOCK_COMMAND** HCI_OPCODE(OGF_STATUS, 0x0007)

Testing Commands

The Testing commands are used to provide the ability to test various functional capabilities of the Bluetooth hardware.

- #define **HCI_READ_LOOPBACK_MODE** HCI_OPCODE(OGF_TESTING, 0x0001)
- #define **HCI_WRITE_LOOPBACK_MODE** HCI_OPCODE(OGF_TESTING, 0x0002)
- #define **HCI_ENABLE_DEVICE_UNDER_TEST_MODE** HCI_OPCODE(OGF_TESTING, 0x0003)
- #define **HCI_WRITE_SIMPLE_PAIRING_DEBUG_MODE** HCI_OPCODE(OGF_TESTING, 0x0004)

LE controller commands

The LE Controller Commands provide access and control to various capabilities of the Bluetooth hardware, as well as methods for the Host to affect how the Link Layer manages the piconet, and controls connections.

- #define **HCI_LE_SET_EVENT_MASK** HCI_OPCODE(OGF_LE, 0x0001)
- #define **HCI_LE_READ_BUFFER_SIZE** HCI_OPCODE(OGF_LE, 0x0002)
- #define **HCI_LE_READ_LOCAL_SUPPORTED_FEATURES** HCI_OPCODE(OGF_LE, 0x0003)
- #define **HCI_LE_SET_RANDOM_ADDRESS** HCI_OPCODE(OGF_LE, 0x0005)
- #define **HCI_LE_SET_ADVERTISING_PARAMETERS** HCI_OPCODE(OGF_LE, 0x0006)
- #define **HCI_LE_READ_ADVERTISING_CHANNEL_TX_POWER** HCI_OPCODE(OGF_LE, 0x0007)
- #define **HCI_LE_SET_ADVERTISING_DATA** HCI_OPCODE(OGF_LE, 0x0008)
- #define **HCI_LE_SET_SCAN_RESPONSE_DATA** HCI_OPCODE(OGF_LE, 0x0009)
- #define **HCI_LE_SET_ADVERTISE_ENABLE** HCI_OPCODE(OGF_LE, 0x000A)
- #define **HCI_LE_SET_SCAN_PARAMETERS** HCI_OPCODE(OGF_LE, 0x000B)
- #define **HCI_LE_SET_SCAN_ENABLE** HCI_OPCODE(OGF_LE, 0x000C)
- #define **HCI_LE_CREATE_CONNECTION** HCI_OPCODE(OGF_LE, 0x000D)

- #define **HCI_LE_CREATE_CONNECTION_CANCEL** HCI_OPCODE(OGF_LE, 0x000E)
- #define **HCI_LE_READ_WHITE_LIST_SIZE** HCI_OPCODE(OGF_LE, 0x000F)
- #define **HCI_LE_CLEAR_WHITE_LIST** HCI_OPCODE(OGF_LE, 0x0010)
- #define **HCI_LE_ADD_DEVICE_TO_WHITE_LIST** HCI_OPCODE(OGF_LE, 0x0011)
- #define **HCI_LE_REMOVE_DEVICE_FROM_WHITE_LIST** HCI_OPCODE(OGF_LE, 0x0012)
- #define **HCI_LE_CONNECTION_UPDATE** HCI_OPCODE(OGF_LE, 0x0013)
- #define **HCI_LE_SET_HOST_CHANNEL_CLASSIFICATION** HCI_OPCODE(OGF_LE, 0x0014)
- #define **HCI_LE_READ_CHANNEL_MAP** HCI_OPCODE(OGF_LE, 0x0015)
- #define **HCI_LE_READ_REMOTE_USED_FEATURES** HCI_OPCODE(OGF_LE, 0x0016)
- #define **HCI_LE_ENCRYPT** HCI_OPCODE(OGF_LE, 0x0017)
- #define **HCI_LE_RAND** HCI_OPCODE(OGF_LE, 0x0018)
- #define **HCI_LE_START_ENCRYPTION** HCI_OPCODE(OGF_LE, 0x0019)
- #define **HCI_LE_LONG_TERM_KEY_REQUEST_REPLY** HCI_OPCODE(OGF_LE, 0x001A)
- #define **HCI_LE_LONG_TERM_KEY_REQUEST_NEGATIVE_REPLY** HCI_OPCODE(OGF_LE, 0x001B)
- #define **HCI_LE_READ_SUPPORTED_STATES** HCI_OPCODE(OGF_LE, 0x001C)
- #define **HCI_LE_RECEIVE_TEST** HCI_OPCODE(OGF_LE, 0x001D)
- #define **HCI_LE_TRANSMITTER_TEST** HCI_OPCODE(OGF_LE, 0x001E)
- #define **HCI_LE_TEST_END** HCI_OPCODE(OGF_LE, 0x001F)

Events

- #define **HCI_EVT_INQUIRY_COMPLETE** 0x01
- #define **HCI_EVT_INQUIRY_RESULT** 0x02
- #define **HCI_EVT_CONNECTION_COMPLETE** 0x03
- #define **HCI_EVT_CONNECTION_REQUEST** 0x04
- #define **HCI_EVT_DISCONNECTION_COMPLETE** 0x05
- #define **HCI_EVT_AUTHENTICATION_COMPLETE** 0x06
- #define **HCI_EVT_REMOTE_NAME_REQUEST_COMPLETE** 0x07
- #define **HCI_EVT_ENCRYPTION_CHANGE** 0x08
- #define **HCI_EVT_CHANGE_CONN_LINK_COMPLETE** 0x09
- #define **HCI_EVT_MASTER_LINK_KEY_COMPLETE** 0x0A
- #define **HCI_EVT_READ_RMT_SUP_FEATURES_COMP** 0x0B
- #define **HCI_EVT_READ_RMT_VERSION_INFO_COMP** 0x0C
- #define **HCI_EVT_QOS_SETUP_COMPLETE** 0x0D
- #define **HCI_EVT_COMMAND_COMPLETE** 0x0E
- #define **HCI_EVT_COMMAND_STATUS** 0x0F
- #define **HCI_EVT_HARDWARE_ERROR** 0x10
- #define **HCI_EVT_FLUSH_OCCURED** 0x11
- #define **HCI_EVT_ROLE_CHANGE** 0x12
- #define **HCI_EVT_NUM_OF_COMPLETED_PACKETS** 0x13
- #define **HCI_EVT_MODE_CHANGE** 0x14
- #define **HCI_EVT_RETURN_LINK_KEYS** 0x15
- #define **HCI_EVT_PIN_CODE_REQUEST** 0x16
- #define **HCI_EVT_LINK_KEY_REQUEST** 0x17
- #define **HCI_EVT_LINK_KEY_NOTIFICATION** 0x18
- #define **HCI_EVT_LOOPBACK_COMMAND** 0x19
- #define **HCI_EVT_DATA_BUFFER_OVERFLOW** 0x1A
- #define **HCI_EVT_MAX_SLOTS_CHANGE** 0x1B
- #define **HCI_EVT_READ_CLOCK_OFFSET_COMPLETE** 0x1C
- #define **HCI_EVT_CONN_PACKET_TYPE_CHANGED** 0x1D
- #define **HCI_EVT_QOS_VIOLATION** 0x1E
- #define **HCI_EVT_PAGE_SCAN_REPET_MODE_CHANGE** 0x20
- #define **HCI_EVT_FLOW_SPECIFICATION_COMPLETE** 0x21
- #define **HCI_EVT_INQUIRY_RESULT_WITH_RSSI** 0x22

- #define **HCI_EVT_READ_RMT_EXT_FEATURES_COMP** 0x23
- #define **HCI_EVT_SYNCH_CONNECTION_COMPLETE** 0x2C
- #define **HCI_EVT_SYNCH_CONNECTION_CHANGED** 0x2D
- #define **HCI_EVT_SNIFF_SUBRATING** 0x2E
- #define **HCI_EVT_EXTENDED_INQUIRY_RESULT** 0x2F
- #define **HCI_EVT_ENCRYPTION_KEY_REFRESH_COMPLETE** 0x30
- #define **HCI_EVT_IO_CAPABILITY_REQUEST** 0x31
- #define **HCI_EVT_IO_CAPABILITY_RESPONSE** 0x32
- #define **HCI_EVT_USER_CONFIRMATION_REQUEST** 0x33
- #define **HCI_EVT_USER_PASSKEY_REQUEST** 0x34
- #define **HCI_EVT_REMOTE_OOB_DATA_REQUEST** 0x35
- #define **HCI_EVT_SIMPLE_PAIRING_COMPLETE** 0x36
- #define **HCI_EVT_LINK_SUPERVISION_TO_CHANGED** 0x38
- #define **HCI_EVT_ENHANCED_FLUSH_COMPLETE** 0x39
- #define **HCI_EVT_USER_PASSKEY_NOTIFICATION** 0x3B
- #define **HCI_EVT_KEYPRESS_NOTIFICATION** 0x3C
- #define **HCI_EVT_RMT_HOST_SUPP_FEATURES_NTF** 0x3D
- #define **HCI_EVT_LE_META_EVENT** 0x3E
- #define **HCI_EVT_LAST** HCI_EVT_LE_META_EVENT
- #define **HCI_EVT_FIRST** HCI_EVT_INQUIRY_COMPLETE

Errors

- #define **HCI_ERR_SUCCESS** 0x00
- #define **HCI_SUCCESS** 0x00
- #define **HCI_ERR_AUTHENTICATION_FAILURE** 0x05
- #define **HCI_ERR_MEMORY_CAPACITY_EXCEEDED** 0x07
- #define **HCI_ERR_CONNECTION_TIMEOUT** 0x08
- #define **HCI_ERR_SCO_CONN_LIMIT_EXCEEDED** 0x0a
- #define **HCI_ERR_ACL_CONN_ALREADY_EXISTS** 0x0b
- #define **HCI_ERR_CONN_REJECT_LIMITED_RESOURCES** 0x0d
- #define **HCI_ERR_INVALID_PARAMETERS** 0x12
- #define **HCI_ERR_UNSPECIFIED** 0x1F
- #define **HCI_ERR_SIMPLE_PAIRING_NOT_SUPPORTED** 0x37

3.2.1 Detailed Description

The Host Controller Interface (HCI) provides a uniform interface method of accessing a Bluetooth Controller's capabilities.

This module describes function and data structures used to send HCI commands and receive responses.

3.2.2 Typedef Documentation

3.2.2.1 typedef void(∗ bt_hci_connect_callback_fp) (bt_byte status, bt_hci_conn_state_t ∗pconn, void ∗param)

HCI connect callback.

This typedef defines a type for the callback function that is called when HCI connect operation initiated by a call to bt_hci_connect() is complete.

Parameters

status	Operation status. `It` is 0 if connection was successfully established.
pconn	pointer to a structure representing the established connection.
param	pointer to arbitrary data passed to the bt_hci_connect() function through its `param` parameter.

3.2.2.2 typedef void(∗ bt_hci_disconnect_callback_fp) (bt_byte status, bt_byte reason, bt_hci_conn_state_t ∗pconn, void ∗param)

HCI disconnect callback.

This typedef defines a type for the callback function that is called when an HCI connection has been terminated.

Parameters

status	Operation status. `It` is 0 if connection has been successfully terminated.
reason	Reason for disconnection.
pconn	pointer to a structure representing the connection.
param	pointer to arbitrary data associated with an event listener.

3.2.2.3 typedef void(∗ bt_hci_start_callback_fp) (bt_bool success, void ∗param)

HCI initialization callback.

This typedef defines a function pointer type for the HCI initialization callback functions. Such a function must be passed to the bt_hci_start() function.

Parameters

success	Specifies whether HCI initialization succeeded or not.

3.2.2.4 typedef void(∗ bt_hci_stop_callback_fp) (void ∗param)

HCI stop callback.

This typedef defines a function pointer type for the HCI stop callback functions. Such a function must be passed to the bt_hci_stop() function.

3.2.3 Function Documentation

3.2.3.1 bt_bool bt_hci_connect (bt_bdaddr_p *dest,* bt_uint *packet_type,* bt_byte *pg_scan_rpt_mode,* bt_byte *role_switch,* bt_uint *acl_config,* bt_hci_connect_callback_fp *callback,* void ∗ *param*)

Connect to a remote device.

This function tries to establish an HCI connection with a remote device specified by the Bluetooth address `dest`. Upon completion, the callback function specified by the `callback` parameter is called.

Parameters

dest	Bluetooth address of the remote device.
packet_type	

role_switch	
acl_config	
callback	Pointer to a callback function that is called when the connect operation completes.
param	Pointer to arbitrary data that is to be passed to the callback function.

Returns

- `TRUE` when the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.2.3.2 void bt_hci_init (void)

Initialize the HCI layer.

This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init.

This function essentially calls bt_hci_init_ex(HCI_LINK_POLICY_ENABLE_ALL) so all link policy setting are enabled.

3.2.3.3 void bt_hci_init_ex (bt_byte *default_link_policy*)

Initialize the HCI layer.

This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt_sys_init_ex.

Parameters

default_link_↩ policy	default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values: • HCI_LINK_POLICY_ENABLE_ROLE_SWITCH • HCI_LINK_POLICY_ENABLE_HOLD_MODE • HCI_LINK_POLICY_ENABLE_SNIFF_MODE • HCI_LINK_POLICY_ENABLE_PARK_STATE To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

3.2.3.4 bt_bool bt_hci_listen (bt_hci_connect_callback_fp *cb*, void ∗ *param*)

Listen for incoming connections.

Parameters

callback	Pointer to a callback function that is called when a new incoming connection has been established.
param	Pointer to arbitrary data that is to be passed to the callback function.

Returns

- `TRUE` when the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.2.3.5 bt_bool bt_hci_reset (bt_hci_cmd_callback_fp *callback*, void ∗ *callback_param*)

Reset controller.

This function resets the BT controller.

Parameters

callback	Completion callback. Called when the controller has been reset.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.2.3.6 bt_bool bt_hci_start (bt_hci_start_callback_fp *callback*, void ∗ *callback_param*, bt_byte *enable_scan*)

Start HCI layer.

This function starts the HCI layer of the stack. Starting the HCI layer consists essentially of two steps:

1. Make the HCI transport receive packets from the controller. This results in a call to bt_oem_recv.
2. Reset and configure the controller.

Upon completion of controller initialization the callback function passed in the `callback` parameter is called.

Parameters

callback	Completion callback. Called when controller initialization is complete.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.
enable_scan	This is a bitmask that defines which scans are enabled during the controller configuration. This value can be a combination of the following values: • HCI_SCAN_INQUIRY (the controller is discoverable) • HCI_SCAN_PAGE (the controller is connectable)

Returns

- `TRUE` when the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.2.3.7 void bt_hci_start_no_init (void)

Start HCI layer without controller configuration.

This function is similar to bt_hci_start but unlike the former it does not perform the controller configuration. I.e., bt_↩ hci_start_no_init simply calls the HCI transport and makes it receive packets from the controller. The main purpose of this function is make the HCI transport ready to exchange packets if controller needs some vendor specific configuration before it can be used with the stack. E.g., controllers based on CRS8811 chip need loading various values that configure its operating mode using CSR's proprietary protocol. So the application after configuring the HCI transport would call bt_hci_init(), bt_hci_start_no_init() and then load configuration values. Once the vendor specific configuration is done, the application will re-initialize the HCI transport and perform full start of the stack with bt_sys_init() and bt_sys_start().

3.2.3.8 void bt_hci_stop (bt_hci_stop_callback_fp *callback*, void ∗ *callback_param*)

Stop HCI layer.

This function makes the HCI layer inoperable. After this call the application must perform the full reset of the HCI transport and stack.

Parameters

callback	Completion callback. Called when the HCI layer has been stopped.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.2.3.9 bt_bool bt_hci_write_local_name (const char ∗ *device_name,* bt_hci_cmd_callback_fp *cb*)

Write local device name.

The Write_Local_Name command provides the ability to modify the userfriendly name for the BR/EDR Controller.

3.3 HCI UART (H4) transport protocol

Functions

- void bt_hcitr_uart_init (void)

  Initialize HCI UART (H4) transport protocol.
- void bt_hcitr_uart_reset (void)

  Re-initialize HCI UART (H4) transport protocol.
- void bt_hcitr_uart_start (void)

  Start HCI UART (H4) transport protocol.

3.3.1 Detailed Description

This module describes functions used to initialize and start HCI UART transport protocol. The transport uses common interface for exchanging data between the host CPU and HCI controller defined in bt_hcitr.h. This interface consist of two functions that must be implemented by the application:

- bt_oem_send()
- bt_oem_recv()

3.3.2 Function Documentation

3.3.2.1 void bt_hcitr_uart_init (void)

Initialize HCI UART (H4) transport protocol.

This function initializes internal structures of the transport. The application must call it as early as possible before bt_hcitr_uart_start and before the stack is initialized and started with bt_sys_init and bt_sys_start.

3.3.2.2 void bt_hcitr_uart_reset (void)

Re-initialize HCI UART (H4) transport protocol.

This function re-initializes the transport. Currently it simply calls bt_hcitr_uart_init. After calling this function the application must perform the full initialization of the stack by calling bt_sys_init, bt_sys_start and initialization functions of all other profile modules the application is intending to use.

3.3.2.3 void bt_hcitr_uart_start (void)

Start HCI UART (H4) transport protocol.

This function starts the transport, i.e., makes it able to receive and send packets.

3.4 Serial Port Profile (SPP)

The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile.

Modules

- SPP Configuration

    This module describes parameters used to configure SPP layer.
- RFCOMM Configuration

    This module describes parameters used to configure RFCOMM layer.

Data Structures

- struct bt_spp_port_t

    Serial port structure.
- struct bt_spp_port_t::_bt_spp_port_flags_t

Typedefs

- typedef void(∗ bt_spp_state_callback_fp) (bt_spp_port_t ∗port, bt_spp_port_event_e evt, void ∗param)

    Serial port state callback.
- typedef void(∗ bt_spp_send_callback_fp) (bt_spp_port_t ∗port, bt_ulong bytes_sent, bt_spp_send_status_e status, void ∗param)

    Serial port send callback.
- typedef void(∗ bt_spp_receive_callback_fp) (bt_spp_port_t ∗port, bt_int bytes_received, void ∗param)

    Serial port receive callback.

Enumerations

- enum bt_spp_port_state_e {
SPP_PORT_STATE_FREE, SPP_PORT_STATE_DISCONNECTED, SPP_PORT_STATE_DISCONNEC←
TING, SPP_PORT_STATE_CONNECTING,
SPP_PORT_STATE_CONNECTED }

    Serial port state.
- enum bt_spp_port_event_e {
SPP_PORT_EVENT_CONNECTED = 1, SPP_PORT_EVENT_DISCONNECTED, SPP_PORT_EVENT_←
CONNECTION_FAILED, SPP_PORT_EVENT_SEND_PROGRESS,
SPP_PORT_EVENT_REMOTE_MODEM_STATUS_CHANGED, SPP_PORT_EVENT_LOCAL_MODEM←
_STATUS_CHANGED, SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED }

    Serial port event.
- enum bt_spp_send_status_e {
SPP_SEND_STATUS_SUCCESS = 0, SPP_SEND_STATUS_TIMEOUT, SPP_SEND_STATUS_NOT_E←
NOUGH_RESOURCES, SPP_SEND_STATUS_CANCELED,
SPP_SEND_STATUS_INTERRUPTED }

    Send operation status.

Functions

- void bt_spp_init (void)

  Initialize the SPP module.
- bt_spp_port_t ∗ bt_spp_allocate (bt_l2cap_mgr_t ∗l2cap_mgr, bt_spp_state_callback_fp callback, void ∗callback_param)

  Allocate a serial port.
- bt_bool bt_spp_listen (bt_spp_port_t ∗port, bt_byte channel)

  Listen for incoming connections.
- bt_bool bt_spp_connect (bt_spp_port_t ∗port, bt_bdaddr_p remote_addr, bt_byte channel)

  Connect to a remote device.
- void bt_spp_disconnect (bt_spp_port_t ∗port)

  Disconnect from the remote device.
- bt_bool bt_spp_deallocate (bt_spp_port_t ∗port)

  Deallocate serial port.
- bt_bool bt_spp_send (bt_spp_port_t ∗port, const void ∗data, bt_ulong data_len, bt_spp_send_callback_fp callback)

  Send data.
- bt_bool bt_spp_receive (bt_spp_port_t ∗port, void ∗buffer, bt_int buffer_len, bt_spp_receive_callback_fp callback)

  Receive data.
- void bt_spp_cancel_send (bt_spp_port_t ∗port)

  Cancel send data.
- void bt_spp_cancel_receive (bt_spp_port_t ∗port)

  Cancel receive data.
- bt_int bt_spp_get_frame_length (bt_spp_port_t ∗port)

  Get frame length.
- bt_bdaddr_t ∗ bt_spp_get_remote_address (const bt_spp_port_t ∗port)

  Get the address of the remote device this device is connected to.
- bt_byte bt_spp_get_local_modem_status (const bt_spp_port_t ∗port)

  Get local device's TS 07.10 control signals.
- bt_byte bt_spp_set_local_modem_status (bt_spp_port_t ∗port, bt_byte ms)

  Set local device's TS 07.10 control signals.
- bt_byte bt_spp_get_remote_modem_status (const bt_spp_port_t ∗port)

  Get remote device's TS 07.10 control signals.
- bt_byte bt_spp_set_dtr (bt_spp_port_t ∗port, bt_bool on)

  Set local device's RS-232 DTR signal.
- bt_byte bt_spp_set_rts (bt_spp_port_t ∗port, bt_bool on)

  Set local device's RS-232 RTS signal.

3.4.1 Detailed Description

The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile.

Here are the steps for using this API:

- Call the bt_spp_init() function.
- Allocate a serial port structure with bt_spp_allocate(). One of the parameters to this function is a pointer to a callback function. That callback function will be called by the stack whenever the state of the serial port changes.

- To connect to a remote device call bt_spp_connect(). The stack will notify when the connection is established by calling the state callback function.
- To wait for a connection from a remote device call bt_spp_listen(). The stack will notify when the connection is established by calling the state callback function.
- When the port is connected you can send data with bt_spp_send() and receive data with bt_spp_receive().
- To terminate the connection call bt_spp_disconnect().
- When you are finished using the port deallocate it with bt_spp_deallocate().

3.4.2 Typedef Documentation

3.4.2.1 typedef void(∗ bt_spp_receive_callback_fp) (bt_spp_port_t ∗port, bt_int bytes_received, void ∗param)

Serial port receive callback.

This callback function is called when a receive operation initiated by bt_spp_receive() completes.

Parameters

port	Serial port on which the receive operation completed.
bytes_received	Number of received bytes.
param	Callback parameter that was specified when bt_spp_allocate() was called.

3.4.2.2 typedef void(∗ bt_spp_send_callback_fp) (bt_spp_port_t ∗port, bt_ulong bytes_sent, bt_spp_send_status_e status, void ∗param)

Serial port send callback.

This callback function is called when a send operation initiated by bt_spp_send() completes.

Parameters

port	Serial port on which the send operation completed.
bytes_sent	Number of bytes sent. This parameter is just a convenience as it always specifies the same number of bytes that was passed to bt_spp_send();
status	Completion status. It is one of the values defined in the bt_spp_send_status_e enumeration.
param	Callback parameter that was specified when bt_spp_allocate() was called.

3.4.2.3 typedef void(∗ bt_spp_state_callback_fp) (bt_spp_port_t ∗port, bt_spp_port_event_e evt, void ∗param)

Serial port state callback.

This callback function is called whenever the state of a serial port is changed.

Parameters

port	Serial port which state has changed.
evt	Event describing the nature of state change. It is one of the values defined in the bt_spp_↩ port_event_e enumeration.
param	Callback parameter that was specified when bt_spp_allocate() was called.

3.4.3 Enumeration Type Documentation

3.4.3.1 enum bt_spp_port_event_e

Serial port event.

Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt_spp_allocate().

Enumerator

SPP_PORT_EVENT_CONNECTED Connection with a remote device was successfully established. This event is reported independent of which side (local or remote) initiated the connection.

SPP_PORT_EVENT_DISCONNECTED Connection with the remote device was terminated. This event is reported independent of which side (local or remote) initiated termination of the connection.

SPP_PORT_EVENT_CONNECTION_FAILED Connection to a remote device failed. This event is reported when an attempt to establish a connection using bt_spp_connect() has failed.

SPP_PORT_EVENT_SEND_PROGRESS Send operation progress. This event is reported periodically during sending data over the serial port connection. It can be used by the application to track send operation progress.

SPP_PORT_EVENT_REMOTE_MODEM_STATUS_CHANGED Remote modem status. This event is reported when a remote device notifies the local device about the status of its V.24 control signals. The actual value of the signals can be obtained by calling bt_spp_get_remote_modem_status(const bt_spp←_port_t∗ port)

SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGED Local modem status. This event is reported when the local device has successfully sent the state of its V.24 signals to the remote party.

SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED Local modem status. This event is reported when the local device failed to send the state of its V.24 signals to the remote party.

3.4.3.2 enum bt_spp_port_state_e

Serial port state.

Values of this enumeration represent states of the serial port.

Enumerator

SPP_PORT_STATE_FREE Used internally to mark a port structure as available for allocation. The application should never encounter a port in this state.

SPP_PORT_STATE_DISCONNECTED Port is not connected.

SPP_PORT_STATE_DISCONNECTING Disconnecting from remote device.

SPP_PORT_STATE_CONNECTING Connecting to a remote device.

SPP_PORT_STATE_CONNECTED Port is connected to a remote device.

3.4.3.3 enum bt_spp_send_status_e

Send operation status.

Enumerator

SPP_SEND_STATUS_SUCCESS The operation completed successfully.

SPP_SEND_STATUS_TIMEOUT The operation timed out.

SPP_SEND_STATUS_NOT_ENOUGH_RESOURCES There was not enough resources to complete the operation.

SPP_SEND_STATUS_CANCELED The operation was canceled.

SPP_SEND_STATUS_INTERRUPTED The operation was interrupted because the connection closed.

3.4.4 Function Documentation

3.4.4.1 bt_spp_port_t∗ bt_spp_allocate (bt_l2cap_mgr_t ∗ *l2cap_mgr,* bt_spp_state_callback_fp *callback,* void ∗ *callback_param*)

Allocate a serial port.

The returned serial port is initially in the SPP_PORT_STATE_DISCONNECTED state. To establish a connection with a remote device, call bt_spp_connect(). To listen for incoming connections from other devices, call bt_spp←_listen(). The `callback` parameter must specify a callback function that will be used to notify about serial port events and state changes. When the port is not needed any more it must be deallocated by bt_spp_deallocate(). The maximum number of serial ports that can be allocated simultaneously is specified by the SPP_MAX_PORTS configuration parameter.

Parameters

l2cap_mgr	L2CAP manager.
callback	Pointer to a callback function used to notify about serial port events. Cannot be NULL.
callback_param	An arbitrary pointer that is passed as a parameter to the callback function.

Returns

A pointer to the ::bt_spp_port_t structure. Returns NULL if the maximum number of ports has been already allocated or the `callback` parameter is NULL.

3.4.4.2 void bt_spp_cancel_receive (bt_spp_port_t ∗ *port*)

Cancel receive data.

If a receive operation is currently in progress this function will cancel it. After calling this function the receive callback specified in bt_spp_receive() will not be called.

If there is no receive operation in progress calling this function has no effect.

Parameters

port	Serial port.

3.4.4.3 void bt_spp_cancel_send (bt_spp_port_t ∗ *port*)

Cancel send data.

If a send operation is currently in progress this function will try to cancel it. When the operation is canceled the send callback function will be called with the SPP_SEND_STATUS_CANCELED status.

If this function is called but the active send operation completes successfuly before the stack can actually cancel it the call back function will still be called with the SPP_SEND_STATUS_CANCELED status.

If there is no send operation in progess calling this function has no effect.

Parameters

port	Serial port.

3.4.4.4 bt_bool bt_spp_connect (bt_spp_port_t ∗ *port,* bt_bdaddr_p *remote_addr,* bt_byte *channel*)

Connect to a remote device.

This function initiates a connection to a remote device. When the connection is successfully established the port's callback is called with the SPP_PORT_EVENT_CONNECTED event. If connection fails the callback is called with the SPP_PORT_EVENT_CONNECTION_FAILED event.

The port must be in SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

Parameters

port	Serial port.
remote_addr	Bluetooth address of the remote device.
channel	RFCOMM server channel on which the connection is to be established.

Returns

`TRUE` if successful, `FALSE` otherwise.

3.4.4.5 bt_bool bt_spp_deallocate (bt_spp_port_t * *port*)

Deallocate serial port.

This function deallocates the specified port structure and other resources associated with it.

The port must be in SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

If the function completes successfully the application must not try to access any fields in the structure and must not use it with any other SPP functions. Also, it becomes available for subsequent allocation by bt_spp_port_allocate().

Parameters

port	Serial port structure to deallocate.

Returns

`TRUE` if successful, `FALSE` otherwise.

3.4.4.6 void bt_spp_disconnect (bt_spp_port_t * *port*)

Disconnect from the remote device.

This function initiates the disconnection process. When it is complete the the port's callback is called with the SPP_PORT_EVENT_DISCONNECTED event.

If the port is already in the disconnected state the function does nothing and the callback is not called.

Parameters

port	Serial port.

3.4.4.7 bt_int bt_spp_get_frame_length (bt_spp_port_t * *port*)

Get frame length.

This function returns the RFCOMM frame length used by the RFCOMM protocol. The frame length depends on configuration of DotStack and configuration of the Bluetooth stack running on the remote device. In order to achieve maximum throughput over the serial port connection the application should send and receive data in chunks that are multiple of this frame length.

Returns

RFCOMM frame length in bytes.

3.4.4.8 bt_byte bt_spp_get_local_modem_status (const bt_spp_port_t * *port*)

Get local device's TS 07.10 control signals.

This function returns current state of the local device's TS 07.10 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DSR SPP_RS232_RTS SPP_RS232_RI SPP_RS232_DCD

Parameters

port	Serial port.

Returns

local device's TS 07.10 control signals.

3.4.4.9 bt_bdaddr_t∗ bt_spp_get_remote_address (const bt_spp_port_t ∗ *port*)

Get the address of the remote device this device is connected to.

Parameters

port	Serial port.

Returns

- `A` pointer to bt_bdaddr structure that contains the address of the remote device.

3.4.4.10 bt_byte bt_spp_get_remote_modem_status (const bt_spp_port_t ∗ *port*)

Get remote device's TS 07.10 control signals.

This function returns current state of the remote device's V.24 controls signals. The signals are defined as a mask of the following constants: SPP_RS232_DTR SPP_RS232_CTS SPP_RS232_RI SPP_RS232_DCD

Parameters

port	Serial port.

Returns

remote device's TS 07.10 control signals.

3.4.4.11 void bt_spp_init (void)

Initialize the SPP module.

This function initializes all internal variables of the SPP module. It must be called prior to using any other functions in this module.

3.4.4.12 bt_bool bt_spp_listen (bt_spp_port_t ∗ *port,* bt_byte *channel*)

Listen for incoming connections.

This function registers the port to accept incoming connections from remote devices on a particular RFCOMM server channel. The specified server channel should be listed in the SDP database. Otherwise, remote devices will not be able to find out which server channel to use.

When a remote device successfully establishes a connection on the specified port the port's callback is called with the SPP_PORT_EVENT_CONNECTED event.

The port must be in SPP_PORT_STATE_DISCONNECTED state. Otherwise, the function will fail.

Parameters

port	Serial port.
channel	The RFCOMM server channel on which to listen for connections.

Returns

TRUE if successful, FALSE otherwise.

3.4.4.13 bt_bool bt_spp_receive (bt_spp_port_t ∗ *port,* void ∗ *buffer,* bt_int *buffer_len,* bt_spp_receive_callback_fp *callback*)

Receive data.

This function receives data from the serial port connection. The caller must provide a buffer and a callback function. Whenever the port receives data they are copied to the provided buffer and the callback function is called. The callback function is passed the length of received data and the same callback parameter that was specified when the port was allocated with bt_spp_allocate(). This function does not wait until the buffer is filled out completely. Any amount of received data will complete the operation.

The port must be in SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started receive operation is still in progress.

Parameters

port	Serial port.
data	Pointer to the data to be sent.
data_len	Length of the data.
callback	Send callback function.

Returns

TRUE if successful, FALSE otherwise.

3.4.4.14 bt_bool bt_spp_send (bt_spp_port_t ∗ *port,* const void ∗ *data,* bt_ulong *data_len,* bt_spp_send_callback_fp *callback*)

Send data.

This function starts sending data over the serial port connection. Along with the data the caller must provide a callback function that is called when all data has been sent. Also, during execution of this operation the port's state callback function is called periodically with the SPP_PORT_EVENT_SEND_PROGRESS event.

The port must be in SPP_PORT_STATE_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started send operation is still in progress.

The callback function is passed the same callback parameter that was specified when the port was allocated with bt_spp_allocate().

Parameters

port	Serial port.
data	Pointer to the data to be sent.
data_len	Length of the data.
callback	Send callback function.

Returns

TRUE if successful, FALSE otherwise.

3.4.4.15 bt_byte bt_spp_set_dtr (bt_spp_port_t ∗ *port,* bt_bool *on*)

Set local device's RS-232 DTR signal.

Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTC signal.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE←D event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

Parameters

port	Serial port.
on	Value indicating weather to set or clear the signal.

Returns

`TRUE` if successful, `FALSE` otherwise.

3.4.4.16 bt_byte bt_spp_set_local_modem_status (bt_spp_port_t ∗ *port,* bt_byte *ms*)

Set local device's TS 07.10 control signals.

Changes the state of local device's TS 07.10 control signals and notifies the remote device of the change.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE←D event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

Parameters

port	Serial port.
ms	Signals mask.

Returns

`TRUE` if successful, `FALSE` otherwise.

3.4.4.17 bt_byte bt_spp_set_rts (bt_spp_port_t ∗ *port,* bt_bool *on*)

Set local device's RS-232 RTS signal.

Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM_MODEM_STATUS_RTR signal.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE←D event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP_PORT_EVENT_LOCAL_MODEM_STATUS_CHANGE_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

Parameters

port	Serial port.
on	Value indicating weather to set or clear the signal.

Returns

`TRUE` if successful, `FALSE` otherwise.

3.5 SPP Configuration

This module describes parameters used to configure SPP layer.

Macros

- #define SPP_MAX_PORTS

  Maximum number of SPP ports.

3.5.1 Detailed Description

This module describes parameters used to configure SPP layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS             ...
#define HCI_MAX_DATA_BUFFERS            ...
#define HCI_MAX_HCI_CONNECTIONS         ...
#define HCI_RX_BUFFER_LEN               ...
#define HCI_TX_BUFFER_LEN               ...
#define HCI_L2CAP_BUFFER_LEN            ...
#define HCI_MAX_CMD_PARAM_LEN           ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS           ...
#define L2CAP_MAX_FRAME_BUFFERS         ...
#define L2CAP_MAX_PSMS                  ...
#define L2CAP_MAX_CHANNELS              ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN      ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN    ...

// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure SPP,
// the following values must be defined:

#define RFCOMM_MAX_SESSIONS             ...
#define RFCOMM_MAX_DLCS                 ...
#define RFCOMM_MAX_SERVER_CHANNELS      ...
#define RFCOMM_INFO_LEN                 ...
#define RFCOMM_MAX_DATA_BUFFERS         ...
#define RFCOMM_MAX_CMD_BUFFERS          ...
#define RFCOMM_LOCAL_CREDIT             ...

#define SPP_MAX_PORTS                   ...

#include "cdbt/bt/bt_oem_config.h"
```

3.5.2 Macro Definition Documentation

3.5.2.1 #define SPP_MAX_PORTS

Maximum number of SPP ports.

This parameter defines the maximum number of SPP port that can be open between the local and remote devices. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is FALSE (default) this value should be equal to RFCOMM↩_MAX_SERVER_CHANNELS. If RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE this value should be between RFCOMM_MAX_SERVER_CHANNELS and RFCOMM_MAX_SERVER_CHANNELS ∗ RFCOMM_MA↩X_SESSIONS.

3.6 RFCOMM Configuration

This module describes parameters used to configure RFCOMM layer.

Macros

- #define RFCOMM_MAX_SESSIONS

  Maximum number of remote devices a local device can be connected to.
- #define RFCOMM_MAX_DLCS

  Maximum number of DLCs.
- #define RFCOMM_MAX_SERVER_CHANNELS

  Maximum number of Server channels.
- #define RFCOMM_INFO_LEN

  Maximum size of the data portion of a UIH frame.
- #define RFCOMM_MAX_CMD_BUFFERS

  Maximum number of command buffers.
- #define RFCOMM_LOCAL_CREDIT

  The number of receive buffers.
- #define RFCOMM_ENABLE_MULTIDEVICE_CHANNELS BT_FALSE

  Enable multi-device server channels.

3.6.1 Detailed Description

This module describes parameters used to configure RFCOMM layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS            ...
#define HCI_MAX_DATA_BUFFERS           ...
#define HCI_MAX_HCI_CONNECTIONS        ...
#define HCI_RX_BUFFER_LEN              ...
#define HCI_TX_BUFFER_LEN              ...
#define HCI_L2CAP_BUFFER_LEN           ...
#define HCI_MAX_CMD_PARAM_LEN          ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS          ...
#define L2CAP_MAX_FRAME_BUFFERS        ...
#define L2CAP_MAX_PSMS                 ...
#define L2CAP_MAX_CHANNELS             ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN     ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN   ...

// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure RFCOMM,
// the following values must be defined:

#define RFCOMM_MAX_SESSIONS            ...
#define RFCOMM_MAX_DLCS                ...
#define RFCOMM_MAX_SERVER_CHANNELS     ...
#define RFCOMM_INFO_LEN                ...
#define RFCOMM_MAX_DATA_BUFFERS        ...
#define RFCOMM_MAX_CMD_BUFFERS         ...
#define RFCOMM_LOCAL_CREDIT            ...

#include "cdbt/bt/bt_oem_config.h"
```

3.6.2 Macro Definition Documentation

3.6.2.1 #define RFCOMM_ENABLE_MULTIDEVICE_CHANNELS BT_FALSE

Enable multi-device server channels.

Normally each server channel can be used only once. I.e. if device A connected to channel 1, device B cannot connect to channel 1 until device A disconnects. With this option it is possible to make channels accept connections from several devices at the same time. I.e., if RFCOMM_ENABLE_MULTIDEVICE_CHANNELS is TRUE both device A and device B can connect to channel 1 at the same time.

3.6.2.2 #define RFCOMM_INFO_LEN

Maximum size of the data portion of a UIH frame.

This parameter defines the maximum size of the data portion of a UIH frame. If CFC is used the actual length of the data portion will be 1 byte less. This value must be less than or equal to HCI_L2CAP_BUFFER_LEN - RFCOMM_FRAME_HEADER_LEN - L2CAP_HEADER_LEN.

3.6.2.3 #define RFCOMM_LOCAL_CREDIT

The number of receive buffers.

This parameter defines the number of received UIH frames that can be stored on the local device. The flow control mechanism used in RFCOMM ensures that the remote side of the link always knows how many free buffers left on the local device. When the number of free buffers reaches 0, the transmitter stops sending data frames until the receiver frees some buffers. The RFCOMM layer does not actually allocate space for buffers. It uses RFCOMM_↩
LOCAL_CREDIT to keep track of free buffers and report them to the remote side. Actual memory allocation is done in SPP layer.

3.6.2.4 #define RFCOMM_MAX_CMD_BUFFERS

Maximum number of command buffers.

This parameter defines the maximum number of commands that can be sent at the same time. It is usually enough to reserve 2 buffers for each DLC excluding control DLC. Therefore, this value can be defined as
#define RFCOMM_MAX_CMD_BUFFERS (RFCOMM_MAX_DLCS - 1) $*$ 2

3.6.2.5 #define RFCOMM_MAX_DLCS

Maximum number of DLCs.

This parameter defines the maximum number of DLCs on each session. This value should be at least 2 because each session uses one DLC to convey multiplexer control messages. All other DLCs are used to emulate serial ports.

3.6.2.6 #define RFCOMM_MAX_SERVER_CHANNELS

Maximum number of Server channels.

This parameter defines the maximum number of server channels exposed by the local device. This value should not exceed RFCOMM_MAX_DLCS - 1.

3.6.2.7 #define RFCOMM_MAX_SESSIONS

Maximum number of remote devices a local device can be connected to.

This parameter defines the maximum number of remote devices a local device can have simultaneous connections to. This value should not exceed HCI_MAX_HCI_CONNECTIONS.

3.7 SDP

This module describe functions and data structures used to start the SDP server and perform SDP queries.

Modules

- Configuration

  This module describes parameters used to configure SDP.

Macros

- #define BEGIN_DE_SEQUENCE(id, len)

  Begin a data element sequence.
- #define END_DE_SEQUENCE(id) }}

  End a data element sequence.
- #define INIT_DE_SEQUENCE(id) init_de_sequence_##id();

  Initialize a data element sequence.
- #define DE_UINT(value) cur_de->type = SDP_DATA_TYPE_UINT; cur_de->data.b = value; cur_de++; if (++i == max_len) return;

  Declare a 1-byte unsigned integer data element.
- #define DE_UINT16(value) cur_de->type = SDP_DATA_TYPE_UINT16; cur_de->data.ui = value; cur_↩ de++; if (++i == max_len) return;

  Declare a 2-byte unsigned integer data element.
- #define DE_INT(value) cur_de->type = SDP_DATA_TYPE_INT; cur_de->data.b = value; cur_de++; if (++i == max_len) return;

  Declare a 1-byte signed integer data element.
- #define DE_STRING(value) cur_de->type = SDP_DATA_TYPE_STRING; cur_de->data.pstr = value; cur↩ _de++;

  Declare a text string data element.
- #define DE_STRING2(value, len) cur_de->type = SDP_DATA_TYPE_UINT; cur_de->data.pstr = value; cur_de->bytecount = len; cur_de++; if (++i == max_len) return;

  Declare a text string data element.
- #define DE_BOOL(value) cur_de->type = SDP_DATA_TYPE_BOOL; cur_de->data.b = value; cur_de++; if (++i == max_len) return;

  Declare a boolean data element.
- #define DE_UUID16(value) cur_de->type = SDP_DATA_TYPE_UUID16; cur_de->data.uuid16 = value; cur_de++; if (++i == max_len) return;

  Declare a 16-bit UUID data element.
- #define DE_UUID32(value) cur_de->type = SDP_DATA_TYPE_UUID32; cur_de->data.uuid32 = value; cur_de++; if (++i == max_len) return;

  Declare a 32-bit UUID data element.
- #define DE_UUID128(value) cur_de->type = SDP_DATA_TYPE_UUID128; cur_de->data.uuid128 = (bt_↩ uuid_t*)&value; cur_de++; if (++i == max_len) return;

  Declare a 128-bit UUID data element.
- #define DE_URL(value) cur_de->type = SDP_DATA_TYPE_URL; cur_de->data.purl = value; cur_de++; if (++i == max_len) return;

  Declare a URL data element.

Functions

- bt_bool bt_sdp_start (bt_l2cap_mgr_p l2cap_mgr, const bt_byte ∗sdp_db, bt_uint sdp_db_len)

  Start SDP server.
- bt_bool bt_sdp_request_service_search (bt_l2cap_channel_t ∗channel, bt_sdp_data_element_p pattern, bt_sdp_service_search_callback_fp callback, void ∗callback_param)

  Search service records.
- bt_bool bt_sdp_request_service_attribute (bt_l2cap_channel_t ∗channel, bt_sr_handle_t sr, bt_sdp_data_↩ element_p pattern, bt_sdp_service_attribute_callback_fp callback, void ∗callback_param)

  Search attributes.

3.7.1 Detailed Description

This module describe functions and data structures used to start the SDP server and perform SDP queries.

3.7.2 Macro Definition Documentation

3.7.2.1 #define BEGIN_DE_SEQUENCE(*id, len*)

Value:

```
static bt_sdp_data_element_t id[len];  \
    static bt_sdp_sequence_t seq_##id = { len, (
      bt_sdp_data_element_p)id}; \
    static void init_de_sequence_##id() {   \
        static bt_bool initialized = FALSE; \
        if (!initialized) { \
            bt_int i = 0;   \
            bt_int max_len = len;   \
            bt_sdp_data_element_t* cur_de = id; \
            initialized = TRUE;
```

Begin a data element sequence.

BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of sdp_data_element structures. The array is used a search pattern in bt_sdp_request_service_search() and bt_sdp_request_service_attribute(). For example, to find a AVRCP Target the following code can be used:

```
const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000,
       SDP_CLSID_AV_REMOTE_CONTROL };
const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000,
       SDP_CLSID_AV_REMOTE_CONTROL_TARGET };

BEGIN_DE_SEQUENCE(avrcp_target_service_search, 2)
   DE_UUID128(AVRCP_AV_REMOTE_CONTROL_CLSID)
   DE_UUID128(AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID)
END_DE_SEQUENCE(avrcp_target_service_search)

.
.
.

void findAvrcpTarget(void)
{
   INIT_DE_SEQUENCE(avrcp_target_service_search);

   bt_sdp_request_service_search(channel, &seq_avrcp_target_service_search, &callback, NULL);
}
```

Parameters

id	The data element sequence identifier.
len	The number of elements in the data element sequence.

3.7.2.2 #define DE_BOOL(*value*) cur_de->type = SDP_DATA_TYPE_BOOL; cur_de->data.b = value; cur_de++; if (++i == max_len) return;

Declare a boolean data element.

This macro adds a boolean data element to a data element sequence. This macro is to be used between BEGIN↩ _DE_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value.

3.7.2.3 #define DE_INT(*value*) cur_de->type = SDP_DATA_TYPE_INT; cur_de->data.b = value; cur_de++; if (++i == max_len) return;

Declare a 1-byte signed integer data element.

This macro adds a 1-byte signed integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value.

3.7.2.4 #define DE_STRING(*value*) cur_de->type = SDP_DATA_TYPE_STRING; cur_de->data.pstr = value; cur_de++;

Declare a text string data element.

This macro adds a text string data element to a data element sequence. The length of the generated data element will be the actual length of the string. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE↩ _SEQUENCE.

Parameters

value	The data element value.

3.7.2.5 #define DE_STRING2(*value*, *len*) cur_de->type = SDP_DATA_TYPE_UINT; cur_de->data.pstr = value; cur_de->bytecount = len; cur_de++; if (++i == max_len) return;

Declare a text string data element.

This macro adds a text string data element to a data element sequence. The length of the generated data element will be the value specified by the "len" parameter even if the actual length of the string is not equal to the "len" value. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value.
len	The length of the data element value.

3.7.2.6 #define DE_UINT(*value*) cur_de->type = SDP_DATA_TYPE_UINT; cur_de->data.b = value; cur_de++; if (++i == max_len) return;

Declare a 1-byte unsigned integer data element.

This macro adds a 1-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value.

3.7.2.7 #define DE_UINT16(*value*) cur_de->type = SDP_DATA_TYPE_UINT16; cur_de->data.ui = value; cur_de++; if (++i == max_len) return;

Declare a 2-byte unsigned integer data element.

This macro adds a 2-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value.

3.7.2.8 #define DE_URL(*value*) cur_de->type = SDP_DATA_TYPE_URL; cur_de->data.purl = value; cur_de++; if (++i == max_len) return;

Declare a URL data element.

This macro adds a URL data element to a data element sequence. This macro is to be used between BEGIN_D↩
E_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value which is a pointer to a string.

3.7.2.9 #define DE_UUID128(*value*) cur_de->type = SDP_DATA_TYPE_UUID128; cur_de->data.uuid128 = (bt_uuid_t∗)&value; cur_de++; if (++i == max_len) return;

Declare a 128-bit UUID data element.

This macro adds a 128-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value. The value must be a name of a variable of type bt_uuid.

3.7.2.10 #define DE_UUID16(*value*) cur_de->type = SDP_DATA_TYPE_UUID16; cur_de->data.uuid16 = value; cur_de++; if (++i == max_len) return;

Declare a 16-bit UUID data element.

This macro adds a 16-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value.

3.7.2.11 #define DE_UUID32(*value*) cur_de->type = SDP_DATA_TYPE_UUID32; cur_de->data.uuid32 = value; cur_de++; if (++i == max_len) return;

Declare a 32-bit UUID data element.

This macro adds a 32-bit UUID data element to a data element sequence. This macro is to be used between BEGIN_DE_SEQUENCE and END_DE_SEQUENCE.

Parameters

value	The data element value.

3.7.2.12 #define END_DE_SEQUENCE(*id*) }}

End a data element sequence.

BEGIN_DE_SEQUENCE and END_DE_SEQUENCE are used to define a data element sequence which is an array of bt_sdp_data_element structures.

Parameters

id	The data element sequence identifier.

3.7.2.13 #define INIT_DE_SEQUENCE(*id*) init_de_sequence_##id();

Initialize a data element sequence.

This macro calls a function defined in BEGIN_DE_SEQUENCE which initializes the data element sequence.

Parameters

id	The data element sequence identifier.

3.7.3 Function Documentation

3.7.3.1 bt_bool bt_sdp_request_service_attribute (bt_l2cap_channel_t * *channel,* bt_sr_handle_t *sr,* bt_sdp_data_element_p *pattern,* bt_sdp_service_attribute_callback_fp *callback,* void * *callback_param*)

Search attributes.

This function retrieves attribute values from a service record.

Parameters

channel	The L2CAP channel used to communicate to the remote SDP server.
sr	The service record handle specifies the service record from which attribute values are to be retrieved.
pattern	The attribute search pattern is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. The pattern buffer must be valid for the duration of the search operation, i.e. until `callback` is called for the first time. To define a data element sequence use the BEGIN_DE_SEQUENCE and END_DE_SEQUENCE macros. These macros will define a variable whose name is the id of the data element sequence passed to the macros prefixed with "seq_". A pointer to this variable can be used as the value for the pattern parameter.

callback	The callback function that will be called when search has completed.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.7.3.2 bt_bool bt_sdp_request_service_search (bt_l2cap_channel_t ∗ *channel,* bt_sdp_data_element_p *pattern,* bt_sdp_service_search_callback_fp *callback,* void ∗ *callback_param*)

Search service records.

This function locates service records on a remote SDP server that match the given service search pattern.

Parameters

channel	The L2CAP channel used to communicate to the remote SDP server.
pattern	The service search pattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12. The pattern buffer must be valid for the duration of the search operation, i.e. until `callback` is called. To define a data element sequence use the BEGIN_DE_S←EQUENCE and END_DE_SEQUENCE macros. These macros will define a variable whose name is the id of the data element sequence passed to the macros prefixed with "seq_". A pointer to this variable can be used as the value for the pattern parameter.
callback	The callback function that will be called when search has completed.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.7.3.3 bt_bool bt_sdp_start (bt_l2cap_mgr_p *l2cap_mgr,* const bt_byte ∗ *sdp_db,* bt_uint *sdp_db_len*)

Start SDP server.

This function starts the SDP server.

Parameters

l2cap_mgr	The L2CAP manager on which the SDP server is to be started.
sdp_db	A pointer to the SDP database define with BEGIN_SDP_DB and END_SDP_DB macros.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.8 Configuration

This module describes parameters used to configure SDP.

Macros

- #define SDP_MAX_PDU_BUFFERS

    Maximum number of SDP server PDU buffers.
- #define SDP_MAX_SEARCH_RESULT_LEN

    Maximum number of service records to find.
- #define SDP_MAX_ATTRIBUTE_RESULT_LEN

    Maximum number of attributes to find.

3.8.1 Detailed Description

This module describes parameters used to configure SDP.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS           ...
#define HCI_MAX_DATA_BUFFERS          ...
#define HCI_MAX_HCI_CONNECTIONS       ...
#define HCI_RX_BUFFER_LEN             ...
#define HCI_TX_BUFFER_LEN             ...
#define HCI_L2CAP_BUFFER_LEN          ...
#define HCI_MAX_CMD_PARAM_LEN         ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS         ...
#define L2CAP_MAX_FRAME_BUFFERS       ...
#define L2CAP_MAX_PSMS                ...
#define L2CAP_MAX_CHANNELS            ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN    ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN  ...

#include "cdbt/bt/bt_oem_config.h"
```

3.8.2 Macro Definition Documentation

3.8.2.1 #define SDP_MAX_ATTRIBUTE_RESULT_LEN

Maximum number of attributes to find.

This parameter defines the maximum number of attributes withing a service record the SDP server will return to the client.

3.8.2.2 #define SDP_MAX_PDU_BUFFERS

Maximum number of SDP server PDU buffers.

This parameter defines the maximum number of responses the SDP server can send at the same time.

3.8.2.3 #define SDP_MAX_SEARCH_RESULT_LEN

Maximum number of service records to find.

This parameter defines the maximum number of service records the SDP server will return to the client.

3.9 HID Profile (HIDP)

Functions

- bt_bool bt_hid_init (void)

    Initialize HID layer.
- bt_bool bt_hid_listen (bt_hid_session_p session, bt_hid_state_callback_fp callback, bt_hid_get_report_↩ callback_fp get_report_callback, bt_hid_set_report_callback_fp set_report_callback)

    Listen for incoming connections.
- bt_bool bt_hid_connect (bt_hid_session_p session, bt_bdaddr_p remote_addr, bt_hid_state_callback_↩ fp callback, bt_hid_get_report_callback_fp get_report_callback, bt_hid_set_report_callback_fp set_report↩ _callback)

    Connect to a remote device.
- bt_bool bt_hid_disconnect (bt_hid_session_p session)

    Close connection.
- void bt_hid_unplug (bt_hid_session_p session)

    Unplug device.
- bt_hid_session_p bt_hid_allocate_session (bt_l2cap_mgr_p l2cap_mgr, void ∗callback_param)

    Allocate HID session.
- void bt_hid_free_session (bt_hid_session_p session)

    Release HID session.
- bt_bool bt_hid_add_report (bt_hid_session_p session, bt_hid_report_t ∗report)

    Add report definition to local HID device.
- bt_bool bt_hid_send_report (bt_hid_session_p session, bt_byte channel, bt_byte report_type, bt_byte report_id, bt_bool send_report_id, bt_byte_p data, bt_int len, bt_byte tran_type, bt_hid_send_report_↩ callback_fp callback)

    Send report.

3.9.1 Detailed Description

3.9.2 Function Documentation

3.9.2.1 bt_bool bt_hid_add_report (bt_hid_session_p *session,* bt_hid_report_t ∗ *report*)

Add report definition to local HID device.

This function add a report definitions to the specified HID session.

Parameters

session	The HID session which a report definition to be added to.
report	The report definition to be added.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.9.2.2 bt_hid_session_p bt_hid_allocate_session (bt_l2cap_mgr_p *l2cap_mgr,* void ∗ *callback_param*)

Allocate HID session.

This function allocates a new HID session.

Parameters

l2cap_mgr	The L2CAP manager on which the HID session is to be created.

Returns

- A pointer to the new HID session structure if the function succeeds.
- `NULL` otherwise.

3.9.2.3 bt_bool bt_hid_connect (bt_hid_session_p *session,* bt_bdaddr_p *remote_addr,* bt_hid_state_callback_fp *callback,* bt_hid_get_report_callback_fp *get_report_callback,* bt_hid_set_report_callback_fp *set_report_callback*)

Connect to a remote device.

This function establishes a HID connection with a remote device. Changes in the session state are reported through a callback function.

Parameters

session	HID session.
remote_addr	Address of the remote device.
callback	The callback function that is called when session state changes.
get_report_↩ callback	The callback function that is called when a remote device requests an INPUT report.
set_report_↩ callback	The callback function that is called when a remote device sends and OUTPUT report.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.9.2.4 bt_bool bt_hid_disconnect (bt_hid_session_p *session*)

Close connection.

This function closes a HID connection with a remote device. Changes in the session state are reported through a callback function set when connection was created by calling bt_hid_connect.

Parameters

session	HID session.

3.9.2.5 void bt_hid_free_session (bt_hid_session_p *session*)

Release HID session.

This function deallocated the specified HID session. This function does not disconnect the session. It just frees the memory used by the bt_hid_session structure. The session has to be disconnected by calling bt_hid_disconnect or bt_hid_unplug first.

Parameters

session	The HID session to be deallocated.

3.9.2.6 bt_bool bt_hid_init (void)

Initialize HID layer.

This function initializes the HID layer of the stack. It must be called prior to any other HID function can be called.

3.9.2.7 bt_bool bt_hid_listen (bt_hid_session_p *session,* bt_hid_state_callback_fp *callback,* bt_hid_get_report_callback_fp *get_report_callback,* bt_hid_set_report_callback_fp *set_report_callback*)

Listen for incoming connections.

This function enables incoming connections on the specified HID session. Changes in the session state are reported through a callback function.

Parameters

session	HID session.
callback	The callback function that is called when session state changes.
get_report_↩ callback	The callback function that is called when a remote device requests an INPUT report.
set_report_↩ callback	The callback function that is called when a remote device sends and OUTPUT report.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.9.2.8 bt_bool bt_hid_send_report (bt_hid_session_p *session,* bt_byte *channel,* bt_byte *report_type,* bt_byte *report_id,* bt_bool *send_report_id,* bt_byte_p *data,* bt_int *len,* bt_byte *tran_type,* bt_hid_send_report_callback_fp *callback*)

Send report.

This function sends a report to a remote device

Parameters

session	The HID session.
channel	The type of the channel (CONTROL or INTERRUPT) used to send the report.
report_type	The type of the report (INPUT, OUTPUR, or FEATURE).
report_id	The id of the report.
send_report_id	The flag that specifies weather the report id is included in the data packet.
data	The pointer to the report data.
len	The length of the report data.
tran_type	The type of the transaction (see definition of the HID_TRANTYPE constants).
callback	The callback function that is called when sending the report has been completed.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.9.2.9 void bt_hid_unplug (bt_hid_session_p *session*)

Unplug device.

This function unplugs a virtually-cabled device Changes in the session state are reported through a callback function set when connection was created by calling bt_hid_connect.

Parameters

session	HID session.

3.10 Hands Free Profile (HFP)

Data Structures

- struct bt_hfp_evt_audio_data_t

  HFP_EVENT_AUDIO_DATA_RECEIVED event parameter.
- struct bt_hfp_evt_slc_connection_state_changed_t

  HFP_EVENT_SLC_CONNECTION_STATE_CHANGED event parameter.
- struct bt_hfp_evt_audio_connection_state_changed_t

  HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event parameter.
- struct bt_hfp_evt_mic_volume_changed_t

  HFP_EVENT_MIC_VOLUME_CHANGED event parameter.
- struct bt_hfp_evt_spk_volume_changed_t

  HFP_EVENT_SPK_VOLUME_CHANGED event parameter.
- struct bt_hfp_evt_indicator_received_t

  HFP_EVENT_INDICATOR_RECEIVED event parameter.
- struct bt_hfp_evt_query_operator_completed_t

  HFP_EVENT_QUERY_OPERATOR_COMPLETED event parameter.
- struct bt_hfp_evt_clip_received_t

  HFP_EVENT_CLIP_RECEIVED event parameter.
- struct bt_hfp_evt_call_waiting_t

  HFP_EVENT_CALL_WAITING event parameter.
- struct bt_hfp_evt_command_completed_t

  HFP_EVENT_CMD_COMPLETED event parameter.
- struct bt_hfp_evt_voice_recognition_changed_t

  HFP_EVENT_VOICE_RECOGNITION_CHANGED event parameter.
- struct bt_hfp_evt_inband_ring_changed_t

  HFP_EVENT_INBAND_RING_CHANGED event parameter.
- struct bt_hfp_evt_dial_request_received_t

  HFP_AG_EVENT_DIAL_REQUEST_RECEIVED event parameter.
- struct bt_hfp_ind

  Stores value of an indicator.
- struct bt_hfp_event_register_t

  Stores value of HF registrations.
- struct bt_hfp_call_t

  Stores information about a call.
- struct bt_hfp_audio_packet_t

  Parameter to HFP_EVENT_AUDIO_DATA_RECEIVED event.

Macros

- #define bt_hfp_allocate_session_hf() bt_hfp_allocate_session(HFP_ROLE_HF)

  Allocate HFP session.
- #define bt_hfp_allocate_session_ag() bt_hfp_allocate_session(HFP_ROLE_AG)

  Allocate HFP session.
- #define bt_hfp_hf_terminate(session) bt_hfp_hf_reject(session)

  Terminate the active call.
- #define bt_hfp_has_active_call(session) (session->call_status_mask & HFP_CALL_STATUS_MASK_AC↩ TIVE)

  Determines if there is one or more active calls in the AG.
- #define bt_hfp_has_held_call(session) (session->call_status_mask & HFP_CALL_STATUS_MASK_HELD)

Determines if there is one or more held calls in the AG.

- #define bt_hfp_has_dialing_call(session) (session->call_status_mask & HFP_CALL_STATUS_MASK_DI↩ ALING)

  Determines if there is one or more dialing calls in the AG.
- #define bt_hfp_has_alerting_call(session) (session->call_status_mask & HFP_CALL_STATUS_MASK_A↩ LERTING)

  Determines if there is one or more alerting calls in the AG.
- #define bt_hfp_has_outgoing_call(session) (session->call_status_mask & (HFP_CALL_STATUS_MASK_↩ ALERTING | HFP_CALL_STATUS_MASK_DIALING))

  Determines if there is one or more outgoing (dialing or alerting) calls in the AG.
- #define bt_hfp_has_incoming_call(session) (session->call_status_mask & HFP_CALL_STATUS_MASK_↩ INCOMING)

  Determines if there is one or more incoming calls in the AG.
- #define bt_hfp_has_waiting_call(session) (session->call_status_mask & HFP_CALL_STATUS_MASK_W↩ AITING)

  Determines if there is one or more waiting calls in the AG.
- #define bt_hfp_get_remote_address(session) (&session->rfcomm_dlc->psess->pch->hci_conn->bdaddr↩ _remote)

  Get the address of the remote device this device is connected.
- #define bt_hfp_ag_set_mic_gain(session, gain) bt_hfp_set_mic_gain(session, gain)

  Set HF microphone gain.
- #define bt_hfp_ag_set_speaker_volume(session, volume) bt_hfp_set_speaker_volume(session, volume)

  Set HF speaker volume.

Typedefs

- typedef void(∗ bt_hfp_session_callback_pf) (bt_hfp_session ∗session, bt_byte evt, void ∗evt_param, void ∗callback_param)

  Notify the application of changes in the HFP session.
- typedef void(∗ bt_hfp_find_remote_callback_pf) (bt_hfp_session ∗session, bt_byte server_channel, bt_bool found)

  Notify the application of the result of searching for the Audio Gateway.
- typedef void(∗ bt_hfp_send_audio_callback_pf) (bt_hfp_session ∗session, void ∗callback_param)

  Notify the application that audio data has been sent to a remote device.

Functions

- bt_bool bt_hfp_init (void)

  Initialize HFP layer.
- bt_hfp_session ∗ bt_hfp_allocate_session (bt_byte role)

  Allocate HFP session.
- void bt_hfp_free_session (bt_hfp_session ∗psess)

  Release HFP session.
- bt_bool bt_hfp_listen (bt_hfp_session ∗session, bt_byte server_channel, bt_hfp_session_callback_pf callback, void ∗callback_param)

  Listen for incoming connections.
- bt_bool bt_hfp_connect (bt_hfp_session ∗session, bt_bdaddr_p pbdaddr_remote, bt_hfp_session_callback↩ _pf callback, void ∗callback_param)

  Connect to a remote device.
- bt_bool bt_hfp_hf_disconnect (bt_hfp_session ∗session)

  Close connection.
- bt_bool bt_hfp_hf_query_operator (bt_hfp_session ∗session)

Request operator's name from AG.
- bt_bool bt_hfp_hf_answer (bt_hfp_session ∗session)

  Answer the incoming call.
- bt_bool bt_hfp_hf_hold (bt_hfp_session ∗session, bt_byte cmd, bt_byte call_index)

  Enhanced call control.
- bt_bool bt_hfp_hf_reject (bt_hfp_session ∗session)

  Reject the incoming call.
- bt_bool bt_hfp_hf_dial_number (bt_hfp_session ∗session, bt_char ∗number)

  Place a call.
- bt_bool bt_hfp_hf_redial (bt_hfp_session ∗session)

  Redial last dialed number.
- bt_bool bt_hfp_hf_enable_call_waiting_notification (bt_hfp_session ∗session, bt_bool enable)

  Enable/disable call waiting notification.
- bt_bool bt_hfp_set_mic_gain (bt_hfp_session ∗session, bt_byte gain)

  Report microphone gain on the HF to the AG.
- bt_bool bt_hfp_set_speaker_volume (bt_hfp_session ∗session, bt_byte volume)

  Report speaker volume on the HF to the AG.
- bt_bool bt_hfp_hf_get_subscriber_number (bt_hfp_session ∗session)

  Request subscriber number information from the AG.
- bt_byte bt_hfp_get_indicator_value (bt_hfp_session ∗session, bt_byte indicator_id)

  Get current value of an indicator.
- bt_bool bt_hfp_has_call_with_status (bt_hfp_session ∗session, bt_byte status)

  Determines if there is one or more calls in the AG with the specified status.
- bt_bool bt_hfp_hf_refresh_call_list (bt_hfp_session ∗session)

  Request a call list from the AG.
- bt_bool bt_hfp_hf_enable_calling_line_identification (bt_hfp_session ∗session, bt_bool enable)

  Enable/disable calling line identification notification.
- bt_bool bt_hfp_hf_find_ag (bt_hfp_session ∗session, bt_bdaddr_t ∗deviceAddress, bt_hfp_find_remote_↩ callback_pf callback)

  Find AG.
- bt_bool bt_hfp_hf_enable_voice_recognition (bt_hfp_session ∗session, bt_bool enable)

  Enable/disable calling voice control in the AG.
- bt_bool bt_hfp_hf_disable_nrec (bt_hfp_session ∗session)

  Disable Echo Canceling and Noise Reduction functions in the AG.
- bt_bool bt_hfp_connect_audio (bt_hfp_session ∗session)

  Transfer audio connection from the AG to the HF.
- bt_bool bt_hfp_disconnect_audio (bt_hfp_session ∗session)

  Transfer audio connection from the HF to the AG.
- bt_hfp_call_t ∗ bt_hfp_ag_incoming_call (bt_hfp_session ∗session, const bt_char ∗number, bt_int type)

  HFP AG incoming call.
- bt_hfp_call_t ∗ bt_hfp_ag_outgoing_call (bt_hfp_session ∗session, const bt_char ∗number)

  HFP AG outgoing call.
- void bt_hfp_ag_reject_outgoing_call (bt_hfp_session ∗session)

  HFP AG reject outgoing call.
- void bt_hfp_ag_outgoing_call_alerting (bt_hfp_session ∗session, bt_hfp_call_t ∗call)

  HFP AG outgoing call alerting.
- void bt_hfp_ag_call_connected (bt_hfp_session ∗session, bt_hfp_call_t ∗call)

  HFP AG call connected.
- void bt_hfp_ag_call_disconnected (bt_hfp_session ∗session, bt_hfp_call_t ∗call)

  HFP AG call disconnected.
- bt_bool bt_hfp_ag_set_service_state (bt_hfp_session ∗session, bt_byte ind_val)

Set Service State indicator value.

- bt_bool bt_hfp_ag_set_signal_strength (bt_hfp_session ∗session, bt_byte ind_val)

  Set Signal Strength indicator value.
- bt_bool bt_hfp_ag_set_roam_state (bt_hfp_session ∗session, bt_byte ind_val)

  Set Roaming State indicator value.
- bt_bool bt_hfp_ag_set_battery_level (bt_hfp_session ∗session, bt_byte ind_val)

  Set Battery Level indicator value.
- bt_bool bt_hfp_ag_set_inband_ring (bt_hfp_session ∗session, bt_byte ind_val)

  Send Inband ring notification to HF device.
- bt_bool bt_hfp_activate_voice_recognition (bt_hfp_session ∗session, bt_bool activate)

  Send voice recognition activation notification to HF device.
- bt_bool bt_hfp_ag_set_operator_name (bt_hfp_session ∗session, const bt_byte ∗name)

  Set Operator name.
- bt_bool bt_hfp_ag_set_subscriber_number (bt_hfp_session ∗session, const bt_byte ∗number, bt_byte type, bt_byte service)

  Set Subscriber number.

3.10.1 Detailed Description

3.10.2 Macro Definition Documentation

3.10.2.1 #define bt_hfp_ag_set_mic_gain(*session,* *gain*) **bt_hfp_set_mic_gain**(session, gain)

Set HF microphone gain.

This function sets microphone gain on the HF. The AG can call this function to set the HF's microphone gain.

Parameters

session	HFP session.
gain	Microphone gain (0 - 15)

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.2.2 #define bt_hfp_ag_set_speaker_volume(*session,* *volume*) **bt_hfp_set_speaker_volume**(session, volume)

Set HF speaker volume.

This function sets speaker volume on the HF. The AG can call this function to set HF's speaker volume.

Parameters

session	HFP session.
gain	Speaker volume (0 - 15)

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.2.3 #define bt_hfp_allocate_session_ag( ) **bt_hfp_allocate_session**(HFP_ROLE_AG)

Allocate HFP session.

This function allocates a new AG HFP session. Currently AG role is not implemented.

Returns

- A pointer to the new HFP session structure if the function succeeds.
- `NULL` otherwise.

3.10.2.4 #define bt_hfp_allocate_session_hf( ) **bt_hfp_allocate_session**(HFP_ROLE_HF)

Allocate HFP session.

This function allocates a new HF HFP session.

Returns

- A pointer to the new HFP session structure if the function succeeds.
- `NULL` otherwise.

3.10.2.5 #define bt_hfp_get_remote_address(*session*) (&session->rfcomm_dlc->psess->pch->hci_conn->bdaddr_remote)

Get the address of the remote device this device is connected.

Parameters

session	HFP session.

Returns

- `A` pointer to bt_bdaddr structure that contains the address of the remote device.

3.10.2.6 #define bt_hfp_has_active_call(*session*) (session->call_status_mask & HFP_CALL_STATUS_MASK_ACTIVE)

Determines if there is one or more active calls in the AG.

This is a define that calls bt_hfp_has_call_with_status with HFP_CALL_STATUS_MASK_ACTIVE as the value for the status parameter.

Parameters

session	HFP session.

Returns

- `TRUE` if there are active calls.
- `FALSE` otherwise.

3.10.2.7 #define bt_hfp_has_alerting_call(*session*) (session->call_status_mask & HFP_CALL_STATUS_MASK_ALERTING)

Determines if there is one or more alerting calls in the AG.

This is a define that calls bt_hfp_has_call_with_status with HFP_CALL_STATUS_MASK_ALERTING as the value for the status parameter.

Parameters

session	HFP session.

Returns

- `TRUE` if there are alerting calls.
- `FALSE` otherwise.

3.10.2.8 #define bt_hfp_has_dialing_call(*session*) (session->call_status_mask & HFP_CALL_STATUS_MASK_DIALING)

Determines if there is one or more dialing calls in the AG.

This is a define that calls bt_hfp_has_call_with_status with HFP_CALL_STATUS_MASK_DIALING as the value for the status parameter.

Parameters

session	HFP session.

Returns

- `TRUE` if there are dialing calls.
- `FALSE` otherwise.

3.10.2.9 #define bt_hfp_has_held_call(*session*) (session->call_status_mask & HFP_CALL_STATUS_MASK_HELD)

Determines if there is one or more held calls in the AG.

This is a define that calls bt_hfp_has_call_with_status with HFP_CALL_STATUS_MASK_HELD as the value for the status parameter.

Parameters

session	HFP session.

Returns

- `TRUE` if there are held calls.
- `FALSE` otherwise.

3.10.2.10 #define bt_hfp_has_incoming_call(*session*) (session->call_status_mask & HFP_CALL_STATUS_MASK_INCOMING)

Determines if there is one or more incoming calls in the AG.

This is a define that calls bt_hfp_has_call_with_status with HFP_CALL_STATUS_MASK_INCOMING as the value for the status parameter.

Parameters

session	HFP session.

Returns

- `TRUE` if there are incoming calls.
- `FALSE` otherwise.

3.10.2.11 #define bt_hfp_has_outgoing_call(*session*) (session->call_status_mask & (HFP_CALL_STATUS_MASK_ALERTI↩ NG | HFP_CALL_STATUS_MASK_DIALING))

Determines if there is one or more outgoing (dialing or alerting) calls in the AG.

This is a define that calls bt_hfp_has_call_with_status with HFP_CALL_STATUS_MASK_ALERTING | HFP_CA↩ LL_STATUS_MASK_DIALING as the value for the status parameter.

Parameters

session	HFP session.

Returns

- `TRUE` if there are outgoing calls.
- `FALSE` otherwise.

3.10.2.12 #define bt_hfp_has_waiting_call(*session*) (session->call_status_mask & HFP_CALL_STATUS_MASK_WAITING)

Determines if there is one or more waiting calls in the AG.

This is a define that calls bt_hfp_has_call_with_status with HFP_CALL_STATUS_MASK_WAITING as the value for the status parameter.

Parameters

session	HFP session.

Returns

- `TRUE` if there are waiting calls.
- `FALSE` otherwise.

3.10.2.13 #define bt_hfp_hf_terminate(*session*) bt_hfp_hf_reject(session)

Terminate the active call.

If there is an active call in the AG the app may call this function to terminate it. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event.

Parameters

session	HFP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.3 Typedef Documentation

3.10.3.1 typedef void(∗ bt_hfp_find_remote_callback_pf) (bt_hfp_session ∗session, bt_byte server_channel, bt_bool found)

Notify the application of the result of searching for the Audio Gateway.

This function is called by the HFP layer when searching for the Audio Gateway on a remote device has completed.

Parameters

session	HFP session.
server_channel	The RFCOMM server channel used by the Audio Gateway. The application must pass this value to bt_hfp_connect. The value is valid only if the `found` parameter is TRUE.
found	`TRUE` if Audio Gateway has been found on the remote device. `FALSE` otherwise.

3.10.3.2 typedef void(∗ bt_hfp_send_audio_callback_pf) (bt_hfp_session ∗session, void ∗callback_param)

Notify the application that audio data has been sent to a remote device.

This function is called by the HFP layer when audio data has been sent to a remote device.

Parameters

session	HFP session.
callback_param	The value of the `callback_param` parameter passed to the call of bt_hfp_send_audio.

3.10.3.3 typedef void(∗ bt_hfp_session_callback_pf) (bt_hfp_session ∗session, bt_byte evt, void ∗evt_param, void ∗callback_param)

Notify the application of changes in the HFP session.

This function is called by the HFP layer when various session parameters have changed. It is also called when commands from the remote device or response codes to the commands sent to the the remote device are received.

Parameters

session	HFP session.
evt	Specifies the id of the event. The value of this arguments is one of the values defined by H←FP_EVENT_ constants. Each event has a parameter that points to an event specific structure that clarifies the event. The event parameter is passed through the `evt_param` parameter.

- HSP_SESSION_RFCOMM_CONNECTION_STATE_CHANGED - The RFCOMM connection state has changed. The application should examine the value of the bt_hsp_session::state member to determine weather the RFCOMM session has connected or disconnected. The RFCOMM session has connected if the HSP_SESSION_STATE_RFCOMM_CONNECTED bit is set in the bt_hsp_session::state member.
- HSP_SESSION_SCO_CONNECTION_STATE_CHANGED - The SCO connection state has changed. The application should examine the value of the bt_hsp_session::state member to determine weather the RFC←OMM session has connected or disconnected. The RFCOMM session has connected if the HSP_SESSIO←N_STATE_SCO_CONNECTED bit is set in the bt_hsp_session::state member.
- HSP_SESSION_CMD_RESPONSE_RECEIVED - The Headset has received a response to the command it sent to the Audio Gateway. The response is a string pointed to by the `what_param` parameter. It can be either HS_RES_OK or HS_RES_ERROR.
- HSP_SESSION_RING - The Audio gateway has sent the RING unsolicited result code. The application must respond to this result code by sending the button press (call bt_hsp_hs_send_button_press).
- HSP_SESSION_AUDIO_DATA_RECEIVED - The audio data has been received. The pointer to a buffer is passed in the `what_param` parameter. The length of the data is passed in the `what_param2` parameter (cast to bt_int).
- HSP_SESSION_HEADSET_CONNECT_ATTEMPT - The attempt to connect to the Audio Gateway has failed.
- HSP_SESSION_MIC_VOLUME_CHANGED - The Audio Gateway has changed the microphone gain. The new value is passed in the `what_param` parameter (cast to bt_byte).
- HSP_SESSION_SPK_VOLUME_CHANGED - The Audio Gateway has changed the speaker volume. The new value is passed in the `what_param` parameter (cast to bt_byte)

Parameters

evt_param	The event's parameter.
callback_param	The value of the callback_param passed to the call of bt_hfp_listen or bt_hfp_connect.

3.10.4 Function Documentation

3.10.4.1 bt_bool bt_hfp_activate_voice_recognition (bt_hfp_session ∗ *session,* bt_bool *activate*)

Send voice recognition activation notification to HF device.

This function should be called by the application to indicate to the HF that the voice recognition in the AG has been activated or deactivated.

This function should be called when Service Level Connection has been established.

Parameters

session	HFP session.
activate	FALSE - voice recognition has been activated in the AG. TRUE - voice recognition has been deactivated in the AG.

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range, or RFCOMM connection is not established.

3.10.4.2 void bt_hfp_ag_call_connected (bt_hfp_session ∗ *session,* bt_hfp_call_t ∗ *call*)

HFP AG call connected.

This function should be called when incoming call has been answered locally, or outgoing call has been answered by remote party. The call is in the connected state. As a result HFP AG sends notification to connected HF device.

Parameters

session	HFP session.
call	HFP call.

3.10.4.3 void bt_hfp_ag_call_disconnected (bt_hfp_session ∗ *session,* bt_hfp_call_t ∗ *call*)

HFP AG call disconnected.

This function should be called when a call has been terminated. It can be local or remote termination of the outgoing or connected call, as well as rejection of the incoming call. As a result HFP AG sends notification to connected HF device.

Parameters

session	HFP session.
call	HFP call.

3.10.4.4 bt_hfp_call_t∗ bt_hfp_ag_incoming_call (bt_hfp_session ∗ *session,* const bt_char ∗ *number,* bt_int *type*)

HFP AG incoming call.

This function should be called when AG receives notification from the cellular baseband about an incoming call. As a result HFP AG saves information about the call and send notification to all connected HF devices.

Parameters

session	HFP session.
number	string type phone number of format specified by `type`
type	• `128` - both the type of number and the numbering plan are unknown • `129` - unknown type of number and ISDN/Telephony numbering plan • `145` - international type of number and ISDN/Telephony numbering plan (contains the character "+")

Returns

- A pointer to the new HFP call structure if the function succeeds.
- `NULL` otherwise.

3.10.4.5 bt_hfp_call_t∗ bt_hfp_ag_outgoing_call (bt_hfp_session ∗ *session,* const bt_char ∗ *number*)

HFP AG outgoing call.

This function should be called when AG receives notification from the cellular baseband about an outgoing call. As a result HFP AG saves information about the call and send notification to all connected HF devices.

Parameters

session	HFP session.
number	string type phone number of format specified by `type`

Returns

- A pointer to the new HFP call structure if the function succeeds.
- `NULL` otherwise.

3.10.4.6 void bt_hfp_ag_outgoing_call_alerting (bt_hfp_session ∗ *session,* bt_hfp_call_t ∗ *call*)

HFP AG outgoing call alerting.

This function should be called when AG receives notification that the outgoing call is alerting remote party. As a result HFP AG sends notification to all connected HF devices.

Parameters

session	HFP session.
call	HFP call.

3.10.4.7 void bt_hfp_ag_reject_outgoing_call (bt_hfp_session ∗ *session*)

HFP AG reject outgoing call.

This function should be called by AG to abort a request from HF to dial a number stored in memory or re-dial last number dialed.

Parameters

session	HFP session.

3.10.4.8 bt_bool bt_hfp_ag_set_battery_level (bt_hfp_session ∗ *session,* bt_byte *ind_val*)

Set Battery Level indicator value.

This function should be called by the application during session initialization and when battery level changes.

Parameters

session	HFP session.
ind_val	Current battery level. Valid values are between 0 for low battery and 5 when battery is full.

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range.

3.10.4.9 bt_bool bt_hfp_ag_set_inband_ring (bt_hfp_session ∗ *session,* bt_byte *ind_val*)

Send Inband ring notification to HF device.

This function should be called by the application to indicate to the HF that the in-band ring tone setting has been locally changed.

This function should be called when Service Level Connection has been established.

Parameters

session	HFP session.
ind_val	Indicator value. 0 - the AG provides no in-band ring tone. 1 - the AG provides an in-band ring tone.

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range, or RFCOMM connection is not established.

3.10.4.10 bt_bool bt_hfp_ag_set_operator_name (bt_hfp_session ∗ *session,* const bt_byte ∗ *name*)

Set Operator name.

This function should be called by the application during session initialization.

Parameters

session	HFP session.
name	Operator's name.

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range.

3.10.4.11 bt_bool bt_hfp_ag_set_roam_state (bt_hfp_session * *session,* bt_byte *ind_val*)

Set Roaming State indicator value.

This function should be called by the application during session initialization and every time roaming state changes.

Parameters

session	HFP session.
ind_val	Current roaming state. Valid values are 1 when the phone is roaming, 0 when phone is in home network.

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range.

3.10.4.12 bt_bool bt_hfp_ag_set_service_state (bt_hfp_session ∗ *session,* bt_byte *ind_val*)

Set Service State indicator value.

This function should be called by the application during session initialization and every time service state goes up or down.

Parameters

session	HFP session.
ind_val	Current service state. Valid values are 1 for service present, 0 otherwise.

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range.

3.10.4.13 bt_bool bt_hfp_ag_set_signal_strength (bt_hfp_session ∗ *session,* bt_byte *ind_val*)

Set Signal Strength indicator value.

This function should be called by the application during session initialization and every time signal strength value changes.

Parameters

session	HFP session.
ind_val	Current signal strength. Valid values are between 0 for low signal and 5 for maximum signal.

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range.

3.10.4.14 bt_bool bt_hfp_ag_set_subscriber_number (bt_hfp_session ∗ *session,* const bt_byte ∗ *number,* bt_byte *type,* bt_byte *service*)

Set Subscriber number.

This function should be called by the application during session initialization.

Parameters

session	HFP session.
number	string type phone number of format specified by `type`
type	• `128` - both the type of number and the numbering plan are unknown • `129` - unknown type of number and ISDN/Telephony numbering plan • `145` - international type of number and ISDN/Telephony numbering plan (contains the character "+")
service	Indicates which service this phone number relates to. Shall be either 4 (voice) or 5 (fax).

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range.

3.10.4.15 bt_hfp_session∗ bt_hfp_allocate_session (bt_byte *role*)

Allocate HFP session.

This function allocates a new HFP session.

Parameters

role	HFP_ROLE_HF or HFP_ROLE_AG.

Returns

- A pointer to the new HFP session structure if the function succeeds.
- `NULL` otherwise.

3.10.4.16 bt_bool bt_hfp_connect (bt_hfp_session ∗ *session,* bt_bdaddr_p *pbdaddr_remote,* bt_hfp_session_callback_pf *callback,* void ∗ *callback_param*)

Connect to a remote device.

This function establishes a HFP connection with a remote device which is running in Audio Gateway mode. Changes in the session state are reported through a callback function.

Parameters

session	HFP session.
remote_addr	Address of the remote device.
callback	A callback function that is called when session state changes.
callback_param	An arbitrary data pointer that will be passed to the callback function specified by the `callback` parameter.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.10.4.17 bt_bool bt_hfp_connect_audio (bt_hfp_session ∗ *session*)

Transfer audio connection from the AG to the HF.

This function establishes a SCO connection to the AG. As a result the AG will route its audio path to the HF. The result of calling this function will be HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event.

Parameters

session	HFP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. the HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event is not reported in this case.

3.10.4.18 bt_bool bt_hfp_disconnect_audio (bt_hfp_session ∗ *session*)

Transfer audio connection from the HF to the AG.

This function terminates the SCO connection to the AG. As a result the AG will route its audio path to itself. The result of calling this function will be HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event.

Parameters

session	HFP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. the HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event is not reported in this case.

3.10.4.19 void bt_hfp_free_session (bt_hfp_session ∗ *psess*)

Release HFP session.

This function deallocated the specified HFP session. This function does not disconnect the session. It just frees the memory used by the bt_hfp_session structure. The session has to be disconnected by a remote device or by calling bt_hfp_hf_disconnect.

Parameters

session	The HFP session to be deallocated.

3.10.4.20 bt_byte bt_hfp_get_indicator_value (bt_hfp_session ∗ *session,* bt_byte *indicator_id*)

Get current value of an indicator.

This function returns a value stored on the local device.

Parameters

session	HFP session.
indicator_id	the ID if the indicator

Returns

- `Current` indicator's value.

3.10.4.21 bt_bool bt_hfp_has_call_with_status (bt_hfp_session ∗ *session,* bt_byte *status*)

Determines if there is one or more calls in the AG with the specified status.

This function does not send any commands to the AG. It uses a list of calls stored on the local device.

Parameters

session	HFP session.
status	a bit mask of HFP_CALL_STATUS_ constants that defines statuses of calls to be checked.

Returns

- `TRUE` if there are calls with specified statuses.
- `FALSE` otherwise.

3.10.4.22 bt_bool bt_hfp_hf_answer (bt_hfp_session ∗ *session*)

Answer the incoming call.

If there is an incoming call in the AG the app may call this function to accept it. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event followed by HFP_EVENT_CALL_STARTED event.

Parameters

session	HSP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.23 bt_bool bt_hfp_hf_dial_number (bt_hfp_session ∗ *session,* bt_char ∗ *number*)

Place a call.

Intiate an outgoing call. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event.

Parameters

session	HFP session.
number	Phone number to dial.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.24 bt_bool bt_hfp_hf_disable_nrec (bt_hfp_session ∗ *session*)

Disable Echo Canceling and Noise Reduction functions in the AG.

This function should be called by the application to disable any Echo Canceling and Noise Reduction functions embedded in the AG. The result of calling this function will be HFP_EVENT_NREC_DISABLED event.

This function should be called when Service Level Connection has been established and before any Audio Connection between the HF and the AG is established.

Parameters

session	HFP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` if parameter if out of range, or RFCOMM connection is not established.

3.10.4.25 bt_bool bt_hfp_hf_disconnect (bt_hfp_session ∗ *session*)

Close connection.

This function closes a HFP connection with a remote device. Changes in the session state are reported through a callback function set when connection was created by calling bt_hfp_connect or bt_hfp_listen.

Parameters

session	HSP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.26 bt_bool bt_hfp_hf_enable_call_waiting_notification (bt_hfp_session ∗ *session,* bt_bool *enable*)

Enable/disable call waiting notification.

Request AG to enable or disable call waiting notification. If call waiting notification is enabled, the AG will notify the HF whenever an incoming call is waiting during an ongoing call. The result of calling this function will be HFP_EV←
ENT_CMD_COMPLETED event. There is usually no need to call this function directly as the call waiting notification is enabled during SLC setup.

Parameters

session	HFP session.
enable	TRUE - to enable call waiting notification, FALSE - to disable it.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.27 bt_bool bt_hfp_hf_enable_calling_line_identification (bt_hfp_session ∗ *session,* bt_bool *enable*)

Enable/disable calling line identification notification.

Request AG to enable or disable calling line identification notification. If calling line identification notification is enabled and the calling number is available from the network, the AG will notify the HF whenever there is an incoming call. The application will receive HFP_EVENT_CLIP_RECEIVED event. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event. There is usually no need to call this function directly as the calling line identification notification is enabled during SLC setup.

Parameters

session	HFP session.
enable	TRUE - to enable calling line identification notification, FALSE - to disable it.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.28 bt_bool bt_hfp_hf_enable_voice_recognition (bt_hfp_session ∗ *session*, bt_bool *enable*)

Enable/disable calling voice control in the AG.

Request AG to enable or disable voice control. The result of calling this function will be HFP_EVENT_CMD_CO←MPLETED event.

Parameters

session	HFP session.
enable	TRUE - to enable voice control, FALSE - to disable it.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.29 bt_bool bt_hfp_hf_find_ag (bt_hfp_session ∗ *session*, bt_bdaddr_t ∗ *deviceAddress*, bt_hfp_find_remote_callback_pf *callback*)

Find AG.

This function connects to a remote device and tries to find the AG on it. If the remote device implements HFP AG this function returns (via callback) the RFCOMM channel which should be used to connect to it.

Parameters

session	HFP session.
deviceAddress	Remote device's address.
callback	A pointer to a function which is called when AG search is complete.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. the callback is not called in this case.

3.10.4.30 bt_bool bt_hfp_hf_get_subscriber_number (bt_hfp_session ∗ *session*)

Request subscriber number information from the AG.

This function sends a command to the AG which causes it to send back subscriber number information. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event followed by HFP_EVENT_SUBSCRIBER←_NUMBER_RECEIVED event. The caller can retrieve the information from the bt_hfp_session::subscriber_number.

Parameters

session	HFP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.10.4.31 bt_bool bt_hfp_hf_hold (bt_hfp_session ∗ *session,* bt_byte *cmd,* bt_byte *call_index*)

Enhanced call control.

If there is an incoming call in the AG the app may call this function to reject it. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event.

Parameters

session	HFP session.
cmd	Command Id.
call	index Call index.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.32 bt_bool bt_hfp_hf_query_operator (bt_hfp_session ∗ *session*)

Request operator's name from AG.

This function requests operator's name from the AG. The result of this call is reported to the app in the form of the HFP_EVENT_QUERY_OPERATOR_COMPLETED event.
There is usually no need to call this function directly as the operator's name is requested and reported to the app during SLC setup.

Parameters

session	HSP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.33 bt_bool bt_hfp_hf_redial (bt_hfp_session ∗ *session*)

Redial last dialed number.

Request AG to dial last dialed number. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event.

Parameters

session	HFP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.34 bt_bool bt_hfp_hf_refresh_call_list (bt_hfp_session ∗ *session*)

Request a call list from the AG.

This function send a command to the AG which causes it to send back a list of all calls it currently has. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event followed by HFP_EVENT_CALL_LIST_R←ECEIVED event.

Parameters

session	HFP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.35 bt_bool bt_hfp_hf_reject (bt_hfp_session ∗ *session*)

Reject the incoming call.

If there is an incoming call in the AG the app may call this function to reject it. The result of calling this function will be HFP_EVENT_CMD_COMPLETED event.

Parameters

session	HFP session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.36 bt_bool bt_hfp_init (void)

Initialize HFP layer.

This function initializes the HFP layer of the stack. It must be called prior to any other HFP function can be called.

3.10.4.37 bt_bool bt_hfp_listen (bt_hfp_session ∗ *session,* bt_byte *server_channel,* bt_hfp_session_callback_pf *callback,* void ∗ *callback_param*)

Listen for incoming connections.

This function enables incoming connections on the specified HFP session. Changes in the session state are reported through a callback function.

Parameters

session	HFP session.
server_channel	An RFCOMM server channel on which the RFCOMM session specified in the call to bt_hfp←_allocate_session will be listening.
callback	A callback function that is called when session state changes or data (command or response) is received from the remote party.
callback_param	An arbitrary data pointer that will be passed to the callback function specified by the `callback` parameter.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The callback function is not called in this case.

3.10.4.38 bt_bool bt_hfp_set_mic_gain (bt_hfp_session ∗ *session,* bt_byte *gain*)

Report microphone gain on the HF to the AG.

This function reports microphone gain on the HF to the AF. This function is called by the HF upon SLC setup to notify the AG about the current level of the microphone gain on the HF. In case physical mechanisms (buttons, dials etc.) means are implemented on the HF to control the volume levels, the HF calls this function to inform the AG of any changes in the microphone gain.

Parameters

session	HFP session.
gain	Microphone gain (0 - 15)

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.10.4.39 bt_bool bt_hfp_set_speaker_volume (bt_hfp_session ∗ *session,* bt_byte *volume*)

Report speaker volume on the HF to the AG.

This function reports speaker volume on the HF to the AG. This function is called by the HF upon SLC setup to notify the AG about the current level of the speaker volume on the HF. In case physical mechanisms (buttons, dials etc.) means are implemented on the HS to control the volume levels, the HF calls this function to inform the AG of any changes in the speaker volume.

Parameters

session	HFP session.
gain	Speaker volume (0 - 15)

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.11 Audio/Video Distribution Protocol (AVDP)

AVDTP is the transport protocol for audio and/or video distribution connections and streaming of audio or video media over the Bluetooth air interface.

Modules

- Configuration

  This module describes parameters used to configure AVDTP layer.

Data Structures

- struct bt_media_packet_t

  Media packet buffer.
- struct bt_avdtp_sep_capabilities_t

  SEP capabilities.
- struct bt_avdtp_sep_t

  SEP description.
- struct bt_avdtp_evt_ctrl_channel_connected_t

  Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event.
- struct bt_avdtp_evt_ctrl_channel_disconnected_t

  Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event.
- struct bt_avdtp_evt_discover_completed_t

  Parameter to AVDTP_EVT_DISCOVER_COMPLETED event.
- struct bt_avdtp_evt_sep_info_received_t

  Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event.
- struct bt_avdtp_evt_get_sep_capabilities_completed_t

  Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event.
- struct bt_avdtp_evt_sep_capabilities_received_t

  Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_R↩ ECEIVED events.
- struct bt_avdtp_evt_set_stream_configuration_completed_t

  Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event.
- struct bt_avdtp_evt_get_stream_configuration_completed_t

  Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event.
- struct bt_avdtp_evt_stream_reconfigure_completed_t

  Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event.
- struct bt_avdtp_evt_open_stream_completed_t

  Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event.
- struct bt_avdtp_evt_start_stream_completed_t

  Parameter to AVDTP_EVT_START_STREAM_COMPLETED event.
- struct bt_avdtp_evt_close_stream_completed_t

  Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event.
- struct bt_avdtp_evt_suspend_stream_completed_t

  Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event.
- struct bt_avdtp_evt_stream_security_control_completed_t

  Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event.
- struct bt_avdtp_evt_set_stream_configuration_requested_t

  Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event.
- struct bt_avdtp_evt_reconfigure_stream_requested_t

Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event.
- struct bt_avdtp_evt_open_stream_requested_t

  Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event.
- struct bt_avdtp_evt_start_stream_requested_t

  Parameter to AVDTP_EVT_START_STREAM_REQUESTED event.
- struct bt_avdtp_evt_suspend_stream_requested_t

  Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event.
- struct bt_avdtp_evt_close_stream_requested_t

  Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event.
- struct bt_avdtp_evt_abort_stream_requested_t

  Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event.
- struct bt_avdtp_evt_delay_report_completed_t

  Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event.
- struct bt_avdtp_evt_stream_configured_t

  Parameter to AVDTP_EVT_STREAM_CONFIGURED event.
- struct bt_avdtp_evt_stream_reconfigured_t

  Parameter to AVDTP_EVT_STREAM_RECONFIGURED event.
- struct bt_avdtp_evt_stream_opened_t

  Parameter to AVDTP_EVT_STREAM_OPENED event.
- struct bt_avdtp_evt_stream_started_t

  Parameter to AVDTP_EVT_STREAM_STARTED event.
- struct bt_avdtp_evt_stream_suspended_t

  Parameter to AVDTP_EVT_STREAM_SUSPENDED event.
- struct bt_avdtp_evt_stream_closed_t

  Parameter to AVDTP_EVT_STREAM_CLOSED event.
- struct bt_avdtp_evt_stream_aborted_t

  Parameter to AVDTP_EVT_STREAM_ABORTED event.
- struct bt_avdtp_evt_media_packet_received_t

  Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event.
- struct bt_avdtp_evt_media_packet_sent_t

  Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event.
- struct bt_avdtp_evt_media_packet_send_failed_t

  Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event.
- union bt_avdtp_event_t

  Parameter to an application callback.
- struct bt_avdtp_transport_channel_t

  Transport channel description.
- struct bt_avdtp_transport_session_t

  Transport session description.
- struct bt_avdtp_stream_t

  Stream description.
- struct bt_avdtp_codec_t

  Codec handler description.
- struct bt_avdtp_codec_op_parse_config_t

  Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation.
- struct bt_avdtp_codec_op_serialize_config_t

  Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation.
- union bt_avdtp_codec_op_param_t

  Parameter to a codec handler.
- struct bt_avdtp_mgr_t

  AVDTP manager.

Macros

- #define bt_avdtp_connect(mgr, remote_addr) _bt_avdtp_open_control_channel_ex(mgr, remote_addr, HC↩ I_CONFIG_ENABLE_AUTHENTICATION | HCI_CONFIG_ENABLE_ENCRYPTION)

  Connect to a remote device.
- #define bt_avdtp_connect_ex(mgr, remote_addr, acl_config) _bt_avdtp_open_control_channel_ex(mgr, remote_addr, acl_config)

  Connect to a remote device.

Typedefs

- typedef bt_byte(∗ bt_avdtp_codec_handler_fp) (struct _bt_avdtp_codec_t ∗codec, bt_byte opcode, bt_↩ avdtp_codec_op_param_t ∗op_param, struct _bt_avdtp_mgr_t ∗mgr)

  Codec handler.
- typedef void(∗ bt_avdtp_mgr_callback_fp) (struct _bt_avdtp_mgr_t ∗mgr, bt_byte evt, bt_avdtp_event_↩ t ∗evt_param, void ∗callback_param)

  AVDTP application callback.

Functions

- bt_avdtp_mgr_t ∗ bt_avdtp_get_mgr (void)

  Return a pointer to an instance of the AVDTP manager.
- void bt_avdtp_init (void)

  Initialize the AVDTP layer.
- bt_bool bt_avdtp_start (bt_avdtp_mgr_t ∗mgr)

  Start the AVDTP layer.
- void bt_avdtp_register_callback (bt_avdtp_mgr_t ∗mgr, bt_avdtp_mgr_callback_fp callback, void ∗callback↩ _param)

  Register a AVDTP application callback.
- bt_byte bt_avdtp_register_sep (bt_avdtp_mgr_t ∗mgr, bt_byte type, const bt_avdtp_sep_capabilities_t ∗caps)

  Register a SEP with the local AVDTP manager.
- bt_avdtp_sep_t ∗ bt_avdtp_get_sep (bt_avdtp_mgr_t ∗mgr, bt_byte sep_id)

  Get a SEP info by its ID.
- bt_bool bt_avdtp_disconnect (bt_avdtp_mgr_t ∗mgr, bt_bdaddr_t ∗remote_addr)

  Disconnect from a remote device.
- bt_bool bt_avdtp_discover (bt_avdtp_mgr_t ∗mgr, bt_bdaddr_t ∗remote_addr)

  Discover SEPs on a remote device.
- bt_bool bt_avdtp_get_capabilities (bt_avdtp_mgr_t ∗mgr, bt_bdaddr_t ∗remote_addr, bt_byte seid_acp)

  Get remote SEP capabilities.
- bt_bool bt_avdtp_get_all_capabilities (bt_avdtp_mgr_t ∗mgr, bt_bdaddr_t ∗remote_addr, bt_byte seid_acp)

  Get remote SEP capabilities.
- bt_byte bt_avdtp_create_stream (bt_avdtp_mgr_t ∗mgr)

  Create a stream.
- bt_bool bt_avdtp_destroy_stream (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Destroy a stream.
- bt_bool bt_avdtp_listen (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_byte sep_id)

  Listen for incoming connections.
- void bt_avdtp_cancel_listen (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_byte sep_id)

  Cancel listening for incoming connections.
- bt_bool bt_avdtp_set_configuration (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_bdaddr_t ∗remote_addr, bt_byte seid_int, bt_byte seid_acp, const bt_avdtp_sep_capabilities_t ∗caps)

Set stream configuration.

- bt_bool bt_avdtp_get_configuration (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get stream configuration.
- bt_bool bt_avdtp_reconfigure_stream (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_avdtp_sep_↩ capabilities_t ∗caps)

  Reconfigure stream.
- bt_byte bt_avdtp_get_stream_state (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get local stream state.
- bt_byte bt_avdtp_get_stream_local_sep_id (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get stream's local SEP ID.
- bt_byte bt_avdtp_get_stream_remote_sep_id (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get stream's remote SEP ID.
- bt_bdaddr_t ∗ bt_avdtp_get_stream_remote_address (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get stream's remote BT address.
- bt_byte bt_avdtp_get_stream_codec_type (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get the type of the codec currently used with the stream.
- void ∗ bt_avdtp_get_stream_codec_config (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get the configuration of the codec currently used with the stream.
- bt_avdtp_sep_capabilities_t ∗ bt_avdtp_get_stream_config (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get stream's configuration.
- bt_bool bt_avdtp_open_stream (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Open a stream.
- bt_bool bt_avdtp_start_stream (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Start a stream.
- bt_bool bt_avdtp_close_stream (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Close a stream.
- bt_bool bt_avdtp_suspend_stream (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Suspend a stream.
- bt_bool bt_avdtp_abort_stream (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Suspend a stream.
- bt_bool bt_avdtp_security_control (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_byte ∗sc_data, bt_byte sc_data_len)

  Exchange content protection control data.
- bt_bool bt_avdtp_report_delay (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_uint delay)

  Report delay value of a Sink to a Source.
- bt_bool bt_avdtp_register_codec (bt_avdtp_mgr_t ∗mgr, bt_avdtp_codec_t ∗codec)

  Register a codec.
- bt_bool bt_avdtp_unregister_codec (bt_avdtp_mgr_t ∗mgr, bt_byte codec_type)

  Unregister a codec.
- bt_avdtp_codec_t ∗ bt_avdtp_find_codec (bt_avdtp_mgr_t ∗mgr, bt_byte codec_type)

  Find a codec.
- bt_bool bt_avdtp_add_media_rx_buffer (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_media_packet_↩ t ∗buffer)

  Add a media packet buffer to a receive queue.
- bt_bool bt_avdtp_remove_media_rx_buffer (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_media_packet↩ _t ∗buffer)

  Remove a media packet buffer from a receive queue.
- bt_bool bt_avdtp_add_media_tx_buffer (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_media_packet_↩ t ∗buffer)

  Add a media packet buffer to a send queue.

- bt_bool bt_avdtp_remove_media_tx_buffer (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle, bt_media_packet←↩ _t ∗buffer)

  Remove a media packet buffer from a send queue.
- bt_hci_conn_state_t ∗ bt_avdtp_get_hci_connection (bt_avdtp_mgr_t ∗mgr, bt_byte strm_handle)

  Get HCI connection for a stream.

Events

The following is a list of events AVDTP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

- #define AVDTP_EVT_CTRL_CHANNEL_CONNECTED 1

  This event is generated when a control channel between two AVDTP entities has been established.
- #define AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED 2

  This event is generated when a control channel between two AVDTP entities has been terminated.
- #define AVDTP_EVT_CTRL_CONNECTION_FAILED 3

  This event is generated when a local device failed to create a control channel between two AVDTP entities.
- #define AVDTP_EVT_DISCOVER_COMPLETED 4

  This event is generated when a local device received a response (either positive or negative) to a "discover" request.
- #define AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED 5

  This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
- #define AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED 6

  This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
- #define AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED 7

  This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
- #define AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED 8

  This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.
- #define AVDTP_EVT_OPEN_STREAM_COMPLETED 9

  This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
- #define AVDTP_EVT_START_STREAM_COMPLETED 10

  This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
- #define AVDTP_EVT_CLOSE_STREAM_COMPLETED 11

  This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
- #define AVDTP_EVT_SUSPEND_STREAM_COMPLETED 12

  This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
- #define AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED 13

  This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
- #define AVDTP_EVT_ABORT_STREAM_COMPLETED 14

  This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
- #define AVDTP_EVT_SEP_INFO_RECEIVED 15

  This event is generated for each SEP contained in a positive response to a "discover" request.
- #define AVDTP_EVT_SEP_CAPABILITIES_RECEIVED 16

  This event is generated when a local device received a positive response to a "get SEP capabilities" request.

- #define AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED 17

  This event is generated when a local device received a positive response to a "get stream configuration" request.
- #define AVDTP_EVT_DELAYREPORT_COMPLETED 18

  This event is generated when a local device received a response (either positive or negative) to a "delay report" request.
- #define AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED 50

  This event is generated when a local device received "set stream configuration" request.
- #define AVDTP_EVT_OPEN_STREAM_REQUESTED 51

  This event is generated when a local device received "open stream" request.
- #define AVDTP_EVT_START_STREAM_REQUESTED 52

  This event is generated when a local device received "start stream" request.
- #define AVDTP_EVT_CLOSE_STREAM_REQUESTED 53

  This event is generated when a local device received "close stream" request.
- #define AVDTP_EVT_SUSPEND_STREAM_REQUESTED 54

  This event is generated when a local device received "suspend stream" request.
- #define AVDTP_EVT_ABORT_STREAM_REQUESTED 55

  This event is generated when a local device received "abort stream" request.
- #define AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED 56

  This event is generated when a local device received "change stream configuration" request.
- #define AVDTP_EVT_MEDIA_PACKET_RECEIVED 57

  This event is generated when a local device received a media packet.
- #define AVDTP_EVT_STREAM_CONFIGURED 58

  This event is generated when a local device has successfully configured a stream.
- #define AVDTP_EVT_STREAM_RECONFIGURED 59

  This event is generated when a local device has successfully reconfigured a stream.
- #define AVDTP_EVT_STREAM_OPENED 60

  This event is generated when a local device has successfully opened a stream.
- #define AVDTP_EVT_STREAM_STARTED 61

  This event is generated when a local device has successfully started a stream.
- #define AVDTP_EVT_STREAM_CLOSED 62

  This event is generated when a local device has successfully closed a stream.
- #define AVDTP_EVT_STREAM_SUSPENDED 63

  This event is generated when a local device has successfully suspended a stream.
- #define AVDTP_EVT_STREAM_ABORTED 64

  This event is generated when a local device has successfully aborted a stream.
- #define AVDTP_EVT_MEDIA_PACKET_SENT 65

  This event is generated when a local device sent a media packet.
- #define AVDTP_EVT_MEDIA_PACKET_SEND_FAILED 66

  This event is generated when a local device failed to send a media packet.

Stream States

- #define AVDTP_STREAM_STATE_IDLE 0

  The stream is idle.
- #define AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS 1

  The stream is opening transport channels.
- #define AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS 2

  The stream is closing transport channels.
- #define AVDTP_STREAM_STATE_CONFIGURED 3

  The stream has been configured.
- #define AVDTP_STREAM_STATE_OPEN 4

The stream has been opened.

- #define AVDTP_STREAM_STATE_STREAMING 5

  The stream has been started.

- #define AVDTP_STREAM_STATE_CLOSING 6

  The stream is closing.

- #define AVDTP_STREAM_STATE_ABORTING 7

  The stream is aborting.

SEP Type

The following is a list of SEP types.

- #define AVDTP_SEP_TYPE_SOURCE 0

  Source (usually a device like a phone, desktop or laptop).

- #define AVDTP_SEP_TYPE_SINK 1

  Sink (usually a device like a headphones or BMW).

Service Categories

The following is a list of service categories a SEP supports.

Note

dotstack supports only AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT and AVDTP_SEP_S↩
ERVICE_CAPABILITY_MEDIA_CODEC.
These constants define values that are transfered OTA. They are not use in API. Constants for initializing bt_avdtp_sep_capabilities_t structure that is used to define a SEP's capabilities are defined with AVDTP_S↩
EP_CAPABILITY_FLAG_... constants.

- #define AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_TRANSPORT 1

  Media. A SEP is capable of transferring media (audio, video or both) packets.

- #define AVDTP_SEP_SERVICE_CAPABILITY_REPORTING 2

  Reporting. A SEP is capable of transferring reporting packets.

- #define AVDTP_SEP_SERVICE_CAPABILITY_RECOVERY 3

  Recovery. A SEP is capable of transferring recovery packets.

- #define AVDTP_SEP_SERVICE_CAPABILITY_CONTENT_PROTECTION 4

  Content Prortection. A SEP is capable of transferring content protection packets.

- #define AVDTP_SEP_SERVICE_CAPABILITY_HEADER_COMPRESSION 5

  Header Compression. A SEP can use header compression for transferring Media or Recovery packets.

- #define AVDTP_SEP_SERVICE_CAPABILITY_MULTIPLEXING 6

  Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.

- #define AVDTP_SEP_SERVICE_CAPABILITY_MEDIA_CODEC 7

  Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.

- #define AVDTP_SEP_SERVICE_CAPABILITY_DELAY_REPORTING 8

  Delay Reporting.

Service Categories Flags

The following is a list of constants that can be used to initialize bt_avdtp_sep_capabilities_t::categories. A combination of these constants defines service capabilities exposed by a SEP to a remote party.

Note

dotstack supports only AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT and AVDTP_SEP_CAP↩ABILITY_FLAG_MEDIA_CODEC. All other service capabilities are ignored. They are defined here only for completeness.

- #define AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT 0x01

  Media. A SEP is capable of transferring media (audio, video or both) packets.
- #define AVDTP_SEP_CAPABILITY_FLAG_REPORTING 0x02

  Reporting. A SEP is capable of transferring reporting packets.
- #define AVDTP_SEP_CAPABILITY_FLAG_RECOVERY 0x04

  Recovery. A SEP is capable of transferring recovery packets.
- #define AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION 0x08

  Content Prortection. A SEP is capable of transferring content protection packets.
- #define AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION 0x10

  Header Compression. A SEP can use header compression for transferring Media or Recovery packets.
- #define AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING 0x20

  Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.
- #define AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC 0x40

  Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.
- #define AVDTP_SEP_CAPABILITY_FLAG_DELAY_REPORTING 0x80

  Delat reporitng.

Transport Session Types

The following is a list of transport sessions a SEP supports.

Note

The only transport session cyrrently supported by dotstack is AVDTP_TRANSPORT_SESSION_TYPE_M↩EDIA.

- #define AVDTP_TRANSPORT_SESSION_TYPE_MEDIA 0

  Media (audio or video).
- #define AVDTP_TRANSPORT_SESSION_TYPE_REPORTING 1

  Reporting (currently not supported).
- #define AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY 2

  Recovery (currently not supported).

Codec Types

The following is a list of codecs a SEP supports.

- #define AVDTP_CODEC_TYPE_SBC 0x00

  SBC (mandatory to support in A2DP profile).
- #define AVDTP_CODEC_TYPE_MPEG1_2_AUDIO 0x01

  MPEG-1,2 (optional).

- #define AVDTP_CODEC_TYPE_MPEG2_4_AAC 0x02

  MPEG-2,4 AAC (optional, used in Apple's products).
- #define AVDTP_CODEC_TYPE_ATRAC 0x04

  ATRAC (proprietary codec owned by Sony Corporation).
- #define AVDTP_CODEC_TYPE_NON_A2DP 0xFF

  Vendor specific.

Media Types

The following is a list of media types a SEP can support.

- #define AVDTP_MEDIA_TYPE_AUDIO 0

  Audio.
- #define AVDTP_MEDIA_TYPE_VIDEO 1

  Video.
- #define AVDTP_MEDIA_TYPE_MULTIMEDIA 2

  Both Audio & Video.

Codec Handler Operations

- #define AVDTP_CODEC_OPCODE_PARSE_CONFIG 0

  Parse codec configuration.
- #define AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG 1

  Serialize codec configuration.

Error Codes

- #define AVDTP_ERROR_SUCCESS 0

  The operation completed with no errors.
- #define AVDTP_ERROR_BAD_HEADER_FORMAT 0x01

  The request packet header format is invalid.
- #define AVDTP_ERROR_BAD_LENGTH 0x11

  The request packet length is not match the assumed length.
- #define AVDTP_ERROR_BAD_ACP_SEID 0x12

  The requested command indicates an invalid ACP SEP ID (not addressable)
- #define AVDTP_ERROR_SEP_IN_USE 0x13

  The SEP is in use.
- #define AVDTP_ERROR_SEP_NOT_IN_USE 0x14

  The SEP is not in use.
- #define AVDTP_ERROR_BAD_SERV_CATEGORY 0x17

  The value of Service Category in the request packet is not defined in AVDTP.
- #define AVDTP_ERROR_BAD_PAYLOAD_FORMAT 0x18

  The requested command has an incorrect payload format.
- #define AVDTP_ERROR_NOT_SUPPORTED_COMMAND 0x19

  The requested command is not supported by the device.
- #define AVDTP_ERROR_INVALID_CAPABILITIES 0x1a

  The reconfigure command is an attempt to reconfigure a transport service capabilities of the SEP. Reconfigure is only permitted for application service capabilities.
- #define AVDTP_ERROR_BAD_RECOVERY_TYPE 0x22

  The requested Recovery Type is not defined in AVDTP.

- #define AVDTP_ERROR_BAD_MEDIA_TRANSPORT_FORMAT 0x23

  The format of Media Transport Capability is not correct.
- #define AVDTP_ERROR_BAD_RECOVERY_FORMAT 0x25

  The format of Recovery Service Capability is not correct.
- #define AVDTP_ERROR_BAD_ROHC_FORMAT 0x26

  The format of Header Compression Service Capability is not correct.
- #define AVDTP_ERROR_BAD_CP_FORMAT 0x27

  The format of Content Protection Service Capability is not correct.
- #define AVDTP_ERROR_BAD_MULTIPLEXING_FORMAT 0x28

  The format of Multiplexing Service Capability is not correct.
- #define AVDTP_ERROR_UNSUPPORTED_CONFIGURAION 0x29

  Configuration not supported.
- #define AVDTP_ERROR_BAD_STATE 0x31

  The stream is in state that does not permit executing commands.
- #define AVDTP_ERROR_FAILED_TO_CONNECT_TRANSPORT 0x40

  An attempt to esctablish a transport channel has failed.
- #define AVDTP_ERROR_FAILED_TO_CONNECT_CONTROL 0x41

  An attempt to esctablish a control channel has failed.

3.11.1 Detailed Description

AVDTP is the transport protocol for audio and/or video distribution connections and streaming of audio or video media over the Bluetooth air interface.

3.11.2 Macro Definition Documentation

3.11.2.1 #define AVDTP_EVT_STREAM_ABORTED 64

This event is generated when a local device has successfully aborted a stream.

This event follows the AVDTP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream abortion was initiated by the local device.

3.11.2.2 #define AVDTP_EVT_STREAM_CLOSED 62

This event is generated when a local device has successfully closed a stream.

This event follows the AVDTP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream closing was initiated by the local device.

3.11.2.3 #define AVDTP_EVT_STREAM_CONFIGURED 58

This event is generated when a local device has successfully configured a stream.

This event follows the AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. This event is not generated if stream configuration was initiated by the local device.

3.11.2.4 #define AVDTP_EVT_STREAM_OPENED 60

This event is generated when a local device has successfully opened a stream.

This event follows the AVDTP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream opening was initiated by the local device.

3.11.2.5 #define AVDTP_EVT_STREAM_RECONFIGURED 59

This event is generated when a local device has successfully reconfigured a stream.

This event follows the AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream reconfiguration was initiated by the local device.

3.11.2.6 #define AVDTP_EVT_STREAM_STARTED 61

This event is generated when a local device has successfully started a stream.

This event follows the AVDTP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream starting was initiated by the local device.

3.11.2.7 #define AVDTP_EVT_STREAM_SUSPENDED 63

This event is generated when a local device has successfully suspended a stream.

This event follows the AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream suspension was initiated by the local device.

3.11.2.8 #define AVDTP_STREAM_STATE_ABORTING 7

The stream is aborting.

This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

3.11.2.9 #define AVDTP_STREAM_STATE_CLOSING 6

The stream is closing.

This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP_STREAM_STATE_IDLE state.

3.11.2.10 #define AVDTP_STREAM_STATE_IDLE 0

The stream is idle.

This can mean two things. The stream specified by strm_handle does not exist or the stream is closed.

3.11.2.11 #define AVDTP_STREAM_STATE_STREAMING 5

The stream has been started.

Depending on the local SEP type (source or sink) it means that the stream can send or receive media packets.

3.11.2.12 #define bt_avdtp_connect(*mgr,* *remote_addr*) _bt_avdtp_open_control_channel_ex(mgr, remote_addr, HCI_CONFIG_ENABLE_AUTHENTICATION | HCI_CONFIG_ENABLE_ENCRYPTION)

Connect to a remote device.

This function opens a control channel connection to a remote device specified by the `remote_addr`. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CH↩ ANNEL_CONNECTION_FAILED.

Parameters

mgr	AVDTP manager.
remote_addr	The address of a remote device.

Returns

- `TRUE` if connection establishment has been started.
- `FALSE` otherwise.

3.11.2.13 #define bt_avdtp_connect_ex(*mgr, remote_addr, acl_config*) _bt_avdtp_open_control_channel_ex(mgr, remote_addr, acl_config)

Connect to a remote device.

This function opens a control channel connection to a remote device specified by the `remote_addr`. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP_EVT_CTRL_CHANNEL_CONNECTED or AVDTP_EVT_CTRL_CH←ANNEL_CONNECTION_FAILED.

Parameters

mgr	AVDTP manager.
remote_addr	The address of a remote device.
acl_config	ACL link configuration. This can be a combination of the following values: • HCI_CONFIG_ENABLE_AUTHENTICATION • HCI_CONFIG_ENABLE_ENCRYPTION • HCI_CONFIG_BECOME_MASTER

Returns

- `TRUE` if connection establishment has been started.
- `FALSE` otherwise.

3.11.3 Typedef Documentation

3.11.3.1 typedef bt_byte(∗ bt_avdtp_codec_handler_fp) (struct _bt_avdtp_codec_t ∗codec, bt_byte opcode, bt_avdtp_codec_op_param_t ∗op_param, struct _bt_avdtp_mgr_t ∗mgr)

Codec handler.

AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request.
This typedef defines the interace for the callback function.

Parameters

codec	A pointer to a structure that describes a codec.
opcode	The code of an operation to execute. The `opcode` can be one of the following values: • AVDTP_CODEC_OPCODE_PARSE_CONFIG: The handler has to parse configuration received from the remote device and store it in a structure defined by the consumer. • AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG: The handler has to serialize the data from a consumer defined structure to a format suitable for sending as a part of a AVDTP request.
op_param	A pointer to the operation's specific parameters. Parameters are passed as a pointer to the bt_avdtp_codec_op_param_t union. Which member of the union points to a valid structure depends on the value of the `opcode`: • bt_avdtp_codec_op_param_t::parse: Valid if `opcode` == AVDTP_CODEC_OPCO↩DE_PARSE_CONFIG. • bt_avdtp_codec_op_param_t::serialize: Valid if `opcode` == AVDTP_CODEC_OPC↩ODE_SERIALIZE_CONFIG.
mgr	AVDTP manager.

3.11.3.2 typedef void(∗ bt_avdtp_mgr_callback_fp) (struct _bt_avdtp_mgr_t ∗mgr, bt_byte evt, bt_avdtp_event_t ∗evt_param, void ∗callback_param)

AVDTP application callback.

In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call that function whenever a new event has been generated.

Parameters

mgr	AVDTP manager.
evt	AVDTP event. The event can be one of the following values: • AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. • AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. • AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). • AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. • AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation. • AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation. • AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation. • AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation. • AVDTP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation. • AVDTP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation. • AVDTP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation. • AVDTP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation. • AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation. • AVDTP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation. • AVDTP_EVT_SEP_INFO_RECEIVED: SEP information received. • AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received. • AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received. • AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration. • AVDTP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream. • AVDTP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream. • AVDTP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream. • AVDTP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream. • AVDTP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream. • AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream. • AVDTP_EVT_MEDIA_PACKET_RECEIVED: [illegible]

callback_param	A pointer to an arbitrary data set by a call to bt_avdtp_register_callback.

3.11.4 Function Documentation

3.11.4.1 bt_bool bt_avdtp_abort_stream (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *strm_handle*)

Suspend a stream.

This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the AVDTP_EVT_ABORT_STREAM_↩ COMPLETED event will be generated. This operation cannot be rejected. The `evt_param.abort_stream↩ _requested.err_code` is always == AVDTP_ERROR_SUCCESS.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.2 bt_bool bt_avdtp_add_media_rx_buffer (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *strm_handle,* bt_media_packet_t ∗ *buffer*)

Add a media packet buffer to a receive queue.

The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated. This function adds a buffer to the receive queue.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist. The stream can be in any state to call this function.

3.11.4.3 bt_bool bt_avdtp_add_media_tx_buffer (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *strm_handle,* bt_media_packet_t ∗ *buffer*)

Add a media packet buffer to a send queue.

When the consumer of AVDTP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a A↩ VDTP_EVT_MEDIA_PACKET_SENT is generated. Otherwise a AVDTP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as AVDTP has removed it from the queue and gave up control over it. As in the case of received buffers, if a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP_EVT_MEDIA_PACKET_SENT event for each buffer and sets the data_len field to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist. The stream can be in any state to call this function.

3.11.4.4 void bt_avdtp_cancel_listen (bt_avdtp_mgr_t ∗ *mgr*, bt_byte *strm_handle*, bt_byte *sep_id*)

Cancel listening for incoming connections.

This function removes a SEP from a list of SEPS which a stream can use for incoming requests.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
sep_id	Local SEP ID.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.11.4.5 bt_bool bt_avdtp_close_stream (bt_avdtp_mgr_t ∗ *mgr*, bt_byte *strm_handle*)

Close a stream.

This function tries to close a stream by sending a request to the remote party. The stream has to be in AVD↩ TP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the `evt↩ _param.bt_avdtp_evt_close_stream_completed_t.err_code` == AVDTP_ERROR_SUCCES↩ S. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the `evt_param.↩ bt_avdtp_evt_close_stream_completed_t.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.

strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.6 bt_byte bt_avdtp_create_stream (bt_avdtp_mgr_t ∗ *mgr*)

Create a stream.

This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort.

Parameters

mgr	AVDTP manager.

Returns

- `Stream` handle if the function succeeds.
- `0` otherwise.

3.11.4.7 bt_bool bt_avdtp_destroy_stream (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *strm_handle*)

Destroy a stream.

This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.11.4.8 bt_bool bt_avdtp_disconnect (bt_avdtp_mgr_t ∗ *mgr,* bt_bdaddr_t ∗ *remote_addr*)

Disconnect from a remote device.

This function closes a control and transport channels on all streams associated with the remote device specified by the `remote_addr`. As a result of this operation the following events will be generated:

- AVDTP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0
- AVDTP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0
- AVDTP_EVT_STREAM_CLOSED: this event is generate if a stream is in "closing" state as a result of a request from the remote device or bt_avdtp_close_stream call before bt_avdtp_disconnect call
- AVDTP_EVT_STREAM_ABORTED: this event is generated if a stream is in "active" state at the time of bt_avdtp_disconnect call.

Parameters

mgr	AVDTP manager.
remote_addr	The address of a remote device.

Returns

- `TRUE` if disconnection has been started.
- `FALSE` otherwise. No events will be generated.

3.11.4.9 bt_bool bt_avdtp_discover (bt_avdtp_mgr_t ∗ *mgr,* bt_bdaddr_t ∗ *remote_addr*)

Discover SEPs on a remote device.

This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated:

- AVDTP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the `evt_param.sep_info_received` contains SEP information.
- AVDTP_EVT_DISCOVER_COMPLETED: this event is generated after last AVDTP_EVT_SEP_INFO_R↩ ECEIVED if the remote accepted the request and the `evt_param.discover_completed.err_↩ code` == AVDTP_ERROR_SUCCESS. if the remote rejected the request the `evt_param.discover↩ _completed.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.
remote_addr	The address of a remote device.

Returns

- `TRUE` if discover request has been sent.
- `FALSE` otherwise. No events will be generated.

3.11.4.10 bt_avdtp_codec_t∗ bt_avdtp_find_codec (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *codec_type*)

Find a codec.

AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function returns a pointer to a structure that holds a pointer to a codec's callback function.

Parameters

mgr	AVDTP manager.
codec_type	Codec type. The codec_type can be one of the following values: • AVDTP_CODEC_TYPE_SBC: SBC • AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) • AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) • AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) • AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails if a callback for a codec type specified in the `codec` parameter
- has not been previously registered with bt_avdtp_register_codec.

3.11.4.11 bt_bool bt_avdtp_get_all_capabilities (bt_avdtp_mgr_t ∗ *mgr*, bt_bdaddr_t ∗ *remote_addr*, bt_byte *seid_acp*)

Get remote SEP capabilities.

This function asks the remote device to send capabilities of a SEP specified by the `seid_acp`. As a result of this operation the following events will be generated:

- AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the `evt_param.sep_capabilities_received` contains SEP capabilities.
- AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EV↩ T_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the `evt_param.get_sep_↩ capabilities_completed.err_code` == AVDTP_ERROR_SUCCESS. if the remote rejected the request the `evt_param.get_sep_capabilities_completed.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.
remote_addr	The address of a remote device.
seid_acp	The ID of a remote SEP.

Returns

- `TRUE` if discover request has been sent.
- `FALSE` otherwise. No events will be generated.

3.11.4.12 bt_bool bt_avdtp_get_capabilities (bt_avdtp_mgr_t ∗ *mgr*, bt_bdaddr_t ∗ *remote_addr*, bt_byte *seid_acp*)

Get remote SEP capabilities.

This function asks the remote device to send capabilities of a SEP specified by the `seid_acp`. As a result of this operation the following events will be generated:

- AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the `evt_param.sep_capabilities_received` contains SEP capabilities.

- AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after AVDTP_EV↩ T_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the `evt_param.get_sep_↩ capabilities_completed.err_code` == AVDTP_ERROR_SUCCESS. if the remote rejected the request the `evt_param.get_sep_capabilities_completed.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.
remote_addr	The address of a remote device.
seid_acp	The ID of a remote SEP.

Returns

- `TRUE` if discover request has been sent.
- `FALSE` otherwise. No events will be generated.

3.11.4.13 bt_bool bt_avdtp_get_configuration (**bt_avdtp_mgr_t** ∗ *mgr,* bt_byte *strm_handle*)

Get stream configuration.

This function requests stream configuration from a remote device. As a result of this operation the following events will be generated:

- AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: this event is generated if the remote accepted the request. the `ebt_para.sep_capabilities_received.caps` will contain current stream configuration.
- AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: If the remote accepted the request the `evt_param.get_stream_configuration_completed.err_code` == AVDTP_ERROR_S↩ UCCESS. if the remote rejected the request the `evt_param.get_stream_configuration_↩ completed.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.14 bt_hci_conn_state_t∗ bt_avdtp_get_hci_connection (**bt_avdtp_mgr_t** ∗ *mgr,* bt_byte *strm_handle*)

Get HCI connection for a stream.

This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- `Pointer` to a structure that describes an HCI connection if the function succeeds.
- `NULL` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist or there is no HCI connection between local and remote devices associated with the stream.

Note

This function has not been implemented.

3.11.4.15 bt_avdtp_mgr_t∗ bt_avdtp_get_mgr (void)

Return a pointer to an instance of the AVDTP manager.

This function returns a pointer to an instance of the AVDTP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all AVDTP functions.

3.11.4.16 bt_avdtp_sep_t∗ bt_avdtp_get_sep (bt_avdtp_mgr_t ∗ *mgr*, bt_byte *sep_id*)

Get a SEP info by its ID.

This function returns a pointer to bt_avdtp_sep_t structure that describes a SEP previously registered with bt_←
avdtp_register_sep.

Parameters

mgr	AVDTP manager.
sep_id	The ID of a SEP.

Returns

- `Pointer` to bt_avdtp_sep_t if the SEP is in the list of registered SEPs.
- `NULL` otherwise.

3.11.4.17 void∗ bt_avdtp_get_stream_codec_config (bt_avdtp_mgr_t ∗ *mgr*, bt_byte *strm_handle*)

Get the configuration of the codec currently used with the stream.

This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MP←
EG-2,4 AAC codecs:

- SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h)
- MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h)
- MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following state:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING

\li NULL otherwise.
```

3.11.4.18 bt_byte bt_avdtp_get_stream_codec_type (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *strm_handle*)

Get the type of the codec currently used with the stream.

This function returns the type of the codec currently used with the stream.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states:

```
  AVDTP_STREAM_STATE_CONFIGURED
  AVDTP_STREAM_STATE_OPEN
  AVDTP_STREAM_STATE_STREAMING

@arg The result will be one of the following values:

   AVDTP_CODEC_TYPE_SBC:            SBC
   AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)
   AVDTP_CODEC_TYPE_MPEG2_4_AAC:   MPEG-2,4 AAC (used in Apple products)
   AVDTP_CODEC_TYPE_ATRAC:         ATRAC (used in Sony products)
   AVDTP_CODEC_TYPE_NON_A2DP:      Non-A2DP Codec

@arg 0xFF otherwise.
```

3.11.4.19 bt_avdtp_sep_capabilities_t∗ bt_avdtp_get_stream_config (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *strm_handle*)

Get stream's configuration.

This function returns a pointer to a structure holding the current configuration of stream.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING

\li NULL otherwise.
```

3.11.4.20 bt_byte bt_avdtp_get_stream_local_sep_id (bt_avdtp_mgr_t * *mgr*, bt_byte *strm_handle*)

Get stream's local SEP ID.

This function returns the ID of the local SEP associated with the stream.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- The ID of the local SEP if strm_handle specifies a valid stream.
- 0 otherwise.

3.11.4.21 bt_bdaddr_t* bt_avdtp_get_stream_remote_address (bt_avdtp_mgr_t * *mgr*, bt_byte *strm_handle*)

Get stream's remote BT address.

This function returns the address of the remote device associated with the stream.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- The address of the remote device if strm_handle specifies a valid stream.
- NULL otherwise.

3.11.4.22 bt_byte bt_avdtp_get_stream_remote_sep_id (bt_avdtp_mgr_t * *mgr*, bt_byte *strm_handle*)

Get stream's remote SEP ID.

This function returns the ID of the remote SEP associated with the stream.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- The ID of the remote SEP if strm_handle specifies a valid stream.
- 0 otherwise.

3.11.4.23 bt_byte bt_avdtp_get_stream_state (bt_avdtp_mgr_t * *mgr*, bt_byte *strm_handle*)

Get local stream state.

This function returns local state of a stream specified by the `strm_handle`. No request is sent to the remote party.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

The state of the stream. The result will be one of the following values:

- AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by `strm_handle` does not exist or the stream is closed.
- AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels.
- AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels.
- AVDTP_STREAM_STATE_CONFIGURED: The stream has been configured.
- AVDTP_STREAM_STATE_OPEN: The stream has been opened.
- AVDTP_STREAM_STATE_STREAMING: The stream has been started. Depending on the local SEP type (source or sink) it means that the stream is can send or receive media packets.
- AVDTP_STREAM_STATE_CLOSING: The stream is closing. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP↩_STREAM_STATE_IDLE state.
- AVDTP_STREAM_STATE_ABORTING: The stream is aborting. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP↩_STREAM_STATE_IDLE state.

3.11.4.24 void bt_avdtp_init (void)

Initialize the AVDTP layer.

This function initializes the AVDTP layer of the stack. It must be called prior to any other AVDTP function can be called.

3.11.4.25 bt_bool bt_avdtp_listen (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *strm_handle,* bt_byte *sep_id*)

Listen for incoming connections.

This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
sep_id	Local SEP ID.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.11.4.26 bt_bool bt_avdtp_open_stream (bt_avdtp_mgr_t ∗ *mgr,* bt_byte *strm_handle*)

Open a stream.

This function tries to open a stream by sending a request to the remote party. The stream has to be already configured with a bt_avdtp_set_configuration call. As a result of this operation the AVDTP_EVT_OPEN_STRE↩ AM_COMPLETED event will be generated. If the stream has been open the `evt_param.open_stream_↩ completed.err_code` == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to open the stream, the `evt_param.open_stream_completed.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.27 bt_bool bt_avdtp_reconfigure_stream (**bt_avdtp_mgr_t** ∗ *mgr,* bt_byte *strm_handle,* **bt_avdtp_sep_capabilities_t** ∗ *caps*)

Reconfigure stream.

This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the `evt_param.stream_reconfigure_completed.err_code` == A↩ VDTP_ERROR_SUCCESS. Otherwise the `evt_param.stream_reconfigure_completed.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
caps	New stream configuration.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.28 void bt_avdtp_register_callback (**bt_avdtp_mgr_t** ∗ *mgr,* **bt_avdtp_mgr_callback_fp** *callback,* void ∗ *callback_param*)

Register a AVDTP application callback.

In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

- AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected.
- AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected.
- AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device).
- AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs.
- AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation.

- AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation.
- AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation.
- AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation.
- AVDTP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation.
- AVDTP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation.
- AVDTP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation.
- AVDTP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation.
- AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation.
- AVDTP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation.
- AVDTP_EVT_SEP_INFO_RECEIVED: SEP information received.
- AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received.
- AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.
- AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration.
- AVDTP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream.
- AVDTP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream.
- AVDTP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream.
- AVDTP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream.
- AVDTP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream.
- AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream.
- AVDTP_EVT_MEDIA_PACKET_RECEIVED: Remote device sent a media packet.
- AVDTP_EVT_STREAM_CONFIGURED: A stream has been configured (This event is generated right after AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the local devices accepted the request).
- AVDTP_EVT_STREAM_RECONFIGURED: A stream has been re-configured (This event is generated right after AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED if the local devices accepted the request).
- AVDTP_EVT_STREAM_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request).
- AVDTP_EVT_STREAM_STARTED: A stream has been started (This event is generated right after AVDTP_EVT_START_STREAM_REQUESTED if the local devices accepted the request).
- AVDTP_EVT_STREAM_CLOSED: A stream has been close (This event is generated right after AVDTP_EVT_CLOSE_STREAM_REQUESTED if the local devices accepted the request).
- AVDTP_EVT_STREAM_SUSPENDED: A stream has been suspended (This event is generated right after AVDTP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request).

- AVDTP_EVT_STREAM_ABORTED: A stream has been aborted (This event is generated right after AVDT↩ P_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request. It is also generated if connection between devices has been terminated by means other than AVDTP signaling, e.g. devices going out of rage).
- AVDTP_EVT_MEDIA_PACKET_SENT: The local device has successfully sent a media packet to the remote device.
- AVDTP_EVT_MEDIA_PACKET_SEND_FAILED: The local device was not able to send a media packet to the remote device.

Parameters

mgr	AVDTP manager.
callback	The callback function that will be called when the AVDTP generates an event.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.11.4.29 bt_bool bt_avdtp_register_codec (**bt_avdtp_mgr_t** ∗ *mgr,* **bt_avdtp_codec_t** ∗ *codec*)

Register a codec.

AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function adds a codec's callback function to an internal list.

Parameters

mgr	AVDTP manager.
codec	Pointer to a structure specifying codec type and a callback.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails if there already is a callback for a codec type specified in the `codec` parameter.

3.11.4.30 bt_byte bt_avdtp_register_sep (**bt_avdtp_mgr_t** ∗ *mgr,* bt_byte *type,* const **bt_avdtp_sep_capabilities_t** ∗ *caps*)

Register a SEP with the local AVDTP manager.

This function is used to make a list of SEPs supported by the local ADVTP entity.

Parameters

mgr	AVDTP manager.
type	The type of a SEP. The type can be one of the following values: • AVDTP_SEP_TYPE_SOURCE: The SEP is a source. • AVDTP_SEP_TYPE_SINK: The SEP is a sink.
caps	The capabilities of a SEP.

Returns

- `ID` of a SEP if the function succeeds.
- `FALSE` otherwise.

3.11.4.31 bt_bool bt_avdtp_remove_media_rx_buffer (bt_avdtp_mgr_t * *mgr*, bt_byte *strm_handle*, bt_media_packet_t * *buffer*)

Remove a media packet buffer from a receive queue.

The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated. This function removes a buffer from the receive queue.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist. The stream can be in any state to call this function.

3.11.4.32 bt_bool bt_avdtp_remove_media_tx_buffer (bt_avdtp_mgr_t * *mgr*, bt_byte *strm_handle*, bt_media_packet_t * *buffer*)

Remove a media packet buffer from a send queue.

When the consumer of AVDTP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt_avdtp_remove_media↩
_tx_buffer.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist. The stream can be in any state to call this function.

Note

This function has not been implemented.

3.11.4.33 bt_bool bt_avdtp_report_delay (**bt_avdtp_mgr_t** ∗ *mgr,* bt_byte *strm_handle,* bt_uint *delay*)

Report delay value of a Sink to a Source.

This function sends the delay value of a Sink to a Source. This enables synchronous playback of audio and video. Delay reports are always sent from the Sink to the Source. If the Sink's delay report has been accepted by the Source the `evt_param.delay_report_completed.err_code` == AVDTP_ERROR_SUCCESS. Otherwise the `evt_param.delay_report_completed.err_code` == the error code sent by the Source.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
delay	The delay value in 1/10 milliseconds.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.34 bt_bool bt_avdtp_security_control (**bt_avdtp_mgr_t** ∗ *mgr,* bt_byte *strm_handle,* bt_byte ∗ *sc_data,* bt_byte *sc_data_len*)

Exchange content protection control data.

This function tries to establish content protection by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE, AVDTP_STREAM_STATE_CLOSING, AVDTP_ST↩ REAM_STATE_ABORTING. As a result of this operation the AVDTP_EVT_STREAM_SECURITY_CONTROL↩ _COMPLETED event will be generated. If the stream's content protection data has been accepted by the remote party the `evt_param.security_control_completed.err_code` == AVDTP_ERROR_SUCC↩ ESS. Otherwise the `evt_param.security_control_completed.err_code` == the error code sent by the remote.

Note

The dotstack does not support content protection. Although the request can be sent it will not affect the operation of the AVDTP in any way.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.35 bt_bool bt_avdtp_set_configuration (**bt_avdtp_mgr_t** ∗ *mgr,* bt_byte *strm_handle,* bt_bdaddr_t ∗ *remote_addr,* bt_byte *seid_int,* bt_byte *seid_acp,* const **bt_avdtp_sep_capabilities_t** ∗ *caps*)

Set stream configuration.

This function tries to configure a stream before opening it. As a result of this operation the AVDTP_EVT_SET_↩ STREAM_CONFIGURATION_COMPLETED event will be generated. If configuration was a success the `evt_↩ param.set_stream_configuration_completed.err_code` == AVDTP_ERROR_SUCCESS. Otherwise the `evt_param.set_stream_configuration_completed.err_code` == the error code sent by the remote and `evt_param.set_stream_configuration_completed.svc_category` == the value of the first Service Category to fail.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.
remote_addr	The address of a remote device.
seid_int	Local SEP ID.
seid_acp	Remote SEP ID.
caps	Stream configuration.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. No events will be generated.

3.11.4.36 bt_bool bt_avdtp_start (bt_avdtp_mgr_t * *mgr*)

Start the AVDTP layer.

This function makes the AVDTP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required.

Parameters

mgr	AVDTP manager.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.11.4.37 bt_bool bt_avdtp_start_stream (bt_avdtp_mgr_t * *mgr*, bt_byte *strm_handle*)

Start a stream.

This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_STR←EAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the AVDTP_EVT_START_STREAM_COMPLETED event will be generated. If the stream has been open the `evt_param.start_stream_requested.err←_code` == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the `evt_param.start_stream_requested.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.38 bt_bool bt_avdtp_suspend_stream (bt_avdtp_mgr_t * *mgr*, bt_byte *strm_handle*)

Suspend a stream.

This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDT↩ P_STREAM_STATE_STREAMING state. As a result of this operation the AVDTP_EVT_SUSPEND_STREAM_↩ COMPLETED event will be generated. If the stream has been suspended the `evt_param.bt_avdtp_evt↩ _suspend_stream_requested_t.err_code` == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the `evt_param.bt_avdtp_evt_↩ suspend_stream_requested_t.err_code` == the error code sent by the remote.

Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.11.4.39 bt_bool bt_avdtp_unregister_codec (**bt_avdtp_mgr_t** ∗ *mgr,* bt_byte *codec_type*)

Unregister a codec.

AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function removes a codec's callback function from an internal list.

Parameters

mgr	AVDTP manager.
codec_type	Codec type. The codec_type can be one of the following values: • AVDTP_CODEC_TYPE_SBC: SBC • AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) • AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) • AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) • AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails if a callback for a codec type specified in the `codec` parameter
- has not been previously registered with bt_avdtp_register_codec.

3.12 Configuration

This module describes parameters used to configure AVDTP layer.

Macros

- #define AVDTP_MAX_SEP

  Maximum number of SEPs that can be exposed by a local device.
- #define AVDTP_MAX_STREAMS

  Maximum number of streams that can be exposed by a local device.
- #define AVDTP_MAX_REMOTE_DEVICES

  Maximum number of remote devices a local device can be connected to.
- #define AVDTP_MAX_CMD_BUFFERS

  Maximum number of command buffers.
- #define AVDTP_MAX_TRANSPORT_CHANNELS

  Maximum number of transport channels.
- #define AVDTP_MAX_TX_BUFFER_LEN

  Size of the transmit buffer.
- #define AVDTP_MAX_CMD_PARAM_LEN

  Maximum length of control command parameters.
- #define AVDTP_CODEC_CONFIG_BUFFER_LEN

  Size of the buffer used to store codec specific configuration.

3.12.1 Detailed Description

This module describes parameters used to configure AVDTP layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI, L2CAP and SDP must always be present

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS            ...
#define HCI_MAX_DATA_BUFFERS           ...
#define HCI_MAX_HCI_CONNECTIONS        ...
#define HCI_RX_BUFFER_LEN              ...
#define HCI_TX_BUFFER_LEN              ...
#define HCI_L2CAP_BUFFER_LEN           ...
#define HCI_MAX_CMD_PARAM_LEN          ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS          ...
#define L2CAP_MAX_FRAME_BUFFERS        ...
#define L2CAP_MAX_PSMS                 ...
#define L2CAP_MAX_CHANNELS             ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN     ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN   ...

// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure AVDTP & A2DP (this one does not need configuration),
// the following values must be defined:

#define BT_INCLUDE_AVDTP               // tells dotstack to compile in AVDTP support
#define AVDTP_MAX_SEP                  ...
#define AVDTP_MAX_STREAMS              ...
#define AVDTP_MAX_REMOTE_DEVICES       ...
#define AVDTP_MAX_CMD_BUFFERS          ...
#define AVDTP_MAX_TRANSPORT_CHANNELS   ...
#define AVDTP_MAX_TX_BUFFER_LEN        ...
#define AVDTP_MAX_CMD_PARAM_LEN        ...
#define AVDTP_CODEC_CONFIG_BUFFER_LEN  ...

#include "cdbt/bt/bt_oem_config.h"
```

3.12.2 Macro Definition Documentation

3.12.2.1 #define AVDTP_CODEC_CONFIG_BUFFER_LEN

Size of the buffer used to store codec specific configuration.

Each codec uses unique configuration which can take different amount of memory. This parameter defines the size of the buffer for storing codec's configuration. The value of 16 is sufficient for SBC, AAC and MPEG1,2. If vendor specific codec is to be used this value may need to increased.

3.12.2.2 #define AVDTP_MAX_CMD_BUFFERS

Maximum number of command buffers.

This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVDTP_MAX_REMOTE_DEVICES $*$ AVDTP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. 2 is usually sufficient.

3.12.2.3 #define AVDTP_MAX_CMD_PARAM_LEN

Maximum length of control command parameters.

This parameter defines the maximum length of all command parameters. The value should not exceed AVDTP_↩ MAX_TX_BUFFER_LEN - 2 (command header).

3.12.2.4 #define AVDTP_MAX_REMOTE_DEVICES

Maximum number of remote devices a local device can be connected to.

This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. control channels). This value should not exceed AVDTP_MAX_STREAMS. For each remote device AVDTP creates one control channel regardless of the number of streams between the local and the remote devices. Assuming that the local devices wants to have only one channel with each remote device and if AVDTP_MAX_REMOTE_DEV↩ ICES $>$ AVDTP_MAX_STREAMS all memory reserved for devices in excess of AVDTP_MAX_STREAMS will be wasted. The minimum value is 1. The maximum value is 255.

3.12.2.5 #define AVDTP_MAX_SEP

Maximum number of SEPs that can be exposed by a local device.

This parameter defines the number of SEPs an application can expose to remote devices. The minimum value is 1. The maximum value is 255.

3.12.2.6 #define AVDTP_MAX_STREAMS

Maximum number of streams that can be exposed by a local device.

This parameter defines the number of streams an application can open between local and remote devices. This value can be different from AVDTP_MAX_SEP but should not exceed it. Since each SEP can only be used once the local device can only have as much streams as there are SEPs. If AVDTP_MAX_STREAMS $>$ AVDTP_MAX_SEP all memory reserved for streams in excess of AVDTP_MAX_SEP will be wasted. The minimum value is 1. The maximum value is 255.

3.12.2.7 #define AVDTP_MAX_TRANSPORT_CHANNELS

Maximum number of transport channels.

Depending on the SEP capabilities (multiplexing, recovery, reporting) each stream may need up to 3 transport channels. E.g., if multiplexing is not supported and recovery and reporting are supported a stream will use 3 transport channels - 1 for media transport session, 1 for recovery transport session and 1 for reporting transport session. If multiplexing is not supported and only reporting is supported the stream will use 2 transport channels - 1 for media transport session and 1 for reporting transport session. If multiplexing is supported the stream will need only 1 transport channel for all supported transport session. With multiplexing even different streams can share the same transport channel. dotstack currently does not support multiplexing, recovery and reporting. So each stream needs its own transport channel for it media transport session. Hence, AVDTP_MAX_TRANSPORT_CHANNELS must be equal to AVDTP_MAX_STREAMS

3.12.2.8 #define AVDTP_MAX_TX_BUFFER_LEN

Size of the transmit buffer.

This parameter defines the size of the buffer used to send AVDTP control commands to L2CAP layer. Each control channel has it own buffer so the total amount of memory allocated for these buffers is AVDTP_MAX_TX_BUFFE←R_LEN) ∗ AVDTP_MAX_REMOTE_DEVICES. The minimum value is 1. The maximum value is 255. The value of 32 is usually sufficient.

3.13 Advanced Audio Distribution Profile (A2DP)

The Advanced Audio Distribution Profile (A2DP) defines the protocols and procedures that realize distribution of audio content of high-quality in mono or stereo on ACL channels.

Data Structures

- struct bt_a2dp_evt_open_and_start_stream_completed_t

  Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event.
- union bt_a2dp_event_t

  Parameter to an application callback.
- struct bt_a2dp_mgr_t

  A2DP manager.

Macros

- #define bt_a2dp_find_codec(mgr, codec_type) bt_avdtp_find_codec(mgr->avdtp_mgr, codec_type)

  Find a codec.
- #define bt_a2dp_connect(mgr, remote_addr) bt_avdtp_connect(mgr->avdtp_mgr, remote_addr)

  Connect to a remote device.
- #define bt_a2dp_connect_ex(mgr, remote_addr, acl_config) bt_avdtp_connect_ex(mgr->avdtp_mgr, remote_addr, acl_config)

  Connect to a remote device.
- #define bt_a2dp_disconnect(mgr, remote_addr) bt_avdtp_disconnect(mgr->avdtp_mgr, remote_addr)

  Disconnect from a remote device.
- #define bt_a2dp_register_sink(mgr, caps) bt_avdtp_register_sep(mgr->avdtp_mgr, AVDTP_SEP_TYPE_↩ SINK, caps)

  Register a Sink SEP with the local A2DP manager.
- #define bt_a2dp_register_source(mgr, caps) bt_avdtp_register_sep(mgr->avdtp_mgr, AVDTP_SEP_TYP↩ E_SOURCE, caps)

  Register a Source SEP with the local A2DP manager.
- #define bt_a2dp_discover(mgr, remote_addr) bt_avdtp_discover(mgr->avdtp_mgr, remote_addr)

  Discover SEPs on a remote device.
- #define bt_a2dp_get_capabilities(mgr, remote_addr, seid_acp) bt_avdtp_get_capabilities(mgr->avdtp_mgr, remote_addr, seid_acp)

  Get remote SEP capabilities.
- #define bt_a2dp_get_all_capabilities(mgr, remote_addr, seid_acp) bt_avdtp_get_all_capabilities(mgr->avdtp_mgr, remote_addr, seid_acp)

  Get remote SEP capabilities.
- #define bt_a2dp_create_stream(mgr) bt_avdtp_create_stream(mgr->avdtp_mgr)

  Create a stream.
- #define bt_a2dp_destroy_stream(mgr, strm_handle) bt_avdtp_destroy_stream(mgr->avdtp_mgr, strm_↩ handle)

  Destroy a stream.
- #define bt_a2dp_listen(mgr, strm_handle, sep_id) bt_avdtp_listen(mgr->avdtp_mgr, strm_handle, sep_id)

  Listen for incoming connections.
- #define bt_a2dp_cancel_listen(mgr, strm_handle, sep_id) bt_avdtp_cancel_listen(mgr->avdtp_mgr, strm_↩ handle, sep_id)

  Cancel listening for incoming connections.
- #define bt_a2dp_reconfigure_stream(mgr, strm_handle, caps) bt_avdtp_reconfigure_stream(mgr->avdtp_↩ mgr, strm_handle, caps)

Reconfigure stream.

- #define bt_a2dp_get_stream_state(mgr, strm_handle) bt_avdtp_get_stream_state(mgr->avdtp_mgr, strm↩ _handle)

  Get local stream state.
- #define bt_a2dp_get_stream_local_sep_id(mgr, strm_handle) bt_avdtp_get_stream_local_sep_id(mgr->avdtp_mgr, strm_handle)

  Get stream's local SEP ID.
- #define bt_a2dp_get_stream_remote_sep_id(mgr, strm_handle) bt_avdtp_get_stream_remote_sep_id(mgr->avdtp_mgr, strm_handle)

  Get stream's remote SEP ID.
- #define bt_a2dp_get_stream_remote_address(mgr, strm_handle) bt_avdtp_get_stream_remote_↩ address(mgr->avdtp_mgr, strm_handle)

  Get stream's remote BT address.
- #define bt_a2dp_get_stream_codec_type(mgr, strm_handle) bt_avdtp_get_stream_codec_type(mgr->avdtp_mgr, strm_handle)

  Get the type of the codec currently used with the stream.
- #define bt_a2dp_get_stream_codec_config(mgr, strm_handle) bt_avdtp_get_stream_codec_config(mgr->avdtp_mgr, strm_handle)

  Get the configuration of the codec currently used with the stream.
- #define bt_avdtp_get_stream_config(mgr, strm_handle) bt_avdtp_get_stream_config(mgr->avdtp_mgr, strm_handle)

  Get stream's configuration.
- #define bt_a2dp_start_stream(mgr, strm_handle) bt_avdtp_start_stream(mgr->avdtp_mgr, strm_handle)

  Start a stream.
- #define bt_a2dp_close_stream(mgr, strm_handle) bt_avdtp_close_stream(mgr->avdtp_mgr, strm_handle)

  Close a stream.
- #define bt_a2dp_suspend_stream(mgr, strm_handle) bt_avdtp_suspend_stream(mgr->avdtp_mgr, strm_↩ handle)

  Suspend a stream.
- #define bt_a2dp_abort_stream(mgr, strm_handle) bt_avdtp_abort_stream(mgr->avdtp_mgr, strm_handle)

  Suspend a stream.
- #define bt_a2dp_get_hci_connection(mgr, strm_handle) bt_avdtp_get_hci_connection(mgr->avdtp_mgr, strm_handle)

  Get HCI connection for a stream.
- #define bt_a2dp_add_media_rx_buffer(mgr, strm_handle, buffer) bt_avdtp_add_media_rx_buffer(mgr->avdtp_mgr, strm_handle, buffer)

  Add a media packet buffer to a receive queue.
- #define bt_a2dp_remove_media_rx_buffer(mgr, strm_handle, buffer) bt_avdtp_remove_media_rx_↩ buffer(mgr->avdtp_mgr, strm_handle, buffer)

  Remove a media packet buffer from a receive queue.
- #define bt_a2dp_add_media_tx_buffer(mgr, strm_handle, buffer) bt_avdtp_add_media_tx_buffer(mgr->avdtp_mgr, strm_handle, buffer)

  Add a media packet buffer to a send queue.
- #define bt_a2dp_remove_media_tx_buffer(mgr, strm_handle, buffer) bt_avdtp_remove_media_tx_↩ buffer(mgr->avdtp_mgr, strm_handle, buffer)

  Remove a media packet buffer from a send queue.

Typedefs

- typedef void(∗ bt_a2dp_find_server_callback_fp) (bt_uint supported_features, bt_bool found, void ∗param)

  Notify the application of the result of searching for a remote A2DP entity (sourse or sink)
- typedef void(∗ bt_a2dp_mgr_callback_fp) (bt_a2dp_mgr_t ∗mgr, bt_byte evt, bt_a2dp_event_t ∗evt_param, void ∗callback_param)

  A2DP application callback.

Functions

- bt_a2dp_mgr_t ∗ bt_a2dp_get_mgr (void)

  Return a pointer to an instance of the A2DP manager.
- void bt_a2dp_init (void)

  Initialize the A2DP layer.
- bt_bool bt_a2dp_start (bt_a2dp_mgr_t ∗mgr)

  Start the A2DP layer.
- void bt_a2dp_register_callback (bt_a2dp_mgr_t ∗mgr, bt_a2dp_mgr_callback_fp callback, void ∗callback_↩ param)

  Register a A2DP application callback.
- bt_bool bt_a2dp_open_and_start_stream (bt_a2dp_mgr_t ∗mgr, bt_byte strm_handle, bt_bdaddr_↩ t ∗remote_addr, bt_byte seid_int, bt_byte seid_acp, const bt_avdtp_sep_capabilities_t ∗caps)

  Open & start a stream.
- bt_bool bt_a2dp_find_source (bt_bdaddr_t ∗deviceAddress, bt_a2dp_find_server_callback_fp callback, bt↩ _sdp_client_callback_fp client_callback, void ∗callback_param)

  Find source.
- bt_bool bt_a2dp_find_sink (bt_bdaddr_t ∗deviceAddress, bt_a2dp_find_server_callback_fp callback, bt_↩ sdp_client_callback_fp client_callback, void ∗callback_param)

  Find sink.
- void bt_a2dp_register_mpeg_codec (bt_a2dp_mgr_t ∗mgr)

  Register default MPEG codec.
- void bt_a2dp_register_aac_codec (bt_a2dp_mgr_t ∗mgr)

  Register default AAC codec.

Source features

The following is a list of features a source can support. The supported features are specified in the corresponding SDP service record and can be retrieved with bt_a2dp_find_source.

- #define A2DP_SOURCE_FEATURE_PLAYER 1

  Player.
- #define A2DP_SOURCE_FEATURE_MICROPHONE 2

  Mic.
- #define A2DP_SOURCE_FEATURE_TUNER 4

  Tuner.
- #define A2DP_SOURCE_FEATURE_MIXER 8

  Mixer.

Sink features

The following is a list of features a sink can support. The supported features are specified in the corresponding SDP service record and can be retrieved with bt_a2dp_find_sink.

- #define A2DP_SINK_FEATURE_HEADPHONE 1

  Headphone.
- #define A2DP_SINK_FEATURE_SPEAKER 2

  Speaker.
- #define A2DP_SINK_FEATURE_RECORDER 4

  Recorder.
- #define A2DP_SINK_FEATURE_AMPLIFIER 8

  Amplifier.

Events

The following is a list of events A2DP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

- #define A2DP_EVT_CTRL_CHANNEL_CONNECTED AVDTP_EVT_CTRL_CHANNEL_CONNECTED

  This event is generated when a control channel between two AVDTP entities has been established.
- #define A2DP_EVT_CTRL_CHANNEL_DISCONNECTED AVDTP_EVT_CTRL_CHANNEL_DISCONNEC↩ TED

  This event is generated when a control channel between two AVDTP entities has been terminated.
- #define A2DP_EVT_CTRL_CONNECTION_FAILED AVDTP_EVT_CTRL_CONNECTION_FAILED

  This event is generated when a local device failed to create a control channel between two AVDTP entities.
- #define A2DP_EVT_DISCOVER_SEP_COMPLETED AVDTP_EVT_DISCOVER_COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "discover" request.
- #define A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED AVDTP_EVT_GET_SEP_CAPABILITIES↩ _COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.
- #define A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED AVDTP_EVT_SET_STREAM_C↩ ONFIGURATION_COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.
- #define A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED AVDTP_EVT_GET_STREAM_C↩ ONFIGURATION_COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.
- #define A2DP_EVT_RECONFIGURE_STREAM_COMPLETED AVDTP_EVT_STREAM_RECONFIGURE↩ _COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.
- #define A2DP_EVT_OPEN_STREAM_COMPLETED AVDTP_EVT_OPEN_STREAM_COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "open stream" request.
- #define A2DP_EVT_START_STREAM_COMPLETED AVDTP_EVT_START_STREAM_COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "start stream" request.
- #define A2DP_EVT_CLOSE_STREAM_COMPLETED AVDTP_EVT_CLOSE_STREAM_COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "close stream" request.
- #define A2DP_EVT_SUSPEND_STREAM_COMPLETED AVDTP_EVT_SUSPEND_STREAM_COMPLE↩ TED

  This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.
- #define A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED AVDTP_EVT_STREAM_SECURI↩ TY_CONTROL_COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.
- #define A2DP_EVT_ABORT_STREAM_COMPLETED AVDTP_EVT_ABORT_STREAM_COMPLETED

  This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.
- #define A2DP_EVT_SEP_INFO_RECEIVED AVDTP_EVT_SEP_INFO_RECEIVED

  This event is generated for each SEP contained in a positive response to a "discover" request.
- #define A2DP_EVT_SEP_CAPABILITIES_RECEIVED AVDTP_EVT_SEP_CAPABILITIES_RECEIVED

  This event is generated when a local device received a positive response to a "get SEP capabilities" request.

- #define A2DP_EVT_STREAM_CONFIGURATION_RECEIVED AVDTP_EVT_STREAM_CONFIGURATI↩ ON_RECEIVED

  This event is generated when a local device received a positive response to a "get stream configuration" request.
- #define A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED AVDTP_EVT_SET_STREAM_C↩ ONFIGURATION_REQUESTED

  This event is generated when a local device received "set stream configuration" request.
- #define A2DP_EVT_OPEN_STREAM_REQUESTED AVDTP_EVT_OPEN_STREAM_REQUESTED

  This event is generated when a local device received "open stream" request.
- #define A2DP_EVT_START_STREAM_REQUESTED AVDTP_EVT_START_STREAM_REQUESTED

  This event is generated when a local device received "start stream" request.
- #define A2DP_EVT_CLOSE_STREAM_REQUESTED AVDTP_EVT_CLOSE_STREAM_REQUESTED

  This event is generated when a local device received "close stream" request.
- #define A2DP_EVT_SUSPEND_STREAM_REQUESTED AVDTP_EVT_SUSPEND_STREAM_REQUES↩ TED

  This event is generated when a local device received "suspend stream" request.
- #define A2DP_EVT_ABORT_STREAM_REQUESTED AVDTP_EVT_ABORT_STREAM_REQUESTED

  This event is generated when a local device received "abort stream" request.
- #define A2DP_EVT_RECONFIGURE_STREAM_REQUESTED AVDTP_EVT_RECONFIGURE_STREAM↩ _REQUESTED

  This event is generated when a local device received "change stream configuration" request.
- #define A2DP_EVT_MEDIA_PACKET_RECEIVED AVDTP_EVT_MEDIA_PACKET_RECEIVED

  This event is generated when a local device received a media packet.
- #define A2DP_EVT_STREAM_CONFIGURED AVDTP_EVT_STREAM_CONFIGURED

  This event is generated when a local device has successfully configured a stream.
- #define A2DP_EVT_STREAM_RECONFIGURED AVDTP_EVT_STREAM_RECONFIGURED

  This event is generated when a local device has successfully reconfigured a stream.
- #define A2DP_EVT_STREAM_OPENED AVDTP_EVT_STREAM_OPENED

  This event is generated when a local device has successfully opened a stream.
- #define A2DP_EVT_STREAM_STARTED AVDTP_EVT_STREAM_STARTED

  This event is generated when a local device has successfully started a stream.
- #define A2DP_EVT_STREAM_CLOSED AVDTP_EVT_STREAM_CLOSED

  This event is generated when a local device has successfully closed a stream.
- #define A2DP_EVT_STREAM_SUSPENDED AVDTP_EVT_STREAM_SUSPENDED

  This event is generated when a local device has successfully suspended a stream.
- #define A2DP_EVT_STREAM_ABORTED AVDTP_EVT_STREAM_ABORTED

  This event is generated when a local device has successfully aborted a stream.
- #define A2DP_EVT_MEDIA_PACKET_SENT AVDTP_EVT_MEDIA_PACKET_SENT

  This event is generated when a local device sent a media packet.
- #define A2DP_EVT_MEDIA_PACKET_SEND_FAILED AVDTP_EVT_MEDIA_PACKET_SEND_FAILED

  This event is generated when a local device failed to send a media packet.
- #define A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED (AVDTP_EVT_LAST + 1)

  This event is generated when a local device completed "open and start" request.

3.13.1 Detailed Description

The Advanced Audio Distribution Profile (A2DP) defines the protocols and procedures that realize distribution of audio content of high-quality in mono or stereo on ACL channels.

3.13.2 Macro Definition Documentation

3.13.2.1 #define A2DP_EVT_STREAM_ABORTED AVDTP_EVT_STREAM_ABORTED

This event is generated when a local device has successfully aborted a stream.

This event follows the A2DP_EVT_ABORT_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream abortion was initiated by the local device.

3.13.2.2 #define A2DP_EVT_STREAM_CLOSED AVDTP_EVT_STREAM_CLOSED

This event is generated when a local device has successfully closed a stream.

This event follows the A2DP_EVT_CLOSE_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream closing was initiated by the local device.

3.13.2.3 #define A2DP_EVT_STREAM_CONFIGURED AVDTP_EVT_STREAM_CONFIGURED

This event is generated when a local device has successfully configured a stream.

This event follows the A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the upper layer has accepted it. This event is not generated if stream configuration was initiated by the local device.

3.13.2.4 #define A2DP_EVT_STREAM_OPENED AVDTP_EVT_STREAM_OPENED

This event is generated when a local device has successfully opened a stream.

This event follows the A2DP_EVT_OPEN_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream opening was initiated by the local device.

3.13.2.5 #define A2DP_EVT_STREAM_RECONFIGURED AVDTP_EVT_STREAM_RECONFIGURED

This event is generated when a local device has successfully reconfigured a stream.

This event follows the A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream reconfiguration was initiated by the local device.

3.13.2.6 #define A2DP_EVT_STREAM_STARTED AVDTP_EVT_STREAM_STARTED

This event is generated when a local device has successfully started a stream.

This event follows the A2DP_EVT_START_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream starting was initiated by the local device.

3.13.2.7 #define A2DP_EVT_STREAM_SUSPENDED AVDTP_EVT_STREAM_SUSPENDED

This event is generated when a local device has successfully suspended a stream.

This event follows the A2DP_EVT_SUSPEND_STREAM_REQUESTED if the upper layer has accepted it. This event is not generated if stream suspension was initiated by the local device.

3.13.2.8 #define bt_a2dp_abort_stream(*mgr*, *strm_handle*) bt_avdtp_abort_stream(mgr->avdtp_mgr, strm_handle)

Suspend a stream.

This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP_STREAM_STATE_IDLE. As a result of this operation the A2DP_EVT_ABORT_STREAM_↩ COMPLETED event will be generated. This operation cannot be rejected. The `evt_param.abort_stream↩ _requested.err_code` is always == AVDTP_ERROR_SUCCESS.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.13.2.9 #define bt_a2dp_add_media_rx_buffer(*mgr*, *strm_handle*, *buffer*) bt_avdtp_add_media_rx_buffer(mgr->avdtp_mgr, strm_handle, buffer)

Add a media packet buffer to a receive queue.

The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the A2DP consumer that the buffer has not been used and can be, for example, deallocated. This function adds a buffer to the receive queue.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist. The stream can be in any state to call this function.

3.13.2.10 #define bt_a2dp_add_media_tx_buffer(*mgr*, *strm_handle*, *buffer*) bt_avdtp_add_media_tx_buffer(mgr->avdtp_mgr, strm_handle, buffer)

Add a media packet buffer to a send queue.

When the consumer of A2DP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. When the packet has been successfully sent a A2DP_EVT↩ _MEDIA_PACKET_SENT is generated. Otherwise a A2DP_EVT_MEDIA_PACKET_SEND_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as A2DP has removed it from the queue and gave up control over it. As in the case of received buffers, if a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP_EVT_MEDIA_PACKET_SENT event for each buffer and sets the data_len field to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist. The stream can be in any state to call this function.

3.13.2.11 #define bt_a2dp_cancel_listen(*mgr,* *strm_handle,* *sep_id*) **bt_avdtp_cancel_listen**(mgr->avdtp_mgr, strm_handle, sep_id)

Cancel listening for incoming connections.

This function removes a SEP from a list of SEPS which a stream can use for incoming requests.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.
sep_id	Local SEP ID.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.13.2.12 #define bt_a2dp_close_stream(*mgr,* *strm_handle*) **bt_avdtp_close_stream**(mgr->avdtp_mgr, strm_handle)

Close a stream.

This function tries to close a stream by sending a request to the remote party. The stream has to be in AVD↩ TP_STREAM_STATE_OPEN or AVDTP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_CLOSE_STREAM_COMPLETED event will be generated. If the stream has been closed the `evt↩ _param.bt_avdtp_evt_close_stream_completed_t.err_code` == AVDTP_ERROR_SUCCES↩ S. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the `evt_param.↩ bt_avdtp_evt_close_stream_completed_t.err_code` == the error code sent by the remote.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.13.2.13 #define bt_a2dp_connect(*mgr,* *remote_addr*) **bt_avdtp_connect**(mgr->avdtp_mgr, remote_addr)

Connect to a remote device.

This function opens a control channel connection to a remote device specified by the `remote_addr`. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are

generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHAN↩
NEL_CONNECTION_FAILED.

Parameters

mgr	A2DP manager.
remote_addr	The address of a remote device.

Returns

- `TRUE` if connection establishment has been started.
- `FALSE` otherwise.

3.13.2.14 #define bt_a2dp_connect_ex(*mgr,* *remote_addr,* *acl_config*) bt_avdtp_connect_ex(mgr->avdtp_mgr, remote_addr, acl_config)

Connect to a remote device.

This function opens a control channel connection to a remote device specified by the `remote_addr`. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP_EVT_CTRL_CHANNEL_CONNECTED or A2DP_EVT_CTRL_CHAN↵ NEL_CONNECTION_FAILED.

Parameters

mgr	A2DP manager.
remote_addr	The address of a remote device.
acl_config	ACL link configuration. This can be a combination of the following values: • HCI_CONFIG_ENABLE_AUTHENTICATION • HCI_CONFIG_ENABLE_ENCRYPTION • HCI_CONFIG_BECOME_MASTER

Returns

- `TRUE` if connection establishment has been started.
- `FALSE` otherwise.

3.13.2.15 #define bt_a2dp_create_stream(*mgr*) bt_avdtp_create_stream(mgr->avdtp_mgr)

Create a stream.

This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort.

Parameters

mgr	A2DP manager.

Returns

- `Stream` handle if the function succeeds.
- `0` otherwise.

3.13.2.16 #define bt_a2dp_destroy_stream(*mgr, strm_handle*) **bt_avdtp_destroy_stream**(mgr->avdtp_mgr, strm_handle)

Destroy a stream.

This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.13.2.17 #define bt_a2dp_disconnect(*mgr, remote_addr*) **bt_avdtp_disconnect**(mgr->avdtp_mgr, remote_addr)

Disconnect from a remote device.

This function closes a control and transport channels on all streams associated with the remote device specified by the `remote_addr`. As a result of this operation the following events will be generated:

- A2DP_EVT_MEDIA_PACKET_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0
- A2DP_EVT_MEDIA_PACKET_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt_media_packet_t::data_len set to 0
- A2DP_EVT_STREAM_CLOSED: this event is generate if a stream is in "closing" state as a result of a request from the remote device or bt_a2dp_close_stream call before bt_a2dp_disconnect call.
- A2DP_EVT_STREAM_ABORTED: this event is generated if a stream is in "active" state at the time of bt_↩ avdtp_disconnect call.

Parameters

mgr	A2DP manager.
remote_addr	The address of a remote device.

Returns

- `TRUE` if disconnection has been started.
- `FALSE` otherwise. No events will be generated.

3.13.2.18 #define bt_a2dp_discover(*mgr, remote_addr*) **bt_avdtp_discover**(mgr->avdtp_mgr, remote_addr)

Discover SEPs on a remote device.

This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated:

- A2DP_EVT_SEP_INFO_RECEIVED: this event is generated for every SEP received from the remote device. the `evt_param.sep_info_received` contains SEP information.
- A2DP_EVT_DISCOVER_COMPLETED: this event is generated after the last A2DP_EVT_SEP_INFO_R↩ ECEIVED if the remote accepted the request and the `evt_param.discover_completed.err_↩ code` == AVDTP_ERROR_SUCCESS. if the remote rejected the request the `evt_param.discover↩ _completed.err_code` == the error code sent by the remote.

Parameters

mgr	A2DP manager.
remote_addr	The address of a remote device.

Returns

- `TRUE` if discover request has been sent.
- `FALSE` otherwise. No events will be generated.

3.13.2.19 #define bt_a2dp_find_codec(*mgr*, *codec_type*) bt_avdtp_find_codec(mgr->avdtp_mgr, codec_type)

Find a codec.

A2DP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make our implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of A2DP has to register a callback function (one per codec type) for each codec it wishes to support. That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function returns a pointer to a structure that holds a pointer to a codec's callback function.

Parameters

mgr	A2DP manager.
codec_type	Codec type. The codec_type can be one of the following values: • AVDTP_CODEC_TYPE_SBC: SBC • AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files) • AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products) • AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products) • AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails if a callback for a codec type specified in the `codec` parameter
- has not been previously registered with bt_avdtp_register_codec.

3.13.2.20 #define bt_a2dp_get_all_capabilities(*mgr*, *remote_addr*, *seid_acp*) bt_avdtp_get_all_capabilities(mgr->avdtp_mgr, remote_addr, seid_acp)

Get remote SEP capabilities.

This function asks the remote device to send capabilities of a SEP specified by the `seid_acp`. As a result of this operation the following events will be generated:

- A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the `evt_param.sep_capabilities_received` contains SEP capabilities.
- A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT↩ _SEP_CAPABILITIES_RECEIVED if the remote accepted the request the `evt_param.get_sep_↩ capabilities_completed.err_code` == AVDTP_ERROR_SUCCESS. if the remote rejected the

request the `evt_param.get_sep_capabilities_completed.err_code` == the error code sent by the remote.

Parameters

mgr	A2DP manager.
remote_addr	The address of a remote device.
seid_acp	The ID of a remote SEP.

Returns

- `TRUE` if discover request has been sent.
- `FALSE` otherwise. No events will be generated.

3.13.2.21 #define bt_a2dp_get_capabilities(*mgr, remote_addr, seid_acp*) **bt_avdtp_get_capabilities**(mgr->avdtp_mgr, remote_addr, seid_acp)

Get remote SEP capabilities.

This function asks the remote device to send capabilities of a SEP specified by the `seid_acp`. As a result of this operation the following events will be generated:

- A2DP_EVT_SEP_CAPABILITIES_RECEIVED: this event is generated if the remote device accepted the request. the `evt_param.sep_capabilities_received` contains SEP capabilities.
- A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: this event is generated right after A2DP_EVT↩_SEP_CAPABILITIES_RECEIVED if the remote accepted the request the `evt_param.get_sep_↩capabilities_completed.err_code` == AVDTP_ERROR_SUCCESS. if the remote rejected the request the `evt_param.get_sep_capabilities_completed.err_code` == the error code sent by the remote.

Parameters

mgr	A2DP manager.
remote_addr	The address of a remote device.
seid_acp	The ID of a remote SEP.

Returns

- `TRUE` if discover request has been sent.
- `FALSE` otherwise. No events will be generated.

3.13.2.22 #define bt_a2dp_get_hci_connection(*mgr, strm_handle*) **bt_avdtp_get_hci_connection**(mgr->avdtp_mgr, strm_handle)

Get HCI connection for a stream.

This function returns a pointer to a structure that describes an HCI connection a stream is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- `Pointer` to a structure that describes an HCI connection if the function succeeds.
- `NULL` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist or there is no HCI connection between local and remote devices associated with the stream.

Note

This function has not been implemented.

3.13.2.23 #define bt_a2dp_get_stream_codec_config(*mgr,* *strm_handle*) bt_avdtp_get_stream_codec_config(mgr->avdtp_mgr, strm_handle)

Get the configuration of the codec currently used with the stream.

This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MP↩
EG-2,4 AAC codecs:

- SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h)
- MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h)
- MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- The codec's configuration if strm_handle specifies a valid stream and the stream is in one of the following state:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING

\li NULL otherwise.
```

3.13.2.24 #define bt_a2dp_get_stream_codec_type(*mgr,* *strm_handle*) bt_avdtp_get_stream_codec_type(mgr->avdtp_mgr, strm_handle)

Get the type of the codec currently used with the stream.

This function returns the type of the codec currently used with the stream.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- The type of the codec if strm_handle specifies a valid stream and the stream is in one of the following states:

```
      AVDTP_STREAM_STATE_CONFIGURED
      AVDTP_STREAM_STATE_OPEN
      AVDTP_STREAM_STATE_STREAMING

    @arg The result will be one of the following values:

       AVDTP_CODEC_TYPE_SBC:            SBC
       AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)
       AVDTP_CODEC_TYPE_MPEG2_4_AAC:   MPEG-2,4 AAC (used in Apple products)
       AVDTP_CODEC_TYPE_ATRAC:         ATRAC (used in Sony products)
       AVDTP_CODEC_TYPE_NON_A2DP:      Non-A2DP Codec

    @arg 0xFF otherwise.
```

3.13.2.25 #define bt_a2dp_get_stream_local_sep_id(*mgr, strm_handle*) bt_avdtp_get_stream_local_sep_id(mgr->avdtp_mgr, strm_handle)

Get stream's local SEP ID.

This function returns the ID of the local SEP associated with the stream.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- The ID of the local SEP if strm_handle specifies a valid stream.
- 0 otherwise.

3.13.2.26 #define bt_a2dp_get_stream_remote_address(*mgr, strm_handle*) bt_avdtp_get_stream_remote_↩ address(mgr->avdtp_mgr, strm_handle)

Get stream's remote BT address.

This function returns the address of the remote device associated with the stream.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- The address of the remote device if strm_handle specifies a valid stream.
- NULL otherwise.

3.13.2.27 #define bt_a2dp_get_stream_remote_sep_id(*mgr, strm_handle*) bt_avdtp_get_stream_remote_sep_↩ id(mgr->avdtp_mgr, strm_handle)

Get stream's remote SEP ID.

This function returns the ID of the remote SEP associated with the stream.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- The ID of the remote SEP if strm_handle specifies a valid stream.
- 0 otherwise.

3.13.2.28 #define bt_a2dp_get_stream_state(*mgr,* *strm_handle*) **bt_avdtp_get_stream_state**(mgr->avdtp_mgr, strm_handle)

Get local stream state.

This function returns local state of a stream specified by the `strm_handle`. No request is sent to the remote party.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

The state of the stream. The result will be one of the following values:

- AVDTP_STREAM_STATE_IDLE: The stream is idle. This can mean two things. The stream specified by `strm_handle` does not exist or the stream is closed.
- AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS: The stream is opening transport channels.
- AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS: The stream is closing transport channels.
- AVDTP_STREAM_STATE_CONFIGURED: The stream has been configured.
- AVDTP_STREAM_STATE_OPEN: The stream has been opened.
- AVDTP_STREAM_STATE_STREAMING: The stream has been started. Depending on the local SEP type (source or sink) it means that the stream is can send or receive media packets.
- AVDTP_STREAM_STATE_CLOSING: The stream is closing. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP↩ _STREAM_STATE_IDLE state.
- AVDTP_STREAM_STATE_ABORTING: The stream is aborting. This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP↩ _STREAM_STATE_IDLE state.

3.13.2.29 #define bt_a2dp_listen(*mgr,* *strm_handle,* *sep_id*) **bt_avdtp_listen**(mgr->avdtp_mgr, strm_handle, sep_id)

Listen for incoming connections.

This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.
sep_id	Local SEP ID.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.13.2.30 #define bt_a2dp_reconfigure_stream(*mgr,* *strm_handle,* *caps*) bt_avdtp_reconfigure_stream(mgr->avdtp_mgr, strm_handle, caps)

Reconfigure stream.

This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the A2DP_EVT_STREAM_RECONFIGURE_COMPLETED event will be generated. If reconfiguration was a success the `evt_param.stream_reconfigure_completed.err_code` == A↩VDTP_ERROR_SUCCESS. Otherwise the `evt_param.stream_reconfigure_completed.err_code` == the error code sent by the remote.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.
caps	New stream configuration.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.13.2.31 #define bt_a2dp_register_sink(*mgr,* *caps*) bt_avdtp_register_sep(mgr->avdtp_mgr, AVDTP_SEP_TYPE_SINK, caps)

Register a Sink SEP with the local A2DP manager.

This function is used to add a sink SEP to a list of SEPs supported by the local A2DP entity.

Parameters

mgr	A2DP manager.
caps	The capabilities of a SEP.

Returns

- `ID` of a SEP if the function succeeds.
- `FALSE` otherwise.

3.13.2.32 #define bt_a2dp_register_source(*mgr,* *caps*) bt_avdtp_register_sep(mgr->avdtp_mgr, AVDTP_SEP_TYPE_SOURCE, caps)

Register a Source SEP with the local A2DP manager.

This function is used to add a source SEP to a list of SEPs supported by the local A2DP entity.

Parameters

mgr	A2DP manager.
caps	The capabilities of a SEP.

Returns

- `ID` of a SEP if the function succeeds.
- `FALSE` otherwise.

3.13.2.33 #define bt_a2dp_remove_media_rx_buffer(*mgr, strm_handle, buffer*) bt_avdtp_remove_media_rx_↩ buffer(mgr->avdtp_mgr, strm_handle, buffer)

Remove a media packet buffer from a receive queue.

The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP_EVT_MEDIA_PACKET_RECEIVED event for each buffer and sets the data_len to 0. This is to inform the A2DP consumer that the buffer has not been used and can be, for example, deallocated. This function removes a buffer from the receive queue.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist. The stream can be in any state to call this function.

3.13.2.34 #define bt_a2dp_remove_media_tx_buffer(*mgr, strm_handle, buffer*) bt_avdtp_remove_media_tx_↩ buffer(mgr->avdtp_mgr, strm_handle, buffer)

Remove a media packet buffer from a send queue.

When the consumer of A2DP wants to send a packet to a remote device it calls bt_avdtp_add_media_tx_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP_STREAM_STATE_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt_a2dp_remove_media_↩ tx_buffer.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

buffer	Pointer to a structure that holds the buffer and its parameters.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. The function fails only if a stream specified by the `strm_handle` parameter
- does not exist. The stream can be in any state to call this function.

Note

This function has not been implemented.

3.13.2.35 #define bt_a2dp_start_stream(*mgr*, *strm_handle*) **bt_avdtp_start_stream**(mgr->avdtp_mgr, strm_handle)

Start a stream.

This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP_S↩ TREAM_STATE_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the A2DP_EVT_START_STREAM_COMPLE↩ TED event will be generated. If the stream has been open the `evt_param.start_stream_requested.↩ err_code` == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the `evt_param.start_stream_requested.err_code` == the error code sent by the remote.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.13.2.36 #define bt_a2dp_suspend_stream(*mgr*, *strm_handle*) **bt_avdtp_suspend_stream**(mgr->avdtp_mgr, strm_handle)

Suspend a stream.

This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVD↩ TP_STREAM_STATE_STREAMING state. As a result of this operation the A2DP_EVT_SUSPEND_STREAM_↩ COMPLETED event will be generated. If the stream has been suspended the `evt_param.bt_avdtp_evt↩ _suspend_stream_requested_t.err_code` == AVDTP_ERROR_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the `evt_param.bt_avdtp_evt_↩ suspend_stream_requested_t.err_code` == the error code sent by the remote.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- `TRUE` if the function succeeds, i.e. the actual request has been sent to the remote party.
- `FALSE` otherwise. No events will be generated.

3.13.2.37 #define bt_avdtp_get_stream_config(*mgr, strm_handle*) bt_avdtp_get_stream_config(mgr->avdtp_mgr, strm_handle)

Get stream's configuration.

This function returns a pointer to a structure holding current configuration of the stream.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

Returns

- The stream's configuration if strm_handle specifies a valid stream and the stream is in one of the following state:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING

\li NULL otherwise.
```

3.13.3 Typedef Documentation

3.13.3.1 typedef void(∗ bt_a2dp_find_server_callback_fp) (bt_uint supported_features, bt_bool found, void ∗param)

Notify the application of the result of searching for a remote A2DP entity (sourse or sink)

This function is called by the A2DP layer when searching for an A2DP entity on a remote device has completed.

Parameters

supported_↩ features	Features supported by a remote A2DP entity.
found	`TRUE` if an A2DP entity has been found on the remote device. `FALSE` otherwise.
param	pointer to arbitrary data passed to the bt_a2dp_find_source() or bt_a2dp_find_sink function through its `callback_param` parameter.

3.13.3.2 typedef void(∗ bt_a2dp_mgr_callback_fp) (bt_a2dp_mgr_t ∗mgr, bt_byte evt, bt_a2dp_event_t ∗evt_param, void ∗callback_param)

A2DP application callback.

In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call that function whenever a new event has been generated.

Parameters

mgr	A2DP manager.
evt	A2DP event. The event can be one of the following values: • A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected. • A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected. • A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device). • A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs. • A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation. • A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation. • A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation. • A2DP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation. • A2DP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation. • A2DP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation. • A2DP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation. • A2DP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation. • A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation. • A2DP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation. • A2DP_EVT_SEP_INFO_RECEIVED: SEP information received. • A2DP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received. • A2DP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received. • A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration. • A2DP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream. • A2DP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream. • A2DP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream. • A2DP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream. • A2DP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream. • A2DP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream.

callback_param	A pointer to an arbitrary data set by a call to bt_avdtp_register_callback.

3.13.4 Function Documentation

3.13.4.1 bt_bool bt_a2dp_find_sink (bt_bdaddr_t ∗ *deviceAddress*, bt_a2dp_find_server_callback_fp *callback*, bt_sdp_client_callback_fp *client_callback*, void ∗ *callback_param*)

Find sink.

This function looks for a sink on a remote device specified by `deviceAddress` and, if found, returns features supported by the sink.

Parameters

deviceAddress	The address of a remote device.
callback	The callback function that will be called when search has completed.
client_callback	The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The `evt` parameter of the callback can be one of the following values: • SDP_CLIENT_STATE_IDLE • SDP_CLIENT_STATE_CONNECTING • SDP_CLIENT_STATE_DISCONNECTING • SDP_CLIENT_STATE_CONNECTED
callback_param	A pointer to arbitrary data to be passed to the `callback` and `client_callback` callbacks.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.13.4.2 bt_bool bt_a2dp_find_source (bt_bdaddr_t ∗ *deviceAddress*, bt_a2dp_find_server_callback_fp *callback*, bt_sdp_client_callback_fp *client_callback*, void ∗ *callback_param*)

Find source.

This function looks for a source on a remote device specified by `deviceAddress` and, if found, returns features supported by the source.

Parameters

deviceAddress	The address of a remote device.
callback	The callback function that will be called when search has completed.
client_callback	The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The `evt` parameter of the callback can be one of the following values: • SDP_CLIENT_STATE_IDLE • SDP_CLIENT_STATE_CONNECTING • SDP_CLIENT_STATE_DISCONNECTING • SDP_CLIENT_STATE_CONNECTED
callback_param	A pointer to arbitrary data to be passed to the `callback` and `client_callback` callbacks.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.13.4.3 bt_a2dp_mgr_t* bt_a2dp_get_mgr (void)

Return a pointer to an instance of the A2DP manager.

This function returns a pointer to an instance of the A manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all A2DP functions.

3.13.4.4 void bt_a2dp_init (void)

Initialize the A2DP layer.

This function initializes the A2DP layer of the stack. It must be called prior to any other A2DP function can be called.

3.13.4.5 bt_bool bt_a2dp_open_and_start_stream (bt_a2dp_mgr_t * *mgr,* bt_byte *strm_handle,* bt_bdaddr_t * *remote_addr,* bt_byte *seid_int,* bt_byte *seid_acp,* const bt_avdtp_sep_capabilities_t * *caps*)

Open & start a stream.

Opening a stream involves sending 3 requests to a remote device - "set configuration", "open stream" and "start stream". Each event generates its own event which must be handled and acted accordingly by the application. To make the use of API easier dotstack combines all these requests in one request called "open & start stream". dotstack sends necessary requests in a proper sequence, handles responses and generates only one event (A2↩
DP_EVT_OPEN_AND_START_STREAM_COMPLETED) at the end. If any of the individual requests has failed the event's parameter bt_a2dp_event_t::open_and_start_stream_completed is populated with the error code and request id which caused it.

Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

remote_addr	The address of a remote device.
seid_int	Local SEP ID.
seid_acp	Remote SEP ID.
caps	Stream configuration.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. No events will be generated.

3.13.4.6 void bt_a2dp_register_aac_codec (bt_a2dp_mgr_t * *mgr*)

Register default AAC codec.

This function adds AAC codec implemented by dotstack to the list of known codecs. For more information about codecs see description of bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use AAC codec it must call this function when it is initializing.

Note

dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration.

Parameters

mgr	A2DP manager.

3.13.4.7 void bt_a2dp_register_callback (bt_a2dp_mgr_t * *mgr*, bt_a2dp_mgr_callback_fp *callback*, void * *callback_param*)

Register a A2DP application callback.

In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

- A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected.
- A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected.
- A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device).
- A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs.
- A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation.
- A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation.
- A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation.
- A2DP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation.
- A2DP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation.
- A2DP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation.

- A2DP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation.
- A2DP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation.
- A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation.
- A2DP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation.
- A2DP_EVT_SEP_INFO_RECEIVED: SEP information received.
- A2DP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received.
- A2DP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.
- A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration.
- A2DP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream.
- A2DP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream.
- A2DP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream.
- A2DP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream.
- A2DP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream.
- A2DP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream.
- A2DP_EVT_MEDIA_PACKET_RECEIVED: Remote device sent a media packet.
- A2DP_EVT_STREAM_CONFIGURED: A stream has been configured (This event is generated right after A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED if the local devices accepted the request).
- A2DP_EVT_STREAM_RECONFIGURED: A stream has been re-configured (This event is generated right after A2DP_EVT_RECONFIGURE_STREAM_REQUESTED if the local devices accepted the request).
- A2DP_EVT_STREAM_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request).
- A2DP_EVT_STREAM_STARTED: A stream has been started (This event is generated right after A2DP_E↩ VT_START_STREAM_REQUESTED if the local devices accepted the request).
- A2DP_EVT_STREAM_CLOSED: A stream has been close (This event is generated right after A2DP_EVT↩ _CLOSE_STREAM_REQUESTED if the local devices accepted the request).
- A2DP_EVT_STREAM_SUSPENDED: A stream has been suspended (This event is generated right after A2DP_EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request).
- A2DP_EVT_STREAM_ABORTED: A stream has been aborted (This event is generated right after A2DP↩ _EVT_SUSPEND_STREAM_REQUESTED if the local devices accepted the request. It is also generated if connection between devices has been terminated by means other than A2DP signaling, e.g. devices going out of range).
- A2DP_EVT_MEDIA_PACKET_SENT: The local device has successfully sent a media packet to the remote device.
- A2DP_EVT_MEDIA_PACKET_SEND_FAILED: The local device was not able to send a media packet to the remote device.
- A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED This event is generated when a local device completed "open and start" request.

Parameters

mgr	AVDTP manager.
callback	The callback function that will be called when the AVDTP generates an event.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.13.4.8 void bt_a2dp_register_mpeg_codec (bt_a2dp_mgr_t ∗ *mgr*)

Register default MPEG codec.

This function adds MPEG codec implemented by dotstack to the list of known codecs. For more information about codecs see description of bt_avdtp_register_codec. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use MPEG-1,2 codec it must call this function when it is initializing.

Note

dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration.

Parameters

mgr	A2DP manager.

3.13.4.9 bt_bool bt_a2dp_start (bt_a2dp_mgr_t ∗ *mgr*)

Start the A2DP layer.

This function makes the A2DP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required.

Parameters

mgr	AVDTP manager.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.14 Audio/Video Control Transport Protocol (AVCTP)

AVCTP is the transport mechanisms used to exchange messages for controlling Audio and/or Video devices.

Modules

- Configuration

  This module describes parameters used to configure AVCTP layer.

Data Structures

- struct bt_avctp_evt_channel_connected_t

  Parameter to AVCTP_EVT_CHANNEL_CONNECTED event.
- struct bt_avctp_evt_channel_disconnected_t

  Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event.
- struct bt_avctp_evt_connection_failed_t

  Parameter to AVCTP_EVT_CONNECTION_FAILED event.
- struct bt_avctp_evt_command_received_t

  Parameter to AVCTP_EVT_COMMAND_RECEIVED event.
- struct bt_avctp_evt_response_received_t

  Parameter to AVCTP_EVT_RESPONSE_RECEIVED event.
- struct bt_avctp_evt_command_sent_t

  Parameter to AVCTP_EVT_COMMAND_SENT event.
- struct bt_avctp_evt_command_cancelled_t

  Parameter to AVCTP_EVT_COMMAND_CANCELLED event.
- struct bt_avctp_evt_response_sent_t

  Parameter to AVCTP_EVT_RESPONSE_SENT event.
- struct bt_avctp_evt_response_cancelled_t

  Parameter to AVCTP_EVT_RESPONSE_CANCELLED event.
- union bt_avctp_event_t

  Parameter to an application callback.
- struct bt_avctp_message_t

  AVCTP message description.
- struct bt_avctp_transport_t

  AVCTP transport description.
- struct bt_avctp_channel_t

  AVCTP channel description.
- struct bt_avctp_mgr_t

  AVCTP manager.

Typedefs

- typedef void(∗ bt_avctp_mgr_callback_fp) (struct _bt_avctp_mgr_t ∗mgr, bt_byte evt, bt_avctp_event_t ∗evt↩_param, void ∗callback_param)

  AVCTP application callback.

Functions

- bt_avctp_mgr_t ∗ bt_avctp_get_mgr (void)

  Return a pointer to an instance of the AVCTP manager.
- void bt_avctp_init (void)

  Initialize the AVCTP layer.
- void bt_avctp_set_callback (bt_avctp_mgr_t ∗mgr, bt_avctp_mgr_callback_fp callback, void ∗callback_↩ param)

  Register a AVCTP application callback.
- bt_avctp_channel_t ∗ bt_avctp_create_channel (bt_avctp_mgr_t ∗mgr, bt_uint profile_id, bt_int psm, bt_byte l2cap_mode)

  Allocate AVCTP channel.
- bt_avctp_channel_t ∗ bt_avctp_create_outgoing_channel (bt_avctp_mgr_t ∗mgr, bt_uint profile_id, bt_int psm, bt_byte l2cap_mode)

  Allocate AVCTP channel.
- bt_bool bt_avctp_destroy_channel (bt_avctp_channel_t ∗channel)

  Destroy AVCTP channel.
- bt_bool bt_avctp_listen (bt_avctp_channel_t ∗channel)

  Listen for incoming connections.
- void bt_avctp_cancel_listen (bt_avctp_channel_t ∗channel)

  Cancel listening for incoming connections.
- bt_byte bt_avctp_get_channel_state (bt_avctp_channel_t ∗channel)

  Get channel state.
- bt_bdaddr_t ∗ bt_avctp_get_channel_remote_address (bt_avctp_channel_t ∗channel)

  Get channel's remote BT address.
- bt_bool bt_avctp_connect (bt_avctp_channel_t ∗channel, bt_bdaddr_t ∗remote_address, bt_uint acl_config)

  Connect to a remote device.
- bt_bool bt_avctp_disconnect (bt_avctp_channel_t ∗channel)

  Disconnect from a remote device.
- bt_bool bt_avctp_send_command (bt_avctp_channel_t ∗channel, bt_byte ∗data, bt_uint data_len, bt_byte ∗tran_id)

  Send a command message to a remote device.
- bt_bool bt_avctp_send_response (bt_avctp_channel_t ∗channel, bt_byte tran_id, bt_byte ∗data, bt_uint data_len)

  Send a response message to a remote device.
- void bt_avctp_cancel_command (bt_avctp_channel_t ∗channel, bt_byte tran_id)

  Cancel a command message.
- void bt_avctp_cancel_response (bt_avctp_channel_t ∗channel, bt_byte tran_id)

  Cancel a response message.
- bt_hci_conn_state_t ∗ bt_avctp_get_hci_connection (bt_avctp_channel_t ∗channel)

  Get HCI connection for a channel.

3.14.1 Detailed Description

AVCTP is the transport mechanisms used to exchange messages for controlling Audio and/or Video devices.

The actual message contents are defined in the applicable A/V control profiles.

3.14.2 Typedef Documentation

3.14.2.1 typedef void(∗ bt_avctp_mgr_callback_fp) (struct _bt_avctp_mgr_t ∗mgr, bt_byte evt, bt_avctp_event_t ∗evt_param, void ∗callback_param)

AVCTP application callback.

In order to be notified of various events a consumer of the AVCTP layer has to register a callback function (done with bt_avctp_set_callback()). The stack will call that function whenever a new event has been generated.

Parameters

mgr	AVCTP manager.
evt	AVCTP event. The event can be one of the following values: • AVCTP_EVT_CHANNEL_CONNECTED Channel connected. • AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected. • AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device). • AVCTP_EVT_COMMAND_RECEIVED Command received. • AVCTP_EVT_RESPONE_RECEIVED Response received. • AVCTP_EVT_COMMAND_SENT Command sent. • AVCTP_EVT_RESPONSE_SENT Response sent. • AVCTP_EVT_COMMAND_CANCELLED Command canceled. • AVCTP_EVT_RESPONSE_CANCELLED Response canceled.
evt_param	Event parameter. Which member of the bt_avctp_event_t union is valid depends on the event: • AVCTP_EVT_CHANNEL_CONNECTED `bt_avctp_evt_channel_connected_t` channel_connected • AVCTP_EVT_CHANNEL_DISCONNECTED `bt_avctp_evt_channel_disconnected_t` channel_disconnected • AVCTP_EVT_CONNECTION_FAILED `bt_avctp_evt_connection_failed_t` connection_failed • AVCTP_EVT_COMMAND_RECEIVED `bt_avctp_evt_command_received_t` command_received • AVCTP_EVT_RESPONE_RECEIVED `bt_avctp_evt_response_received_t` response_received • AVCTP_EVT_COMMAND_SENT `bt_avctp_evt_command_sent_t` command_sent • AVCTP_EVT_RESPONSE_SENT `bt_avctp_evt_response_sent_t` response_sent • AVCTP_EVT_COMMAND_CANCELLED `bt_avctp_evt_command_cancelled_t` command_cancelled • AVCTP_EVT_RESPONSE_CANCELLED `bt_avctp_evt_response_cancelled_t` response_cancelled
callback_param	A pointer to an arbitrary data set by a call to bt_avctp_set_callback.

3.14.3 Function Documentation

3.14.3.1 void bt_avctp_cancel_command (bt_avctp_channel_t ∗ *channel,* bt_byte *tran_id*)

Cancel a command message.

If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function.

Parameters

channel	AVCTP channel.
tran_id	Transaction Id. This value is obtained by calling bt_avctp_send_command.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. No events will be generated.

3.14.3.2 void bt_avctp_cancel_listen (bt_avctp_channel_t ∗ *channel*)

Cancel listening for incoming connections.

This function stops listening for incoming connections on the specified channel.

Parameters

channel	AVCTP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.14.3.3 void bt_avctp_cancel_response (bt_avctp_channel_t ∗ *channel,* bt_byte *tran_id*)

Cancel a response message.

If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function.

Parameters

channel	AVCTP channel.
tran_id	Transaction Id. This value is obtained by calling bt_avctp_send_command.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. No events will be generated.

3.14.3.4 bt_bool bt_avctp_connect (bt_avctp_channel_t ∗ *channel,* bt_bdaddr_t ∗ *remote_address,* bt_uint *acl_config*)

Connect to a remote device.

This function establishes a connection to a remote device specified by the `remote_address`. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVCTP callback. The events generated will either be AVCTP_EVT_CHANNEL_CONNECTED or AVCTP_EVT_CONNE←CTION_FAILED.

Parameters

channel	AVCTP channel.
remote_address	The address of a remote device.
acl_config	ACL link configuration. This can be a combination of the following values: • HCI_CONFIG_ENABLE_AUTHENTICATION • HCI_CONFIG_ENABLE_ENCRYPTION • HCI_CONFIG_BECOME_MASTER

Returns

- `TRUE` if connection establishment has been started.
- `FALSE` otherwise.

3.14.3.5 bt_avctp_channel_t* bt_avctp_create_channel (bt_avctp_mgr_t * *mgr,* bt_uint *profile_id,* bt_int *psm,* bt_byte *l2cap_mode*)

Allocate AVCTP channel.

This function allocates a new incoming AVCTP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one channel for each combination of `profile_id` and `psm`.

Parameters

mgr	AVCTP manager.
profile_id	Profile Id
psm	The PSM on which the underlying L2CAP channel will listen and accept incoming connections.
l2cap_mode	Underlying L2CAP channel mode. This currently can only be CMODE_BASIC.

Returns

- A pointer to the new AVCTP channel if the function succeeds.
- `NULL` otherwise.

3.14.3.6 bt_avctp_channel_t* bt_avctp_create_outgoing_channel (bt_avctp_mgr_t * *mgr,* bt_uint *profile_id,* bt_int *psm,* bt_byte *l2cap_mode*)

Allocate AVCTP channel.

This function allocates a new outgoing AVCTP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple channels with the same `profile_id` and `psm`.

Parameters

mgr	AVCTP manager.
profile_id	Profile Id
psm	The PSM on which the underlying L2CAP channel will listen and accept incoming connections.

l2cap_mode	Underlying L2CAP channel mode. This currently can only be CMODE_BASIC.

Returns

- A pointer to the new AVCTP channel if the function succeeds.
- `NULL` otherwise.

3.14.3.7 bt_bool bt_avctp_destroy_channel (bt_avctp_channel_t ∗ *channel*)

Destroy AVCTP channel.

This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called.

Parameters

channel	AVCTP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.14.3.8 bt_bool bt_avctp_disconnect (bt_avctp_channel_t ∗ *channel*)

Disconnect from a remote device.

This function closes a connection to a remote device.

Parameters

channel	AVCTP channel.

Returns

- `TRUE` if disconnection has been started.
- `FALSE` otherwise. No events will be generated.

3.14.3.9 bt_bdaddr_t∗ bt_avctp_get_channel_remote_address (bt_avctp_channel_t ∗ *channel*)

Get channel's remote BT address.

This function returns the address of the remote device associated with the channel.

Parameters

channel	AVCTP channel.

Returns

- The address of the remote device if channel is connected.
- NULL otherwise.

3.14.3.10 bt_byte bt_avctp_get_channel_state (bt_avctp_channel_t ∗ *channel*)

Get channel state.

This function return current state of the specified channel

Parameters

channel	AVCTP channel.

Returns

- AVCTP_CHANNEL_STATE_FREE
- AVCTP_CHANNEL_STATE_IDLE
- AVCTP_CHANNEL_STATE_CONNECTING
- AVCTP_CHANNEL_STATE_CONNECTED
- AVCTP_CHANNEL_STATE_DISCONNECTING

3.14.3.11 bt_hci_conn_state_t* bt_avctp_get_hci_connection (bt_avctp_channel_t * *channel*)

Get HCI connection for a channel.

This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect.

Parameters

channel	AVCTP channel.

Returns

- `Pointer` to a structure that describes an HCI connection if the function succeeds.
- `NULL` otherwise. The function fails only if a channel specified by the `channel` parameter
- does not exist or there is no HCI connection between local and remote devices associated with the channel.

3.14.3.12 bt_avctp_mgr_t* bt_avctp_get_mgr (void)

Return a pointer to an instance of the AVCTP manager.

This function returns a pointer to an instance of the AVCTP manager. There is only one instance of the manager allocated by the stack.

3.14.3.13 void bt_avctp_init (void)

Initialize the AVCTP layer.

This function initializes the AVCTP layer of the stack. It must be called prior to any other AVCTP function can be called.

3.14.3.14 bt_bool bt_avctp_listen (bt_avctp_channel_t * *channel*)

Listen for incoming connections.

This function enables incoming connections on the specified AVCTP channel.

Parameters

channel	AVCTP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.14.3.15 bt_bool bt_avctp_send_command (bt_avctp_channel_t ∗ *channel,* bt_byte ∗ *data,* bt_uint *data_len,* bt_byte ∗ *tran_id*)

Send a command message to a remote device.

This function sends a command message to a remote device.

Parameters

channel	AVCTP channel.
data	Message body.
data_len	Message body length.
tran_id	Pointer to a bt_byte where AVRCP will write transaction id assigned to the message.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. No events will be generated.

3.14.3.16 bt_bool bt_avctp_send_response (bt_avctp_channel_t ∗ *channel,* bt_byte *tran_id,* bt_byte ∗ *data,* bt_uint *data_len*)

Send a response message to a remote device.

This function sends a response message to a remote device.

Parameters

channel	AVCTP channel.
tran_id	Transaction Id. This value is obtained by calling bt_avctp_send_command.
data	Message body.
data_len	Message body length.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. No events will be generated.

3.14.3.17 void bt_avctp_set_callback (bt_avctp_mgr_t ∗ *mgr,* bt_avctp_mgr_callback_fp *callback,* void ∗ *callback_param*)

Register a AVCTP application callback.

In order to be notified of various events a consumer of the AVCTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

- AVCTP_EVT_CHANNEL_CONNECTED Channel connected.

- AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected.
- AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device).
- AVCTP_EVT_COMMAND_RECEIVED Command received.
- AVCTP_EVT_RESPONE_RECEIVED Response received.
- AVCTP_EVT_COMMAND_SENT Command sent.
- AVCTP_EVT_RESPONSE_SENT Response sent.
- AVCTP_EVT_COMMAND_CANCELLED Command canceled.
- AVCTP_EVT_RESPONSE_CANCELLED Response canceled.

Parameters

mgr	AVCTP manager.
callback	The callback function that will be called when the AVCTP generates an event.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.15 Configuration

This module describes parameters used to configure AVCTP layer.

Macros

- #define AVCTP_MAX_CHANNELS

  Maximum number of AVCTP channels.
- #define AVCTP_MAX_TRANSPORT_CHANNELS

  Maximum number of AVCTP transports.
- #define AVCTP_ALLOCATE_BUFFERS_VARS()

  Maximum size of received message.

3.15.1 Detailed Description

This module describes parameters used to configure AVCTP layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI, L2CAP and SDP must always be present

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS            ...
#define HCI_MAX_DATA_BUFFERS           ...
#define HCI_MAX_HCI_CONNECTIONS        ...
#define HCI_RX_BUFFER_LEN              ...
#define HCI_TX_BUFFER_LEN              ...
#define HCI_L2CAP_BUFFER_LEN           ...
#define HCI_MAX_CMD_PARAM_LEN          ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS          ...
#define L2CAP_MAX_FRAME_BUFFERS        ...
#define L2CAP_MAX_PSMS                 ...
#define L2CAP_MAX_CHANNELS             ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN     ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN   ...

// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure AVRCP,
// the following values must be defined:

#define BT_INCLUDE_AVCTP               // tells dotstack to compile in AVCTP support
#define AVCTP_MAX_CHANNELS             ...
#define AVCTP_MAX_TRANSPORT_CHANNELS   ...
#define AVCTP_MAX_RX_MESSAGE_LEN       ...
#define AVCTP_MAX_MESSAGE_BUFFERS      ...

#include "cdbt/bt/bt_oem_config.h"
```

3.15.2 Macro Definition Documentation

3.15.2.1 #define AVCTP_ALLOCATE_BUFFERS_VARS( )

Value:

```
bt_avctp_channel_t                 _avctp_channels[AVCTP_MAX_CHANNELS];    \
    const bt_byte                      _avctp_max_channels = AVCTP_MAX_CHANNELS; \
    bt_avctp_transport_t               _avctp_transports[
      AVCTP_MAX_TRANSPORT_CHANNELS];  \
    const bt_byte                      _avctp_max_transports =
      AVCTP_MAX_TRANSPORT_CHANNELS; \
    const bt_uint                      _avctp_max_rx_message_len = AVCTP_MAX_RX_MESSAGE_LEN; \
    bt_byte                            _avctp_rx_buffers[(AVCTP_MAX_RX_MESSAGE_LEN) * (
      AVCTP_MAX_TRANSPORT_CHANNELS)]; \
    bt_buffer_header_t                 _avctp_message_buffer_headers[(AVCTP_MAX_MESSAGE_BUFFERS) * (
      AVCTP_MAX_TRANSPORT_CHANNELS)]; \
    bt_avctp_message_t                 _avctp_message_buffers[(AVCTP_MAX_MESSAGE_BUFFERS) * (
      AVCTP_MAX_TRANSPORT_CHANNELS)]; \
    const bt_byte                      _avctp_max_message_buffers = AVCTP_MAX_MESSAGE_BUFFERS; \
    \
    AVCTP_ALLOCATE_BUFFERS_RAM_SIZE_VAR \
```

Maximum size of received message.

This parameter defines the maximum size of a fragmented message (command or response) a local device can accept from a remote device. If fragmented message are not expected this parameter can be set to 1. Maximum number of message buffers

This parameter defines the maximum number of buffer available for sending message.

3.15.2.2 #define AVCTP_MAX_CHANNELS

Maximum number of AVCTP channels.

This parameter defines the maximum number of channels a local device can have with remote devices.

3.15.2.3 #define AVCTP_MAX_TRANSPORT_CHANNELS

Maximum number of AVCTP transports.

This parameter defines the maximum number of transports a local device can have with remote devices. This value should not exceed AVCTP_MAX_CHANNELS.

3.16 Audio/Video Remote Control Profile (AVRCP)

The Audio/Video Remote Control Profile (AVRCP) defines the features and procedures required in order to ensure interoperability between Bluetooth devices with audio/video control functions in the Audio/Video distribution scenarios.

Modules

- Configuration

  This module describes parameters used to configure AVRCP layer.

Data Structures

- struct bt_avrcp_evt_search_completed_t

  Parameter to AVRCP_EVT_SEARCH_COMPLETED event.
- struct bt_avrcp_evt_channel_connected_t

  Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event.
- struct bt_avrcp_evt_channel_disconnected_t

  Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event.
- struct bt_avrcp_evt_connection_failed_t

  Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event.
- struct bt_avrcp_evt_panel_response_received_t

  Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event.
- struct bt_avrcp_evt_panel_command_received_t

  Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event.
- union bt_avrcp_event_t

  Parameter to an application callback.
- struct bt_avrcp_channel_t

  AVRCP channel description.
- struct bt_avrcp_mgr_t

  AVRCP manager.
- struct bt_av_response_t

  AV/C response header.
- struct bt_av_element_id_t

  Media element UID.
- struct bt_av_capability_company_id_t

  Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event.
- struct bt_av_capability_event_id_t

  Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event.
- struct bt_av_player_settings_t

  Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event.
- struct bt_av_player_setting_values_t

  Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event.
- struct bt_av_player_setting_current_values_t

  Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event.
- struct bt_av_player_settings_text_t

  Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event.
- struct bt_av_player_setting_values_text_t

  Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event.
- struct bt_av_element_attribute_t

Media element attribute.
- struct bt_av_element_attributes_t

  Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event.
- struct bt_av_play_status_t

  Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event.
- struct bt_av_battery_status_of_ct_t

  Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event.
- struct bt_av_displayable_character_set_t

  Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event.
- struct bt_av_notification_playback_status_changed_t

  Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event.
- struct bt_av_notification_track_changed_t

  Parameter to AVRCP_EVT_TRACK_CHANGED event.
- struct bt_av_notification_playback_pos_changed_t

  Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event.
- struct bt_av_notification_battery_status_t

  Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event.
- struct bt_av_notification_system_status_changed_t

  Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event.
- struct bt_av_notification_addressed_player_changed_t

  Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event.
- struct bt_av_notification_uids_changed_t

  Parameter to AVRCP_EVT_UIDS_CHANGED event.
- struct bt_av_notification_volume_changed_t

  Parameter to AVRCP_EVT_VOLUME_CHANGED event.
- struct bt_av_notification_t

  Parameter to the following events:
- struct bt_av_set_absolute_volume_t

  Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event.
- struct bt_av_set_addressed_player_t

  Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event.
- struct bt_av_play_item_t

  Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event.
- struct bt_av_add_to_now_playing_t

  Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event.
- struct bt_av_get_element_attributes_t

  Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event.
- struct bt_av_register_notification_t

  Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event.

Typedefs

- typedef void(∗ bt_avrcp_mgr_callback_fp) (bt_avrcp_mgr_t ∗mgr, bt_byte evt, bt_avrcp_event_t ∗evt_param, void ∗callback_param)

  AVRCP application callback.
- typedef void(∗ bt_avrcp_find_callback_fp) (bt_bool found, void ∗param)

  Find Controller/Target callback.

Functions

- bt_avrcp_mgr_t ∗ bt_avrcp_get_mgr (void)

  Return a pointer to an instance of the AVRCP manager.
- void bt_avrcp_init_target (bt_ulong company_id, bt_uint supported_events)

  Initialize AVRCP to be used in target mode.
- void bt_avrcp_init_controller (void)

  Initialize AVRCP to be used in controller mode.
- void bt_avrcp_start (bt_avrcp_mgr_t ∗mgr, bt_avrcp_mgr_callback_fp callback, void ∗callback_param)

  Start the AVRCP layer.
- bt_avrcp_channel_t ∗ bt_avrcp_create_channel (bt_avrcp_mgr_t ∗mgr, bt_bool create_browsing_channel)

  Allocate AVRCP channel.
- bt_avrcp_channel_t ∗ bt_avrcp_create_outgoing_channel (bt_avrcp_mgr_t ∗mgr, bt_bool create_browsing↩_channel)

  Allocate AVRCP channel.
- bt_bool bt_avrcp_destroy_channel (bt_avrcp_channel_t ∗channel)

  Destroy AVRCP channel.
- bt_bool bt_avrcp_listen (bt_avrcp_channel_t ∗channel)

  Listen for incoming connections.
- void bt_avrcp_cancel_listen (bt_avrcp_channel_t ∗channel)

  Cancel listening for incoming connections.
- bt_byte bt_avrcp_get_control_channel_state (bt_avrcp_channel_t ∗channel)

  Get AVCTP control channel state.
- bt_byte bt_avrcp_get_browsing_channel_state (bt_avrcp_channel_t ∗channel)

  Get AVCTP browsing channel state.
- bt_bdaddr_t ∗ bt_avrcp_get_channel_remote_address (bt_avrcp_channel_t ∗channel)

  Get channel's remote BT address.
- bt_bool bt_avrcp_connect (bt_avrcp_channel_t ∗channel, bt_bdaddr_t ∗remote_address)

  Connect to a remote device.
- bt_bool bt_avrcp_disconnect (bt_avrcp_channel_t ∗channel)

  Disconnect from a remote device.
- bt_hci_conn_state_p ∗ bt_avrcp_get_hci_connection (bt_avrcp_channel_t ∗channel)

  Get HCI connection for a channel.
- bt_bool bt_avrcp_send_cmd (bt_avrcp_channel_t ∗channel, bt_av_command_t ∗command)

  Send AVRCP command.
- bt_bool bt_avrcp_add_to_now_playing (bt_avrcp_channel_t ∗channel, bt_byte scope, bt_av_element_id_↩t ∗element_id, bt_uint counter)

  Add to "now playing" list.
- bt_bool bt_avrcp_get_company_id_list (bt_avrcp_channel_t ∗channel)

  Get Company ID list.
- bt_bool bt_avrcp_get_supported_event_id_list (bt_avrcp_channel_t ∗channel)

  Get supported events.
- bt_bool bt_avrcp_get_current_player_application_setting_value (bt_avrcp_channel_t ∗channel, bt_av_↩player_setting_current_values_t ∗response_buffer)

  Get current player setting values.
- bt_bool bt_avrcp_get_element_attributes (bt_avrcp_channel_t ∗channel, bt_av_element_id_t ∗element_id, bt_uint attr_mask)

  Get media element attributes.
- bt_bool bt_avrcp_get_play_status (bt_avrcp_channel_t ∗channel, bt_uint repeat_interval)

  Get playback status.
- bt_bool bt_avrcp_get_player_application_setting_attr_text (bt_avrcp_channel_t ∗channel, bt_av_player_↩settings_text_t ∗response_buffer)

Get player setting attribute text.

- bt_bool bt_avrcp_get_player_application_setting_value_text (bt_avrcp_channel_t ∗channel, bt_byte attr_id, bt_av_player_setting_values_text_t ∗response_buffer)

    Get player setting value text.
- bt_bool bt_avrcp_inform_battery_status (bt_avrcp_channel_t ∗channel, bt_byte status)

    Inform controller's battery status.
- bt_bool bt_avrcp_inform_displayable_character_set (bt_avrcp_channel_t ∗channel, bt_uint ∗charset_list, bt_byte charset_count)

    Inform displayable character set.
- bt_bool bt_avrcp_list_player_application_setting_attributes (bt_avrcp_channel_t ∗channel, bt_av_player_←settings_t ∗response_buffer)

    Get supported player setting attributes.
- bt_bool bt_avrcp_list_player_application_setting_values (bt_avrcp_channel_t ∗channel, bt_byte attr_id, bt←_av_player_setting_values_t ∗response_buffer)

    Get player setting attribute values.
- bt_bool bt_avrcp_play_item (bt_avrcp_channel_t ∗channel, bt_byte scope, bt_av_element_id_t ∗element_id, bt_uint counter)

    Play media item.
- bt_bool bt_avrcp_register_notification (bt_avrcp_channel_t ∗channel, bt_byte event_id, bt_ulong playback←_interval)

    Register notification.
- bt_bool bt_avrcp_register_notifications (bt_avrcp_channel_t ∗channel, bt_uint event_mask)

    Register notifications.
- bt_bool bt_avrcp_set_absolute_volume (bt_avrcp_channel_t ∗channel, bt_byte volume)

    Set absolute volume.
- bt_bool bt_avrcp_set_addressed_player (bt_avrcp_channel_t ∗channel, bt_uint player_id)

    Set addressed player.
- bt_bool bt_avrcp_set_player_application_setting_value (bt_avrcp_channel_t ∗channel, bt_byte ∗attr_id_list, bt_byte ∗attr_value_list, bt_byte attr_id_count)

    Set player setting attribute values.
- bt_bool bt_avrcp_find_targets (bt_byte search_length)

    Find Targets.
- bt_bool bt_avrcp_cancel_find (void)

    Cancel finding Targets.
- bt_bool bt_avrcp_find_target (bt_bdaddr_t ∗deviceAddress, bt_avrcp_find_callback_fp callback, bt_sdp_←client_callback_fp client_callback, void ∗callback_param)

    Find Target.
- bt_bool bt_avrcp_find_controller (bt_bdaddr_t ∗deviceAddress, bt_avrcp_find_callback_fp callback, bt_sdp←_client_callback_fp client_callback, void ∗callback_param)

    Find Controller.
- void bt_avrcp_tg_set_play_status (bt_ulong song_length, bt_ulong song_position, bt_byte play_status)

    Set playback status.
- void bt_avrcp_tg_set_current_track (bt_av_element_id_t ∗track_id, bt_ulong song_length, bt_ulong song_←position)

    Set current track.
- void bt_avrcp_tg_set_absolute_volume (bt_byte volume)

    Set absolute volume.
- void bt_avrcp_tg_set_battery_status (bt_byte status)

    Set battery status.
- void bt_avrcp_tg_set_system_status (bt_byte status)

    Set system status.

- bt_bool bt_avrcp_tg_send_element_attributes (bt_avrcp_channel_t ∗channel, bt_byte tran_id, bt_av_↩ element_attribute_t ∗attrs, bt_byte attr_count)

  Send media element attributes.
- bt_bool bt_avrcp_get_unit_info (bt_avrcp_channel_t ∗channel)

  Get unit info.
- bt_bool bt_avrcp_get_subuint_info (bt_avrcp_channel_t ∗channel)

  Get subunit info.
- bt_bool bt_avrcp_send_simple_panel_cmd (bt_avrcp_channel_t ∗channel, bt_byte ctype, bt_byte opid, bt↩ _byte button_state)

  Send AV/C Panel Subunit PASS THROUGH command.
- bt_bool bt_avrcp_send_press_panel_control (bt_avrcp_channel_t ∗channel, bt_byte opid)

  Send AV/C Panel Subunit "pressed" PASS THROUGH command.
- bt_bool bt_avrcp_send_release_panel_control (bt_avrcp_channel_t ∗channel, bt_byte opid)

  Send AV/C Panel Subunit "released" PASS THROUGH command.
- bt_bool bt_avrcp_send_panel_control (bt_avrcp_channel_t ∗channel, bt_byte opid, bt_byte button_state)

  Send AV/C Panel Subunit "control" PASS THROUGH command.
- bt_bool bt_avrcp_send_button_click (bt_avrcp_channel_t ∗channel, bt_byte button_id)

  Send AV/C Panel Subunit "click" PASS THROUGH command.

Events

The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

- #define **AVCTP_EVT_NOTHING** 0
- #define AVCTP_EVT_CHANNEL_CONNECTED 1

  This event is generated when a channel between two AVCTP entities has been established.
- #define AVCTP_EVT_CHANNEL_DISCONNECTED 2

  This event is generated when a channel between two AVCTP entities has been terminated.
- #define AVCTP_EVT_CONNECTION_FAILED 3

  This event is generated when a local device failed to create a channel between two AVCTP entities.
- #define AVCTP_EVT_COMMAND_RECEIVED 50

  This event is generated when a local device received a command.
- #define AVCTP_EVT_RESPONE_RECEIVED 51

  This event is generated when a local device received a response.
- #define AVCTP_EVT_COMMAND_SENT 52

  This event is generated when a local device finished sending a command.
- #define AVCTP_EVT_RESPONSE_SENT 53

  This event is generated when a local device finished sending a response.
- #define AVCTP_EVT_COMMAND_CANCELLED 54

  This event is generated when a command has been canceled.
- #define AVCTP_EVT_RESPONSE_CANCELLED 55

  This event is generated when a response has been canceled.

Events

The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

- #define **AVRCP_EVT_NOTHING** 0
- #define AVRCP_EVT_CONTROL_CHANNEL_CONNECTED 1

This event is generated when a control channel between two AVRCP entities has been established.

- #define AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED 2

  This event is generated when a control channel between two AVRCP entities has been terminated.
- #define AVRCP_EVT_CONTROL_CONNECTION_FAILED 3

  This event is generated when a local device failed to create a control channel between two AVRCP entities.
- #define AVRCP_EVT_BROWSING_CHANNEL_CONNECTED 4

  This event is generated when a browsing channel between two AVRCP entities has been established.
- #define AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED 5

  This event is generated when a browsing channel between two AVRCP entities has been terminated.
- #define AVRCP_EVT_BROWSING_CONNECTION_FAILED 6

  This event is generated when a local device failed to create a browsing channel between two AVRCP entities.
- #define AVRCP_EVT_SEARCH_COMPLETED 7

  This event is generated when a local device completed searching for nearby targets.
- #define AVRCP_EVT_PANEL_RESPONSE_RECEIVED 50

  This event is generated when a local device received a response to a PASS THROUGH command.
- #define AVRCP_EVT_COMPANY_ID_LIST_RECEIVED 51

  This event is generated when a local device received a response to a "get company id" request.
- #define AVRCP_EVT_EVENT_ID_LIST_RECEIVED 52

  This event is generated when a local device received a response to a "get supported events" request.
- #define AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED 53

  This event is generated when a local device received a response to a "get supported player setting attributes" request.
- #define AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED 54

  This event is generated when a local device received a response to a "get player setting attribute values" request.
- #define AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED 55

  This event is generated when a local device received a response to a "get current player setting attribute values" request.
- #define AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED 56

  This event is generated when a local device received a response to a "set player setting attribute values" request.
- #define AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED 57

  This event is generated when a local device received a response to a "get player setting attributes displayable text" request.
- #define AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED 58

  This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.
- #define AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED 59

  This event is generated when a local device received a response to a "inform displayable character set" request.
- #define AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED 60

  This event is generated when a local device received a response to a "inform battery status" request.
- #define AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED 61

  This event is generated when a local device received a response to a "get media element attributes" request.
- #define AVRCP_EVT_GET_PLAY_STATUS_RECEIVED 62

  This event is generated when a local device received a response to a "get play status" request.
- #define AVRCP_EVT_PLAYBACK_STATUS_CHANGED 63

  This event is generated when a local device received a "play status changed" notification.
- #define AVRCP_EVT_TRACK_CHANGED 64

  This event is generated when a local device received a "track changed changed" notification.
- #define AVRCP_EVT_TRACK_REACHED_END 65

  This event is generated when a local device received a "track reached end" notification.
- #define AVRCP_EVT_TRACK_REACHED_START 66

  This event is generated when a local device received a "track reached start" notification.
- #define AVRCP_EVT_PLAYBACK_POS_CHANGED 67

This event is generated when a local device received a "playback position changed" notification.

- #define AVRCP_EVT_BATT_STATUS_CHANGED 68

  This event is generated when a local device received a "battery status changed" notification.
- #define AVRCP_EVT_SYSTEM_STATUS_CHANGED 69

  This event is generated when a local device received a "system status changed" notification.
- #define AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED 70

  This event is generated when a local device received a "player application setting changed" notification.
- #define AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED 71

  This event is generated when a local device received a "now playing content changed" notification.
- #define AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED 72

  This event is generated when a local device received a "available players changed" notification.
- #define AVRCP_EVT_ADDRESSED_PLAYER_CHANGED 73

  This event is generated when a local device received a "addressed player changed" notification.
- #define AVRCP_EVT_UIDS_CHANGED 74

  This event is generated when a local device received a "UIDs changed" notification.
- #define AVRCP_EVT_VOLUME_CHANGED 75

  This event is generated when a local device received a "volume changed" notification.
- #define AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED 76

  This event is generated when a local device received a response to a "set absolute volume" request.
- #define AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED 77

  This event is generated when a local device received a response to a "set addressed player" request.
- #define AVRCP_EVT_PLAY_ITEM_COMPLETED 78

  This event is generated when a local device received a response to a "play item" request.
- #define AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED 79

  This event is generated when a local device received a response to a "add to now playing" request.
- #define AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED 80

  This event is generated when a local device received a response to a "register notification" request.
- #define AVRCP_EVT_PANEL_COMMAND_RECEIVED 81

  This event is generated when a local device received a PASS THROUGH command.
- #define AVRCP_EVT_SET_ABSOLUTE_VOLUME_REQUESTED 82

  This event is generated when a local device received a "set absolute volume" request.
- #define AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED 83

  This event is generated when a local device received a "battery status of controller" command.
- #define AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED 84

  This event is generated when a local device received a "displayable chracter set command" request.
- #define AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED 85

  This event is generated when a local device received a "get element attributes" request.
- #define AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED 86

  This event is generated when a local device received a "register notification" request.

Command types

- #define **AVC_CTYPE_CONTROL** 0
- #define **AVC_CTYPE_STATUS** 1
- #define **AVC_CTYPE_SPECIFIC_IQUIRY** 2
- #define **AVC_CTYPE_NOTIFY** 3
- #define **AVC_CTYPE_GENERAL_INQUORY** 4

Response types

- #define **AVC_RESPONSE_NOT_IMPLEMENTED** 0x8
- #define **AVC_RESPONSE_ACCEPTED** 0x9
- #define **AVC_RESPONSE_REJECTED** 0xA
- #define **AVC_RESPONSE_IN_TRANSITION** 0xB
- #define **AVC_RESPONSE_STABLE** 0xC
- #define **AVC_RESPONSE_IMPLEMENTED** 0xC
- #define **AVC_RESPONSE_CHANGED** 0xD
- #define **AVC_RESPONSE_INTERIM** 0xF
- #define **AVC_RESPONSE_TIMEOUT** 0xF0

Subunit types

- #define **AVC_SUBUNIT_TYPE_MONITOR** 0x00
- #define **AVC_SUBUNIT_TYPE_AUDIO** 0x01
- #define **AVC_SUBUNIT_TYPE_PRINTER** 0x02
- #define **AVC_SUBUNIT_TYPE_DISC** 0x03
- #define **AVC_SUBUNIT_TYPE_TAPE_RECORDER_PLAYER** 0x04
- #define **AVC_SUBUNIT_TYPE_TUNER** 0x05
- #define **AVC_SUBUNIT_TYPE_CA** 0x06
- #define **AVC_SUBUNIT_TYPE_CAMERA** 0x07
- #define **AVC_SUBUNIT_TYPE_PANEL** 0x09
- #define **AVC_SUBUNIT_TYPE_BULLETIN_BOARD** 0x0A
- #define **AVC_SUBUNIT_TYPE_CAMERA_STORAGE** 0x0B
- #define **AVC_SUBUNIT_TYPE_VENDOR_UNIQUE** 0x1C
- #define **AVC_SUBUNIT_TYPE_EXTENDED_TO_NEXT_BYTE** 0x1E
- #define **AVC_SUBUNIT_TYPE_UNIT** 0x1F

AV/C Panel PASS THROUGH operation IDs

- #define **AVC_PANEL_OPID_SELECT** 0x00
- #define **AVC_PANEL_OPID_UP** 0x01
- #define **AVC_PANEL_OPID_DOWN** 0x02
- #define **AVC_PANEL_OPID_LEFT** 0x03
- #define **AVC_PANEL_OPID_RIGHT** 0x04
- #define **AVC_PANEL_OPID_RIGHT_UP** 0x05
- #define **AVC_PANEL_OPID_RIGHT_DOWN** 0x06
- #define **AVC_PANEL_OPID_LEFT_UP** 0x07
- #define **AVC_PANEL_OPID_LEFT_DOWN** 0x08
- #define **AVC_PANEL_OPID_ROOT_MENU** 0x09
- #define **AVC_PANEL_OPID_SETUP_MENU** 0x0A
- #define **AVC_PANEL_OPID_CONTENTS_MENU** 0x0B
- #define **AVC_PANEL_OPID_FAVORITE_MENU** 0x0C
- #define **AVC_PANEL_OPID_EXIT** 0x0D
- #define **AVC_PANEL_OPID_ON_DEMAND_MENU** 0x0E
- #define **AVC_PANEL_OPID_APPS_MENU** 0x0F
- #define **AVC_PANEL_OPID_0** 0x20
- #define **AVC_PANEL_OPID_1** 0x21
- #define **AVC_PANEL_OPID_2** 0x22
- #define **AVC_PANEL_OPID_3** 0x23
- #define **AVC_PANEL_OPID_4** 0x24
- #define **AVC_PANEL_OPID_5** 0x25
- #define **AVC_PANEL_OPID_6** 0x26

- #define **AVC_PANEL_OPID_7** 0x27
- #define **AVC_PANEL_OPID_8** 0x28
- #define **AVC_PANEL_OPID_9** 0x29
- #define **AVC_PANEL_OPID_DOT** 0x2A
- #define **AVC_PANEL_OPID_ENTER** 0x2B
- #define **AVC_PANEL_OPID_CLEAR** 0x2C
- #define **AVC_PANEL_OPID_CHANNEL_UP** 0x30
- #define **AVC_PANEL_OPID_CHANNEL_DOWN** 0x31
- #define **AVC_PANEL_OPID_PREVIOUS_CHANNEL** 0x32
- #define **AVC_PANEL_OPID_SOUND_SELECT** 0x33
- #define **AVC_PANEL_OPID_INPUT_SELECT** 0x34
- #define **AVC_PANEL_OPID_DISPLAY_INFORMATION** 0x35
- #define **AVC_PANEL_OPID_HELP** 0x36
- #define **AVC_PANEL_OPID_PAGE_UP** 0x37
- #define **AVC_PANEL_OPID_PAGE_DOWN** 0x38
- #define **AVC_PANEL_OPID_LIVE_TV** 0x39
- #define **AVC_PANEL_OPID_ZOOM** 0x3A
- #define **AVC_PANEL_OPID_LOCK** 0x3B
- #define **AVC_PANEL_OPID_SKIP** 0x3C
- #define **AVC_PANEL_OPID_NEXT_DAY** 0x3D
- #define **AVC_PANEL_OPID_PREVIOUS_DAY** 0x3E
- #define **AVC_PANEL_OPID_LINKED_CONTENT** 0x3F
- #define **AVC_PANEL_OPID_POWER_TOGGLE** 0x40
- #define **AVC_PANEL_OPID_VOLUME_UP** 0x41
- #define **AVC_PANEL_OPID_VOLUME_DOWN** 0x42
- #define **AVC_PANEL_OPID_MUTE** 0x43
- #define **AVC_PANEL_OPID_PLAY** 0x44
- #define **AVC_PANEL_OPID_STOP** 0x45
- #define **AVC_PANEL_OPID_PAUSE** 0x46
- #define **AVC_PANEL_OPID_RECORD** 0x47
- #define **AVC_PANEL_OPID_REWIND** 0x48
- #define **AVC_PANEL_OPID_FAST_FORWARD** 0x49
- #define **AVC_PANEL_OPID_EJECT** 0x4A
- #define **AVC_PANEL_OPID_FORWARD** 0x4B
- #define **AVC_PANEL_OPID_BACKWARD** 0x4C
- #define **AVC_PANEL_OPID_LIST** 0x4D
- #define **AVC_PANEL_OPID_ANGLE** 0x50
- #define **AVC_PANEL_OPID_SUBPICTURE** 0x51
- #define **AVC_PANEL_OPID_PIP_MOVE** 0x52
- #define **AVC_PANEL_OPID_PIP_DOWN** 0x53
- #define **AVC_PANEL_OPID_PIP_UP** 0x54
- #define **AVC_PANEL_OPID_RF_BYPASS** 0x55
- #define **AVC_PANEL_OPID_PLAY_FUNCTION** 0x60
- #define **AVC_PANEL_OPID_PAUSE_PLAY_FUNCTION** 0x61
- #define **AVC_PANEL_OPID_RECORD_FUNCTION** 0x62
- #define **AVC_PANEL_OPID_PAUSE_RECORD_FUNCTION** 0x63
- #define **AVC_PANEL_OPID_STOP_FUNCTION** 0x64
- #define **AVC_PANEL_OPID_MUTE_FUNCTION** 0x65
- #define **AVC_PANEL_OPID_RESTORE_FOLUME_FUNCTION** 0x66
- #define **AVC_PANEL_OPID_TUNE_FUNCTION** 0x67
- #define **AVC_PANEL_OPID_SELECT_DISK_FUNCTION** 0x68
- #define **AVC_PANEL_OPID_SELECT_AV_INPUT_FUNCTION** 0x69
- #define **AVC_PANEL_OPID_SELECT_AUDIO_INPUT_FUNCTION** 0x6A
- #define **AVC_PANEL_OPID_POWER_STATE_FUNCTION** 0x6B
- #define **AVC_PANEL_OPID_KEYBORD_FUNCTION** 0x6C

- #define **AVC_PANEL_OPID_F1** 0x71
- #define **AVC_PANEL_OPID_F2** 0x72
- #define **AVC_PANEL_OPID_F3** 0x73
- #define **AVC_PANEL_OPID_F4** 0x74
- #define **AVC_PANEL_OPID_F5** 0x75
- #define **AVC_PANEL_OPID_F6** 0x76
- #define **AVC_PANEL_OPID_F7** 0x77
- #define **AVC_PANEL_OPID_F8** 0x78
- #define **AVC_PANEL_OPID_F9** 0x79
- #define **AVC_PANEL_OPID_A** 0x7A
- #define **AVC_PANEL_OPID_B** 0x7B
- #define **AVC_PANEL_OPID_C** 0x7C
- #define **AVC_PANEL_OPID_D** 0x7D
- #define **AVC_PANEL_OPID_VENDOR_UNIQUE** 0x7E

Battery status

- #define **AVC_BATTERY_STATUS_NORMAL** 0
- #define **AVC_BATTERY_STATUS_WARNING** 1
- #define **AVC_BATTERY_STATUS_CRITICAL** 2
- #define **AVC_BATTERY_STATUS_EXTERNAL** 3
- #define **AVC_BATTERY_STATUS_FULL_CHARGE** 4

Media attribute IDs

- #define **AVC_MEDIA_ATTR_ID_TITLE** 1
- #define **AVC_MEDIA_ATTR_ID_ARTIST** 2
- #define **AVC_MEDIA_ATTR_ID_ALBUM** 3
- #define **AVC_MEDIA_ATTR_ID_NUMBER** 4
- #define **AVC_MEDIA_ATTR_ID_TOTAL_NUMBER** 5
- #define **AVC_MEDIA_ATTR_ID_GENRE** 6
- #define **AVC_MEDIA_ATTR_ID_PLAYING_TIME** 7

Media attribute bitmask

- #define **AVC_MEDIA_ATTR_FLAG_TITLE** 0x01
- #define **AVC_MEDIA_ATTR_FLAG_ARTIST** 0x02
- #define **AVC_MEDIA_ATTR_FLAG_ALBUM** 0x04
- #define **AVC_MEDIA_ATTR_FLAG_NUMBER** 0x08
- #define **AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER** 0x10
- #define **AVC_MEDIA_ATTR_FLAG_GENRE** 0x20
- #define **AVC_MEDIA_ATTR_FLAG_PLAYING_TIME** 0x40
- #define **AVC_MEDIA_ATTR_FLAG_ALL** 0x7F

Play status

- #define **AVC_PLAY_STATUS_STOPPED** 0
- #define **AVC_PLAY_STATUS_PLAYING** 1
- #define **AVC_PLAY_STATUS_PAUSED** 2
- #define **AVC_PLAY_STATUS_FW_SEEK** 3
- #define **AVC_PLAY_STATUS_REV_SEEK** 4
- #define **AVC_PLAY_STATUS_ERROR** 0xFF

Notifications

- #define AVC_EVENT_PLAYBACK_STATUS_CHANGED 0x01

  Change in playback status of the current track.
- #define AVC_EVENT_TRACK_CHANGED 0x02

  Change of current track.
- #define AVC_EVENT_TRACK_REACHED_END 0x03

  Reached end of a track.
- #define AVC_EVENT_TRACK_REACHED_START 0x04

  Reached start of a track.
- #define AVC_EVENT_PLAYBACK_POS_CHANGED 0x05

  Change in playback position. Returned after the specified playback notification change notification interval.
- #define AVC_EVENT_BATT_STATUS_CHANGED 0x06

  Change in battery status.
- #define AVC_EVENT_SYSTEM_STATUS_CHANGED 0x07

  Change in system status.
- #define AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED 0x08

  Change in player application setting.
- #define AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED 0x09

  The content of the Now Playing list has changed, see 6.9.5.
- #define AVC_EVENT_AVAILABLE_PLAYERS_CHANGED 0x0a

  The available players have changed, see 6.9.
- #define AVC_EVENT_ADDRESSED_PLAYER_CHANGED 0x0b

  The Addressed Player has been changed, see 6.9.2.
- #define AVC_EVENT_UIDS_CHANGED 0x0c

  The UIDs have changed, see 6.10.3.3.
- #define AVC_EVENT_VOLUME_CHANGED 0x0d

  The volume has been changed locally on the TG, see 6.13.3.

Notifications mask

- #define **AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED** 0x0001
- #define **AVC_EVENT_FLAG_TRACK_CHANGED** 0x0002
- #define **AVC_EVENT_FLAG_TRACK_REACHED_END** 0x0004
- #define **AVC_EVENT_FLAG_TRACK_REACHED_START** 0x0008
- #define **AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED** 0x0010
- #define **AVC_EVENT_FLAG_BATT_STATUS_CHANGED** 0x0020
- #define **AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED** 0x0040
- #define **AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED** 0x0080
- #define **AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED** 0x0100
- #define **AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED** 0x0200
- #define **AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED** 0x0400
- #define **AVC_EVENT_FLAG_UIDS_CHANGED** 0x0800
- #define **AVC_EVENT_FLAG_VOLUME_CHANGED** 0x1000
- #define **AVC_EVENT_FLAG_ALL** 0x1FFF

Media navigation scope

- #define **AVC_SCOPE_MEDIA_PLAYER_LIST** 0x00
- #define **AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM** 0x01
- #define **AVC_SEARCH** 0x02
- #define **AVC_NOW_PLAYING** 0x03

3.16.1 Detailed Description

The Audio/Video Remote Control Profile (AVRCP) defines the features and procedures required in order to ensure interoperability between Bluetooth devices with audio/video control functions in the Audio/Video distribution scenarios.

3.16.2 Typedef Documentation

3.16.2.1 typedef void(∗ bt_avrcp_find_callback_fp) (bt_bool found, void ∗param)

Find Controller/Target callback.

This callback is called when search for Controller/Target has finished.

Parameters

found	This can be one of the following values: • `TRUE` if Controller/Target was found. • `FALSE` otherwise.
callback_param	A pointer to an arbitrary data set by a call to bt_avrcp_find_target/bt_avrcp_find_controller.

3.16.2.2 typedef void(∗ bt_avrcp_mgr_callback_fp) (bt_avrcp_mgr_t ∗mgr, bt_byte evt, bt_avrcp_event_t ∗evt_param, void ∗callback_param)

AVRCP application callback.

In order to be notified of various events a consumer of the AVRCP layer has to register a callback function (done with bt_avrcp_start()). The stack will call that function whenever a new event has been generated.

Parameters

mgr	AVRCP manager.
evt	AVRCP event. The event can be one of the following values: • AVRCP_EVT_CONTROL_CHANNEL_CONNECTED This event is generated when a control channel between two AVRCP entities has been established. • AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED This event is generated when a control channel between two AVRCP entities has been terminated. • AVRCP_EVT_CONTROL_CONNECTION_FAILED This event is generated when a local device failed to create a control channel between two AVRCP entities. • AVRCP_EVT_BROWSING_CHANNEL_CONNECTED This event is generated when a browsing channel between two AVRCP entities has been established. • AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED This event is generated when a browsing channel between two AVRCP entities has been terminated. • AVRCP_EVT_BROWSING_CONNECTION_FAILED This event is generated when a local device failed to create a browsing channel between two AVRCP entities. • AVRCP_EVT_SEARCH_COMPLETED This event is generated when a local device completed searching for nearby targets. • AVRCP_EVT_PANEL_RESPONSE_RECEIVED This event is generated when a local device received a response to a PASS THROUGH command. • AVRCP_EVT_COMPANY_ID_LIST_RECEIVED This event is generated when a local device received a response to a "get company id" request. • AVRCP_EVT_EVENT_ID_LIST_RECEIVED This event is generated when a local device received a response to a "get supported events" request. • AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED This event is generated when a local device received a response to a "get supported player setting attributes" request. • AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED This event is generated when a local device received a response to a "get player setting attribute values" request. • AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED This event is generated when a local device received a response to a "get current player setting attribute values" request. • AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED This event is generated when a local device received a response to a "set player setting attribute values" request. • AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED This event is generated when a local device received a response to a "get player setting attributes displayable text" request. • AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED This event is generated when a local device received a response to a "get player setting attribute values displayable text" request. • AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED This event is generated when a local device received a response to a "inform displayable character set" request. • AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED This event is generated when a local device received a response to a "inform battery status" request.
	• AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED This event is generated when a local device received a response to a "get media element attributes" request. • AVRCP_EVT_GET_PLAY_STATUS_RECEIVED This event is generated when a local device received a response to a "get play status" request.

callback_param	A pointer to an arbitrary data set by a call to bt_avrcp_start.

3.16.3 Function Documentation

3.16.3.1 bt_bool bt_avrcp_add_to_now_playing (bt_avrcp_channel_t ∗ *channel,* bt_byte *scope,* bt_av_element_id_t ∗ *element_id,* bt_uint *counter*)

Add to "now playing" list.

This function adds a media element specified by `element_id` to the "now playing" list on the target.

Parameters

channel	AVRCP channel.
scope	The scope in which the `element_id` is valid. This value can be on the following values: • AVC_SCOPE_MEDIA_PLAYER_LIST • AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM • AVC_SEARCH • AVC_NOW_PLAYING
element_id	UID of the media element to be added to the "now playing" list.
counter	UID counter.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.2 bt_bool bt_avrcp_cancel_find (void)

Cancel finding Targets.

This function stops AVRCP layer looking for targets on nearby devices. As a result of this operation the AVRCP_↩
EVT_SEARCH_COMPLETED event will be generated.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. the callback is not called in this case.

3.16.3.3 void bt_avrcp_cancel_listen (bt_avrcp_channel_t ∗ *channel*)

Cancel listening for incoming connections.

This function stops listening for incoming connections on a specified channel.

Parameters

channel	AVRCP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.4 bt_bool bt_avrcp_connect (bt_avrcp_channel_t ∗ *channel,* bt_bdaddr_t ∗ *remote_address*)

Connect to a remote device.

This function establishes a connection to a remote device specified by the `remote_address`. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVRCP callback. The events generated will either be AVRCP_EVT_CONTROL_CHANNEL_CONNECTED or AVRCP_E←VT_CONTROL_CONNECTION_FAILED.

Parameters

channel	AVRCP channel.
remote_address	The address of a remote device.

Returns

- `TRUE` if connection establishment has been started.
- `FALSE` otherwise.

3.16.3.5 bt_avrcp_channel_t∗ bt_avrcp_create_channel (bt_avrcp_mgr_t ∗ *mgr,* bt_bool *create_browsing_channel*)

Allocate AVRCP channel.

This function allocates a new incoming AVRCP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one incoming channel.

Parameters

mgr	AVRCP manager.
create_←browsing_←channel	Defines weather a browsing channel will be created.

Returns

- A pointer to the new AVRCP channel if the function succeeds.
- `NULL` otherwise.

3.16.3.6 bt_avrcp_channel_t∗ bt_avrcp_create_outgoing_channel (bt_avrcp_mgr_t ∗ *mgr,* bt_bool *create_browsing_channel*)

Allocate AVRCP channel.

This function allocates a new outgoing AVRCP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple outgoing channels.

Parameters

mgr	AVRCP manager.
create_←browsing_←channel	Defines weather a browsing channel will be created.

Returns

- A pointer to the new AVRCP channel if the function succeeds.
- `NULL` otherwise.

3.16.3.7 bt_bool bt_avrcp_destroy_channel (bt_avrcp_channel_t ∗ *channel*)

Destroy AVRCP channel.

This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called.

Parameters

channel	AVRCP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.8 bt_bool bt_avrcp_disconnect (bt_avrcp_channel_t ∗ *channel*)

Disconnect from a remote device.

This function closes a connection to a remote device.

Parameters

channel	AVRCP channel.

Returns

- `TRUE` if disconnection has been started.
- `FALSE` otherwise. No events will be generated.

3.16.3.9 bt_bool bt_avrcp_find_controller (bt_bdaddr_t ∗ *deviceAddress,* bt_avrcp_find_callback_fp *callback,* bt_sdp_client_callback_fp *client_callback,* void ∗ *callback_param*)

Find Controller.

This function looks for a controller on a remote device specified by `deviceAddress`.

Parameters

deviceAddress	The address of a remote device.
callback	The callback function that will be called when search has completed.
client_callback	The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The `evt` parameter of the callback can be one of the following values: • SDP_CLIENT_STATE_IDLE • SDP_CLIENT_STATE_CONNECTING • SDP_CLIENT_STATE_DISCONNECTING • SDP_CLIENT_STATE_CONNECTED
callback_param	A pointer to arbitrary data to be passed to the `callback` and `client_callback` callbacks.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. the callback is not called in this case.

3.16.3.10 bt_bool bt_avrcp_find_target (bt_bdaddr_t ∗ *deviceAddress*, bt_avrcp_find_callback_fp *callback*, bt_sdp_client_callback_fp *client_callback*, void ∗ *callback_param*)

Find Target.

This function looks for a target on a remote device specified by `deviceAddress`.

Parameters

deviceAddress	The address of a remote device.
callback	The callback function that will be called when search has completed.
client_callback	The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The `evt` parameter of the callback can be one of the following values: • SDP_CLIENT_STATE_IDLE • SDP_CLIENT_STATE_CONNECTING • SDP_CLIENT_STATE_DISCONNECTING • SDP_CLIENT_STATE_CONNECTED
callback_param	A pointer to arbitrary data to be passed to the `callback` and `client_callback` callbacks.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. the callback is not called in this case.

3.16.3.11 bt_bool bt_avrcp_find_targets (bt_byte *search_length*)

Find Targets.

This function looks for targets on nearby devices. The AVRCP_EVT_SEARCH_COMPLETED event is generated when the search has completed.

Parameters

search_length	The amount of time the search will be performed for.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise. the callback is not called in this case.

3.16.3.12 bt_byte bt_avrcp_get_browsing_channel_state (bt_avrcp_channel_t ∗ *channel*)

Get AVCTP browsing channel state.

This function returns status of the AVCTP browsing channel.

Parameters

channel	AVRCP channel.

Returns

Returns of the following values:

- AVCTP_CHANNEL_STATE_FREE
- AVCTP_CHANNEL_STATE_IDLE
- AVCTP_CHANNEL_STATE_CONNECTING
- AVCTP_CHANNEL_STATE_CONNECTED
- AVCTP_CHANNEL_STATE_DISCONNECTING

3.16.3.13 bt_bdaddr_t∗ bt_avrcp_get_channel_remote_address (bt_avrcp_channel_t ∗ *channel*)

Get channel's remote BT address.

This function returns the address of the remote device associated with the channel.

Parameters

channel	AVRCP channel.

Returns

- The address of the remote device if channel is connected.
- NULL otherwise.

3.16.3.14 bt_bool bt_avrcp_get_company_id_list (bt_avrcp_channel_t ∗ *channel*)

Get Company ID list.

This function requests a list of company id's supported by the remote device

Parameters

channel	AVRCP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.15 bt_byte bt_avrcp_get_control_channel_state (bt_avrcp_channel_t ∗ *channel*)

Get AVCTP control channel state.

This function returns status of the AVCTP control channel.

Parameters

channel	AVRCP channel.

Returns

Returns of the following values:

- AVCTP_CHANNEL_STATE_FREE
- AVCTP_CHANNEL_STATE_IDLE
- AVCTP_CHANNEL_STATE_CONNECTING
- AVCTP_CHANNEL_STATE_CONNECTED
- AVCTP_CHANNEL_STATE_DISCONNECTING

3.16.3.16 bt_bool bt_avrcp_get_current_player_application_setting_value (bt_avrcp_channel_t ∗ *channel*, bt_av_player_setting_current_values_t ∗ *response_buffer*)

Get current player setting values.

This function requests a list of current set values for the player application on the target. The list of attribute ids whose values have to be returned is passed via the `response_buffer` parameter. The caller has to set bt_av_player_setting_current_values_t::setting_id_list to a list of player setting attribute ids, bt_av_player_setting_current_values_t::count to the number of entries in the list, bt_av_player_setting_current_values_t::setting_value_id_list to a buffer where returned values will be stored.

Parameters

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_setting_current_values_t structure initialized as stated above.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.17 bt_bool bt_avrcp_get_element_attributes (bt_avrcp_channel_t ∗ *channel*, bt_av_element_id_t ∗ *element_id*, bt_uint *attr_mask*)

Get media element attributes.

This function requests the attributes of the element specified with `element_id`.

Note

Currently `element_id` is ignored. The AVRCP specification defines that only UID 0 can be used to return the attributes of the current track.

Parameters

channel	AVRCP channel.
element_id	UID of the media element whose attributes are requested.
attr_mask	Bitmask that defines which attributes are requested. This value can be a combination of the following values: • AVC_MEDIA_ATTR_FLAG_TITLE • AVC_MEDIA_ATTR_FLAG_ARTIST • AVC_MEDIA_ATTR_FLAG_ALBUM • AVC_MEDIA_ATTR_FLAG_NUMBER • AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER • AVC_MEDIA_ATTR_FLAG_GENRE • AVC_MEDIA_ATTR_FLAG_PLAYING_TIME • AVC_MEDIA_ATTR_FLAG_ALL

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.18 bt_hci_conn_state_p∗ bt_avrcp_get_hci_connection (bt_avrcp_channel_t ∗ *channel*)

Get HCI connection for a channel.

This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect.

Parameters

channel	AVRCP channel.

Returns

- `Pointer` to a structure that describes an HCI connection if the function succeeds.
- `NULL` otherwise. The function fails only if a channel specified by the `channel` parameter
- does not exist or there is no HCI connection between local and remote devices associated with the channel.

3.16.3.19 bt_avrcp_mgr_t∗ bt_avrcp_get_mgr (void)

Return a pointer to an instance of the AVRCP manager.

This function returns a pointer to an instance of the AVRCP manager. There is only one instance of the manager allocated by the stack.

3.16.3.20 bt_bool bt_avrcp_get_play_status (bt_avrcp_channel_t ∗ *channel,* bt_uint *repeat_interval*)

Get playback status.

This function requests the status of the currently playing media at the target.

Parameters

channel	AVRCP channel.
repeat_interval	Interval in milliseconds at which AVRCP polls the target for playback status. If 0 is passed polling is stopped.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.21 bt_bool bt_avrcp_get_player_application_setting_attr_text (bt_avrcp_channel_t ∗ *channel,* bt_av_player_settings_text_t ∗ *response_buffer*)

Get player setting attribute text.

This function requests the target device to provide supported player application setting attribute displayable text for the provided player application setting attributes. The list of attribute ids whose displayable text have to be returned is passed via the `response_buffer` parameter. The caller has to set bt_av_player_settings_text_t::setting_↩ id_list to a list of player setting attribute ids, bt_av_player_settings_text_t::count to the number of entries in the list, bt_av_player_settings_text_t::setting_text_list to a buffer where returned values will be stored.

Parameters

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_settings_text_t structure initialized as stated above.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.22 bt_bool bt_avrcp_get_player_application_setting_value_text (bt_avrcp_channel_t ∗ *channel*, bt_byte *attr_id*, bt_av_player_setting_values_text_t ∗ *response_buffer*)

Get player setting value text.

This function request the target device to provide target supported player application setting value displayable text for the provided player application setting attribute values. The list of attribute ids whose value displayable text have to be returned is passed via the `response_buffer` parameter. The caller has to set bt_av_player_setting_↩values_text_t::setting_value_id_list to a list of player setting attribute value ids, bt_av_player_setting_values_text↩_t::count to the number of entries in the list, bt_av_player_setting_values_text_t::setting_value_text_list to a buffer where returned values will be stored.

Parameters

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_setting_values_text_t structure initialized as stated above.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.23 bt_bool bt_avrcp_get_subuint_info (bt_avrcp_channel_t ∗ *channel*)

Get subunit info.

This function is used to request subunit info from the target.

Parameters

channel	AVRCP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.24 bt_bool bt_avrcp_get_supported_event_id_list (bt_avrcp_channel_t ∗ *channel*)

Get supported events.

This function requests a list of events supported by the remote device

Parameters

channel	AVRCP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.25 bt_bool bt_avrcp_get_unit_info (bt_avrcp_channel_t ∗ *channel*)

Get unit info.

This function is used to request unit info from the target.

Parameters

channel	AVRCP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.26 bt_bool bt_avrcp_inform_battery_status (bt_avrcp_channel_t ∗ *channel,* bt_byte *status*)

Inform controller's battery status.

This function is used to inform the target about the controller's battery status.

Parameters

channel	AVRCP channel.
status	Battery status. This can be one of the following values: • AVC_BATTERY_STATUS_NORMAL • AVC_BATTERY_STATUS_WARNING • AVC_BATTERY_STATUS_CRITICAL • AVC_BATTERY_STATUS_EXTERNAL • AVC_BATTERY_STATUS_FULL_CHARGE

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.27 bt_bool bt_avrcp_inform_displayable_character_set (bt_avrcp_channel_t ∗ *channel,* bt_uint ∗ *charset_list,* bt_byte *charset_count*)

Inform displayable character set.

This function informs the list of character set supported by the controller to the target.

Parameters

channel	AVRCP channel.
charset_list	List of displayable character sets.
charset_count	Number of entries in the list.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.28 void bt_avrcp_init_controller (void)

Initialize AVRCP to be used in controller mode.

This function initializes the AVRCP layer of the stack in controller mode. It must be called prior to any other AVRCP function can be called.

3.16.3.29 void bt_avrcp_init_target (bt_ulong *company_id,* bt_uint *supported_events*)

Initialize AVRCP to be used in target mode.

This function initializes the AVRCP layer of the stack in target mode. It must be called prior to any other AVRCP function can be called.

Parameters

company_id	The 24-bit unique ID obtained from the IEEE Registration Authority Committee. If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used.
supported_↩ events	Bitmask that specifies events supported by the target. This value can be a combination of the following values: • AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED • AVC_EVENT_FLAG_TRACK_CHANGED • AVC_EVENT_FLAG_TRACK_REACHED_END • AVC_EVENT_FLAG_TRACK_REACHED_START • AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED • AVC_EVENT_FLAG_BATT_STATUS_CHANGED • AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED • AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED • AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED • AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED • AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED • AVC_EVENT_FLAG_UIDS_CHANGED • AVC_EVENT_FLAG_VOLUME_CHANGED

3.16.3.30 bt_bool bt_avrcp_list_player_application_setting_attributes (bt_avrcp_channel_t ∗ *channel,* bt_av_player_settings_t ∗ *response_buffer*)

Get supported player setting attributes.

This function request the target device to provide target supported player application setting attributes. The list of attribute ids is stored in the setting_id_list member of the `response_buffer` parameter. The caller has to set bt_av_player_settings_t::setting_id_list to a buffer where returned values will be stored and bt_av_player_settings←_t::count to the number of entries in the list.

Parameters

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_settings_t structure initialized as stated above.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.31 bt_bool bt_avrcp_list_player_application_setting_values (bt_avrcp_channel_t ∗ *channel,* bt_byte *attr_id,* bt_av_player_setting_values_t ∗ *response_buffer*)

Get player setting attribute values.

This function requests the target device to list the set of possible values for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the `response_buffer` parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list.

Parameters

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_setting_values_t structure initialized as stated above.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.32 bt_bool bt_avrcp_listen (bt_avrcp_channel_t ∗ *channel*)

Listen for incoming connections.

This function enables incoming connections on the specified AVRCP channel.

Parameters

channel	AVRCP channel.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.33 bt_bool bt_avrcp_play_item (bt_avrcp_channel_t * *channel*, bt_byte *scope*, bt_av_element_id_t * *element_id*, bt_uint *counter*)

Play media item.

This function starts playing an item indicated by the UID.

Parameters

channel	AVRCP channel.
scope	The scope in which the `element_id` is valid. This value can be on the following values: • AVC_SCOPE_MEDIA_PLAYER_LIST • AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM • AVC_SEARCH • AVC_NOW_PLAYING
element_id	UID of the media element to be played.
counter	UID counter.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.34 bt_bool bt_avrcp_register_notification (bt_avrcp_channel_t ∗ *channel,* bt_byte *event_id,* bt_ulong *playback_interval*)

Register notification.

This function registers with the target to receive notifications asynchronously based on specific events occurring.

Parameters

channel	AVRCP channel.
event_id	Event Id. This value can be one of the following values: • AVC_EVENT_PLAYBACK_STATUS_CHANGED • AVC_EVENT_TRACK_CHANGED • AVC_EVENT_TRACK_REACHED_END • AVC_EVENT_TRACK_REACHED_START • AVC_EVENT_PLAYBACK_POS_CHANGED • AVC_EVENT_BATT_STATUS_CHANGED • AVC_EVENT_SYSTEM_STATUS_CHANGED • AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED • AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED • AVC_EVENT_AVAILABLE_PLAYERS_CHANGED • AVC_EVENT_ADDRESSED_PLAYER_CHANGED • AVC_EVENT_UIDS_CHANGED • AVC_EVENT_VOLUME_CHANGED

playback_↩ interval	The time interval (in seconds) at which the change in playback position will be notified. Applicable for AVC_EVENT_PLAYBACK_POS_CHANGED event.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.35 bt_bool bt_avrcp_register_notifications (bt_avrcp_channel_t ∗ *channel,* bt_uint *event_mask*)

Register notifications.

This function registers with the target to receive notifications asynchronously based on specific events occurring. This function is used to register multiple notifications with one call.

Note

This function cannot be used to register for AVC_EVENT_PLAYBACK_POS_CHANGED event.

Parameters

channel	AVRCP channel.
event_mask	Bitmask that specifies which events to register for. This value can be a combination of the following values: • AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED • AVC_EVENT_FLAG_TRACK_CHANGED • AVC_EVENT_FLAG_TRACK_REACHED_END • AVC_EVENT_FLAG_TRACK_REACHED_START • AVC_EVENT_FLAG_BATT_STATUS_CHANGED • AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED • AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED • AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED • AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED • AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED • AVC_EVENT_FLAG_UIDS_CHANGED • AVC_EVENT_FLAG_VOLUME_CHANGED • AVC_EVENT_FLAG_ALL

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.36 bt_bool bt_avrcp_send_button_click (bt_avrcp_channel_t ∗ *channel,* bt_byte *button_id*)

Send AV/C Panel Subunit "click" PASS THROUGH command.

This function is used to send a button click. Two PATH THROUGTH commands are sent. The first command has button state set to AVC_PANEL_BUTTON_PRESSED. The second command gas button state set to AVC_PAN↩
EL_BUTTON_RELEASED

Parameters

channel	AVRCP channel.
button_id	Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.37 bt_bool bt_avrcp_send_cmd (bt_avrcp_channel_t ∗ *channel,* bt_av_command_t ∗ *command*)

Send AVRCP command.

This function sends a command to the remote device.

Parameters

channel	AVRCP channel.
command	Command to be sent.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.38 bt_bool bt_avrcp_send_panel_control (bt_avrcp_channel_t ∗ *channel,* bt_byte *opid,* bt_byte *button_state*)

Send AV/C Panel Subunit "control" PASS THROUGH command.

This function is used to send AV/C Panel Subunit PASS THROUGH command with command type set to AVC_↩ CTYPE_CONTROL.

Parameters

channel	AVRCP channel.
opid	Operation Id. This value can be on of the AVC_PANEL_OPID_... constants
button_state	Button state. This can be on of the following values: • AVC_PANEL_BUTTON_PRESSED • AVC_PANEL_BUTTON_RELEASED

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.39 bt_bool bt_avrcp_send_press_panel_control (bt_avrcp_channel_t ∗ *channel,* bt_byte *opid*)

Send AV/C Panel Subunit "pressed" PASS THROUGH command.

This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PA↩ NEL_BUTTON_PRESSED.

Parameters

channel	AVRCP channel.
opid	Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.40 bt_bool bt_avrcp_send_release_panel_control (bt_avrcp_channel_t ∗ *channel*, bt_byte *opid*)

Send AV/C Panel Subunit "released" PASS THROUGH command.

This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC_PA←NEL_BUTTON_RELEASED.

Parameters

channel	AVRCP channel.
opid	Operation Id. This value can be on of the AVC_PANEL_OPID_... constants

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.41 bt_bool bt_avrcp_send_simple_panel_cmd (bt_avrcp_channel_t ∗ *channel*, bt_byte *ctype*, bt_byte *opid*, bt_byte *button_state*)

Send AV/C Panel Subunit PASS THROUGH command.

This function is used to send AV/C Panel Subunit PASS THROUGH command to the target.

Parameters

channel	AVRCP channel.
ctype	Command type. This value can be on of the following values: • AVC_CTYPE_CONTROL 0 • AVC_CTYPE_STATUS 1 • AVC_CTYPE_SPECIFIC_IQUIRY 2 • AVC_CTYPE_NOTIFY 3 • AVC_CTYPE_GENERAL_INQUORY 4
opid	Operation Id. This value can be on of the AVC_PANEL_OPID_... constants
button_state	Button state. This can be on of the following values: • AVC_PANEL_BUTTON_PRESSED • AVC_PANEL_BUTTON_RELEASED

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.42 bt_bool bt_avrcp_set_absolute_volume (bt_avrcp_channel_t ∗ *channel*, bt_byte *volume*)

Set absolute volume.

This function is used to set an absolute volume to be used by the rendering device.

Parameters

channel	AVRCP channel.
volume	Volume

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.43 bt_bool bt_avrcp_set_addressed_player (bt_avrcp_channel_t ∗ *channel*, bt_uint *player_id*)

Set addressed player.

This function is used to inform the target of which media player the controller wishes to control.

Parameters

channel	AVRCP channel.
player_id	Player Id.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.44 bt_bool bt_avrcp_set_player_application_setting_value (bt_avrcp_channel_t ∗ *channel*, bt_byte ∗ *attr_id_list*, bt_byte ∗ *attr_value_list*, bt_byte *attr_id_count*)

Set player setting attribute values.

This function requests to set the player application setting list of player application setting values on the target device for the corresponding defined list of setting attributes. for the requested player application setting attribute. The list of attribute value ids is stored in the setting_value_id_list member of the `response_buffer` parameter. The caller has to set bt_av_player_setting_values_t::setting_value_id_list to a buffer where returned values will be stored and bt_av_player_setting_values_t::count to the number of entries in the list.

Parameters

channel	AVRCP channel.
attr_id_list	List of setting attribute ids.
attr_value_list	List of setting attribute value ids.
attr_id_count	The number of entries in both lists.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.16.3.45 void bt_avrcp_start (bt_avrcp_mgr_t ∗ *mgr,* bt_avrcp_mgr_callback_fp *callback,* void ∗ *callback_param*)

Start the AVRCP layer.

In order to be notified of various events a consumer of the AVRCP layer has to register a callback function. The stack will call the callback function whenever a new event has been generated passing the code of the event as the second parameter. bt_avrcp_start() stores pointers to the `callback` and `callback_param` in the bt_avrcp↩_mgr_t structure.

Parameters

mgr	AVRCP manager.
callback	The callback function that will be called when the AVRCP generates an event.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.16.3.46 bt_bool bt_avrcp_tg_send_element_attributes (bt_avrcp_channel_t ∗ *channel,* bt_byte *tran_id,* bt_av_element_attribute_t ∗ *attrs,* bt_byte *attr_count*)

Send media element attributes.

This function is used to send the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified.

Parameters

status	Battery status

3.16.3.47 void bt_avrcp_tg_set_absolute_volume (bt_byte *volume*)

Set absolute volume.

This function is used to set the absolute volume when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_VOLUME_CHANGED event will be notified.

Parameters

track_id	Track Id
song_length	The length of the current track.
song_position	The position of the current track.

3.16.3.48 void bt_avrcp_tg_set_battery_status (bt_byte *status*)

Set battery status.

This function is used to set the battery status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_BATT_STATUS_CHANGED event will be notified.

Parameters

status	Battery status

3.16.3.49 void bt_avrcp_tg_set_current_track (bt_av_element_id_t ∗ *track_id,* bt_ulong *song_length,* bt_ulong *song_position*)

Set current track.

This function is used to set the current track when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_TRACK_CHANGED event will be notified.

Parameters

track_id	Track Id
song_length	The length of the current track.
song_position	The position of the current track.

3.16.3.50 void bt_avrcp_tg_set_play_status (bt_ulong *song_length,* bt_ulong *song_position,* bt_byte *play_status*)

Set playback status.

This function is used to set the playback status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_PLAYBACK_STATUS_CHANGED event will be notified.

Parameters

song_length	The length of the current track.
song_position	The position of the current track.
play_status	Playback status. This value can be one of the following values: • AVC_PLAY_STATUS_STOPPED • AVC_PLAY_STATUS_PLAYING • AVC_PLAY_STATUS_PAUSED • AVC_PLAY_STATUS_FW_SEEK • AVC_PLAY_STATUS_REV_SEEK

3.16.3.51 void bt_avrcp_tg_set_system_status (bt_byte *status*)

Set system status.

This function is used to set the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC_EVENT_SYSTEM_STATUS_CHANGED event will be notified.

Parameters

status	Battery status

3.17 Configuration

This module describes parameters used to configure AVRCP layer.

Macros

- #define AVRCP_MAX_CHANNELS

  Maximum number of remote devices a local device can be connected to.
- #define AVRCP_MAX_CMD_BUFFERS 1

  Maximum number of command buffers.
- #define AVRCP_MAX_CMD_PARAM_LEN 512

  Maximum length of command parameters.
- #define AVRCP_MAX_SEARCH_RESULTS 7

  Maximum number of devices to find.
- #define AVRCP_MAX_DEVICE_NAME_LEN 20

  Maximum length of device name.
- #define AVRCP_CMD_TIMEOUT 10000

  Command timeout.

3.17.1 Detailed Description

This module describes parameters used to configure AVRCP layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI, L2CAP and SDP must always be present

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS             ...
#define HCI_MAX_DATA_BUFFERS            ...
#define HCI_MAX_HCI_CONNECTIONS         ...
#define HCI_RX_BUFFER_LEN               ...
#define HCI_TX_BUFFER_LEN               ...
#define HCI_L2CAP_BUFFER_LEN            ...
#define HCI_MAX_CMD_PARAM_LEN           ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS           ...
#define L2CAP_MAX_FRAME_BUFFERS         ...
#define L2CAP_MAX_PSMS                  ...
#define L2CAP_MAX_CHANNELS              ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN      ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN    ...

// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure AVRCP,
// the following values must be defined:

#define BT_INCLUDE_AVCTP                // tells dotstack to compile in AVCTP support
#define AVCTP_MAX_CHANNELS              ...
#define AVCTP_MAX_TRANSPORT_CHANNELS    ...
#define AVCTP_MAX_RX_MESSAGE_LEN        ...
#define AVCTP_MAX_MESSAGE_BUFFERS       ...

#define BT_INCLUDE_AVRCP                // tells dotstack to compile in AVRCP support
#define AVRCP_MAX_CHANNELS              ...
#define AVRCP_MAX_CMD_BUFFERS           ...
#define AVRCP_MAX_CMD_PARAM_LEN         ...
#define AVRCP_MAX_SEARCH_RESULTS        ...
#define AVRCP_MAX_DEVICE_NAME_LEN       ...
#define AVRCP_CMD_TIMEOUT               ...

#include "cdbt/bt/bt_oem_config.h"
```

3.17.2 Macro Definition Documentation

3.17.2.1 #define AVRCP_CMD_TIMEOUT 10000

Command timeout.

This parameter defines the amount of time in milliseconds AVRCP waits for a response to a request. If not defined the default value of 10000 (10 secconds) is used.

3.17.2.2 #define AVRCP_MAX_CHANNELS

Maximum number of remote devices a local device can be connected to.

This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. channels). This value should not exceed AVCTP_MAX_CHANNELS.

3.17.2.3 #define AVRCP_MAX_CMD_BUFFERS 1

Maximum number of command buffers.

This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVRCP_MAX_CHANNELS ∗ AVR↩
CP_MAX_CMD_BUFFERS. The minimum value is 1. The maximum value is 255. If not define one buffer for each channel is reserved.

3.17.2.4 #define AVRCP_MAX_CMD_PARAM_LEN 512

Maximum length of command parameters.

This parameter defines the maximum length of all command parameters. If not defined the default value of 512 is used.

3.17.2.5 #define AVRCP_MAX_DEVICE_NAME_LEN 20

Maximum length of device name.

This parameter defines the size of the buffer used to store device's name while searching for nearby targets with bt_avrcp_find_targets. If the name of the device is longer than AVRCP_MAX_DEVICE_NAME_LEN it is truncated to AVRCP_MAX_DEVICE_NAME_LEN. If not defined the default value of 20 is used.

3.17.2.6 #define AVRCP_MAX_SEARCH_RESULTS 7

Maximum number of devices to find.

This parameter defines the maximum number of devices bt_avrcp_find_targets can find. If not defined the default value of 7 is used.

3.18 ATT

The attribute protocol allows a device referred to as the server to expose a set of attributes and their associated values to a peer device referred to as the client.

Modules

- ATT Server

  This module describes functions and data structures used to implement an ATT server (peripheral).
- ATT Client

  This module describes functions and data structures used to implement an ATT client (central).

Error Codes

- #define ATT_ERROR_SUCCESS 0x00

  The operation completed with no errors.
- #define ATT_ERROR_INVALID_HANDLE 0x01

  The attribute handle given was not valid on this server.
- #define ATT_ERROR_READ_NOT_PERMITTED 0x02

  The attribute cannot be read.
- #define ATT_ERROR_WRITE_NOT_PERMITTED 0x03

  The attribute cannot be written.
- #define ATT_ERROR_INVALID_PDU 0x04

  The attribute PDU was invalid.
- #define ATT_ERROR_INSUFFICIENT_AUTHENTICATION 0x05

  The attribute requires authentication before it can be read or written.
- #define ATT_ERROR_REQUEST_NOT_SUPPORTED 0x06

  Attribute server does not support the request received from the client.
- #define ATT_ERROR_INVALID_OFFSET 0x07

  Offset specified was past the end of the attribute.
- #define ATT_ERROR_INSUFFICIENT_AUTHORIZATION 0x08

  The attribute requires authorization before it can be read or written.
- #define ATT_ERROR_PREPARE_QUEUE_FULL 0x09

  Too many prepare writes have been queued.
- #define ATT_ERROR_ATTRIBUTE_NOT_FOUND 0x0A

  No attribute found within the given attribute handle range.
- #define ATT_ERROR_ATTRIBUTE_NOT_LONG 0x0B

  The attribute cannot be read or written using the Read Blob Request.
- #define ATT_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE 0x0C

  The Encryption Key Size used for encrypting this link is insufficient.
- #define ATT_ERROR_INVALID_ATTRIBUTE_VALUE_LENGTH 0x0D

  The attribute value length is invalid for the operation.
- #define ATT_ERROR_UNLIKELY_ERROR 0x0E

  The attribute request that was requested has encountered an error.
- #define ATT_ERROR_INSUFFICIENT_ENCRYPTION 0x0F

  The attribute requires encryption before it can be read or written.
- #define ATT_ERROR_UNSUPPORTED_GROUP_TYPE 0x10

  The attribute type is not a supported.
- #define ATT_ERROR_INSUFFICIENT_RESOURCES 0x11

  Insufficient Resources to complete the request.

3.18.1 Detailed Description

The attribute protocol allows a device referred to as the server to expose a set of attributes and their associated values to a peer device referred to as the client.

These attributes exposed by the server can be discovered, read, and written by a client, and can be indicated and notified by the server.

3.18.2 Macro Definition Documentation

3.18.2.1 #define ATT_ERROR_UNLIKELY_ERROR 0x0E

The attribute request that was requested has encountered an error.

that was unlikely, and therefore could not be completed as requested.

3.18.2.2 #define ATT_ERROR_UNSUPPORTED_GROUP_TYPE 0x10

The attribute type is not a supported.

grouping attribute as defined by a higher layer specification.

3.19 ATT Server

This module describes functions and data structures used to implement an ATT server (peripheral).

Modules

- Configuration

  This module describes parameters used to configure ATT server layer.

Data Structures

- struct bt_att_evt_session_connected_t

  Parameter to ATT_SERVER_EVT_SESSION_CONNECTED event.
- struct bt_att_evt_session_disconnected_t

  Parameter to ATT_SERVER_EVT_SESSION_DISCONNECTED event.
- struct bt_att_evt_attr_value_changed_t

  Parameter to ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_SERVER and ATT_SERVER_EVT_ATTR_V↩ ALUE_CHANGED_BY_CLIENT events.
- struct bt_att_evt_attr_value_read_t

  Parameter to ATT_SERVER_EVT_ATTR_VALUE_READ event.
- struct bt_att_evt_attr_notification_sent_t

  Parameter to ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED event.
- struct bt_att_evt_attr_indication_sent_t

  Parameter to ATT_SERVER_EVT_ATTR_VALUE_INDICATED event.
- struct bt_att_evt_tran_timeout_t

  Parameter to ATT_SERVER_EVT_TRAN_TIMEOUT event.
- struct bt_att_evt_authorization_requested_t

  Parameter to ATT_SERVER_EVT_AUTHORIZATION_REQUESTED event.
- struct bt_att_mgr_t

  ATT manager.

Typedefs

- typedef void(∗ bt_att_mgr_callback_fp) (bt_int evt, void ∗evt_param, void ∗param)

  ATT application callback.

Functions

- bt_bool bt_att_init (void)

  Initialize the ATT layer.
- bt_att_mgr_t ∗ bt_att_get_mgr (void)

  Return a pointer to an instance of the ATT manager.
- bt_bool bt_att_update_conn_parameters (bt_att_session_t ∗session, bt_uint min_interval, bt_uint max_↩ interval, bt_uint slave_latency, bt_uint supervision_timeout)

  Request connection parameters update.
- bt_bdaddr_t ∗ bt_att_get_remote_address (const bt_att_session_t ∗session)

  Get the address of the remote device this device is connected to.
- bt_bool bt_att_server_start (bt_byte ∗att_var_db, bt_uint att_var_db_len, const bt_byte ∗att_const_db, bt_uint att_const_db_len, bt_bool listen_on_dynamic_channel)

  Start ATT server.

- void bt_att_server_stop (void)

    Stop ATT server.
- void bt_att_server_register_callback (bt_att_server_callback_fp callback, void ∗param)

    Register a ATT application callback.
- bt_byte bt_att_server_write_attribute_value (bt_uint handle, const bt_byte ∗value, bt_uint value_len, bt_uint offset)

    Write attribute's value.
- bt_bool bt_att_server_disconnect (bt_att_session_t ∗session)

    Disconnect from a remote device.
- void bt_att_server_indicate_to_all (bt_att_attribute_t ∗attr)

    Indicate attribute's value to all connected clients.
- void bt_att_server_indicate (bt_att_session_t ∗session, bt_att_attribute_t ∗attr)

    Indicate attribute's value.
- void bt_att_server_notify (bt_att_attribute_t ∗attr)

    Notify attribute's value to all connected clients.
- void bt_att_server_authorize_access (bt_att_session_t ∗session, bt_att_attribute_t ∗attr, bt_byte opcode, bt_bool authorize)

    Authorize access to an attribute.

Events

- #define ATT_SERVER_EVT_SESSION_CONNECTED ATT_MGR_EVT_SESSION_CONNECTED

    A client connected to the server.
- #define ATT_SERVER_EVT_SESSION_DISCONNECTED ATT_MGR_EVT_SESSION_DISCONNECTED

    A client disconnected from the server.
- #define ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_SERVER ATT_MGR_EVT_ATTR_VALUE_↩ CHANGED_BY_SERVER

    Attribute's value has been changed locally on the server.
- #define ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_CLIENT ATT_MGR_EVT_ATTR_VALUE_↩ CHANGED_BY_CLIENT

    A client has changed the attribute's value.
- #define ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED ATT_MGR_EVT_ATTR_VALUE_NOTIFIED

    A value notification has bee sent to the client.
- #define ATT_SERVER_EVT_ATTR_VALUE_INDICATED ATT_MGR_EVT_ATTR_VALUE_INDICATED

    A value indication has been sent to the client.
- #define ATT_SERVER_EVT_TRAN_TIMEOUT ATT_MGR_EVT_TRAN_TIMEOUT

    Operation timed out.
- #define ATT_SERVER_EVT_AUTHORIZATION_REQUESTED ATT_MGR_EVT_AUTHORIZATION_RE↩ QUESTED

    Authorization is required in order to access the attribute's value.
- #define ATT_SERVER_EVT_ATTR_VALUE_READ_BY_CLIENT ATT_MGR_EVT_ATTR_VALUE_READ↩ _BY_CLIENT

    A client has read the attribute's value.
- #define ATT_SERVER_EVT_STOPPED 50

    Server stooped.

3.19.1 Detailed Description

This module describes functions and data structures used to implement an ATT server (peripheral).

3.19.2 Typedef Documentation

3.19.2.1 typedef void(∗ bt_att_mgr_callback_fp) (bt_int evt, void ∗evt_param, void ∗param)

ATT application callback.

In order to be notified of various events a consumer of the ATT layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the first parameter.

Parameters

evt	ATT event
evt_param	Event parameter. This can be one of the following values: • ATT_SERVER_EVT_SESSION_CONNECTED A client connected to the server • ATT_SERVER_EVT_SESSION_DISCONNECTED A client disconnected from the server • ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_SERVER Attribute's value has been changed locally on the server • ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_CLIENT A client has changed the attribute's value • ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED A value notification has bee sent to the client • ATT_SERVER_EVT_ATTR_VALUE_INDICATED A value indication has been sent to the client • ATT_SERVER_EVT_TRAN_TIMEOUT Operation timed out • ATT_SERVER_EVT_AUTHORIZATION_REQUESTED Authorization is required in order to access the attribute's value • ATT_SERVER_EVT_STOPPED Server stooped

callback_param	A pointer to an arbitrary data set by a call to bt_att_server_register_callback.

3.19.3 Function Documentation

3.19.3.1 bt_att_mgr_t∗ bt_att_get_mgr (void)

Return a pointer to an instance of the ATT manager.

This function returns a pointer to an instance of the ATT manager. There is only one instance of the manager allocated by the stack.

3.19.3.2 bt_bdaddr_t∗ bt_att_get_remote_address (const bt_att_session_t ∗ *session*)

Get the address of the remote device this device is connected to.

Parameters

session	ATT session.

Returns

- A pointer to bt_bdaddr structure that contains the address of the remote device.

3.19.3.3 bt_bool bt_att_init (void)

Initialize the ATT layer.

This function initializes the ATT layer of the stack. It must be called prior to any other ATT function can be called.

3.19.3.4 void bt_att_server_authorize_access (bt_att_session_t ∗ *session,* bt_att_attribute_t ∗ *attr,* bt_byte *opcode,* bt_bool *authorize*)

Authorize access to an attribute.

If an attribute requires authorization before its value can be read or written, ATT generates a ATT_SERVER_E←VT_AUTHORIZATION_REQUESTED event. In response to this event the application should obtain authorization (how this is done is implementation specific) and call bt_att_server_authorize_access.

Parameters

session	ATT session.
attr	Attribute.
opcode	Attribute Opcode. The opcode that requires authorization is passed in the ATT_SERV↩ ER_EVT_AUTHORIZATION_REQUESTED event. The application should use the passed opcode when calling bt_att_server_authorize_access. The opcode can be one the following values: • ATT_OPCODE_READ_REQUEST • ATT_OPCODE_READ_BLOB_REQUEST • ATT_OPCODE_READ_BY_TYPE_REQUEST • ATT_OPCODE_READ_BY_GROUP_TYPE_REQUEST • ATT_OPCODE_READ_MULTIPLE_REQUEST • ATT_OPCODE_WRITE_REQUEST • ATT_OPCODE_PREPARE_WRITE_REQUEST
authorize	Specifies whether access to the attribute has been authorized or not.

3.19.3.5 bt_bool bt_att_server_disconnect (bt_att_session_t ∗ *session*)

Disconnect from a remote device.

This function closes a connection to a remote device.

Parameters

session	ATT session.

Returns

- `TRUE` if disconnection has been started.
- `FALSE` otherwise. No events will be generated.

3.19.3.6 void bt_att_server_indicate (bt_att_session_t ∗ *session,* bt_att_attribute_t ∗ *attr*)

Indicate attribute's value.

This function sends the attribute's value specified with the `attr` parameter to a client sepecified with the `session` parameter.

Parameters

session	ATT session.
attr	Attribute.

3.19.3.7 void bt_att_server_indicate_to_all (bt_att_attribute_t ∗ *attr*)

Indicate attribute's value to all connected clients.

This function sends the attribute's value specified with the `attr` parameter to all connected clients.

Parameters

attr	Attribute.

3.19.3.8 void bt_att_server_notify (bt_att_attribute_t ∗ *attr*)

Notify attribute's value to all connected clients.

This function sends the attribute's value specified with the `attr` parameter to all connected clients.

Parameters

attr	Attribute.

3.19.3.9 void bt_att_server_register_callback (bt_att_server_callback_fp *callback,* void ∗ *param*)

Register a ATT application callback.

In order to be notified of various events a consumer of the ATT layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the first parameter. The event can be one of the following values:

- ATT_SERVER_EVT_SESSION_CONNECTED A client connected to the server
- ATT_SERVER_EVT_SESSION_DISCONNECTED A client disconnected from the server
- ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_SERVER Attribute's value has been changed locally on the server
- ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_CLIENT A client has changed the attribute's value
- ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED A value notification has bee sent to the client
- ATT_SERVER_EVT_ATTR_VALUE_INDICATED A value indication has been sent to the client
- ATT_SERVER_EVT_TRAN_TIMEOUT Transaction timed out
- ATT_SERVER_EVT_AUTHORIZATION_REQUESTED Authorization is required in order to access the attribute's value
- ATT_SERVER_EVT_STOPPED Server stooped

Parameters

callback	The callback function that will be called when ATT generates an event.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.19.3.10 bt_bool bt_att_server_start (bt_byte ∗ *att_var_db,* bt_uint *att_var_db_len,* const bt_byte ∗ *att_const_db,* bt_uint *att_const_db_len,* bt_bool *listen_on_dynamic_channel*)

Start ATT server.

This function starts the ATT server.

Parameters

att_var_db	A pointer to ATT database that holds writable attributes.
att_var_db_len	The length of the writable ATT database.
att_const_db	A pinter to ATT database that holds read-only attributes.
att_const_db_↩ len	The length of the read-only ATT database.
listen_on_↩ dynamic_↩ channel	Specifies whether ATT server should accept connections on BR/EDR links.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.19.3.11 void bt_att_server_stop (void)

Stop ATT server.

This function stops the ATT server.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.19.3.12 bt_byte bt_att_server_write_attribute_value (bt_uint *handle,* const bt_byte ∗ *value,* bt_uint *value_len,* bt_uint *offset*)

Write attribute's value.

This function writes attribute's value.

Parameters

handle	Attribute handle.
value	Attribute value.
value_len	Attribute value length.
offset	Offset from which to write the value.

Returns

Error code. This can be one of the following values:

- ATT_ERROR_SUCCESS
- ATT_ERROR_INVALID_HANDLE
- ATT_ERROR_READ_NOT_PERMITTED
- ATT_ERROR_WRITE_NOT_PERMITTED
- ATT_ERROR_INVALID_PDU
- ATT_ERROR_INSUFFICIENT_AUTHENTICATION
- ATT_ERROR_REQUEST_NOT_SUPPORTED
- ATT_ERROR_INVALID_OFFSET
- ATT_ERROR_INSUFFICIENT_AUTHORIZATION
- ATT_ERROR_PREPARE_QUEUE_FULL
- ATT_ERROR_ATTRIBUTE_NOT_FOUND

- ATT_ERROR_ATTRIBUTE_NOT_LONG
- ATT_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE
- ATT_ERROR_INVALID_ATTRIBUTE_VALUE_LENGTH
- ATT_ERROR_UNLIKELY_ERROR
- ATT_ERROR_INSUFFICIENT_ENCRYPTION
- ATT_ERROR_UNSUPPORTED_GROUP_TYPE
- ATT_ERROR_INSUFFICIENT_RESOURCES

3.19.3.13 bt_bool bt_att_update_conn_parameters (bt_att_session_t ∗ *session,* bt_uint *min_interval,* bt_uint *max_interval,* bt_uint *slave_latency,* bt_uint *supervision_timeout*)

Request connection parameters update.

This function sends a request to the client (central) to update connection parameters.

Parameters

session	ATT session.
min_interval	Minimum connection interval expressed in 1.25ms units.
max_interval	Maximum connection interval expressed in 1.25ms units.
slave_latency	Slave latency expressed as number of connection events.
supervision_↩ timeout	Link supervision timeout expressed in 10ms units.

Returns

- `TRUE` if request has been sent to the central.
- `FALSE` otherwise.

3.20 Configuration

This module describes parameters used to configure ATT server layer.

Macros

- #define ATT_MAX_CLIENTS

  Maximum number of clients.
- #define ATT_MAX_FOUND_ATTRIBUTES

  Maximum number of attributes that can be returned in one multi-attribute response.
- #define ATT_MAX_QUEUED_WRITE_BUFFER_SIZE 0

  Size of the buffer for storing queued writes.

3.20.1 Detailed Description

This module describes parameters used to configure ATT server layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS            ...
#define HCI_MAX_DATA_BUFFERS           ...
#define HCI_MAX_HCI_CONNECTIONS        ...
#define HCI_RX_BUFFER_LEN              ...
#define HCI_TX_BUFFER_LEN              ...
#define HCI_L2CAP_BUFFER_LEN           ...
#define HCI_MAX_CMD_PARAM_LEN          ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS          ...
#define L2CAP_MAX_FRAME_BUFFERS        ...
#define L2CAP_MAX_PSMS                 ...
#define L2CAP_MAX_CHANNELS             ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN     ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN   ...

// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure ATT,
// the following values must be defined:

#define BT_INCLUDE_ATT               // tells dotstack to compile in ATT support
#define ATT_MAX_CLIENTS                          ...
#define ATT_MAX_FOUND_ATTRIBUTES                 ...
#define ATT_MAX_QUEUED_WRITE_BUFFER_SIZE         ...

#include "cdbt/bt/bt_oem_config.h"
```

3.20.2 Macro Definition Documentation

3.20.2.1 #define ATT_MAX_CLIENTS

Maximum number of clients.

This parameter defines the maximum number of clients that can be simultaneously connected to the server. If ATT is used only over LE this number must be set to 1. Any other numbers will be a waste of RAM because an LE slave (server) can have only one connection to a master (client) at any given moment.

3.20.2.2 #define ATT_MAX_FOUND_ATTRIBUTES

Maximum number of attributes that can be returned in one multi-attribute response.

This parameter defines the maximum number of attributes that can be returned in one multi-attribute response. The minimum value is 1. The maximum value is 255. This number must be set to as large value as possible (given there is enough RAM) to minimize the number of request needed by the client to find all requested attributes.

3.20.2.3 #define ATT_MAX_QUEUED_WRITE_BUFFER_SIZE 0

Size of the buffer for storing queued writes.

This parameter defines the Size of the buffer for storing queued writes. The minimum value is 0. The maximum value is 65535. If this is set to 0 queued writes will not be supported.

3.21 ATT Client

This module describes functions and data structures used to implement an ATT client (central).

Modules

- Configuration

  This module describes parameters used to configure ATT client layer.

Data Structures

- struct bt_att_find_info_response_t

  Structure to store response to a "find info" request.
- struct bt_att_find_by_type_value_response_t

  Structure to store response to a "find by type value" request.
- struct bt_att_read_by_type_response_t

  Structure to store response to a "read by type" request.
- struct bt_att_read_by_group_type_response_t

  Structure to store response to a "read by group type" request.
- struct bt_att_client_evt_mtu_response_t

  Parameter to ATT_CLIENT_EVT_EXCHANGE_MTU_RESPONSE event.
- struct bt_att_client_evt_info_response_t

  Parameter to ATT_CLIENT_EVT_FIND_INFO_RESPONSE event.
- struct bt_att_client_evt_find_by_type_value_response_t

  Parameter to ATT_CLIENT_EVT_FIND_BY_TYPE_VALUE_RESPONSE event.
- struct bt_att_client_evt_read_by_type_response_t

  Parameter to ATT_CLIENT_EVT_READ_BY_TYPE_RESPONSE event.
- struct bt_att_client_evt_read_response_t

  Parameter to ATT_CLIENT_EVT_READ_RESPONSE event.
- struct bt_att_client_evt_read_blob_response_t

  Parameter to ATT_CLIENT_EVT_READ_BLOB_RESPONSE event.
- struct bt_att_client_evt_read_multiple_response_t

  Parameter to ATT_CLIENT_EVT_READ_MULTIPLE_RESPONSE event.
- struct bt_att_client_evt_read_by_group_type_response_t

  Parameter to ATT_CLIENT_EVT_READ_BY_GROUP_TYPE_RESPONSE event.
- struct bt_att_client_evt_write_response_t

  Parameter to ATT_CLIENT_EVT_WRITE_RESPONSE event.
- struct bt_att_client_evt_prepare_write_response_t

  Parameter to ATT_CLIENT_EVT_PREPARE_WRITE_RESPONSE event.
- struct bt_att_client_evt_execute_write_response_t

  Parameter to ATT_CLIENT_EVT_EXECUTE_WRITE_RESPONSE event.
- struct bt_att_client_evt_value_notification_t

  Parameter to ATT_CLIENT_EVT_VALUE_NOTIFICATION event.
- struct bt_att_client_evt_value_indication_t

  Parameter to ATT_CLIENT_EVT_VALUE_INDICATION event.
- struct bt_att_client_evt_conn_param_update_t

  Parameter to ATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST event.
- struct bt_att_client_evt_conn_param_update_completed_t

  Parameter to ATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED event.
- union bt_att_client_evt_t

  Parameter to ATT client application callback.

Functions

- bt_bool bt_att_client_init (void)

  Initialize the ATT client.
- bt_att_client_mgr_t ∗ bt_att_get_client_mgr (void)

  Return a pointer to an instance of the ATT client manager.
- bt_att_client_session_t ∗ bt_att_client_allocate_session (bt_att_client_session_callback_fp callback, void ∗callback_param)

  Allocate ATT client session.
- void bt_att_client_free_session (bt_att_client_session_t ∗session)

  Destroy ATT client session.
- bt_bool bt_att_client_connect (bt_att_client_session_t ∗session, bt_bdaddr_t ∗addr)

  Connect to a remote device (peripheral).
- bt_bool bt_att_client_disconnect (bt_att_client_session_t ∗session)

  Disconnect from a remote device.
- bt_bool bt_att_client_exchange_mtu (bt_att_client_session_t ∗session, bt_uint mtu)

  Exchange MTU.
- bt_bool bt_att_client_find_info (bt_att_client_session_t ∗session, bt_uint start_handle, bt_uint end_handle, bt_att_find_info_response_t ∗result, bt_uint max_results)

  Find Information.
- bt_bool bt_att_client_find_by_type_value (bt_att_client_session_t ∗session, bt_uint start_handle, bt_uint end_handle, bt_uint type, const bt_byte ∗value, bt_uint len, bt_att_find_by_type_value_response_t ∗result, bt_uint max_results)

  Find By Type Value.
- bt_bool bt_att_client_read_by_type (bt_att_client_session_t ∗session, bt_uint start_handle, bt_uint end_↩ handle, bt_uint type, bt_att_read_by_type_response_t ∗result, bt_uint max_results)

  Read By Type (16-bit).
- bt_bool bt_att_client_read_by_type_80 (bt_att_client_session_t ∗session, bt_uint start_handle, bt_uint end↩ _handle, bt_uuid_t ∗type, bt_att_read_by_type_response_t ∗result, bt_uint max_results)

  Read By Type (128-bit).
- bt_bool bt_att_client_read (bt_att_client_session_t ∗session, bt_uint handle)

  Read Attribute value.
- bt_bool bt_att_client_read_blob (bt_att_client_session_t ∗session, bt_uint handle, bt_uint offset)

  Read Blob.
- bt_bool bt_att_client_read_multiple (bt_att_client_session_t ∗session, const bt_uint ∗handles, bt_uint count)

  Read Multiple.
- bt_bool bt_att_client_read_by_group_type (bt_att_client_session_t ∗session, bt_uint start_handle, bt_uint end_handle, bt_uint group_type, bt_att_read_by_group_type_response_t ∗result, bt_uint max_results)

  Read By Group Type (16-bit).
- bt_bool bt_att_client_read_by_group_type_80 (bt_att_client_session_t ∗session, bt_uint start_handle, bt↩ _uint end_handle, bt_uuid_t ∗group_type, bt_att_read_by_group_type_response_t ∗result, bt_uint max_↩ results)

  Read By Group Type (128-bit).
- bt_bool bt_att_client_write (bt_att_client_session_t ∗session, bt_uint handle, const bt_byte ∗value, bt_uint len)

  Write Attribute Value.
- bt_bool bt_att_client_write_cmd (bt_att_client_session_t ∗session, bt_uint handle, const bt_byte ∗value, bt↩ _uint len)

  Write Command.
- bt_bool bt_att_client_signed_write (bt_att_client_session_t ∗session, bt_uint handle, const bt_byte ∗value, bt_uint len, const bt_byte ∗signature)

  Signed Write Command.

- bt_bool bt_att_client_prepare_write (bt_att_client_session_t ∗session, bt_uint handle, bt_uint value_offset, const bt_byte ∗value, bt_uint len)

  Prepare Write.
- bt_bool bt_att_client_execute_write (bt_att_client_session_t ∗session, bt_byte flags)

  Execute Write.
- bt_bool bt_att_client_accept_conn_param_update (bt_att_client_session_t ∗session, bt_uint min_interval, bt_uint max_interval, bt_uint slave_latency, bt_uint supervision_timeout, bt_uint min_ce_length, bt_uint max_ce_length)

  Accept connection parameters update.
- void bt_att_client_reject_conn_param_update (bt_att_client_session_t ∗session)

  Reject connection parameters update.
- bt_bdaddr_t ∗ bt_att_client_get_remote_address (const bt_att_client_session_t ∗session)

  Get the address of the remote device this device is connected to.

Events

- #define ATT_CLIENT_SESSION_EVT_SESSION_CONNECTED 1

  This event is generated when client connected to the server.
- #define ATT_CLIENT_SESSION_EVT_SESSION_DISCONNECTED 2

  This event is generated when client disconnected from the server.
- #define ATT_CLIENT_SESSION_EVT_CONNECTION_FAILED 3

  This event is generated when client failed to connect to the server.
- #define ATT_CLIENT_SESSION_EVT_TRAN_TIMEOUT 7

  This event is generated if operation (find, read, write) timed out.
- #define ATT_CLIENT_EVT_EXCHANGE_MTU_RESPONSE 10

  This event is generated when client received a response (either positive or negative) to a "exchange MTU" request.
- #define ATT_CLIENT_EVT_FIND_INFO_RESPONSE 11

  This event is generated when client received a response (either positive or negative) to a "find info" request.
- #define ATT_CLIENT_EVT_FIND_BY_TYPE_VALUE_RESPONSE 12

  This event is generated when client received a response (either positive or negative) to a "find by type value" request.
- #define ATT_CLIENT_EVT_READ_BY_TYPE_RESPONSE 13

  This event is generated when client received a response (either positive or negative) to a "read by type" request.
- #define ATT_CLIENT_EVT_READ_RESPONSE 14

  This event is generated when client received a response (either positive or negative) to a "read" request.
- #define ATT_CLIENT_EVT_READ_BLOB_RESPONSE 15

  This event is generated when client received a response (either positive or negative) to a "read blob" request.
- #define ATT_CLIENT_EVT_READ_MULTIPLE_RESPONSE 16

  This event is generated when client received a response (either positive or negative) to a "read multiple" request.
- #define ATT_CLIENT_EVT_READ_BY_GROUP_TYPE_RESPONSE 17

  This event is generated when client received a response (either positive or negative) to a "read by group type" request.
- #define ATT_CLIENT_EVT_WRITE_RESPONSE 18

  This event is generated when client received a response (either positive or negative) to a "write" request.
- #define ATT_CLIENT_EVT_PREPARE_WRITE_RESPONSE 19

  This event is generated when client received a response (either positive or negative) to a "preapare write" request.
- #define ATT_CLIENT_EVT_EXECUTE_WRITE_RESPONSE 20

  This event is generated when client received a response (either positive or negative) to a "execute write" request.
- #define ATT_CLIENT_EVT_VALUE_NOTIFICATION 21

  This event is generated when client received attribute value notification.
- #define ATT_CLIENT_EVT_VALUE_INDICATION 22

  This event is generated when client received attribute value indication.
- #define ATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST 50

This event is generated when client received a "connection parameter update" request.

- #define ATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED 51

  This event is generated after the new connection parameters have been set.

3.21.1 Detailed Description

This module describes functions and data structures used to implement an ATT client (central).

3.21.2 Function Documentation

3.21.2.1 bt_bool bt_att_client_accept_conn_param_update (bt_att_client_session_t ∗ *session,* bt_uint *min_interval,* bt_uint *max_interval,* bt_uint *slave_latency,* bt_uint *supervision_timeout,* bt_uint *min_ce_length,* bt_uint *max_ce_length*)

Accept connection parameters update.

When a server sends "connection parameters update" request a ATT_CLIENT_EVT_CONN_PARAM_UPDATE↩_REQUEST event is generated. Client has to either accept the request or deny it. `bt_att_client_accept↩_conn_param_update` is used to accept the request.

Parameters

session	ATT session.
min_interval	Minimum connection interval expressed in 1.25ms units.
max_interval	Maximum connection interval expressed in 1.25ms units.
slave_latency	Slave latency expressed as number of connection events.
supervision_↩ timeout	Link supervision timeout expressed in 10ms units.
min_ce_length	Information parameter about the minimum length of connection needed for this LE connection expressed in 0.625ms units.
max_ce_length	Information parameter about the maximum length of connection needed for this LE connection expressed in 0.625ms units.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.2 bt_att_client_session_t∗ bt_att_client_allocate_session (bt_att_client_session_callback_fp *callback,* void ∗ *callback_param*)

Allocate ATT client session.

This function allocates a new ATT client session.

Parameters

callback	The callback function that will be called when the ATT client generates an event.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

Returns

- A pointer to the new ATT client session structure if the function succeeds.
- `NULL` otherwise.

3.21.2.3 bt_bool bt_att_client_connect (bt_att_client_session_t ∗ *session,* bt_bdaddr_t ∗ *addr*)

Connect to a remote device (peripheral).

This function establishes a connection to a remote device specified by the `addr`. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the ATT client callback. The events generated will either be ATT_CLIENT_SESSION_EVT_SESSION_CONNECTED or ATT_CLIENT_SES←SION_EVT_CONNECTION_FAILED.

Parameters

session	ATT session.
addr	The address of a remote device.

Returns

- `TRUE` if connection establishment has been started.
- `FALSE` otherwise.

3.21.2.4 bt_bool bt_att_client_disconnect (bt_att_client_session_t ∗ *session*)

Disconnect from a remote device.

This function closes a connection to a remote device. After the connection has been terminated the ATT client callback is called with ATT_CLIENT_SESSION_EVT_SESSION_CONNECTED event.

Parameters

session	ATT session.

Returns

- `TRUE` if disconnection has been started.
- `FALSE` otherwise. No events will be generated.

3.21.2.5 bt_bool bt_att_client_exchange_mtu (bt_att_client_session_t ∗ *session,* bt_uint *mtu*)

Exchange MTU.

This function informs the server about the client's MTU. In response to the "exchange MTU" request the server sends its MTU to the client.

Parameters

session	ATT session.
mtu	Client's MTU.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.6 bt_bool bt_att_client_execute_write (bt_att_client_session_t ∗ *session,* bt_byte *flags*)

Execute Write.

This function sends a "execute write request to the client. ATT_CLIENT_EVT_EXECUTE_WRITE_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
flags	This can be one of the following values: • 0x00 - Cancel all prepared writes. • 0x01 - Immediately write all pending prepared values.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.7 bt_bool bt_att_client_find_by_type_value (bt_att_client_session_t ∗ *session*, bt_uint *start_handle*, bt_uint *end_handle*, bt_uint *type*, const bt_byte ∗ *value*, bt_uint *len*, bt_att_find_by_type_value_response_t ∗ *result*, bt_uint *max_results*)

Find By Type Value.

This function sends a "find by type value request to the client. ATT_CLIENT_EVT_FIND_BY_TYPE_VALUE_RE↩SPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
type	Attribute type.
value	Attribute value to find.
len	Length of the `value`.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that `result` can store.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.8 bt_bool bt_att_client_find_info (bt_att_client_session_t ∗ *session*, bt_uint *start_handle*, bt_uint *end_handle*, bt_att_find_info_response_t ∗ *result*, bt_uint *max_results*)

Find Information.

This function sends a "find information request to the client. ATT_CLIENT_EVT_FIND_INFO_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.

result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that `result` can store.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.9 void bt_att_client_free_session (bt_att_client_session_t ∗ *session*)

Destroy ATT client session.

This function frees memory used by the session. The session has to exist and be in the "idle" state for this function to succeed. I.e. the session has to be disconnected before this function can be called.

Parameters

session	ATT session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.21.2.10 bt_bdaddr_t∗ bt_att_client_get_remote_address (const bt_att_client_session_t ∗ *session*)

Get the address of the remote device this device is connected to.

Parameters

session	ATT client session.

Returns

- `A` pointer to bt_bdaddr structure that contains the address of the remote device.

3.21.2.11 bt_bool bt_att_client_init (void)

Initialize the ATT client.

This function initializes the ATT client of the stack. It must be called prior to any other ATT client function can be called.

3.21.2.12 bt_bool bt_att_client_prepare_write (bt_att_client_session_t ∗ *session,* bt_uint *handle,* bt_uint *value_offset,* const bt_byte ∗ *value,* bt_uint *len*)

Prepare Write.

This function sends a "prepare write request to the client. ATT_CLIENT_EVT_PREPARE_WRITE_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
handle	The handle of the attribute to be written.
value_offset	The offset of the first octet to be written.
value	The value to be written to the attribute.
len	The length of the `value`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.13 bt_bool bt_att_client_read (bt_att_client_session_t ∗ *session*, bt_uint *handle*)

Read Attribute value.

This function sends a "read request to the client. ATT_CLIENT_EVT_READ_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
handle	The handle of the attribute to be read.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.14 bt_bool bt_att_client_read_blob (bt_att_client_session_t ∗ *session*, bt_uint *handle*, bt_uint *offset*)

Read Blob.

This function sends a "read blob request to the client. ATT_CLIENT_EVT_READ_BLOB_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
handle	The handle of the attribute to be read.
offset	The offset of the first octet to be read.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.15 bt_bool bt_att_client_read_by_group_type (bt_att_client_session_t ∗ *session*, bt_uint *start_handle*, bt_uint *end_handle*, bt_uint *group_type*, bt_att_read_by_group_type_response_t ∗ *result*, bt_uint *max_results*)

Read By Group Type (16-bit).

This function sends a "read by group type request to the client. ATT_CLIENT_EVT_READ_BY_GROUP_TYPE_↩ RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
group_type	16-bit Attribute Group Type.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that `result` can store.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.16 bt_bool bt_att_client_read_by_group_type_80 (bt_att_client_session_t ∗ *session,* bt_uint *start_handle,* bt_uint *end_handle,* bt_uuid_t ∗ *group_type,* bt_att_read_by_group_type_response_t ∗ *result,* bt_uint *max_results*)

Read By Group Type (128-bit).

This function sends a "read by group type request to the client. ATT_CLIENT_EVT_READ_BY_GROUP_TYPE_↩ RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
group_type	128-bit Attribute Group Type.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that `result` can store.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.17 bt_bool bt_att_client_read_by_type (bt_att_client_session_t ∗ *session,* bt_uint *start_handle,* bt_uint *end_handle,* bt_uint *type,* bt_att_read_by_type_response_t ∗ *result,* bt_uint *max_results*)

Read By Type (16-bit).

This function sends a "read by type request to the client. ATT_CLIENT_EVT_READ_BY_TYPE_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
type	16-bit attribute type.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that `result` can store.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.18 bt_bool bt_att_client_read_by_type_80 (bt_att_client_session_t ∗ *session,* bt_uint *start_handle,* bt_uint *end_handle,* bt_uuid_t ∗ *type,* bt_att_read_by_type_response_t ∗ *result,* bt_uint *max_results*)

Read By Type (128-bit).

This function sends a "read by type request to the client. ATT_CLIENT_EVT_READ_BY_TYPE_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
type	128-bit attribute type.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that `result` can store.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.19 bt_bool bt_att_client_read_multiple (bt_att_client_session_t ∗ *session,* const bt_uint ∗ *handles,* bt_uint *count*)

Read Multiple.

This function sends a "read blob request to the client. ATT_CLIENT_EVT_READ_MULTIPLE_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
handles	The list of two or more handles.
count	The number of handles in `handles`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.20 void bt_att_client_reject_conn_param_update (bt_att_client_session_t ∗ *session*)

Reject connection parameters update.

When a server sends "connection parameters update" request a ATT_CLIENT_EVT_CONN_PARAM_UPDATE↩ _REQUEST event is generated. Client has to either accept the request or deny it. `bt_att_client_reject↩ _conn_param_update` is used to deny the request.

Parameters

session	ATT session.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.21 bt_bool bt_att_client_signed_write (bt_att_client_session_t ∗ *session*, bt_uint *handle*, const bt_byte ∗ *value*, bt_uint *len*, const bt_byte ∗ *signature*)

Signed Write Command.

This function sends a "signed write command request to the client. No event is generated.

Parameters

session	ATT session.
handle	The handle of the attribute to be written.
value	The value to be written to the attribute.
len	The length of the `value`.
signature	Authentication signature (12 bytes).

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.22 bt_bool bt_att_client_write (bt_att_client_session_t ∗ *session*, bt_uint *handle*, const bt_byte ∗ *value*, bt_uint *len*)

Write Attribute Value.

This function sends a "write request to the client. ATT_CLIENT_EVT_WRITE_RESPONSE event is generated when the response from the client has been received.

Parameters

session	ATT session.
handle	The handle of the attribute to be written.
value	The value to be written to the attribute.
len	The length of the `value`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.23 bt_bool bt_att_client_write_cmd (bt_att_client_session_t ∗ *session*, bt_uint *handle*, const bt_byte ∗ *value*, bt_uint *len*)

Write Command.

This function sends a "write command request to the client. No event is generated.

Parameters

session	ATT session.
handle	The handle of the attribute to be written.
value	The value to be written to the attribute.
len	The length of the `value`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.21.2.24 bt_att_client_mgr_t* bt_att_get_client_mgr (void)

Return a pointer to an instance of the ATT client manager.

This function returns a pointer to an instance of the ATT client manager. There is only one instance of the manager allocated by the stack.

Returns

- A pointer to the ATT client manager.

3.22 Configuration

This module describes parameters used to configure ATT client layer.

Macros

- #define ATT_CLIENT_MAX_SESSIONS

  Maximum number of ATT sessions.

3.22.1 Detailed Description

This module describes parameters used to configure ATT client layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS            ...
#define HCI_MAX_DATA_BUFFERS           ...
#define HCI_MAX_HCI_CONNECTIONS        ...
#define HCI_RX_BUFFER_LEN              ...
#define HCI_TX_BUFFER_LEN              ...
#define HCI_L2CAP_BUFFER_LEN           ...
#define HCI_MAX_CMD_PARAM_LEN          ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS          ...
#define L2CAP_MAX_FRAME_BUFFERS        ...
#define L2CAP_MAX_PSMS                 ...
#define L2CAP_MAX_CHANNELS             ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN     ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN   ...

// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure ATT,
// the following values must be defined:

#define BT_INCLUDE_ATT               // tells dotstack to compile in ATT support
#define ATT_CLIENT_MAX_SESSIONS      ...

#include "cdbt/bt/bt_oem_config.h"
```

3.22.2 Macro Definition Documentation

3.22.2.1 #define ATT_CLIENT_MAX_SESSIONS

Maximum number of ATT sessions.

This parameter defines the maximum number of connections a client can have with servers. The minimum value is 1. The maximum value is 7.

3.23 GATT

The GATT profile is designed to be used by an application or another profile, so that a client can communicate with a server.

Modules

- GATT Server

  This module describes functions and data structures used to implement a GATT server (peripheral).
- GATT Client

  This module describes functions and data structures used to implement a GATT client (central).

3.23.1 Detailed Description

The GATT profile is designed to be used by an application or another profile, so that a client can communicate with a server.

The server contains a number of attributes, and the GATT Profile defines how to use the Attribute Protocol to discover, read, write and obtain indications of these attributes, as well as configuring broadcast of attributes.

The Generic Attribute Profile (GATT) defines a service framework using the Attribute Protocol. This framework defines procedures and formats of services and their characteristics. The procedures defined include discovering, reading, writing, notifying and indicating characteristics, as well as configuring the broadcast of characteristics.

3.24 GATT Server

This module describes functions and data structures used to implement a GATT server (peripheral).

Data Structures

- struct bt_gatt_evt_header_t

  Common to all event parameters header.
- struct bt_gatt_evt_client_config_changed_t

  Parameter to GATT_SERVER_EVT_CLIENT_CONFIG_CHANGED event.
- struct bt_gatt_evt_server_config_changed_t

  Parameter to GATT_SERVER_EVT_SERVER_CONFIG_CHANGED event.
- struct bt_gatt_evt_ext_properties_changed_t

  Parameter to GATT_SERVER_EVT_EXTENDED_PROPERTIES_CHANGED event.
- struct bt_gatt_evt_value_changed_t

  Parameter to GATT_SERVER_EVT_VALUE_CHANGED event.
- struct bt_gatt_evt_value_read_t

  Parameter to GATT_SERVER_EVT_VALUE_READ event.

Macros

- #define bt_gatt_server_start(att_var_db, att_var_db_len, att_const_db, att_const_db_len) bt_gatt_server_↵ start_ex(att_var_db, att_var_db_len, att_const_db, att_const_db_len, BT_FALSE)

  Start GATT server.
- #define bt_gatt_server_authorize_access(session, attr, opcode, authorize) bt_att_server_authorize_↵ access(session, attr, opcode, authorize)

  Authorize access to an attribute.

Functions

- bt_bool bt_gatt_server_start_ex (bt_byte ∗att_var_db, bt_uint att_var_db_len, const bt_byte ∗att_const_db, bt_uint att_const_db_len, bt_bool listen_on_dynamic_channel)

  Start GATT server (extended version)
- void bt_gatt_server_register_callback (bt_gatt_server_callback_fp callback, void ∗param)

  Register a GATT application callback.
- bt_bool bt_gatt_server_register_listener (bt_gatt_listener_t ∗listener)

  Register a GATT event listener.
- void bt_gatt_server_unregister_listener (bt_gatt_listener_t ∗listener)

  Unregister a GATT event listener.
- bt_byte bt_gatt_server_write_char_value (bt_uuid16 service_type, bt_uuid16 service_id, bt_uuid16 characteristic_type, const bt_byte ∗value, bt_uint value_len, bt_uint offset)

  Write characteristic's value (16-bit).
- bt_byte bt_gatt_server_write_char_value_80 (bt_uuid_t ∗service_type, bt_uuid_t ∗service_id, bt_uuid_↵ t ∗characteristic_type, const bt_byte ∗value, bt_uint value_len, bt_uint offset)

  Write characteristic's value (128-bit).
- bt_byte bt_gatt_server_notify_char_value (bt_uuid16 service_type, bt_uuid16 service_id, bt_uuid16 characteristic_type)

  Notify characteristic's value (16-bit).
- bt_byte bt_gatt_server_notify_char_value_80 (bt_uuid_t ∗service_type, bt_uuid_t ∗service_id, bt_uuid_↵ t ∗characteristic_type)

  Notify characteristic's value (128-bit).

- bt_byte bt_gatt_server_indicate_char_value (bt_uuid16 service_type, bt_uuid16 service_id, bt_uuid16 characteristic_type)

  Indicate characteristic's value (16-bit).
- bt_byte bt_gatt_server_indicate_char_value_80 (bt_uuid_t ∗service_type, bt_uuid_t ∗service_id, bt_uuid_t ∗characteristic_type)

  Indicate characteristic's value (128-bit).

Events

- #define GATT_SERVER_EVT_SESSION_CONNECTED ATT_SERVER_EVT_SESSION_CONNECTED

  A client connected to the server.
- #define GATT_SERVER_EVT_SESSION_DISCONNECTED ATT_SERVER_EVT_SESSION_DISCONN←ECTED

  A client disconnected from the server.
- #define GATT_SERVER_EVT_ATTR_VALUE_NOTIFIED ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED

  A value notification has bee sent to the client.
- #define GATT_SERVER_EVT_ATTR_VALUE_INDICATED ATT_SERVER_EVT_ATTR_VALUE_INDICA←TED

  A value indication has been sent to the client.
- #define GATT_SERVER_EVT_AUTHORIZATION_REQUESTED ATT_SERVER_EVT_AUTHORIZATIO←N_REQUESTED

  Authorization is required in order to access the attribute's value.
- #define GATT_SERVER_EVT_CLIENT_CONFIG_CHANGED 100

  Characteristic's client configuration changed.
- #define GATT_SERVER_EVT_EXTENDED_PROPERTIES_CHANGED 101

  Characteristic's extended properties changed.
- #define GATT_SERVER_EVT_SERVER_CONFIG_CHANGED 102

  Characteristic's server configuration changed.
- #define GATT_SERVER_EVT_VALUE_CHANGED 103

  Characteristic's value changed.
- #define GATT_SERVER_EVT_VALUE_READ 104

  Characteristic's value has been read.

3.24.1 Detailed Description

This module describes functions and data structures used to implement a GATT server (peripheral).

3.24.2 Macro Definition Documentation

3.24.2.1 #define bt_gatt_server_authorize_access(*session,* *attr,* *opcode,* *authorize*) bt_att_server_authorize_access(session, attr, opcode, authorize)

Authorize access to an attribute.

If an attribute requires authorization before its value can be read or written, GATT generates a GATT_SERVER_←EVT_AUTHORIZATION_REQUESTED event. In response to this event the application should obtain authorization (how this is done is implementation specific) and call bt_gatt_server_authorize_access.

Parameters

session	ATT session.
attr	Attribute.
opcode	Attribute Opcode. The opcode that requires authorization is passed in the ATT_SERV↩ ER_EVT_AUTHORIZATION_REQUESTED event. The application should use the passed opcode when calling bt_att_server_authorize_access. The opcode can be one the following values: • ATT_OPCODE_READ_REQUEST • ATT_OPCODE_READ_BLOB_REQUEST • ATT_OPCODE_READ_BY_TYPE_REQUEST • ATT_OPCODE_READ_BY_GROUP_TYPE_REQUEST • ATT_OPCODE_READ_MULTIPLE_REQUEST • ATT_OPCODE_WRITE_REQUEST • ATT_OPCODE_PREPARE_WRITE_REQUEST
authorize	Specifies whether access to the attribute has been authorized or not.

3.24.2.2 #define bt_gatt_server_start(*att_var_db, att_var_db_len, att_const_db, att_const_db_len*) bt_gatt_server_start_ex(att_var_db, att_var_db_len, att_const_db, att_const_db_len, BT_FALSE)

Start GATT server.

This function starts the GATT server. This function start the server only on LE links.

Parameters

att_var_db	A pointer to ATT database that holds writable attributes.
att_var_db_len	The length of the writable ATT database.
att_const_db	A pinter to ATT database that holds read-only attributes.
att_const_db_↩ len	The length of the read-only ATT database.
listen_on_↩ dynamic_↩ channel	Specifies whether ATT server should accept connections on BR/EDR links.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.24.3 Function Documentation

3.24.3.1 bt_byte bt_gatt_server_indicate_char_value (bt_uuid16 *service_type,* bt_uuid16 *service_id,* bt_uuid16 *characteristic_type*)

Indicate characteristic's value (16-bit).

This function sends current characteristic's value to the client. The value is sent regardless of characteristic's client configuration. GATT_SERVER_EVT_ATTR_VALUE_INDICATED event is generated after receiving confirmation from the client.

Parameters

service_type	16-bit service type UUID.
service_id	16-bit service type UUID.
characteristic_↩ type	16-bit characteristic type UUID.

Returns

Error code. This can be one of the following values:

- ATT_ERROR_SUCCESS
- ATT_ERROR_INVALID_HANDLE
- ATT_ERROR_READ_NOT_PERMITTED
- ATT_ERROR_WRITE_NOT_PERMITTED
- ATT_ERROR_INVALID_PDU
- ATT_ERROR_INSUFFICIENT_AUTHENTICATION
- ATT_ERROR_REQUEST_NOT_SUPPORTED
- ATT_ERROR_INVALID_OFFSET
- ATT_ERROR_INSUFFICIENT_AUTHORIZATION
- ATT_ERROR_PREPARE_QUEUE_FULL
- ATT_ERROR_ATTRIBUTE_NOT_FOUND
- ATT_ERROR_ATTRIBUTE_NOT_LONG
- ATT_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE
- ATT_ERROR_INVALID_ATTRIBUTE_VALUE_LENGTH
- ATT_ERROR_UNLIKELY_ERROR
- ATT_ERROR_INSUFFICIENT_ENCRYPTION
- ATT_ERROR_UNSUPPORTED_GROUP_TYPE
- ATT_ERROR_INSUFFICIENT_RESOURCES

3.24.3.2 bt_byte bt_gatt_server_indicate_char_value_80 (bt_uuid_t ∗ *service_type,* bt_uuid_t ∗ *service_id,* bt_uuid_t ∗ *characteristic_type*)

Indicate characteristic's value (128-bit).

This function sends current characteristic's value to the client. The value is sent regardless of characteristic's client configuration. GATT_SERVER_EVT_ATTR_VALUE_INDICATED event is generated after receiving confirmation from the client. In order to use standard 16-bit UUIDs with this function they have to be converted to 128-bit UUID by combining with a base UUID which is 00000000 (this part is replaced with 16-bit UUID) - 00001000 - 80000080 - 5F9B34FB.

Parameters

service_type	128-bit service type UUID.
service_id	128-bit service type UUID.
characteristic_↩ type	128-bit characteristic type UUID.

Returns

Error code. This can be one of the following values:

- ATT_ERROR_SUCCESS
- ATT_ERROR_INVALID_HANDLE

- ATT_ERROR_READ_NOT_PERMITTED
- ATT_ERROR_WRITE_NOT_PERMITTED
- ATT_ERROR_INVALID_PDU
- ATT_ERROR_INSUFFICIENT_AUTHENTICATION
- ATT_ERROR_REQUEST_NOT_SUPPORTED
- ATT_ERROR_INVALID_OFFSET
- ATT_ERROR_INSUFFICIENT_AUTHORIZATION
- ATT_ERROR_PREPARE_QUEUE_FULL
- ATT_ERROR_ATTRIBUTE_NOT_FOUND
- ATT_ERROR_ATTRIBUTE_NOT_LONG
- ATT_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE
- ATT_ERROR_INVALID_ATTRIBUTE_VALUE_LENGTH
- ATT_ERROR_UNLIKELY_ERROR
- ATT_ERROR_INSUFFICIENT_ENCRYPTION
- ATT_ERROR_UNSUPPORTED_GROUP_TYPE
- ATT_ERROR_INSUFFICIENT_RESOURCES

3.24.3.3 bt_byte bt_gatt_server_notify_char_value (bt_uuid16 *service_type,* bt_uuid16 *service_id,* bt_uuid16 *characteristic_type*)

Notify characteristic's value (16-bit).

This function sends current characteristic's value to the client. The value is sent regardless of characteristic's client configuration. GATT_SERVER_EVT_ATTR_VALUE_NOTIFIED is generated right after the value has been sent.

Parameters

service_type	16-bit service type UUID.
service_id	16-bit service type UUID.
characteristic_↩ type	16-bit characteristic type UUID.

Returns

Error code. This can be one of the following values:

- ATT_ERROR_SUCCESS
- ATT_ERROR_INVALID_HANDLE
- ATT_ERROR_READ_NOT_PERMITTED
- ATT_ERROR_WRITE_NOT_PERMITTED
- ATT_ERROR_INVALID_PDU
- ATT_ERROR_INSUFFICIENT_AUTHENTICATION
- ATT_ERROR_REQUEST_NOT_SUPPORTED
- ATT_ERROR_INVALID_OFFSET
- ATT_ERROR_INSUFFICIENT_AUTHORIZATION
- ATT_ERROR_PREPARE_QUEUE_FULL
- ATT_ERROR_ATTRIBUTE_NOT_FOUND
- ATT_ERROR_ATTRIBUTE_NOT_LONG
- ATT_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE
- ATT_ERROR_INVALID_ATTRIBUTE_VALUE_LENGTH
- ATT_ERROR_UNLIKELY_ERROR
- ATT_ERROR_INSUFFICIENT_ENCRYPTION
- ATT_ERROR_UNSUPPORTED_GROUP_TYPE
- ATT_ERROR_INSUFFICIENT_RESOURCES

3.24.3.4 bt_byte bt_gatt_server_notify_char_value_80 (bt_uuid_t ∗ *service_type,* bt_uuid_t ∗ *service_id,* bt_uuid_t ∗ *characteristic_type*)

Notify characteristic's value (128-bit).

This function sends current characteristic's value to the client. The value is sent regardless of characteristic's client configuration. GATT_SERVER_EVT_ATTR_VALUE_NOTIFIED is generated right after the value has been sent. In order to use standard 16-bit UUIDs with this function they have to be converted to 128-bit UUID by combining with a base UUID which is 00000000 (this part is replaced with 16-bit UUID) - 00001000 - 80000080 - 5F9B34FB.

Parameters

service_type	128-bit service type UUID.
service_id	128-bit service type UUID.
characteristic_↩ type	128-bit characteristic type UUID.

Returns

Error code. This can be one of the following values:

- ATT_ERROR_SUCCESS
- ATT_ERROR_INVALID_HANDLE
- ATT_ERROR_READ_NOT_PERMITTED
- ATT_ERROR_WRITE_NOT_PERMITTED
- ATT_ERROR_INVALID_PDU
- ATT_ERROR_INSUFFICIENT_AUTHENTICATION
- ATT_ERROR_REQUEST_NOT_SUPPORTED
- ATT_ERROR_INVALID_OFFSET
- ATT_ERROR_INSUFFICIENT_AUTHORIZATION
- ATT_ERROR_PREPARE_QUEUE_FULL
- ATT_ERROR_ATTRIBUTE_NOT_FOUND
- ATT_ERROR_ATTRIBUTE_NOT_LONG
- ATT_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE
- ATT_ERROR_INVALID_ATTRIBUTE_VALUE_LENGTH
- ATT_ERROR_UNLIKELY_ERROR
- ATT_ERROR_INSUFFICIENT_ENCRYPTION
- ATT_ERROR_UNSUPPORTED_GROUP_TYPE
- ATT_ERROR_INSUFFICIENT_RESOURCES

3.24.3.5 void bt_gatt_server_register_callback (bt_gatt_server_callback_fp *callback,* void ∗ *param*)

Register a GATT application callback.

In order to be notified of various events a consumer of the GATT layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the first parameter. The event can be one of the following values:

- GATT_SERVER_EVT_SESSION_CONNECTED A client connected to the server.
- GATT_SERVER_EVT_SESSION_DISCONNECTED A client disconnected from the server.
- GATT_SERVER_EVT_ATTR_VALUE_NOTIFIED A value notification has bee sent to the client.
- GATT_SERVER_EVT_ATTR_VALUE_INDICATED A value indication has been sent to the client.

- GATT_SERVER_EVT_AUTHORIZATION_REQUESTED Authorization is required in order to access the attribute's value.
- GATT_SERVER_EVT_CLIENT_CONFIG_CHANGED Characteristic's client configuration changed
- GATT_SERVER_EVT_EXTENDED_PROPERTIES_CHANGED Characteristic's extended properties changed
- GATT_SERVER_EVT_SERVER_CONFIG_CHANGED Characteristic's server configuration changed
- GATT_SERVER_EVT_VALUE_CHANGED Characteristic's value changed

Parameters

callback	The callback function that will be called when GATT generates an event.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.24.3.6 bt_bool bt_gatt_server_register_listener (bt_gatt_listener_t ∗ *listener*)

Register a GATT event listener.

In order to be notified of various events a consumer of the GATT layer may register an event listener. The listener is a structure that contains a pointer to a callback and callback parameter - a pointer to arbitrary data to be passed to the callback . The stack will call the callback whenever a new event has been generated passing the code of the event as the first parameter. The event can be one of the following values:

- GATT_SERVER_EVT_SESSION_CONNECTED A client connected to the server.
- GATT_SERVER_EVT_SESSION_DISCONNECTED A client disconnected from the server.
- GATT_SERVER_EVT_ATTR_VALUE_NOTIFIED A value notification has bee sent to the client.
- GATT_SERVER_EVT_ATTR_VALUE_INDICATED A value indication has been sent to the client.
- GATT_SERVER_EVT_AUTHORIZATION_REQUESTED Authorization is required in order to access the attribute's value.
- GATT_SERVER_EVT_CLIENT_CONFIG_CHANGED Characteristic's client configuration changed
- GATT_SERVER_EVT_EXTENDED_PROPERTIES_CHANGED Characteristic's extended properties changed
- GATT_SERVER_EVT_SERVER_CONFIG_CHANGED Characteristic's server configuration changed
- GATT_SERVER_EVT_VALUE_CHANGED Characteristic's value changed

Parameters

listener	Listener.

3.24.3.7 bt_bool bt_gatt_server_start_ex (bt_byte ∗ *att_var_db,* bt_uint *att_var_db_len,* const bt_byte ∗ *att_const_db,* bt_uint *att_const_db_len,* bt_bool *listen_on_dynamic_channel*)

Start GATT server (extended version)

This function starts the GATT server. This function can be used to start the server on BR/EDR links.

Parameters

att_var_db	A pointer to ATT database that holds writable attributes.
att_var_db_len	The length of the writable ATT database.
att_const_db	A pinter to ATT database that holds read-only attributes.
att_const_db_↩ len	The length of the read-only ATT database.
listen_on_↩ dynamic_↩ channel	Specifies whether ATT server should accept connections on BR/EDR links.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.24.3.8 void bt_gatt_server_unregister_listener (bt_gatt_listener_t ∗ *listener*)

Unregister a GATT event listener.

Remove a listener previosuly registered with bt_gatt_server_register_listener.

Parameters

listener	Listener.

3.24.3.9 bt_byte bt_gatt_server_write_char_value (bt_uuid16 *service_type,* bt_uuid16 *service_id,* bt_uuid16 *characteristic_type,* const bt_byte ∗ *value,* bt_uint *value_len,* bt_uint *offset*)

Write characteristic's value (16-bit).

This function updates characteristic's value. If the client configured characteristic to be notified or indicated this function sends new value to the client.

Parameters

service_type	16-bit service type UUID.
service_id	16-bit service type UUID.
characteristic_↩ type	16-bit characteristic type UUID.

Returns

Error code. This can be one of the following values:

- ATT_ERROR_SUCCESS
- ATT_ERROR_INVALID_HANDLE
- ATT_ERROR_READ_NOT_PERMITTED
- ATT_ERROR_WRITE_NOT_PERMITTED
- ATT_ERROR_INVALID_PDU
- ATT_ERROR_INSUFFICIENT_AUTHENTICATION
- ATT_ERROR_REQUEST_NOT_SUPPORTED
- ATT_ERROR_INVALID_OFFSET
- ATT_ERROR_INSUFFICIENT_AUTHORIZATION
- ATT_ERROR_PREPARE_QUEUE_FULL
- ATT_ERROR_ATTRIBUTE_NOT_FOUND

- ATT_ERROR_ATTRIBUTE_NOT_LONG
- ATT_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE
- ATT_ERROR_INVALID_ATTRIBUTE_VALUE_LENGTH
- ATT_ERROR_UNLIKELY_ERROR
- ATT_ERROR_INSUFFICIENT_ENCRYPTION
- ATT_ERROR_UNSUPPORTED_GROUP_TYPE
- ATT_ERROR_INSUFFICIENT_RESOURCES

3.24.3.10 bt_byte bt_gatt_server_write_char_value_80 (bt_uuid_t ∗ *service_type,* bt_uuid_t ∗ *service_id,* bt_uuid_t ∗ *characteristic_type,* const bt_byte ∗ *value,* bt_uint *value_len,* bt_uint *offset*)

Write characteristic's value (128-bit).

This function updates characteristic's value. If the client configured characteristic to be notified or indicated this function sends new value to the client. In order to use standard 16-bit UUIDs with this function they have to be converted to 128-bit UUID by combining with a base UUID which is 00000000 (this part is replaced with 16-bit UUID) - 00001000 - 80000080 - 5F9B34FB.

Parameters

service_type	128-bit service type UUID.
service_id	128-bit service type UUID.
characteristic_↩ type	128-bit characteristic type UUID.

Returns

Error code. This can be one of the following values:

- ATT_ERROR_SUCCESS
- ATT_ERROR_INVALID_HANDLE
- ATT_ERROR_READ_NOT_PERMITTED
- ATT_ERROR_WRITE_NOT_PERMITTED
- ATT_ERROR_INVALID_PDU
- ATT_ERROR_INSUFFICIENT_AUTHENTICATION
- ATT_ERROR_REQUEST_NOT_SUPPORTED
- ATT_ERROR_INVALID_OFFSET
- ATT_ERROR_INSUFFICIENT_AUTHORIZATION
- ATT_ERROR_PREPARE_QUEUE_FULL
- ATT_ERROR_ATTRIBUTE_NOT_FOUND
- ATT_ERROR_ATTRIBUTE_NOT_LONG
- ATT_ERROR_INSUFFICIENT_ENCRYPTION_KEY_SIZE
- ATT_ERROR_INVALID_ATTRIBUTE_VALUE_LENGTH
- ATT_ERROR_UNLIKELY_ERROR
- ATT_ERROR_INSUFFICIENT_ENCRYPTION
- ATT_ERROR_UNSUPPORTED_GROUP_TYPE
- ATT_ERROR_INSUFFICIENT_RESOURCES

3.25 GATT Client

This module describes functions and data structures used to implement a GATT client (central).

Modules

- Configuration

  This module describes parameters used to configure GATT client.

Data Structures

- struct bt_att_client_uuid_t

  UUID.
- struct bt_gatt_client_service_definition_t

  Service Definition.
- struct bt_gatt_client_inc_service_declaration_t

  Included Service Declaration.
- struct bt_gatt_client_char_declaration_t

  Characteristic Declaration.
- struct bt_gatt_client_char_value_t

  Characteristic Value.
- struct bt_gatt_client_char_descriptor_t

  Characteristic Descriptor Declaration.
- struct bt_gatt_client_evt_exchange_mtu_completed_t

  Parameter to GATT_CLIENT_EVT_EXCHANGE_MTU_COMPLETED event.
- struct bt_gatt_client_evt_discover_all_services_completed_t

  Parameter to GATT_CLIENT_EVT_DISCOVER_ALL_SERVICES_COMPLETED event.
- struct bt_gatt_client_evt_discover_service_by_uuid_completed_t

  Parameter to GATT_CLIENT_EVT_DISCOVER_SERVICE_BY_UUID_COMPLETED event.
- struct bt_gatt_client_evt_find_included_completed_t

  Parameter to GATT_CLIENT_EVT_FIND_INCLUDED_SERVICES_COMPLETED event.
- struct bt_gatt_client_evt_discover_all_chars_completed_t

  Parameter to GATT_CLIENT_EVT_DISCOVER_ALL_CHARS_COMPLETED event.
- struct bt_gatt_client_evt_discover_char_by_uuid_completed_t

  Parameter to GATT_CLIENT_EVT_DISCOVER_CHAR_BY_UUID_COMPLETED event.
- struct bt_gatt_client_evt_read_char_value_completed_t

  Parameter to GATT_CLIENT_EVT_READ_CHAR_VALUE_COMPLETED event.
- struct bt_gatt_client_evt_read_by_char_uuid_completed_t

  Parameter to GATT_CLIENT_EVT_READ_USING_CHAR_UUID_COMPLETED event.
- struct bt_gatt_client_evt_read_multiple_char_values_completed_t

  Parameter to GATT_CLIENT_EVT_READ_MULTIPLE_CHAR_VALUES_COMPLETED event.
- struct bt_gatt_client_evt_read_char_long_value_completed_t

  Parameter to GATT_CLIENT_EVT_READ_CHAR_LONG_VALUE_COMPLETED event.
- struct bt_gatt_client_evt_write_char_value_completed_t

  Parameter to GATT_CLIENT_EVT_WRITE_CHAR_VALUE_COMPLETED event.
- struct bt_gatt_client_evt_write_char_long_value_completed_t

  Parameter to GATT_CLIENT_EVT_WRITE_CHAR_LONG_VALUE_COMPLETED event.
- struct bt_gatt_client_evt_value_notification_t

  Parameter to GATT_CLIENT_EVT_VALUE_NOTIFICATION event.
- struct bt_gatt_client_evt_discover_descriptors_completed_t

Parameter to GATT_CLIENT_EVT_DISCOVER_CHAR_DESCRIPTORS_COMPLETED event.

- struct bt_gatt_client_evt_read_char_descriptor_completed_t

  Parameter to GATT_CLIENT_EVT_READ_CHAR_DESCRIPTOR_COMPLETED event.
- struct bt_gatt_client_evt_read_char_long_descriptor_completed_t

  Parameter to GATT_CLIENT_EVT_READ_CHAR_LONG_DESCRIPTOR_COMPLETED event.
- struct bt_gatt_client_evt_write_char_descriptor_completed_t

  Parameter to GATT_CLIENT_EVT_WRITE_CHAR_DESCRIPTOR_COMPLETED event.
- struct bt_gatt_client_evt_write_char_long_descriptor_completed_t

  Parameter to GATT_CLIENT_EVT_WRITE_CHAR_LONG_DESCRIPTOR_COMPLETED event.
- struct bt_gatt_client_evt_conn_param_update_t

  Parameter to GATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST event.
- struct bt_gatt_client_evt_conn_param_update_completed_t

  Parameter to GATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED event.
- union bt_gatt_client_evt_t

  Parameter to GATT client application callback.

Macros

- #define bt_gatt_client_accept_conn_param_update(session, min_interval, max_interval, slave_latency, supervision_timeout, min_ce_length, max_ce_length) bt_att_client_accept_conn_param_update(session->att_session, min_interval, max_interval, slave_latency, supervision_timeout, min_ce_length, max_ce_↩ length)

  Accept connection parameters update.
- #define bt_gatt_client_reject_conn_param_update(session) bt_att_client_reject_conn_param_update(session->att_session)

  Reject connection parameters update.

Functions

- bt_bool bt_gatt_client_init (void)

  Initialize the GATT client.
- bt_gatt_client_mgr_t ∗ bt_gatt_get_client_mgr (void)

  Return a pointer to an instance of the GATT client manager.
- bt_gatt_client_session_t ∗ bt_gatt_client_allocate_session (bt_gatt_client_session_callback_fp callback, void ∗callback_param)

  Allocate GATT client session.
- void bt_gatt_client_free_sessions (bt_gatt_client_session_t ∗session)

  Destroy GATT client session.
- bt_bool bt_gatt_client_connect (bt_gatt_client_session_t ∗session, bt_bdaddr_t ∗addr)

  Connect to a remote device (peripheral).
- bt_bool bt_gatt_client_disconnect (bt_gatt_client_session_t ∗session)

  Disconnect from a remote device.
- bt_bool bt_gatt_exchange_mtu (bt_gatt_client_session_t ∗session, bt_uint mtu)

  Exchange MTU.
- bt_bool bt_gatt_client_discover_all_services_ex (bt_gatt_client_session_t ∗session, bt_uint start_handle, bt_gatt_client_service_definition_t ∗result, bt_uint max_results)

  Discover All Services.
- bt_bool bt_gatt_client_discover_by_service_uuid (bt_gatt_client_session_t ∗session, bt_uint service_uuid, bt_gatt_client_service_definition_t ∗result, bt_uint max_results)

  Discover Services by UUID (16-bit).

- bt_bool bt_gatt_client_find_included_services_ex (bt_gatt_client_session_t ∗session, const bt_gatt_client↩_service_definition_t ∗service, bt_uint start_handle, bt_gatt_client_inc_service_declaration_t ∗result, bt_uint max_results)

  Find Included Services.
- bt_bool bt_gatt_client_discover_by_service_uuid_80 (bt_gatt_client_session_t ∗session, bt_uuid_↩t ∗service_uuid, bt_gatt_client_service_definition_t ∗result, bt_uint max_results)

  Discover Services by UUID (128-bit).
- bt_bool bt_gatt_client_discover_all_chars_ex (bt_gatt_client_session_t ∗session, const bt_gatt_client_↩service_definition_t ∗service, bt_uint start_handle, bt_gatt_client_char_declaration_t ∗result, bt_uint max_↩results)

  Discover All Characteristics of a Service.
- bt_bool bt_gatt_client_discover_char_by_uuid (bt_gatt_client_session_t ∗session, const bt_gatt_client_↩service_definition_t ∗service, bt_uint char_uuid, bt_gatt_client_char_declaration_t ∗result, bt_uint max_↩results)

  Discover Characteristics by UUID (16-bit).
- bt_bool bt_gatt_client_discover_char_by_uuid_80 (bt_gatt_client_session_t ∗session, const bt_gatt_client↩_service_definition_t ∗service, bt_uuid_t ∗char_uuid, bt_gatt_client_char_declaration_t ∗result, bt_uint max_results)

  Discover Characteristics by UUID (128-bit).
- bt_bool bt_gatt_client_read_char_value (bt_gatt_client_session_t ∗session, const bt_gatt_client_char_↩declaration_t ∗characteristic, bt_byte ∗buffer, bt_uint len)

  Read Characteristic Value.
- bt_bool bt_gatt_client_read_by_char_uuid (bt_gatt_client_session_t ∗session, const bt_gatt_client_service↩_definition_t ∗service, bt_uint char_uuid, bt_gatt_client_char_value_t ∗result, bt_uint max_results)

  Read Using Characteristic UUID (16-bit).
- bt_bool bt_gatt_client_read_by_char_uuid_80 (bt_gatt_client_session_t ∗session, const bt_gatt_client_↩service_definition_t ∗service, bt_uuid_t ∗char_uuid, bt_gatt_client_char_value_t ∗result, bt_uint max_results)

  Read Using Characteristic UUID (128-bit).
- bt_bool bt_gatt_client_read_char_long_value (bt_gatt_client_session_t ∗session, const bt_gatt_client_↩char_declaration_t ∗characteristic, bt_uint offset, bt_byte ∗buffer, bt_uint len)

  Read Long Characteristic Value.
- bt_bool bt_gatt_client_read_multiple_char_values (bt_gatt_client_session_t ∗session, const bt_uint ∗handles, bt_uint count, bt_byte ∗buffer, bt_uint len)

  Read Multiple Characteristic Values.
- bt_bool bt_gatt_client_write_char_value (bt_gatt_client_session_t ∗session, const bt_gatt_client_char_↩declaration_t ∗characteristic, const bt_byte ∗value, bt_uint len)

  Write Characteristic Value.
- bt_bool bt_gatt_client_set_char_value (bt_gatt_client_session_t ∗session, const bt_gatt_client_char_↩declaration_t ∗characteristic, const bt_byte ∗value, bt_uint len)

  Write Without Response.
- bt_bool bt_gatt_client_write_char_long_value (bt_gatt_client_session_t ∗session, const bt_gatt_client_↩char_declaration_t ∗characteristic, bt_uint value_offset, const bt_byte ∗value, bt_uint len)

  Write Long Characteristic Value.
- bt_bool bt_gatt_client_discover_char_descriptors (bt_gatt_client_session_t ∗session, const bt_gatt_client↩_char_declaration_t ∗characteristic, bt_gatt_client_char_descriptor_t ∗result, bt_uint max_results)

  Discover All Characteristic Descriptors.
- bt_bool bt_gatt_client_read_char_descriptor (bt_gatt_client_session_t ∗session, const bt_gatt_client_char↩_descriptor_t ∗descriptor, bt_byte ∗buffer, bt_uint len)

  Read Characteristic Descriptor.
- bt_bool bt_gatt_client_read_char_long_descriptor (bt_gatt_client_session_t ∗session, const bt_gatt_client↩_char_descriptor_t ∗descriptor, bt_uint offset, bt_byte ∗buffer, bt_uint len)

  Read Long Characteristic Descriptor.
- bt_bool bt_gatt_client_write_char_descriptor (bt_gatt_client_session_t ∗session, const bt_gatt_client_char↩_descriptor_t ∗descriptor, const bt_byte ∗value, bt_uint len)

Write Characteristic Descriptor.
- bt_bool bt_gatt_client_write_char_long_descriptor (bt_gatt_client_session_t ∗session, const bt_gatt_client←_char_descriptor_t ∗descriptor, bt_uint value_offset, const bt_byte ∗value, bt_uint len)

  Write Long Characteristic Descriptor.
- bt_hci_conn_state_t ∗ bt_gatt_client_get_hci_connection (const bt_gatt_client_session_t ∗session)

  Get HCI connection for a session.

Events

- #define GATT_CLIENT_EVT_SESSION_CONNECTED 1

  This event is generated when the client connected to the server.
- #define GATT_CLIENT_EVT_SESSION_DISCONNECTED 2

  This event is generated when the client disconnected from the server.
- #define GATT_CLIENT_EVT_CONNECTION_FAILED 3

  This event is generated when the client failed to connect to the server.
- #define GATT_CLIENT_EVT_TRAN_TIMEOUT 7

  This event is generated if operation (discover, read, write) timed out.
- #define GATT_CLIENT_EVT_DISCOVER_ALL_SERVICES_COMPLETED 10

  This event is generated when the "discover all services" operation has completed.
- #define GATT_CLIENT_EVT_DISCOVER_SERVICE_BY_UUID_COMPLETED 11

  This event is generated when the "discover service by UUID" operation has completed.
- #define GATT_CLIENT_EVT_DISCOVER_ALL_CHARS_COMPLETED 12

  This event is generated when the "discover all cgaracteristics" operation has completed.
- #define GATT_CLIENT_EVT_READ_CHAR_VALUE_COMPLETED 13

  This event is generated when the "read characteristic value" operation has completed.
- #define GATT_CLIENT_EVT_READ_CHAR_LONG_VALUE_COMPLETED 14

  This event is generated when the "read long characteristic value" operation has completed.
- #define GATT_CLIENT_EVT_WRITE_CHAR_VALUE_COMPLETED 15

  This event is generated when the "write characteristic value" operation has completed.
- #define GATT_CLIENT_EVT_WRITE_CHAR_LONG_VALUE_COMPLETED 16

  This event is generated when the "write long characteristic value" operation has completed.
- #define GATT_CLIENT_EVT_VALUE_NOTIFICATION 17

  This event is generated when the client received value notification or indication.
- #define GATT_CLIENT_EVT_DISCOVER_CHAR_DESCRIPTORS_COMPLETED 18

  This event is generated when the "discover characteristic descriptiors" operation has completed.
- #define GATT_CLIENT_EVT_WRITE_CHAR_DESCRIPTOR_COMPLETED 19

  This event is generated when the "write characteristic descriptor" operation has completed.
- #define GATT_CLIENT_EVT_WRITE_CHAR_LONG_DESCRIPTOR_COMPLETED 20

  This event is generated when the "write characteristic long descriptor" operation has completed.
- #define GATT_CLIENT_EVT_READ_CHAR_DESCRIPTOR_COMPLETED 21

  This event is generated when the "read characteristic descriptor" operation has completed.
- #define GATT_CLIENT_EVT_READ_CHAR_LONG_DESCRIPTOR_COMPLETED 22

  This event is generated when the "read characteristic long descriptor" operation has completed.
- #define GATT_CLIENT_EVT_EXCHANGE_MTU_COMPLETED 23

  This event is generated when the "exchange MTU" operation has completed.
- #define GATT_CLIENT_EVT_FIND_INCLUDED_SERVICES_COMPLETED 24

  This event is generated when the "find included services" operation has completed.
- #define GATT_CLIENT_EVT_DISCOVER_CHAR_BY_UUID_COMPLETED 25

  This event is generated when the "discover characteristic by UUID" operation has completed.
- #define GATT_CLIENT_EVT_READ_USING_CHAR_UUID_COMPLETED 26

This event is generated when the "read characteristic value using characteristic UUID" operation has completed.
- #define GATT_CLIENT_EVT_READ_MULTIPLE_CHAR_VALUES_COMPLETED 27

  This event is generated when the "read multiple characteristic values" operation has completed.
- #define GATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST 28

  This event is generated when the client received "connection parameter update" request.
- #define GATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED 29

  This event is generated after the new connection parameters have been set.
- #define **GATT_CLIENT_EVT_PROFILE_FINDER_STARTED** 100
- #define **GATT_CLIENT_EVT_PROFILE_FINDER_COMPLETED** 101
- #define **GATT_CLIENT_EVT_PROFILE_FOUND** 102

3.25.1 Detailed Description

This module describes functions and data structures used to implement a GATT client (central).

3.25.2 Macro Definition Documentation

3.25.2.1 #define bt_gatt_client_accept_conn_param_update(*session,* *min_interval,* *max_interval,* *slave_latency,* *supervision_timeout,* *min_ce_length,* *max_ce_length*) **bt_att_client_accept_conn_param_↩ update**(session->att_session, min_interval, max_interval, slave_latency, supervision_timeout, min_ce_length, max_ce_length)

Accept connection parameters update.

When a server sends "connection parameters update" request a GATT_CLIENT_EVT_CONN_PARAM_UPDA↩ TE_REQUEST event is generated. Client has to either accept the request or deny it. `bt_gatt_client_↩ accept_conn_param_update` is used to accept the request.

Parameters

session	GATT session.
min_interval	Minimum connection interval expressed in 1.25ms units.
max_interval	Maximum connection interval expressed in 1.25ms units.
slave_latency	Slave latency expressed as number of connection events.
supervision_↩ timeout	Link supervision timeout expressed in 10ms units.
min_ce_length	Information parameter about the minimum length of connection needed for this LE connection expressed in 0.625ms units.
max_ce_length	Information parameter about the maximum length of connection needed for this LE connection expressed in 0.625ms units.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.2.2 #define bt_gatt_client_reject_conn_param_update(*session*) **bt_att_client_reject_conn_param_↩ update**(session->att_session)

Reject connection parameters update.

When a server sends "connection parameters update" request a GATT_CLIENT_EVT_CONN_PARAM_UPDA↩ TE_REQUEST event is generated. Client has to either accept the request or deny it. `bt_gatt_client_↩ reject_conn_param_update` is used to deny the request.

Parameters

session	GATT session.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3 Function Documentation

3.25.3.1 bt_gatt_client_session_t∗ bt_gatt_client_allocate_session (bt_gatt_client_session_callback_fp *callback*, void ∗ *callback_param*)

Allocate GATT client session.

This function allocates a new GATT client session.

Parameters

callback	The callback function that will be called when the GATT client generates an event.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

Returns

- A pointer to the new GATT client session structure if the function succeeds.
- `NULL` otherwise.

3.25.3.2 bt_bool bt_gatt_client_connect (bt_gatt_client_session_t ∗ *session*, bt_bdaddr_t ∗ *addr*)

Connect to a remote device (peripheral).

This function establishes a connection to a remote device specified by the `addr`. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the GATT client callback. The events generated will either be GATT_CLIENT_SESSION_EVT_SESSION_CONNECTED or GATT_CLIENT_S↩ESSION_EVT_CONNECTION_FAILED.

Parameters

session	GATT session.
addr	The address of a remote device.

Returns

- `TRUE` if connection establishment has been started.
- `FALSE` otherwise.

3.25.3.3 bt_bool bt_gatt_client_disconnect (bt_gatt_client_session_t ∗ *session*)

Disconnect from a remote device.

This function closes a connection to a remote device. After the connection has been terminated the GATT client callback is called with GATT_CLIENT_SESSION_EVT_SESSION_DISCONNECTED event.

Parameters

session	GATT session.

Returns

- `TRUE` if disconnection has been started.
- `FALSE` otherwise. No events will be generated.

3.25.3.4 bt_bool bt_gatt_client_discover_all_chars_ex (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_service_definition_t ∗ *service*, bt_uint *start_handle*, bt_gatt_client_char_declaration_t ∗ *result*, bt_uint *max_results*)

Discover All Characteristics of a Service.

This function finds all the characteristic declarations within a service definition on a server.

Parameters

session	GATT session.
service	Service definition.
start_handle	First attribute handle from which the server has to start searching for characteristic declarations.
result	Pointer to a buffer where the client will store found characteristic declarations.
max_result	The maximum number of characteristic declarations that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.5 bt_bool bt_gatt_client_discover_all_services_ex (bt_gatt_client_session_t ∗ *session*, bt_uint *start_handle*, bt_gatt_client_service_definition_t ∗ *result*, bt_uint *max_results*)

Discover All Services.

This function discovers all the primary services on the server.

Parameters

session	GATT session.
start_handle	First attribute handle from which the server has to start searching for services.
result	Pointer to a buffer where the client will store definitions of the found services.
max_result	The maximum number of service definitions that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.6 bt_bool bt_gatt_client_discover_by_service_uuid (bt_gatt_client_session_t ∗ *session*, bt_uint *service_uuid*, bt_gatt_client_service_definition_t ∗ *result*, bt_uint *max_results*)

Discover Services by UUID (16-bit).

This function discovers a specific primary service on the server when only the Service UUID is known.

Parameters

session	GATT session.
service_uuid	16-bit Service UUID.
result	Pointer to a buffer where the client will store definitions of the found services.
max_result	The maximum number of service definitions that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.7 bt_bool bt_gatt_client_discover_by_service_uuid_80 (bt_gatt_client_session_t ∗ *session*, bt_uuid_t ∗ *service_uuid*, bt_gatt_client_service_definition_t ∗ *result*, bt_uint *max_results*)

Discover Services by UUID (128-bit).

This function discovers a specific primary service on the server when only the Service UUID is known.

Parameters

session	GATT session.
service_uuid	128-bit Service UUID.
result	Pointer to a buffer where the client will store definitions of the found services.
max_result	The maximum number of service definitions that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.8 bt_bool bt_gatt_client_discover_char_by_uuid (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_service_definition_t ∗ *service*, bt_uint *char_uuid*, bt_gatt_client_char_declaration_t ∗ *result*, bt_uint *max_results*)

Discover Characteristics by UUID (16-bit).

This function finds characteristic declarations within a service definition on a server by the characteristic UUID.

Parameters

session	GATT session.
service	Service definition.
char_uuid	16-bit characteristic UUID.
result	Pointer to a buffer where the client will store found characteristic declarations.
max_result	The maximum number of characteristic declarations that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.9 bt_bool bt_gatt_client_discover_char_by_uuid_80 (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_service_definition_t ∗ *service*, bt_uuid_t ∗ *char_uuid*, bt_gatt_client_char_declaration_t ∗ *result*, bt_uint *max_results*)

Discover Characteristics by UUID (128-bit).

This function finds characteristic declarations within a service definition on a server by the characteristic UUID.

Parameters

session	GATT session.
service	Service definition.
char_uuid	128-bit characteristic UUID.
result	Pointer to a buffer where the client will store found characteristic declarations.
max_result	The maximum number of characteristic declarations that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.10 bt_bool bt_gatt_client_discover_char_descriptors (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_char_declaration_t ∗ *characteristic*, bt_gatt_client_char_descriptor_t ∗ *result*, bt_uint *max_results*)

Discover All Characteristic Descriptors.

This function finds all the characteristic descriptors.

Parameters

session	GATT session.
characteristic	Characteristic declaration.
start_handle	First attribute handle from which the server has to start searching for characteristic declarations.
result	Pointer to a buffer where the client will store found characteristic descriptors.
max_result	The maximum number of characteristic descriptors that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.11 bt_bool bt_gatt_client_find_included_services_ex (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_↩ service_definition_t ∗ *service*, bt_uint *start_handle*, bt_gatt_client_inc_service_declaration_t ∗ *result*, bt_uint *max_results*)

Find Included Services.

This function finds include service declarations within a service definition on a server.

Parameters

session	GATT session.
service	Service definition.
start_handle	First attribute handle from which the server has to start searching for include service declarations.
result	Pointer to a buffer where the client will store found include service declarations.
max_result	The maximum number of include service declarations that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.12 void bt_gatt_client_free_sessions (bt_gatt_client_session_t ∗ *session*)

Destroy GATT client session.

This function frees memory used by the session. The session has to exist and be in the "idle" state for this function to succeed. I.e. the session has to be disconnected before this function can be called.

Parameters

session	GATT session.

Returns

- `TRUE` if the function succeeds.
- `FALSE` otherwise.

3.25.3.13 bt_hci_conn_state_t∗ bt_gatt_client_get_hci_connection (const bt_gatt_client_session_t ∗ *session*)

Get HCI connection for a session.

This function returns a pointer to a structure that describes an HCI connection a session is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt_hci_disconnect.

Parameters

session	GATT session.

Returns

- `Pointer` to a structure that describes an HCI connection if the function succeeds.
- `NULL` otherwise. The function fails only if a session specified by the `session` parameter
- does not exist or there is no HCI connection between local and remote devices associated with the session.

3.25.3.14 bt_bool bt_gatt_client_init (void)

Initialize the GATT client.

This function initializes the GATT client of the stack. It must be called prior to any other GATT client function can be called.

Returns

Always `TRUE`

3.25.3.15 bt_bool bt_gatt_client_read_by_char_uuid (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_service_definition_t ∗ *service*, bt_uint *char_uuid*, bt_gatt_client_char_value_t ∗ *result*, bt_uint *max_results*)

Read Using Characteristic UUID (16-bit).

This function reads characteristic values from a server when only the characteristic UUID is known.

Parameters

session	GATT session.
service	Service definition.
char_uuid	16-bit characteristic UUID.
result	Pointer to a buffer where the client will store characteristic values.
max_result	The maximum number of characteristic values that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.16 bt_bool bt_gatt_client_read_by_char_uuid_80 (bt_gatt_client_session_t ∗ *session,* const bt_gatt_client_service_definition_t ∗ *service,* bt_uuid_t ∗ *char_uuid,* bt_gatt_client_char_value_t ∗ *result,* bt_uint *max_results*)

Read Using Characteristic UUID (128-bit).

This function reads characteristic values from a server when only the characteristic UUID is known.

Parameters

session	GATT session.
service	Service definition.
char_uuid	128-bit characteristic UUID.
result	Pointer to a buffer where the client will store characteristic values.
max_result	The maximum number of characteristic values that can be stored in `result`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.17 bt_bool bt_gatt_client_read_char_descriptor (bt_gatt_client_session_t ∗ *session,* const bt_gatt_client_char_descriptor_t ∗ *descriptor,* bt_byte ∗ *buffer,* bt_uint *len*)

Read Characteristic Descriptor.

This function reads characteristic descriptor.

Parameters

session	GATT session.
descriptor	Characteristic descriptor.
buffer	Pointer to a buffer where the client will store the characteristic's descriptor.
len	The length of the `buffer`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.18 bt_bool bt_gatt_client_read_char_long_descriptor (bt_gatt_client_session_t ∗ *session,* const bt_gatt_client_char_descriptor_t ∗ *descriptor,* bt_uint *offset,* bt_byte ∗ *buffer,* bt_uint *len*)

Read Long Characteristic Descriptor.

This function reads long characteristic descriptor.

Parameters

session	GATT session.
descriptor	Characteristic descriptor.
offset	The offset of the first octet to read.
buffer	Pointer to a buffer where the client will store the characteristic's descriptor.
len	The length of the `buffer`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.19 bt_bool bt_gatt_client_read_char_long_value (bt_gatt_client_session_t ∗ *session,* const bt_gatt_client_char_declaration_t ∗ *characteristic,* bt_uint *offset,* bt_byte ∗ *buffer,* bt_uint *len*)

Read Long Characteristic Value.

This function reads characteristic value.

Parameters

session	GATT session.
characteristic	Characteristic declaration.
offset	The offset of the first octet to read.
buffer	Pointer to a buffer where the client will store the characteristic's value.
len	The length of the `buffer`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.20 bt_bool bt_gatt_client_read_char_value (bt_gatt_client_session_t ∗ *session,* const bt_gatt_client_char_declaration_t ∗ *characteristic,* bt_byte ∗ *buffer,* bt_uint *len*)

Read Characteristic Value.

This function reads characteristic value.

Parameters

session	GATT session.
characteristic	Characteristic declaration.
buffer	Pointer to a buffer where the client will store the characteristic's value.
len	The length of the `buffer`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.21 bt_bool bt_gatt_client_read_multiple_char_values (bt_gatt_client_session_t ∗ *session,* const bt_uint ∗ *handles,* bt_uint *count,* bt_byte ∗ *buffer,* bt_uint *len*)

Read Multiple Characteristic Values.

This function reads multiple characteristic values from a server when the characteristic value handles are known.

Parameters

session	GATT session.
handles	An array of characteristic value handles.
count	The number of handles.
buffer	Pointer to a buffer where the client will store the characteristic values.
len	The length of the `buffer`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.22 bt_bool bt_gatt_client_set_char_value (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_char_declaration_t ∗ *characteristic*, const bt_byte ∗ *value*, bt_uint *len*)

Write Without Response.

This function writes characteristic value to a server using ATT "write" command, i.e. the server will not send a response after writing the value.

Parameters

session	GATT session.
characteristic	Characteristic declaration.
value	Characteristic value.
len	The length of the `value`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.23 bt_bool bt_gatt_client_write_char_descriptor (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_char_descriptor_t ∗ *descriptor*, const bt_byte ∗ *value*, bt_uint *len*)

Write Characteristic Descriptor.

This function writes characteristic descriptor.

Parameters

session	GATT session.
descriptor	Characteristic descriptor.
value	Characteristic descriptor value.
len	The length of the `value`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.24 bt_bool bt_gatt_client_write_char_long_descriptor (bt_gatt_client_session_t ∗ *session*, const bt_gatt_client_char_descriptor_t ∗ *descriptor*, bt_uint *value_offset*, const bt_byte ∗ *value*, bt_uint *len*)

Write Long Characteristic Descriptor.

This function writes characteristic descriptor.

Parameters

session	GATT session.
descriptor	Characteristic descriptor.
value_offset	The offset of the first octet to be written.
value	Characteristic descriptor value.
len	The length of the `buffer`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.25 bt_bool bt_gatt_client_write_char_long_value (bt_gatt_client_session_t ∗ *session,* const bt_gatt_client_char_declaration_t ∗ *characteristic,* bt_uint *value_offset,* const bt_byte ∗ *value,* bt_uint *len*)

Write Long Characteristic Value.

This function writes characteristic value to a server using ATT "prepare write" and "execute write" requests.

Parameters

session	GATT session.
characteristic	Characteristic declaration.
value_offset	The offset of the first octet to be written.
value	Characteristic value.
len	The length of the `value`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.26 bt_bool bt_gatt_client_write_char_value (bt_gatt_client_session_t ∗ *session,* const bt_gatt_client_char_declaration_t ∗ *characteristic,* const bt_byte ∗ *value,* bt_uint *len*)

Write Characteristic Value.

This function writes characteristic value to a server using ATT "write" request, i.e. the server will send a response after writing the value.

Parameters

session	GATT session.
characteristic	Characteristic declaration.
value	Characteristic value.
len	The length of the `value`.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.27 bt_bool bt_gatt_exchange_mtu (bt_gatt_client_session_t ∗ *session,* bt_uint *mtu*)

Exchange MTU.

This function informs the server about the client's MTU. In response to the "exchange MTU" request the server sends its MTU to the client.

Parameters

session	GATT session.
mtu	Client's MTU.

Returns

- `TRUE` if a request has been sent.
- `FALSE` otherwise. No events will be generated.

3.25.3.28 bt_gatt_client_mgr_t* bt_gatt_get_client_mgr (void)

Return a pointer to an instance of the GATT client manager.

This function returns a pointer to an instance of the GATT client manager. There is only one instance of the manager allocated by the stack.

Returns

- A pointer to the GATT client manager.

3.26 Configuration

This module describes parameters used to configure GATT client.

This module describes parameters used to configure GATT client.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"

// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE

// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS            ...
#define HCI_MAX_DATA_BUFFERS           ...
#define HCI_MAX_HCI_CONNECTIONS        ...
#define HCI_RX_BUFFER_LEN              ...
#define HCI_TX_BUFFER_LEN              ...
#define HCI_L2CAP_BUFFER_LEN           ...
#define HCI_MAX_CMD_PARAM_LEN          ...

// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS          ...
#define L2CAP_MAX_FRAME_BUFFERS        ...
#define L2CAP_MAX_PSMS                 ...
#define L2CAP_MAX_CHANNELS             ...

// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN     ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN   ...

// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure GATT,
// the following values must be defined:

#define BT_INCLUDE_ATT               // tells dotstack to compile in ATT support
#define ATT_CLIENT_MAX_SESSIONS      ...

#define BT_INCLUDE_GATT_CLIENT       // tells dotstack to compile in GATT support

// By default all GATT client functions can be used by the application. In most cases only a subset of the
       API will be used.
// To to save quite significant amount of code space unused functions can be disabled. To disable a
       function the correspondign symbol
// must be defined in this configuration file. For example do disable bt_gatt_client_read_by_char_uuid the
       GATT_NO_READ_BY_CHAR_UUID
// must be defined:
//   #define GATT_NO_READ_BY_CHAR_UUID

#include "cdbt/bt/bt_oem_config.h"
```

- GATT_NO_DISCOVER_ALL_SERVICES - Disable bt_gatt_client_discover_all_services_ex()
- GATT_NO_DISCOVER_BY_SERVICE_UUID - Disable bt_gatt_client_discover_by_service_uuid()
- GATT_NO_FIND_INCLUDED_SERVICES - Disable bt_gatt_client_find_included_services_ex()
- GATT_NO_DISCOVER_ALL_CHARS - Disable bt_gatt_client_discover_all_chars_ex()
- GATT_NO_DISCOVER_CHAR_BY_UUID - Disable bt_gatt_client_discover_char_by_uuid()
- GATT_NO_READ_CHAR_VALUE - Disable bt_gatt_client_read_char_value()
- GATT_NO_READ_BY_CHAR_UUID - Disable bt_gatt_client_read_by_char_uuid()
- GATT_NO_READ_MULTIPLE_CHAR_VALUES - Disable bt_gatt_client_read_multiple_char_values()
- GATT_NO_READ_CHAR_DESCRIPTOR - Disable bt_gatt_client_read_char_descriptor()
- GATT_NO_READ_CHAR_LONG_VALUE - Disable bt_gatt_client_read_char_long_value()
- GATT_NO_READ_CHAR_LONG_DESCRIPTOR - Disable bt_gatt_client_read_char_long_descriptor()

- GATT_NO_WRITE_CHAR_VALUE - Disable bt_gatt_client_write_char_value()
- GATT_NO_WRITE_CHAR_DESCRIPTOR - Disable bt_gatt_client_write_char_descriptor()
- GATT_NO_WRITE_CHAR_LONG_VALUE - Disable bt_gatt_client_write_char_long_value()
- GATT_NO_WRITE_CHAR_LONG_DESCRIPTOR - Disable bt_gatt_client_write_char_long_descriptor()
- GATT_NO_DISCOVER_CHAR_DESCRIPTORS - Disable bt_gatt_client_discover_char_descriptors()

3.27 Security Manager

This module describes functions and data structures used to implement Security Manager.

This module describes functions and data structures used to implement Security Manager.

3.28 OEM - HCI Communication Interface

Modules

- HCI UART (H4) transport protocol

Typedefs

- typedef void(∗ bt_oem_send_callback_fp) (void)

    Send callback.
- typedef void(∗ bt_oem_recv_callback_fp) (bt_uint len)

    Receive callback.

Functions

- void bt_oem_send (const bt_byte ∗buffer, bt_uint len, bt_oem_send_callback_fp callback)

    Send data.
- void bt_oem_recv (bt_byte ∗buffer, bt_uint len, bt_oem_recv_callback_fp callback)

    Receive data.

3.28.1 Detailed Description

The DotStack library provides several HCI transport protocols (e.g., H4, 3-wire protocol). However, the code that actually moves octets of data between the CPU and HCI controller is application specific.

This module declares an interface that allows DotStack to communicate with the HCI controller. The application has to implement this interface.

The interface consist of the following functions that must be implemented by the application:

- bt_oem_send()
- bt_oem_recv()

3.28.2 Typedef Documentation

3.28.2.1 typedef void(∗ bt_oem_recv_callback_fp) (bt_uint len)

Receive callback.

This callback function is called when a receive operation initiated by bt_oem_recv() has completed.

Parameters

len	Number of received bytes. The value of this parameter should always be the same as the number of bytes requested in a call to bt_oem_recv().

3.28.2.2 typedef void(∗ bt_oem_send_callback_fp) (void)

Send callback.

This callback function is called when a send operation initiated by bt_oem_send() has completed.

3.28.3 Function Documentation

3.28.3.1 void bt_oem_recv (bt_byte ∗ *buffer,* bt_uint *len,* bt_oem_recv_callback_fp *callback*)

Receive data.

This function is called by the HCI layer when it needs more data from the HCI controller. Implementation of this function must receive the specified number of bytes from the HCI controller and call the provided callback function.

Parameters

buffer	Pointer to a buffer for the received data. The buffer must be long enough to accommodate the number of bytes specified by the

len parameter.

Parameters

len	Number of bytes to receive.
callback	A callback function that must be called when the requested number of bytes have been received.

3.28.3.2 void bt_oem_send (const bt_byte ∗ *buffer,* bt_uint *len,* bt_oem_send_callback_fp *callback*)

Send data.

This function is called by the HCI layer when it needs to send data to the HCI controller. Implementation of this function must send the specified number of bytes to the HCI controller and call the provided callback function.

Parameters

buffer	Pointer to the data to be sent .
len	Number of bytes to send.
callback	A callback function that must be called when all data have been sent.

3.29 OEM - Timer Interface

Typedefs

- typedef void(∗ bt_timer_callback_fp) (void)

  Timer callback.

Functions

- void bt_oem_timer_set (bt_uint timerId, bt_ulong milliseconds, bt_timer_callback_fp callback)

  Set timer.
- void bt_oem_timer_clear (bt_uint timerId)

  Clear timer.

3.29.1 Detailed Description

DotStack requires a facility to measure various time intervals. This module declares an interface that the application has to implement in order to provide DotStack with such functionality.

The minimum required timer resolution is 100 milliseconds.

The maximum number of timers is defined by the `BT_TIMER_MAX` constant.

Timer ID values used in the interface are from `0` to `BT_TIMER_MAX-1`.

The interface consists of the following function:

- bt_oem_timer_set()
- bt_oem_timer_clear()

3.29.2 Typedef Documentation

3.29.2.1 typedef void(∗ bt_timer_callback_fp) (void)

Timer callback.

This callback is called when a timer expires.

3.29.3 Function Documentation

3.29.3.1 void bt_oem_timer_clear (bt_uint *timerId*)

Clear timer.

This function must be implemented by the application. When this function is called the application must clear the specified timer. If it is already expired and a callback is currently pending, the application should also take measures to cancel the callback.

Parameters

timerId	ID of the timer to clear.

3.29.3.2 void bt_oem_timer_set (bt_uint *timerId,* bt_ulong *milliseconds,* bt_timer_callback_fp *callback*)

Set timer.

This function must be implemented by the application. When it is called, the application must set the specified timer. When the timer expires, the application must call the passed callback function. The function must not wait until the timer expires. It must set the timer and exit immediately.

Parameters

timerId	ID of the timer to set.
milliseconds	Timer interval in milliseconds
callback	Timer expiration callback function.

3.30 OEM - Non-volatile Storage Interface

Functions

- bt_uint bt_oem_storage_get_capacity (void)

    Get non-volatile storage capacity.
- void bt_oem_storage_start (void)

    Begin a sequence of non-volatile storage operations.
- void bt_oem_storage_stop (void)

    End a sequence of non-volatile storage operations.
- void bt_oem_storage_write (bt_int addr, const bt_byte ∗data, bt_int len, bt_storage_callback_fp callback)

    Write to non-volatile storage.
- void bt_oem_storage_read (bt_int addr, bt_byte ∗buffer, bt_int len, bt_storage_callback_fp callback)

    Read from the non-volatile storage.

3.30.1 Detailed Description

DotStack requires a non-volatile storage for storing link keys. This module declares an interface for accessing such storage. The application must provide implementations of all functions of this interface.

3.30.2 Function Documentation

3.30.2.1 bt_uint bt_oem_storage_get_capacity (void)

Get non-volatile storage capacity.

Implementation of this function must return the capacity of its non-volatile storage.

3.30.2.2 void bt_oem_storage_read (bt_int *addr,* bt_byte ∗ *buffer,* bt_int *len,* bt_storage_callback_fp *callback*)

Read from the non-volatile storage.

This function is called by the stack to read from the non-volatile storage. This function must be implemented by the application. When this function is called the application must start a read operation. When the number of bytes specified by the `len` parameter is read, the application must call the callback function specified by the `callback` parameter. The application does not have to read the whole number of bytes during the call to this function. It may complete reading later and then call the completion callback. The stack guarantees that the destination data buffer will be available until the application calls the completion callback.

Parameters

addr	The non-volatile storage address where to read data from.
buffer	The receiving buffer.
len	The number of bytes to read.
callback	The completion callback function.

3.30.2.3 void bt_oem_storage_start (void)

Begin a sequence of non-volatile storage operations.

DotStack calls this function when it starts a sequence of non-volatile storage operations. When the sequence is finished, DotStack will call bt_oem_storage_stop().

3.30.2.4 void bt_oem_storage_stop (void)

End a sequence of non-volatile storage operations.

DotStack calls this function when it finishes executing a sequence of non-volatile storage operations.

3.30.2.5 void bt_oem_storage_write (bt_int *addr,* const bt_byte ∗ *data,* bt_int *len,* bt_storage_callback_fp *callback*)

Write to non-volatile storage.

This function is called by the stack to write data to the non-volatile storage. This function must be implemented by the application. When this function is called, the application must start writing specified data to the non-volatile storage. When all data has been written, the application must call the callback function passed in the `callback` parameter. The application does not have to complete the write operation during the call to this function. It may complete the operation later and then call the callback function. In this case, the application does not have to store the data in an internal buffer. The stack guarantees that the passed data will be present until the completion callback is called by the application.

Parameters

addr	The persitent storage address where to write data to.
data	Pointer to data.
len	Data length.
callback	The completion callback function.

3.31 OEM - Logging Interface

Functions

- void bt_oem_log_write (const char ∗msg)

  Output log message.

3.31.1 Detailed Description

This module declares an interface that DotStack is using to output its debug and diagnostic information. The application must implement this interface.

Though the simplest implementation may consis of just empty stubs it is strongly encouraged to provide a useful implementation that allows for viewing and capturing of the data passed through this interface.

3.31.2 Function Documentation

3.31.2.1 void bt_oem_log_write (const char ∗ *msg*)

Output log message.

DotStack calls this function to output its debug information. Implementation should output or store the specified message to whatever device or medium where it can be examined and analyzed.

3.32 Vendor specific extensions to HCI

Modules

- CSR

3.32.1 Detailed Description

This module defines functions and data structures used to access and control various capabilities of CSR's controllers.

3.33 CSR

Functions

- void btx_csr_init (void)

  Initialize CSR support layer.
- void btx_csr_autobaud (btx_csr_autobaud_buffer_t ∗buffer, btx_csr_autobaud_callback_fp callback, void ∗callback_param)

  Configure controller's UART speed.
- void btx_csr_bc7_sel_host_interface_h4 (btx_csr_autobaud_buffer_t ∗buffer, bt_byte interval, btx_csr_↩ autobaud_callback_fp callback, void ∗callback_param)

  Configure controller's UART speed and host interface.
- void btx_csr_exec_script (const btx_csr_script_t ∗script, btx_csr_exec_script_buffer_t ∗buffer, btx_csr_↩ exec_script_callback_fp callback, void ∗callback_param)

  Patch controller's firmware.
- void btx_csr_set_ps_vars (const bt_uint ∗ps_vars, btx_csr_set_ps_vars_buffer_t ∗buffer, btx_csr_set_ps_↩ vars_callback_fp callback, void ∗callback_param)

  Write PS variables.
- void btx_csr_set_ps_vars_ex (const bt_uint ∗ps_vars, btx_csr_set_ps_vars_buffer_t ∗buffer, bt_uint store, btx_csr_set_ps_vars_callback_fp callback, void ∗callback_param)

  Write PS variables.
- bt_bool btx_csr_warm_reset (void)

  Warm reset.
- bt_bool btx_csr_warm_reset_ex (bt_hci_cmd_callback_fp callback, void ∗callback_param)

  Warm reset.
- bt_bool btx_csr_enable_tx (bt_bool enable, bt_hci_cmd_callback_fp callback, void ∗callback_param)

  Enable/disable transmitter.
- bt_bool btx_csr_get_cached_temperature (btx_csr_get_var_callback_fp callback, void ∗callback_param)

  Get chip's cached temperature.
- bt_bool btx_csr_get_rssi_acl (bt_hci_hconn_t hconn, btx_csr_get_var_callback_fp callback, void ∗callback↩ _param)

  Get RSSI.
- void btx_csr_patch_controller (const btx_csr_get_script_fp ∗scripts, bt_int script_count, btx_csr_exec_↩ script_buffer_t ∗buffer, btx_csr_exec_script_callback_fp callback, void ∗callback_param)

  Patch controller's firmware.
- const btx_csr_script_t ∗ btx_csr_get_script__PB_27_R20_BC6ROM_A04 (void)

  Return script for patching BlueCore 6.
- const btx_csr_script_t ∗ btx_csr_get_script__PB_90_REV6 (void)

  Return script for patching CSR8810 (BlueCore 7)
- const btx_csr_script_t ∗ btx_csr_get_script__PB_101_CSR8811_CSP28_UART (void)

  Return script for patching CSR8x11 A06 (BlueCore 7)
- const btx_csr_script_t ∗ btx_csr_get_script__PB_109_CSR8811_REV16 (void)

  Return script for patching CSR8x11 A08 (BlueCore 7)
- const btx_csr_script_t ∗ btx_csr_get_script__dsp_script__PB_109_DSP_rev8 (void)

  Return script for patching DSP in CSR8x11 A08 (BlueCore 7)
- const btx_csr_script_t ∗ btx_csr_get_script__PB_173_CSR8X11_REV1 (void)

  Return script for patching CSR8x11 A12 (BlueCore 7)

3.33.1 Detailed Description

3.33.2 Function Documentation

3.33.2.1 void btx_csr_autobaud (btx_csr_autobaud_buffer_t ∗ *buffer,* btx_csr_autobaud_callback_fp *callback,* void ∗ *callback_param*)

Configure controller's UART speed.

This function makes the controller auto-configure its UART speed. The host transport must be set to H4. This function works only with BC6 controllers.

3.33.2.2 void btx_csr_bc7_sel_host_interface_h4 (btx_csr_autobaud_buffer_t ∗ *buffer,* bt_byte *interval,* btx_csr_autobaud_callback_fp *callback,* void ∗ *callback_param*)

Configure controller's UART speed and host interface.

This function makes the controller auto-configure its UART speed and select H4 as host interface. PS_KEY_HO←ST_INTERFACE must not be set. PS_KEY_UART_BITRATE must be set to 0. This function works only with BC7 controllers.

3.33.2.3 bt_bool btx_csr_enable_tx (bt_bool *enable,* bt_hci_cmd_callback_fp *callback,* void ∗ *callback_param*)

Enable/disable transmitter.

Parameters

enable	Specifies whether the transmitter should be enable or disabled.
callback	The callback function that will be called after the command has completed.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.33.2.4 void btx_csr_exec_script (const **btx_csr_script_t** ∗ *script,* btx_csr_exec_script_buffer_t ∗ *buffer,* btx_csr_exec_script_callback_fp *callback,* void ∗ *callback_param*)

Patch controller's firmware.

This function executes a script that patches the controller's firmware. The `script` must point to a structure that contain a complete patch script for a particular controller model and revision. If the revision specified in the script and revision read from the controller are the same btx_csr_patch_controller() loads the script to the controller and calls the `callback` with the first parameter TRUE. Otherwise the `callback` is called with the first parameter FALSE.

If support for multiple firmware revisions is neede use btx_csr_patch_controller().

Parameters

script	Array of patch scripts.
buffer	A buffer for storing temporary data needed for script execution.
callback	The callback function that will be called when the script has been executed.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback..

3.33.2.5 bt_bool btx_csr_get_cached_temperature (btx_csr_get_var_callback_fp *callback,* void ∗ *callback_param*)

Get chip's cached temperature.

Parameters

callback	The callback function that will be called after the command has completed.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.33.2.6 bt_bool btx_csr_get_rssi_acl (bt_hci_hconn_t *hconn*, btx_csr_get_var_callback_fp *callback*, void ∗ *callback_param*)

Get RSSI.

This function retrieves the RSSI for a given HCI ACL handle.

Parameters

hconn	ACL connection handle.
callback	The callback function that will be called after the command has completed.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

3.33.2.7 void btx_csr_init (void)

Initialize CSR support layer.

This function initializes all internal variables of the CSR support layer. CSR controllers use vendor specific event (0xFF) to carry the BCCMD protocol. They also do not report number of completed packets for BCCMD commands. This function installs a vendor specific event handler that makes sure that callback are called on corresponding vendor specific commands and the number of free command buffers in the controller is kept correct.

3.33.2.8 void btx_csr_patch_controller (const btx_csr_get_script_fp ∗ *scripts*, bt_int *script_count*, btx_csr_exec_script_buffer_t ∗ *buffer*, btx_csr_exec_script_callback_fp *callback*, void ∗ *callback_param*)

Patch controller's firmware.

This function executes a script that patches the controller's firmware. Each entry of the `scripts` array must be a complete patch script for a particular controller model and revision. btx_csr_patch_controller() reads the revision number from the controller then tries to find the corresponding script in the `scripts`. If there is a matching script it is loaded to the controller and `callback` is called with the first parameter TRUE. If no suitable script found `callback` is called with the first parameter FALSE.

Parameters

scripts	Array of patch scripts.
script_count	The number of scripts in `scripts`.
buffer	A buffer for storing temporary data needed for script execution.
callback	The callback function that will be called when the script has been executed.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback..

3.33.2.9 void btx_csr_set_ps_vars (const bt_uint ∗ *ps_vars*, btx_csr_set_ps_vars_buffer_t ∗ *buffer*, btx_csr_set_ps_vars_callback_fp *callback*, void ∗ *callback_param*)

Write PS variables.

Parameters

ps_vars	PS values

buffer	A buffer for storing temporary data during function execution.
callback	The callback function that will be called when all PS values have been sent to the controller or error occurred.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback..

3.33.2.10 void btx_csr_set_ps_vars_ex (const bt_uint ∗ *ps_vars*, btx_csr_set_ps_vars_buffer_t ∗ *buffer*, bt_uint *store*, btx_csr_set_ps_vars_callback_fp *callback*, void ∗ *callback_param*)

Write PS variables.

Parameters

ps_vars	PS values
buffer	A buffer for storing temporary data during function execution.
store	
callback	The callback function that will be called when all PS values have been sent to the controller or error occurred.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback..

3.33.2.11 bt_bool btx_csr_warm_reset (void)

Warm reset.

This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact.

3.33.2.12 bt_bool btx_csr_warm_reset_ex (bt_hci_cmd_callback_fp *callback*, void ∗ *callback_param*)

Warm reset.

This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact. Since the controller does not respond to the warm reset command as it starts resetting immediately upon receiving the command, the `callback` is called right after the command packet has been transmitted to the controller.

Parameters

callback	The callback function that will be called after the warm reset command has been sent to the controller.
callback_param	A pointer to arbitrary data to be passed to the `callback` callback.

Chapter 4

Data Structure Documentation

4.1 bt_spp_port_t::_bt_spp_port_flags_t Struct Reference

4.2 bt_a2dp_aac_config_t Struct Reference

4.3 bt_a2dp_event_t Union Reference

Parameter to an application callback.

```
#include <a2dp.h>
```

Data Fields

- bt_a2dp_evt_open_and_start_stream_completed_t open_and_start_stream_completed

  Valid if event is A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED.
- bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected

  Valid if event is A2DP_EVT_CTRL_CHANNEL_CONNECTED.
- bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected

  Valid if event is A2DP_EVT_CTRL_CHANNEL_DISCONNECTED.
- bt_avdtp_evt_discover_completed_t discover_completed

  Valid if event is A2DP_EVT_DISCOVER_COMPLETED.
- bt_avdtp_evt_sep_info_received_t sep_info_received

  Valid if event is A2DP_EVT_SEP_INFO_RECEIVED.
- bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed

  Valid if event is A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED.
- bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received

  Valid if event is A2DP_EVT_SEP_CAPABILITIES_RECEIVED.
- bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed

  Valid if event is A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED.
- bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed

  Valid if event is A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED.
- bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed

  Valid if event is A2DP_EVT_STREAM_RECONFIGURE_COMPLETED.
- bt_avdtp_evt_open_stream_completed_t open_stream_completed

  Valid if event is A2DP_EVT_OPEN_STREAM_COMPLETED.
- bt_avdtp_evt_start_stream_completed_t start_stream_completed

  Valid if event is A2DP_EVT_START_STREAM_COMPLETED.

- bt_avdtp_evt_close_stream_completed_t close_stream_completed

  Valid if event is A2DP_EVT_CLOSE_STREAM_COMPLETED.
- bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed

  Valid if event is A2DP_EVT_SUSPEND_STREAM_COMPLETED.
- bt_avdtp_evt_stream_security_control_completed_t security_control_completed

  Valid if event is A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED.
- bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested

  Valid if event is A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED.
- bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested

  Valid if event is A2DP_EVT_RECONFIGURE_STREAM_REQUESTED.
- bt_avdtp_evt_open_stream_requested_t open_stream_requested

  Valid if event is A2DP_EVT_OPEN_STREAM_REQUESTED.
- bt_avdtp_evt_start_stream_requested_t start_stream_requested

  Valid if event is A2DP_EVT_START_STREAM_REQUESTED.
- bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested

  Valid if event is A2DP_EVT_SUSPEND_STREAM_REQUESTED.
- bt_avdtp_evt_close_stream_requested_t close_stream_requested

  Valid if event is A2DP_EVT_CLOSE_STREAM_REQUESTED.
- bt_avdtp_evt_abort_stream_requested_t abort_stream_requested

  Valid if event is A2DP_EVT_ABORT_STREAM_REQUESTED.
- bt_avdtp_evt_stream_configured_t stream_configured

  Valid if event is A2DP_EVT_STREAM_CONFIGURED.
- bt_avdtp_evt_stream_reconfigured_t stream_reconfigured

  Valid if event is A2DP_EVT_STREAM_RECONFIGURED.
- bt_avdtp_evt_stream_opened_t stream_opened

  Valid if event is A2DP_EVT_STREAM_OPENED.
- bt_avdtp_evt_stream_started_t stream_started

  Valid if event is A2DP_EVT_STREAM_STARTED.
- bt_avdtp_evt_stream_suspended_t stream_suspended

  Valid if event is A2DP_EVT_STREAM_SUSPENDED.
- bt_avdtp_evt_stream_closed_t stream_closed

  Valid if event is A2DP_EVT_STREAM_CLOSED.
- bt_avdtp_evt_stream_aborted_t stream_aborted

  Valid if event is A2DP_EVT_STREAM_ABORTED.

4.3.1 Detailed Description

Parameter to an application callback.

This union is used to pass event specific data to the A2DP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

4.4 bt_a2dp_evt_open_and_start_stream_completed_t Struct Reference

Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event.

```
#include <a2dp.h>
```

Data Fields

- bt_byte failed_cmd

  ID of the failed command.
- bt_byte err_code

  The result of the request.
- bt_byte strm_handle

  Stream handle.
- bt_avdtp_evt_set_stream_configuration_completed_t set_config

  If "set configuration" request failed this member contains description of an error returned by the remote party.
- bt_avdtp_evt_open_stream_completed_t open

  If open stream" request failed this member contains description of an error returned by the remote party.
- bt_avdtp_evt_start_stream_completed_t start

  If "start stream" request failed this member contains description of an error returned by the remote party.

4.4.1 Detailed Description

Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event.

A pointer to this structure is passed to the A2DP application callback as a valid member of the bt_a2dp_event↩_t union - bt_a2dp_event_t::open_and_start_stream_completed - when A2DP completed a "open & start stream" request.

4.4.2 Field Documentation

4.4.2.1 bt_byte err_code

The result of the request.

- If the remote accepted all 3 requests sent during execution of the "open & start" request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.4.2.2 bt_byte failed_cmd

ID of the failed command.

This can be one of the following values:

- AVDTP_CMD_SET_CONFIGURATION
- AVDTP_CMD_OPEN
- AVDTP_CMD_START

4.5 bt_a2dp_mgr_t Struct Reference

A2DP manager.

```
#include <a2dp.h>
```

Data Fields

- bt_byte state

  Manager state.
- bt_avdtp_mgr_t ∗ avdtp_mgr

  AVDTP manager.
- bt_avdtp_stream_t ∗ connection_stream

  Pointer to a stream being open with bt_a2dp_open_and_start_stream.
- bt_a2dp_mgr_callback_fp callback

  Pointer to a function which a A2DP consumer must register in order to be notified of various events.
- void ∗ callback_param

  Pointer to arbitrary data to be passed to the `callback`.

4.5.1 Detailed Description

A2DP manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with bt_a2dp_get_mgr.

4.5.2 Field Documentation

4.5.2.1 bt_byte state

Manager state.

This value can be one of the following values:

- A2DP_MANAGER_STATE_IDLE
- A2DP_MANAGER_STATE_CONNECTING

4.6 bt_a2dp_mpeg_config_t Struct Reference

4.7 bt_a2dp_sbc_config_t Struct Reference

4.8 bt_a2dp_sbc_packet_info_t Struct Reference

4.9 bt_att_attribute_t Struct Reference

4.10 bt_att_client_evt_conn_param_update_completed_t Struct Reference

Parameter to ATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_int hci_status

  HCI command status.

4.10.1 Detailed Description

Parameter to ATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED event.

A pointer to this structure is passed to the ATT client application callback after the new connection parameters have been set.

4.11 bt_att_client_evt_conn_param_update_t Struct Reference

Parameter to ATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST event.

```
#include <att_client.h>
```

Data Fields

- bt_uint min_interval

  Minimum connection interval expressed in 1.25ms units.
- bt_uint max_interval

  Maximum connection interval expressed in 1.25ms units.
- bt_uint slave_latency

  Slave latency expressed as number of connection events.
- bt_uint supervision_timeout

  Link supervision timeout expressed in 10ms units.

4.11.1 Detailed Description

Parameter to ATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST event.

A pointer to this structure is passed to the ATT client application callback when client received update connection parameters request.

4.12 bt_att_client_evt_execute_write_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_EXECUTE_WRITE_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint err_handle

  The attribute handle that generated error response.

4.12.1 Detailed Description

Parameter to ATT_CLIENT_EVT_EXECUTE_WRITE_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "execute write" request.

4.13 bt_att_client_evt_find_by_type_value_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_FIND_BY_TYPE_VALUE_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status
  Operation status.
- bt_uint start_handle
  First attribute handle found.
- bt_uint end_handle
  Last attribute handle found.
- bt_uint count
  The number of attribute handles found.

4.13.1 Detailed Description

Parameter to ATT_CLIENT_EVT_FIND_BY_TYPE_VALUE_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "find by type value info" request.

4.14 bt_att_client_evt_info_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_FIND_INFO_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status
  Operation status.
- bt_uint start_handle
  First attribute handle found.
- bt_uint end_handle
  Last attribute handle found.
- bt_uint count
  The number of attribute handles found.

4.14.1 Detailed Description

Parameter to ATT_CLIENT_EVT_FIND_INFO_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "find info" request.

4.15 bt_att_client_evt_mtu_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_EXCHANGE_MTU_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint mtu

  Server's MTU.

4.15.1 Detailed Description

Parameter to ATT_CLIENT_EVT_EXCHANGE_MTU_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "exchange MTU" request.

4.16 bt_att_client_evt_prepare_write_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_PREPARE_WRITE_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint err_handle

  The attribute handle that generated error response.

4.16.1 Detailed Description

Parameter to ATT_CLIENT_EVT_PREPARE_WRITE_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "prepare write" request.

4.17 bt_att_client_evt_read_blob_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_READ_BLOB_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint handle

  Attribute handle.
- bt_uint offset

  The offset of the first octet.
- const bt_byte ∗ value

  Part of the attribute value.
- bt_uint len

  Part of the attribute value length.

4.17.1 Detailed Description

Parameter to ATT_CLIENT_EVT_READ_BLOB_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read blob" request.

4.18 bt_att_client_evt_read_by_group_type_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_READ_BY_GROUP_TYPE_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint start_handle

  First attribute handle found.
- bt_uint end_handle

  Last attribute handle found.
- bt_uint count

  The number of attribute handles found.

4.18.1 Detailed Description

Parameter to ATT_CLIENT_EVT_READ_BY_GROUP_TYPE_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read by group type" request.

4.19 bt_att_client_evt_read_by_type_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_READ_BY_TYPE_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint start_handle

  First attribute handle found.
- bt_uint end_handle

  Last attribute handle found.
- bt_uint count

  The number of attribute handles found.

4.19.1 Detailed Description

Parameter to ATT_CLIENT_EVT_READ_BY_TYPE_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read by type" request.

4.20 bt_att_client_evt_read_multiple_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_READ_MULTIPLE_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- const bt_byte ∗ values

  Attribute values.
- bt_uint len

  The length of attribute values.

4.20.1 Detailed Description

Parameter to ATT_CLIENT_EVT_READ_MULTIPLE_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read multiple" request.

4.21 bt_att_client_evt_read_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_READ_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint handle

  Attribute handle.
- const bt_byte ∗ value

  Attribute value.
- bt_uint len

  Attribute value length.

4.21.1 Detailed Description

Parameter to ATT_CLIENT_EVT_READ_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read" request.

4.22 bt_att_client_evt_t Union Reference

Parameter to ATT client application callback.

```
#include <att_client.h>
```

Data Fields

- bt_att_client_evt_mtu_response_t mtu
  Valid if event is ATT_CLIENT_EVT_EXCHANGE_MTU_RESPONSE.
- bt_att_client_evt_info_response_t find_info
  Valid if event is ATT_CLIENT_EVT_FIND_INFO_RESPONSE.
- bt_att_client_evt_find_by_type_value_response_t find_by_type_value
  Valid if event is ATT_CLIENT_EVT_FIND_BY_TYPE_VALUE_RESPONSE.
- bt_att_client_evt_read_by_type_response_t read_by_type
  Valid if event is ATT_CLIENT_EVT_READ_BY_TYPE_RESPONSE.
- bt_att_client_evt_read_response_t read
  Valid if event is ATT_CLIENT_EVT_READ_RESPONSE.
- bt_att_client_evt_read_blob_response_t read_blob
  Valid if event is ATT_CLIENT_EVT_READ_BLOB_RESPONSE.
- bt_att_client_evt_read_multiple_response_t read_multiple
  Valid if event is ATT_CLIENT_EVT_READ_MULTIPLE_RESPONSE.
- bt_att_client_evt_read_by_group_type_response_t read_by_group_type
  Valid if event is ATT_CLIENT_EVT_READ_BY_GROUP_TYPE_RESPONSE.
- bt_att_client_evt_write_response_t write
  Valid if event is ATT_CLIENT_EVT_WRITE_RESPONSE.
- bt_att_client_evt_prepare_write_response_t prepare_write
  Valid if event is ATT_CLIENT_EVT_PREPARE_WRITE_RESPONSE.
- bt_att_client_evt_execute_write_response_t execute_write
  Valid if event is ATT_CLIENT_EVT_EXECUTE_WRITE_RESPONSE.
- bt_att_client_evt_value_notification_t value_notification
  Valid if event is ATT_CLIENT_EVT_VALUE_NOTIFICATION.
- bt_att_client_evt_value_indication_t value_indication
  Valid if event is ATT_CLIENT_EVT_VALUE_INDICATION.
- bt_att_client_evt_conn_param_update_t conn_param_update
  Valid if event is ATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST.
- bt_att_client_evt_conn_param_update_completed_t conn_param_update_completed
  Valid if event is ATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED.

4.22.1 Detailed Description

Parameter to ATT client application callback.

This union is used to pass event specific data to the ATT client consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

4.23 bt_att_client_evt_value_indication_t Struct Reference

Parameter to ATT_CLIENT_EVT_VALUE_INDICATION event.

```
#include <att_client.h>
```

Data Fields

- bt_uint handle

  Attribute handle.
- const bt_byte ∗ value

  Attribute value.
- bt_uint len

  Attribute value length.

4.23.1 Detailed Description

Parameter to ATT_CLIENT_EVT_VALUE_INDICATION event.

A pointer to this structure is passed to the ATT client application callback when client received attribute value indication.

4.24 bt_att_client_evt_value_notification_t Struct Reference

Parameter to ATT_CLIENT_EVT_VALUE_NOTIFICATION event.

```
#include <att_client.h>
```

Data Fields

- bt_uint handle

  Attribute handle.
- const bt_byte ∗ value

  Attribute value.
- bt_uint len

  Attribute value length.

4.24.1 Detailed Description

Parameter to ATT_CLIENT_EVT_VALUE_NOTIFICATION event.

A pointer to this structure is passed to the ATT client application callback when client received attribute value notification.

4.25 bt_att_client_evt_write_response_t Struct Reference

Parameter to ATT_CLIENT_EVT_WRITE_RESPONSE event.

```
#include <att_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint handle

  The attribute handle.

4.25.1 Detailed Description

Parameter to ATT_CLIENT_EVT_WRITE_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "write" request.

4.26 bt_att_client_mgr_t Struct Reference

4.27 bt_att_client_session_t Struct Reference

4.28 bt_att_client_uuid_t Struct Reference

UUID.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte len
  UUID length.
- bt_uint uuid16
  16-bit UUID.
- bt_uuid_t uuid128
  128-bit UUID.

4.28.1 Detailed Description

UUID.

This structure is used to hold a UUID - 16 or 128 bit.

4.28.2 Field Documentation

4.28.2.1 bt_byte len

UUID length.

This can be one of the following values:

- ATT_CLIENT_UUID_LEN_16
- ATT_CLIENT_UUID_LEN_128

4.29 bt_att_enum_context_t Struct Reference

4.30 bt_att_evt_attr_indication_sent_t Struct Reference

Parameter to ATT_SERVER_EVT_ATTR_VALUE_INDICATED event.

```
#include <att.h>
```

Data Fields

- bt_byte status

  Error code.
- bt_att_session_t ∗ session

  ATT session.
- bt_uint handle

  Attribute handle.

4.30.1 Detailed Description

Parameter to ATT_SERVER_EVT_ATTR_VALUE_INDICATED event.

A pointer to this structure is passed to the ATT application callback when a value indication has bee sent to the client.

4.31 bt_att_evt_attr_notification_sent_t Struct Reference

Parameter to ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED event.

```
#include <att.h>
```

Data Fields

- bt_byte status

  Error code.
- bt_att_session_t ∗ session

  ATT session.
- bt_uint handle

  Attribute handle.

4.31.1 Detailed Description

Parameter to ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED event.

A pointer to this structure is passed to the ATT application callback when a value notification has bee sent to the client.

4.32 bt_att_evt_attr_value_changed_t Struct Reference

Parameter to ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_SERVER and ATT_SERVER_EVT_ATTR_↩
VALUE_CHANGED_BY_CLIENT events.

```
#include <att.h>
```

Data Fields

- bt_att_attribute_t ∗ attr

  ATT attribute.

4.32.1 Detailed Description

Parameter to ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_SERVER and ATT_SERVER_EVT_ATTR_↩
VALUE_CHANGED_BY_CLIENT events.

A pointer to this structure is passed to the ATT application callback when the value of an attribute has been changed locally on the server or by a client.

4.33 bt_att_evt_attr_value_read_t Struct Reference

Parameter to ATT_SERVER_EVT_ATTR_VALUE_READ event.

```
#include <att.h>
```

Data Fields

- bt_att_attribute_t ∗ attr

  ATT attribute.
- bt_uint value_offset

  Offset from which value of the attribute has been read.

4.33.1 Detailed Description

Parameter to ATT_SERVER_EVT_ATTR_VALUE_READ event.

A pointer to this structure is passed to the ATT application callback when the value of an attribute has been read by a client.

4.34 bt_att_evt_authorization_requested_t Struct Reference

Parameter to ATT_SERVER_EVT_AUTHORIZATION_REQUESTED event.

```
#include <att.h>
```

Data Fields

- bt_att_session_t ∗ session

  ATT session.
- bt_byte opcode

  Operation code.
- bt_att_attribute_t ∗ attr

  ATT attribute.

4.34.1 Detailed Description

Parameter to ATT_SERVER_EVT_AUTHORIZATION_REQUESTED event.

A pointer to this structure is passed to the ATT application callback if authorization is required in order to access the attribute's value.

4.35 bt_att_evt_data_received_t Struct Reference

4.36 bt_att_evt_session_connected_t Struct Reference

Parameter to ATT_SERVER_EVT_SESSION_CONNECTED event.

```
#include <att.h>
```

Data Fields

- bt_att_session_t ∗ session
  ATT session.

4.36.1 Detailed Description

Parameter to ATT_SERVER_EVT_SESSION_CONNECTED event.

A pointer to this structure is passed to the ATT application callback when a client connected to the server.

4.37 bt_att_evt_session_disconnected_t Struct Reference

Parameter to ATT_SERVER_EVT_SESSION_DISCONNECTED event.

```
#include <att.h>
```

Data Fields

- bt_att_session_t ∗ session
  ATT session.

4.37.1 Detailed Description

Parameter to ATT_SERVER_EVT_SESSION_DISCONNECTED event.

A pointer to this structure is passed to the ATT application callback when a client disconnected from the server.

4.38 bt_att_evt_tran_timeout_t Struct Reference

Parameter to ATT_SERVER_EVT_TRAN_TIMEOUT event.

```
#include <att.h>
```

Data Fields

- bt_att_session_t ∗ session
  ATT session.

4.38.1 Detailed Description

Parameter to ATT_SERVER_EVT_TRAN_TIMEOUT event.

A pointer to this structure is passed to the ATT application callback if operation has timed out.

4.39 bt_att_find_by_type_value_response_t Struct Reference

Structure to store response to a "find by type value" request.

```
#include <att_client.h>
```

Data Fields

- bt_uint handle

  Found Attribute Handle.
- bt_uint group_end_handle

  Group End Handle.

4.39.1 Detailed Description

Structure to store response to a "find by type value" request.

An array of this structures is passed to bt_att_client_find_by_type_value(). It is application's responsibility to allocate memory for the array. When response is received the actual number of valid entries in the array is stored in bt_↩ att_client_evt_find_by_type_value_response_t::count.

4.40 bt_att_find_info_response_t Struct Reference

Structure to store response to a "find info" request.

```
#include <att_client.h>
```

Data Fields

- bt_byte format

  Length of the attribute UUID.
- bt_uint handle

  Attribute handle.
- union {
  bt_uint uuid16
  16-bit UUID
  bt_uuid_t uuid128
  128-bit UUID
  } type

  Attribute UUID.

4.40.1 Detailed Description

Structure to store response to a "find info" request.

An array of this structures is passed to bt_att_client_find_info(). It is application's responsibility to allocate memory for the array. When response is received the actual number of valid entries in the array is stored in bt_att_client_↩ evt_info_response_t::count.

4.40.2 Field Documentation

4.40.2.1 bt_byte format

Length of the attribute UUID.

- 0x01 = 16-bit UUID
- 0x02 = 128-bit UUID

4.41 bt_att_found_attribyte_t Struct Reference

4.42 bt_att_listener_t Struct Reference

4.43 bt_att_mgr_t Struct Reference

ATT manager.

`#include <att.h>`

4.43.1 Detailed Description

ATT manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with `bt_att_get_mgr()`.

4.44 bt_att_read_by_group_type_response_t Struct Reference

Structure to store response to a "read by group type" request.

`#include <att_client.h>`

Data Fields

- bt_uint handle

    Attribute handle.
- bt_uint end_group_handle

    Group End Handle.
- const bt_byte ∗ value

    Attribute value.
- bt_byte len

    Attribute value length.

4.44.1 Detailed Description

Structure to store response to a "read by group type" request.

An array of this structures is passed to bt_att_client_read_by_group_type(). It is application's responsibility to allocate memory for the array. When response is received the actual number of valid entries in the array is stored in bt_att_client_evt_read_by_group_type_response_t::count.

4.45 bt_att_read_by_type_response_t Struct Reference

Structure to store response to a "read by type" request.

```
#include <att_client.h>
```

Data Fields

- bt_uint handle

    Attribute handle.
- const bt_byte ∗ value

    Attribute value.
- bt_byte len

    Attribute value length.

4.45.1 Detailed Description

Structure to store response to a "read by type" request.

An array of this structures is passed to bt_att_client_read_by_type(). It is application's responsibility to allocate memory for the array. When response is received the actual number of valid entries in the array is stored in bt_←
att_client_evt_read_by_type_response_t::count.

4.46 bt_att_session_evt_authorization_requested_t Struct Reference

4.47 bt_att_session_evt_packet_sent_t Struct Reference

4.48 bt_att_session_t Struct Reference

4.49 bt_av_add_to_now_playing_t Struct Reference

Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

    Common response header.
- bt_byte status

    The result of the request.

4.49.1 Detailed Description

Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::add_to_now_playing_status - when a local device received a response to a "add to now playing" request.

4.50 bt_av_battery_status_of_ct_t Struct Reference

Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event.

`#include <avrcp_command.h>`

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte status

  Battery status.

4.50.1 Detailed Description

Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event←_t union - bt_avrcp_event_t::battery_status_of_ct - when a local device received a "battery status of controller" command.

4.50.2 Field Documentation

4.50.2.1 bt_byte status

Battery status.

This can be one of the following values:

- AVC_BATTERY_STATUS_NORMAL
- AVC_BATTERY_STATUS_WARNING
- AVC_BATTERY_STATUS_CRITICAL
- AVC_BATTERY_STATUS_EXTERNAL
- AVC_BATTERY_STATUS_FULL_CHARGE

4.51 bt_av_capability_company_id_t Struct Reference

Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event.

`#include <avrcp_command.h>`

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte count

  The number of supported company ids.
- bt_ulong ∗ company_id_list

  List of supported company ids.

4.51.1 Detailed Description

Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::company_id - when a local device received a response to a "get company id" request.

4.52 bt_av_capability_event_id_t Struct Reference

Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte count

  The number of supported events ids.
- bt_byte ∗ event_id_list

  List of supported event ids.

4.52.1 Detailed Description

Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::supported_event_id - when a local device received a response to a "get supported events" request.

4.53 bt_av_command_t Struct Reference

4.54 bt_av_displayable_character_set_t Struct Reference

Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte count

  The number of supported characters sets by the controller.
- bt_uint ∗ charset_list

  List of supported characters sets by the controller.

4.54.1 Detailed Description

Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event↩_t union - bt_avrcp_event_t::displayable_character_set - when a local device received a "displayable chracter set command" request.

4.55 bt_av_element_attribute_t Struct Reference

Media element attribute.

```
#include <avrcp_command.h>
```

Data Fields

- bt_ulong id
  Attribute Id.
- bt_uint charset
  Charcater set.
- bt_uint len
  Value length.
- bt_byte ∗ value
  Attribute value.

4.55.1 Detailed Description

Media element attribute.

This structure is used to store media element attribute.

4.56 bt_av_element_attributes_t Struct Reference

Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header
  Common response header.
- bt_byte count
  The number of attributes returned.
- bt_av_element_attribute_t ∗ attr_list
  List of attribute values.

4.56.1 Detailed Description

Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::element_attributes - when a local device received a response to a "get media element attributes" request.

4.57 bt_av_element_id_t Struct Reference

Media element UID.

```
#include <avrcp_command.h>
```

Data Fields

- bt_ulong id_lo

  4 least significant bytes of UID.
- bt_ulong id_hi

  4 most significant bytes of UID.

4.57.1 Detailed Description

Media element UID.

This structure is used to store media element UID.

4.58 bt_av_get_element_attributes_t Struct Reference

Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_av_element_id_t id

  Media element UID.
- bt_byte attributes

  Bitmask that defines attributes requested.

4.58.1 Detailed Description

Parameter to AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::get_element_attributes - when a local device received a "get element attributes" request.

4.58.2 Field Documentation

4.58.2.1 bt_byte attributes

Bitmask that defines attributes requested.

This can be a combination of the following values:

- AVC_MEDIA_ATTR_FLAG_TITLE
- AVC_MEDIA_ATTR_FLAG_ARTIST
- AVC_MEDIA_ATTR_FLAG_ALBUM

- AVC_MEDIA_ATTR_FLAG_NUMBER
- AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER
- AVC_MEDIA_ATTR_FLAG_GENRE
- AVC_MEDIA_ATTR_FLAG_PLAYING_TIME

4.59 bt_av_notification_addressed_player_changed_t Struct Reference

Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_uint player_id

  New player Id.
- bt_uint uid_counter

  UID counter.

4.59.1 Detailed Description

Parameter to AVRCP_EVT_ADDRESSED_PLAYER_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:addressed_player - when a local device received a "addressed player changed" notification.

4.60 bt_av_notification_app_setting_changed_t Struct Reference

4.61 bt_av_notification_battery_status_t Struct Reference

Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_byte status

  New battery status.This can be one of the following values:

4.61.1 Detailed Description

Parameter to AVRCP_EVT_BATT_STATUS_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event←_t union - bt_avrcp_event_t::notification::params:battery_status - when a local device received a "battery status changed" notification.

4.61.2 Field Documentation

4.61.2.1 bt_byte status

New battery status.This can be one of the following values:

- AVC_BATTERY_STATUS_NORMAL
- AVC_BATTERY_STATUS_WARNING
- AVC_BATTERY_STATUS_CRITICAL
- AVC_BATTERY_STATUS_EXTERNAL
- AVC_BATTERY_STATUS_FULL_CHARGE

4.62 bt_av_notification_playback_pos_changed_t Struct Reference

Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_ulong position

  New playback position.

4.62.1 Detailed Description

Parameter to AVRCP_EVT_PLAYBACK_POS_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event←_t union - bt_avrcp_event_t::notification::params:playback_pos - when a local device received a "playback position changed" notification.

4.63 bt_av_notification_playback_status_changed_t Struct Reference

Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte status

  Play status.

4.63.1 Detailed Description

Parameter to AVRCP_EVT_PLAYBACK_STATUS_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:play_status - when a local device received a "play status changed" notification.

4.63.2 Field Documentation

4.63.2.1 bt_byte status

Play status.

This can be on of the following values:

- AVC_PLAY_STATUS_STOPPED
- AVC_PLAY_STATUS_PLAYING
- AVC_PLAY_STATUS_PAUSED
- AVC_PLAY_STATUS_FW_SEEK
- AVC_PLAY_STATUS_REV_SEEK
- AVC_PLAY_STATUS_ERROR

4.64 bt_av_notification_system_status_changed_t Struct Reference

Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event.

`#include <avrcp_command.h>`

Data Fields

- bt_byte status

  New system status.

4.64.1 Detailed Description

Parameter to AVRCP_EVT_SYSTEM_STATUS_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event↩_t union - bt_avrcp_event_t::notification::params:system_status - when a local device received a "system status changed" notification.

4.65 bt_av_notification_t Struct Reference

Parameter to the following events:

`#include <avrcp_command.h>`

Data Fields

- bt_av_response_t header

  Common response header.
- bt_av_notification_playback_status_changed_t play_status

  Valid if notification is AVRCP_EVT_PLAYBACK_STATUS_CHANGED.
- bt_av_notification_track_changed_t track

  Valid if notification is AVRCP_EVT_TRACK_CHANGED.
- bt_av_notification_playback_pos_changed_t playback_pos

  Valid if notification is AVRCP_EVT_PLAYBACK_POS_CHANGED.

- bt_av_notification_battery_status_t battery_status

  Valid if notification is AVRCP_EVT_BATT_STATUS_CHANGED.
- bt_av_notification_system_status_changed_t system_status

  Valid if notification is AVRCP_EVT_SYSTEM_STATUS_CHANGED.
- bt_av_notification_addressed_player_changed_t addressed_player

  Valid if notification is AVRCP_EVT_ADDRESSED_PLAYER_CHANGED.
- bt_av_notification_uids_changed_t uids

  Valid if notification is AVRCP_EVT_UIDS_CHANGED.
- bt_av_notification_volume_changed_t volume

  Valid if notification is AVRCP_EVT_VOLUME_CHANGED.
- bt_av_notification_app_setting_changed_t app_setting

  Valid if notification is AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED.

4.65.1 Detailed Description

Parameter to the following events:

- AVRCP_EVT_PLAYBACK_STATUS_CHANGED
- AVRCP_EVT_TRACK_CHANGED
- AVRCP_EVT_PLAYBACK_POS_CHANGED
- AVRCP_EVT_BATT_STATUS_CHANGED
- AVRCP_EVT_SYSTEM_STATUS_CHANGED
- AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED
- AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED
- AVRCP_EVT_ADDRESSED_PLAYER_CHANGED
- AVRCP_EVT_UIDS_CHANGED
- AVRCP_EVT_VOLUME_CHANGED

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event↩ _t union - bt_avrcp_event_t::notification - when a local device received one of the following notifications from the target:

- Play status changed
- Track changed changed
- Playback position changed
- Battery status changed
- System status changed
- Addressed player changed
- UIDs changed
- Volume changed
- Player application setting changed The notification code defines which member of the bt_av_notification_t↩ ::params union is valid
- AVRCP_EVT_PLAYBACK_STATUS_CHANGED `bt_av_notification_playback_status_↩ changed_t` play_status

- AVRCP_EVT_TRACK_CHANGED `bt_av_notification_track_changed_t` track
- AVRCP_EVT_PLAYBACK_POS_CHANGED `bt_av_notification_playback_pos_changed_t` playback_pos
- AVRCP_EVT_BATT_STATUS_CHANGED `bt_av_notification_battery_status_t` battery_status
- AVRCP_EVT_SYSTEM_STATUS_CHANGED `bt_av_notification_system_status_changed_t` system_status
- AVRCP_EVT_ADDRESSED_PLAYER_CHANGED `bt_av_notification_addressed_player_changed_t` addressed_player
- AVRCP_EVT_UIDS_CHANGED `bt_av_notification_uids_changed_t` uids
- AVRCP_EVT_VOLUME_CHANGED `bt_av_notification_volume_changed_t` volume
- AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED `bt_av_notification_app_setting_changed_t` app_setting

4.66 bt_av_notification_track_changed_t Struct Reference

Parameter to AVRCP_EVT_TRACK_CHANGED event.

`#include <avrcp_command.h>`

Data Fields

- bt_av_element_id_t id

  New track UID.

4.66.1 Detailed Description

Parameter to AVRCP_EVT_TRACK_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:track - when a local device received a "track changed" notification.

4.67 bt_av_notification_uids_changed_t Struct Reference

Parameter to AVRCP_EVT_UIDS_CHANGED event.

`#include <avrcp_command.h>`

Data Fields

- bt_uint uid_counter

  UID counter.

4.67.1 Detailed Description

Parameter to AVRCP_EVT_UIDS_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:uids - when a local device received a "UIDs changed" notification.

4.68 bt_av_notification_volume_changed_t Struct Reference

Parameter to AVRCP_EVT_VOLUME_CHANGED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_byte volume

  Volume.

4.68.1 Detailed Description

Parameter to AVRCP_EVT_VOLUME_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::notification::params:volume - when a local device received a "UIDs changed" notification.

4.69 bt_av_play_item_t Struct Reference

Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte status

  The result of the request.

4.69.1 Detailed Description

Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_item_status - when a local device received a response to a "play item" request.

4.70 bt_av_play_status_t Struct Reference

Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_ulong song_length

  Current track length.

- bt_ulong song_position

  Current track position.
- bt_byte play_status

  Playback status.

4.70.1 Detailed Description

Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::play_status - when a local device received a response to a "get play status" request.

4.70.2 Field Documentation

4.70.2.1 bt_byte play_status

Playback status.

This can be one of the following values:

- AVC_PLAY_STATUS_STOPPED
- AVC_PLAY_STATUS_PLAYING
- AVC_PLAY_STATUS_PAUSED
- AVC_PLAY_STATUS_FW_SEEK
- AVC_PLAY_STATUS_REV_SEEK
- AVC_PLAY_STATUS_ERROR

4.71 bt_av_player_setting_current_values_t Struct Reference

Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte count

  The number of player setting attribute ids to be returned from the target.
- bt_byte ∗ setting_id_list

  List of player setting attribute ids to be returned from the target.
- bt_byte ∗ setting_value_id_list

  List of current player setting attribute value ids.

4.71.1 Detailed Description

Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_current_values - when a local device received a response to a "get current player setting attribute values" request.

4.72 bt_av_player_setting_values_t Struct Reference

Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header
  Common response header.
- bt_byte count
  The number of supported player setting attribute value ids.
- bt_byte ∗ setting_value_id_list
  List of supported player setting attribute value ids.

4.72.1 Detailed Description

Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values - when a local device received a response to a "get player setting attribute values" request.

4.73 bt_av_player_setting_values_text_t Struct Reference

Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header
  Common response header.
- bt_byte count
  The number of player setting attribute value ids for which displayable text is requested.
- bt_byte ∗ setting_value_id_list
  List of player setting attribute value ids for which displayable text is requested.
- bt_av_player_text_t ∗ setting_value_text_list
  List of player setting attribute values displayable text.

4.73.1 Detailed Description

Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_setting_values_text - when a local device received a response to a "get player setting attribute values displayable text" request.

4.74 bt_av_player_settings_t Struct Reference

Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte count

  The number of supported player setting attribute ids.
- bt_byte ∗ setting_id_list

  List of supported player setting attribute ids.

4.74.1 Detailed Description

Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event↩_t union - bt_avrcp_event_t::player_settings - when a local device received a response to a "get supported player setting attributes" request.

4.75 bt_av_player_settings_text_t Struct Reference

Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte count

  The number of player setting attribute ids for which displayable text is requested.
- bt_byte ∗ setting_id_list

  List of player setting attribute ids for which displayable text is requested.
- bt_av_player_text_t ∗ setting_text_list

  List of player setting attributes displayable text.

4.75.1 Detailed Description

Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::player_settings_text - when a local device received a response to a "get player setting attributes displayable text" request.

4.76 bt_av_player_text_t Struct Reference

4.77 bt_av_register_notification_t Struct Reference

Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte event_id

  Event Id.
- bt_ulong playback_pos

  Playback position. Used only if `event_id` is AVC_EVENT_PLAYBACK_POS_CHANGED.

4.77.1 Detailed Description

Parameter to AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::register_notification - when a local device received a "register notification" request.

4.77.2 Field Documentation

4.77.2.1 bt_byte event_id

Event Id.

This can be one of the following values:

- AVC_EVENT_PLAYBACK_STATUS_CHANGED
- AVC_EVENT_TRACK_CHANGED
- AVC_EVENT_TRACK_REACHED_END
- AVC_EVENT_TRACK_REACHED_START
- AVC_EVENT_PLAYBACK_POS_CHANGED
- AVC_EVENT_BATT_STATUS_CHANGED
- AVC_EVENT_SYSTEM_STATUS_CHANGED
- AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED
- AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED
- AVC_EVENT_AVAILABLE_PLAYERS_CHANGED
- AVC_EVENT_ADDRESSED_PLAYER_CHANGED
- AVC_EVENT_UIDS_CHANGED
- AVC_EVENT_VOLUME_CHANGED

4.78 bt_av_response_t Struct Reference

AV/C response header.

```
#include <avrcp_command.h>
```

Data Fields

- bt_avrcp_channel_t ∗ channel

  AVRCP channel.
- bt_byte ctype

  Response type.
- bt_byte tran_id

  Transaction Id.

4.78.1 Detailed Description

AV/C response header.

This structure is used to store fields present in every AV/C response.

4.78.2 Field Documentation

4.78.2.1 bt_byte ctype

Response type.

This can be one of the following values:

- AVC_RESPONSE_NOT_IMPLEMENTED
- AVC_RESPONSE_ACCEPTED
- AVC_RESPONSE_REJECTED
- AVC_RESPONSE_IN_TRANSITION
- AVC_RESPONSE_STABLE
- AVC_RESPONSE_IMPLEMENTED
- AVC_RESPONSE_CHANGED
- AVC_RESPONSE_INTERIM
- AVC_RESPONSE_TIMEOUT

4.79 bt_av_set_absolute_volume_t Struct Reference

Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event.

`#include <avrcp_command.h>`

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte volume

  Volume.

4.79.1 Detailed Description

Parameter to AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::absolute_volume - when a local device received a response to a "set absolute volume" request.

4.80 bt_av_set_addressed_player_t Struct Reference

Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event.

```
#include <avrcp_command.h>
```

Data Fields

- bt_av_response_t header

  Common response header.
- bt_byte status

  The result of changing the addressed player.

4.80.1 Detailed Description

Parameter to AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::addressed_player - when a local device received a response to a "set addressed player" request.

4.81 bt_avctp_channel_t Struct Reference

AVCTP channel description.

```
#include <avctp.h>
```

4.81.1 Detailed Description

AVCTP channel description.

This structure is used to hold information about an AVCTP channel.

4.82 bt_avctp_event_t Union Reference

Parameter to an application callback.

```
#include <avctp.h>
```

Data Fields

- bt_avctp_evt_channel_connected_t channel_connected

  Valid if event is AVCTP_EVT_CHANNEL_CONNECTED.
- bt_avctp_evt_channel_disconnected_t channel_disconnected

Valid if event is AVCTP_EVT_CHANNEL_DISCONNECTED.
- bt_avctp_evt_connection_failed_t connection_failed

  Valid if event is AVCTP_EVT_CONNECTION_FAILED.
- bt_avctp_evt_command_received_t command_received

  Valid if event is AVCTP_EVT_COMMAND_RECEIVED.
- bt_avctp_evt_response_received_t response_received

  Valid if event is AVCTP_EVT_RESPONE_RECEIVED.
- bt_avctp_evt_command_sent_t command_sent

  Valid if event is AVCTP_EVT_COMMAND_SENT.
- bt_avctp_evt_response_sent_t response_sent

  Valid if event is AVCTP_EVT_RESPONSE_SENT.
- bt_avctp_evt_command_cancelled_t command_cancelled

  Valid if event is AVCTP_EVT_COMMAND_CANCELLED.
- bt_avctp_evt_response_cancelled_t response_cancelled

  Valid if event is AVCTP_EVT_RESPONSE_CANCELLED.

4.82.1 Detailed Description

Parameter to an application callback.

This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

4.83 bt_avctp_evt_channel_connected_t Struct Reference

Parameter to AVCTP_EVT_CHANNEL_CONNECTED event.

```
#include <avctp.h>
```

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.

4.83.1 Detailed Description

Parameter to AVCTP_EVT_CHANNEL_CONNECTED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_connected - when a channel between two devices has been established.

4.84 bt_avctp_evt_channel_disconnected_t Struct Reference

Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event.

```
#include <avctp.h>
```

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.

4.84.1 Detailed Description

Parameter to AVCTP_EVT_CHANNEL_DISCONNECTED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::channel_disconnected - when a channel between two devices has been terminated.

4.85 bt_avctp_evt_command_cancelled_t Struct Reference

Parameter to AVCTP_EVT_COMMAND_CANCELLED event.

`#include <avctp.h>`

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.
- bt_avctp_message_t ∗ command

  AVCTP command message.

4.85.1 Detailed Description

Parameter to AVCTP_EVT_COMMAND_CANCELLED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_cancelled - when sending a command message has been canceled.

4.86 bt_avctp_evt_command_received_t Struct Reference

Parameter to AVCTP_EVT_COMMAND_RECEIVED event.

`#include <avctp.h>`

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.
- bt_avctp_message_t ∗ command

  AVCTP command message.

4.86.1 Detailed Description

Parameter to AVCTP_EVT_COMMAND_RECEIVED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_received - when a local device received a command message.

4.87 bt_avctp_evt_command_sent_t Struct Reference

Parameter to AVCTP_EVT_COMMAND_SENT event.

`#include <avctp.h>`

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.
- bt_avctp_message_t ∗ command

  AVCTP command message.

4.87.1 Detailed Description

Parameter to AVCTP_EVT_COMMAND_SENT event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::command_sent - when a local device finished sending a command message.

4.88 bt_avctp_evt_connection_failed_t Struct Reference

Parameter to AVCTP_EVT_CONNECTION_FAILED event.

```
#include <avctp.h>
```

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.

4.88.1 Detailed Description

Parameter to AVCTP_EVT_CONNECTION_FAILED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::connection_failed - when a channel between two devices could not be established.

4.89 bt_avctp_evt_response_cancelled_t Struct Reference

Parameter to AVCTP_EVT_RESPONSE_CANCELLED event.

```
#include <avctp.h>
```

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.
- bt_avctp_message_t ∗ response

  AVCTP response message.

4.89.1 Detailed Description

Parameter to AVCTP_EVT_RESPONSE_CANCELLED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_cancelled - when sending a response message has been canceled.

4.90 bt_avctp_evt_response_received_t Struct Reference

Parameter to AVCTP_EVT_RESPONSE_RECEIVED event.

```
#include <avctp.h>
```

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.
- bt_avctp_message_t ∗ response

  AVCTP response message.

4.90.1 Detailed Description

Parameter to AVCTP_EVT_RESPONSE_RECEIVED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_received - when a local device received a response message.

4.91 bt_avctp_evt_response_sent_t Struct Reference

Parameter to AVCTP_EVT_RESPONSE_SENT event.

```
#include <avctp.h>
```

Data Fields

- bt_avctp_channel_t ∗ channel

  AVCTP channel.
- bt_avctp_message_t ∗ response

  AVCTP response message.

4.91.1 Detailed Description

Parameter to AVCTP_EVT_RESPONSE_SENT event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt_avctp_event_t union - bt_avctp_event_t::response_sent - when a local device finished sending a response message.

4.92 bt_avctp_message_t Struct Reference

AVCTP message description.

```
#include <avctp.h>
```

4.92.1 Detailed Description

AVCTP message description.

This structure is used to hold information about an AVCTP message.

4.93 bt_avctp_mgr_t Struct Reference

AVCTP manager.

`#include <avctp.h>`

Data Fields

- bt_byte state

  Manager state.
- bt_byte flags

  Additional manager state.
- bt_avctp_channel_t ∗ channels

  List of available AVCTP channels.
- bt_avctp_transport_t ∗ transports

  List of available AVCTP transports.
- bt_avctp_mgr_callback_fp callback

  Pointer to a function used to notify the AVCTP consumer about various events.
- void ∗ callback_param

  Pointer to arbitrary data to be passed to the `callback`*.*
- bt_avctp_channel_t ∗ opening_channel

  Pointer to a channle being open.

4.93.1 Detailed Description

AVCTP manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with `bt_avctp_get_mgr()`.

4.93.2 Field Documentation

4.93.2.1 bt_byte flags

Additional manager state.

This value can be a combination of the following values:

- AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET

4.93.2.2 bt_byte state

Manager state.

This value can be one of the following values:

- AVCTP_MANAGER_STATE_IDLE
- AVCTP_MANAGER_STATE_CONNECTING

4.94 bt_avctp_transport_t Struct Reference

AVCTP transport description.

`#include <avctp.h>`

4.94.1 Detailed Description

AVCTP transport description.

This structure is used to hold information about an AVCTP transport.

4.95 bt_avdtp_codec_op_decode_t Struct Reference

4.96 bt_avdtp_codec_op_encode_t Struct Reference

4.97 bt_avdtp_codec_op_param_t Union Reference

Parameter to a codec handler.

`#include <avdtp.h>`

Data Fields

- bt_avdtp_codec_op_parse_config_t parse

  Valid if operation is AVDTP_CODEC_OPCODE_PARSE_CONFIG.
- bt_avdtp_codec_op_serialize_config_t serialize

  Valid if operation is AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG.
- bt_avdtp_codec_op_parse_packet_t parse_packet

  This member is currently not used.

4.97.1 Detailed Description

Parameter to a codec handler.

This union is used to pass operation specific data to a codec handler. Which member of the union points to a valid structure depends on the operation.

4.98 bt_avdtp_codec_op_parse_config_t Struct Reference

Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation.

`#include <avdtp.h>`

Data Fields

- void ∗ codec_config

  A pointer to a structure defined by the AVDTP consumer where codec's configuration will be stored by the handler.
- bt_byte codec_config_max_size

  The maximum size of a buffer pointed to by `codec_config` field.
- bt_byte ∗ buffer

  A pointer to a buffer holding codec's configuration in OTA format.
- bt_int buffer_len

  The length of the `buffer`.
- bt_int offset

  The `buffer` points to a complete packet received from the remoted device.

4.98.1 Detailed Description

Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation.

A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param↩_t union - bt_avdtp_codec_op_param_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.

4.98.2 Field Documentation

4.98.2.1 void∗ codec_config

A pointer to a structure defined by the AVDTP consumer where codec's configuration will be stored by the handler.

The format of the structure is totally up to the AVDTP consumer. The dotstack defines such structures for SBC, MPEG-1,2 and MPEG-2,4 AAC:

- SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h)
- MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h)
- MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)

4.98.2.2 bt_int offset

The `buffer` points to a complete packet received from the remoted device.

The `offset` points to a location in the `buffer` where codec's configuration starts.

4.99 bt_avdtp_codec_op_parse_packet_t Struct Reference

4.100 bt_avdtp_codec_op_serialize_config_t Struct Reference

Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation.

`#include <avdtp.h>`

Data Fields

- void ∗ codec_config

  A pointer to a structure defined by the AVDTP consumer where codec's configuration will be read from by the handler.
- bt_byte ∗ buffer

  A pointer to a buffer where the handler has to write codec's configuration in OTA format.
- bt_int buffer_len

  The length of the `buffer`.
- bt_int offset

  The `buffer` points to a complete packet that will be sent to the remote device.

4.100.1 Detailed Description

Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation.

A pointer to this structure is passed to the codec handler as a valid member of the bt_avdtp_codec_op_param_t union - bt_avdtp_codec_op_param_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.

4.100.2 Field Documentation

4.100.2.1 void∗ codec_config

A pointer to a structure defined by the AVDTP consumer where codec's configuration will be read from by the handler.

The format of the structure is totally up to the AVDTP consumer. The dotstack defines such structures for SBC, MPEG-1,2 and MPEG-2,4 AAC:

- SBC: bt_a2dp_sbc_config_t (defined in a2dp_sbc_codec.h)
- MPEG-1,2: bt_a2dp_mpeg_config_t (defined in a2dp_mpeg_codec.h)
- MPEG-2,4 AAC: bt_a2dp_aac_config_t (defined in a2dp_aac_codec.h)

4.100.2.2 bt_int offset

The `buffer` points to a complete packet that will be sent to the remote device.

The `offset` points to a location in the `buffer` where codec's configuration has to be written.

4.101 bt_avdtp_codec_t Struct Reference

Codec handler description.

`#include <avdtp.h>`

Data Fields

- bt_byte codec_type
  Codec type.
- bt_avdtp_codec_handler_fp codec_handler
  A pointer to a codec handler.

4.101.1 Detailed Description

Codec handler description.

This structure is used to register a codec handler for parsing/serializing codec capabilities and configuration. See description of the bt_avdtp_register_codec for more details.

4.101.2 Field Documentation

4.101.2.1 bt_byte codec_type

Codec type.

The `codec_type` can be one of the following values:

- AVDTP_CODEC_TYPE_SBC: SBC
- AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)
- AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products)
- AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products)
- AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

4.102 bt_avdtp_event_t Union Reference

Parameter to an application callback.

`#include <avdtp.h>`

Data Fields

- bt_avdtp_evt_ctrl_channel_connected_t ctrl_channel_connected
  Valid if event is AVDTP_EVT_CTRL_CHANNEL_CONNECTED.
- bt_avdtp_evt_ctrl_channel_disconnected_t ctrl_channel_disconnected
  Valid if event is AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED.
- bt_avdtp_evt_discover_completed_t discover_completed
  Valid if event is AVDTP_EVT_DISCOVER_COMPLETED.
- bt_avdtp_evt_sep_info_received_t sep_info_received
  Valid if event is AVDTP_EVT_SEP_INFO_RECEIVED.
- bt_avdtp_evt_get_sep_capabilities_completed_t get_sep_capabilities_completed
  Valid if event is AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED.
- bt_avdtp_evt_sep_capabilities_received_t sep_capabilities_received
  Valid if event is AVDTP_EVT_SEP_CAPABILITIES_RECEIVED.
- bt_avdtp_evt_set_stream_configuration_completed_t set_stream_configuration_completed
  Valid if event is AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED.
- bt_avdtp_evt_get_stream_configuration_completed_t get_stream_configuration_completed
  Valid if event is AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED.
- bt_avdtp_evt_stream_reconfigure_completed_t stream_reconfigure_completed
  Valid if event is AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED.
- bt_avdtp_evt_open_stream_completed_t open_stream_completed
  Valid if event is AVDTP_EVT_OPEN_STREAM_COMPLETED.
- bt_avdtp_evt_start_stream_completed_t start_stream_completed
  Valid if event is AVDTP_EVT_START_STREAM_COMPLETED.
- bt_avdtp_evt_close_stream_completed_t close_stream_completed
  Valid if event is AVDTP_EVT_CLOSE_STREAM_COMPLETED.
- bt_avdtp_evt_suspend_stream_completed_t suspend_stream_completed
  Valid if event is AVDTP_EVT_SUSPEND_STREAM_COMPLETED.
- bt_avdtp_evt_stream_security_control_completed_t security_control_completed
  Valid if event is AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED.
- bt_avdtp_evt_set_stream_configuration_requested_t set_stream_configuration_requested
  Valid if event is AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED.
- bt_avdtp_evt_reconfigure_stream_requested_t reconfigure_stream_requested
  Valid if event is AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED.
- bt_avdtp_evt_open_stream_requested_t open_stream_requested
  Valid if event is AVDTP_EVT_OPEN_STREAM_REQUESTED.
- bt_avdtp_evt_start_stream_requested_t start_stream_requested
  Valid if event is AVDTP_EVT_START_STREAM_REQUESTED.
- bt_avdtp_evt_suspend_stream_requested_t suspend_stream_requested
  Valid if event is AVDTP_EVT_SUSPEND_STREAM_REQUESTED.
- bt_avdtp_evt_close_stream_requested_t close_stream_requested
  Valid if event is AVDTP_EVT_CLOSE_STREAM_REQUESTED.
- bt_avdtp_evt_abort_stream_requested_t abort_stream_requested
  Valid if event is AVDTP_EVT_ABORT_STREAM_REQUESTED.
- bt_avdtp_evt_delay_report_completed_t delay_report_completed

Valid if event is AVDTP_EVT_DELAYREPORT_COMPLETED.

- bt_avdtp_evt_stream_configured_t stream_configured

    Valid if event is AVDTP_EVT_STREAM_CONFIGURED.
- bt_avdtp_evt_stream_reconfigured_t stream_reconfigured

    Valid if event is AVDTP_EVT_STREAM_RECONFIGURED.
- bt_avdtp_evt_stream_opened_t stream_opened

    Valid if event is AVDTP_EVT_STREAM_OPENED.
- bt_avdtp_evt_stream_started_t stream_started

    Valid if event is AVDTP_EVT_STREAM_STARTED.
- bt_avdtp_evt_stream_suspended_t stream_suspended

    Valid if event is AVDTP_EVT_STREAM_SUSPENDED.
- bt_avdtp_evt_stream_closed_t stream_closed

    Valid if event is AVDTP_EVT_STREAM_CLOSED.
- bt_avdtp_evt_stream_aborted_t stream_aborted

    Valid if event is AVDTP_EVT_STREAM_ABORTED.
- bt_avdtp_evt_media_packet_received_t media_packet_received

    Valid if event is AVDTP_EVT_MEDIA_PACKET_RECEIVED.
- bt_avdtp_evt_media_packet_sent_t media_packet_sent

    Valid if event is AVDTP_EVT_MEDIA_PACKET_SENT.
- bt_avdtp_evt_media_packet_send_failed_t media_packet_send_failed

    Valid if event is AVDTP_EVT_MEDIA_PACKET_SEND_FAILED.

4.102.1 Detailed Description

Parameter to an application callback.

This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

4.103 bt_avdtp_evt_abort_stream_requested_t Struct Reference

Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte err_code

    The result to be sent to the remote party.
- bt_byte strm_handle

    The handle of a stream to abort.

4.103.1 Detailed Description

Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::abort_stream_requested - when AVDTP received a "abort stream" request.

4.103.2 Field Documentation

4.103.2.1 bt_byte err_code

The result to be sent to the remote party.

- If local device accepts the configuration requested by the remote device it should set `err_code` to AVDT↩ P_ERROR_SUCCESS. Otherwise it should set `err_code` to one of the AVDTP_ERROR_ constants.

4.104 bt_avdtp_evt_close_stream_completed_t Struct Reference

Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result of the request.
- bt_byte strm_handle

  Stream handle.

4.104.1 Detailed Description

Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_completed - when AVDTP received a response to a "close stream" request.

4.104.2 Field Documentation

4.104.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.105 bt_avdtp_evt_close_stream_requested_t Struct Reference

Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result to be sent to the remote party.
- bt_byte strm_handle

  The handle of a stream to close.

4.105.1 Detailed Description

Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::close_stream_requested - when AVDTP received a "close stream" request.

4.105.2 Field Documentation

4.105.2.1 bt_byte err_code

The result to be sent to the remote party.

- If local device accepts the configuration requested by the remote device it should set `err_code` to AVDT←P_ERROR_SUCCESS. Otherwise it should set `err_code` to one of the AVDTP_ERROR_ constants.

4.106 bt_avdtp_evt_ctrl_channel_connected_t Struct Reference

Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event.

```
#include <avdtp.h>
```

Data Fields

- bt_bdaddr_t ∗ bdaddr

  the address of a remote device

4.106.1 Detailed Description

Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_←event_t union - bt_avdtp_event_t::ctrl_channel_connected - when a control channel between two devices has been established.

4.107 bt_avdtp_evt_ctrl_channel_disconnected_t Struct Reference

Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event.

```
#include <avdtp.h>
```

Data Fields

- bt_bdaddr_t ∗ bdaddr

  the address of a remote device

4.107.1 Detailed Description

Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event←_t union - bt_avdtp_event_t::ctrl_channel_disconnected - when a control channel between two devices has been terminated.

4.108 bt_avdtp_evt_delay_report_completed_t Struct Reference

Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result of the request.

4.108.1 Detailed Description

Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::delay_report_completed - when AVDTP received a response to a "delay report" request.

4.108.2 Field Documentation

4.108.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.109 bt_avdtp_evt_discover_completed_t Struct Reference

Parameter to AVDTP_EVT_DISCOVER_COMPLETED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result of discovering.

4.109.1 Detailed Description

Parameter to AVDTP_EVT_DISCOVER_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::discover_completed - when AVDTP completed discovering SEPs available on a remote device.

4.109.2 Field Documentation

4.109.2.1 bt_byte err_code

The result of discovering.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.110 bt_avdtp_evt_get_sep_capabilities_completed_t Struct Reference

Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte err_code

  The result of the request.

4.110.1 Detailed Description

Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_↩ event_t union - bt_avdtp_event_t::get_sep_capabilities_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_SEP_CAPABILITIES_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.

4.110.2 Field Documentation

4.110.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.111 bt_avdtp_evt_get_stream_configuration_completed_t Struct Reference

Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event.

```
#include <avdtp.h>
```

4.111.1 Detailed Description

Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event↩ _t union - bt_avdtp_event_t::get_stream_configuration_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP_EVT_STREAM_CONFIGU↩ RATION_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.

4.112 bt_avdtp_evt_media_packet_received_t Struct Reference

Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event.

`#include <avdtp.h>`

Data Fields

- bt_byte strm_handle

  The handle of a stream which received a packet.
- bt_media_packet_t ∗ packet

  A pointer to a media packet buffer.

4.112.1 Detailed Description

Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event↩_t union - bt_avdtp_event_t::media_packet_received - when ADVDTP received a media packet from the remote device.

4.113 bt_avdtp_evt_media_packet_send_failed_t Struct Reference

Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event.

`#include <avdtp.h>`

Data Fields

- bt_byte strm_handle

  The handle of a stream which received a packet.
- bt_media_packet_t ∗ packet

  A pointer to a media packet buffer.

4.113.1 Detailed Description

Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_send_failed - when ADVDTP failed to send a media packet to the remote device.

4.114 bt_avdtp_evt_media_packet_sent_t Struct Reference

Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event.

`#include <avdtp.h>`

Data Fields

- bt_byte strm_handle

  The handle of a stream which received a packet.
- bt_media_packet_t ∗ packet

  A pointer to a media packet buffer.

4.114.1 Detailed Description

Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::media_packet_sent - when ADVDTP sent a media packet to the remote device.

4.115 bt_avdtp_evt_open_stream_completed_t Struct Reference

Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result of the request.
- bt_byte strm_handle

  Stream handle.

4.115.1 Detailed Description

Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_completed - when AVDTP received a response to a "open stream" request.

4.115.2 Field Documentation

4.115.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.116 bt_avdtp_evt_open_stream_requested_t Struct Reference

Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

    The result to be sent to the remote party.
- bt_byte strm_handle

    The handle of a stream to open.

4.116.1 Detailed Description

Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::open_stream_requested - when AVDTP received a "open stream" request.

4.116.2 Field Documentation

4.116.2.1 bt_byte err_code

The result to be sent to the remote party.

- If local device accepts the configuration requested by the remote device it should set `err_code` to AVDT↩ P_ERROR_SUCCESS. Otherwise it should set `err_code` to one of the AVDTP_ERROR_ constants.

4.117 bt_avdtp_evt_reconfigure_stream_requested_t Struct Reference

Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte err_code

    The result to be sent to the remote party.
- bt_byte err_category

    If local device cannot accept the request it should set `err_category` to the value of the forst Service Category that failed.
- bt_avdtp_sep_t ∗ sep

    Description of the local SEP.
- bt_avdtp_sep_capabilities_t ∗ config

    Stream configuration requested by the remote party.
- bt_byte strm_handle

    Stream handle.

4.117.1 Detailed Description

Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::reconfigure_stream_requested - when AVDTP received a "change stream configuration" request.

4.117.2 Field Documentation

4.117.2.1 bt_byte err_code

The result to be sent to the remote party.

- If local device accepts the configuration requested by the remote device it should set `err_code` to AVDT↩ P_ERROR_SUCCESS. Otherwise it should set `err_code` to one of the AVDTP_ERROR_ constants.

4.118 bt_avdtp_evt_sep_capabilities_received_t Struct Reference

Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_↩ RECEIVED events.

`#include <avdtp.h>`

Data Fields

- bt_avdtp_sep_capabilities_t ∗ caps

    SEP capabilities or stream configuration.

4.118.1 Detailed Description

Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_CONFIGURATION_↩ RECEIVED events.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event↩ _t union - bt_avdtp_event_t::sep_capabilities_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.

4.119 bt_avdtp_evt_sep_info_received_t Struct Reference

Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event.

`#include <avdtp.h>`

Data Fields

- bt_byte sep_id

    SEP ID.
- bt_byte sep_type

    SEP type. This can be either AVDTP_SEP_TYPE_SOURCE or AVDTP_SEP_TYPE_SINK.
- bt_bool in_use

    A flag indicating if a SEP is already being used.
- bt_byte media_type

    Type of media supported by this SEP.

4.119.1 Detailed Description

Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::sep_info_received - when AVDTP received positive result to a "discover" request. AVD↩ TP_EVT_SEP_INFO_RECEIVED is generated for every SEP received from the remote device.

4.119.2 Field Documentation

4.119.2.1 bt_byte media_type

Type of media supported by this SEP.

This can be on of the following values:

- AVDTP_MEDIA_TYPE_AUDIO
- AVDTP_MEDIA_TYPE_VIDEO
- AVDTP_MEDIA_TYPE_MULTIMEDIA

4.120 bt_avdtp_evt_set_stream_configuration_completed_t Struct Reference

Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte err_code

  The result of the request.
- bt_byte strm_handle

  Stream handle.
- bt_byte svc_category

  The value of the first Service Category to fail if the remote rejected the request.

4.120.1 Detailed Description

Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_completed - when AVDTP received a response to a "set stream configuration" request.

4.120.2 Field Documentation

4.120.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.121 bt_avdtp_evt_set_stream_configuration_requested_t Struct Reference

Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result to be sent to the remote party.
- bt_byte err_category

  If local device cannot accept the request it should set `err_category` to the value of the forst Service Category that failed.
- bt_avdtp_sep_t ∗ sep

  Description of the local SEP.
- bt_byte int_sep_id

  The ID of the remote SEP.
- bt_avdtp_sep_capabilities_t ∗ config

  Stream configuration requested by the remote party.
- bt_byte strm_handle

  Stream handle.

4.121.1 Detailed Description

Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::set_stream_configuration_requested - when AVDTP received a "set stream configuration" request.

4.121.2 Field Documentation

4.121.2.1 bt_byte err_code

The result to be sent to the remote party.

- If local device accepts the configuration requested by the remote device it should set `err_code` to AVDT←P_ERROR_SUCCESS. Otherwise it should set `err_code` to one of the AVDTP_ERROR_ constants.

4.122 bt_avdtp_evt_start_stream_completed_t Struct Reference

Parameter to AVDTP_EVT_START_STREAM_COMPLETED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result of the request.
- bt_byte strm_handle

  Stream handle.

4.122.1 Detailed Description

Parameter to AVDTP_EVT_START_STREAM_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_completed - when AVDTP received a response to a "start stream" request.

4.122.2 Field Documentation

4.122.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.123 bt_avdtp_evt_start_stream_requested_t Struct Reference

Parameter to AVDTP_EVT_START_STREAM_REQUESTED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result to be sent to the remote party.
- bt_byte strm_handle

  The handle of a stream to start.

4.123.1 Detailed Description

Parameter to AVDTP_EVT_START_STREAM_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::start_stream_requested - when AVDTP received a "start stream" request.

4.123.2 Field Documentation

4.123.2.1 bt_byte err_code

The result to be sent to the remote party.

- If local device accepts the configuration requested by the remote device it should set `err_code` to AVDT↩ P_ERROR_SUCCESS. Otherwise it should set `err_code` to one of the AVDTP_ERROR_ constants.

4.124 bt_avdtp_evt_stream_aborted_t Struct Reference

Parameter to AVDTP_EVT_STREAM_ABORTED event.

`#include <avdtp.h>`

Data Fields

- bt_byte strm_handle

  The handle of a stream that has been aborted.

4.124.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_ABORTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event↩ _t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully aborted.

4.125 bt_avdtp_evt_stream_closed_t Struct Reference

Parameter to AVDTP_EVT_STREAM_CLOSED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte strm_handle

  The handle of a stream that has been closed.

4.125.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_CLOSED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event↩ _t union - bt_avdtp_event_t::stream_closed - to notify the AVDTP consumer that a stream has been successfully closed.

4.126 bt_avdtp_evt_stream_configured_t Struct Reference

Parameter to AVDTP_EVT_STREAM_CONFIGURED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte strm_handle

  The handle of a stream that has been configured.

4.126.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_CONFIGURED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.

4.127 bt_avdtp_evt_stream_opened_t Struct Reference

Parameter to AVDTP_EVT_STREAM_OPENED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte strm_handle

  The handle of a stream that has been opened.

4.127.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_OPENED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_opened - to notify the AVDTP consumer that a stream has been successfully opened.

4.128 bt_avdtp_evt_stream_reconfigure_completed_t Struct Reference

Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte err_code

  The result of the request.
- bt_byte strm_handle

  Stream handle.
- bt_byte svc_category

  The value of the first Service Category to fail if the remote rejected the request.

4.128.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigure_completed - when AVDTP received a response to a "change stream configuration" request.

4.128.2 Field Documentation

4.128.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.129 bt_avdtp_evt_stream_reconfigured_t Struct Reference

Parameter to AVDTP_EVT_STREAM_RECONFIGURED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte strm_handle

  The handle of a stream that has been reconfigured.

4.129.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_RECONFIGURED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.

4.130 bt_avdtp_evt_stream_security_control_completed_t Struct Reference

Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte err_code

  The result of the request.

4.130.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::security_control_completed - when AVDTP received a response to a "exchange content protection control data" request.

4.130.2 Field Documentation

4.130.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.131 bt_avdtp_evt_stream_started_t Struct Reference

Parameter to AVDTP_EVT_STREAM_STARTED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte strm_handle

  The handle of a stream that has been started.

4.131.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_STARTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event↵_t union - bt_avdtp_event_t::stream_started - to notify the AVDTP consumer that a stream has been successfully started.

4.132 bt_avdtp_evt_stream_suspended_t Struct Reference

Parameter to AVDTP_EVT_STREAM_SUSPENDED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte strm_handle

  The handle of a stream that has been suspended.

4.132.1 Detailed Description

Parameter to AVDTP_EVT_STREAM_SUSPENDED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::stream_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.

4.133 bt_avdtp_evt_suspend_stream_completed_t Struct Reference

Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event.

```
#include <avdtp.h>
```

Data Fields

- bt_byte err_code

  The result of the request.
- bt_byte strm_handle

  Stream handle.

4.133.1 Detailed Description

Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_completed - when AVDTP received a response to a "suspend stream" request.

4.133.2 Field Documentation

4.133.2.1 bt_byte err_code

The result of the request.

- If the remote accepted the request `err_code` == AVDTP_ERROR_SUCCESS.
- Otherwise `err_code` == the error code returned by the remote party.

4.134 bt_avdtp_evt_suspend_stream_requested_t Struct Reference

Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event.

`#include <avdtp.h>`

Data Fields

- bt_byte err_code

  The result to be sent to the remote party.
- bt_byte strm_handle

  The handle of a stream to suspend.

4.134.1 Detailed Description

Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt_avdtp_event_t union - bt_avdtp_event_t::suspend_stream_requested - when AVDTP received a "suspend stream" request.

4.134.2 Field Documentation

4.134.2.1 bt_byte err_code

The result to be sent to the remote party.

- If local device accepts the configuration requested by the remote device it should set `err_code` to AVDT↩
P_ERROR_SUCCESS. Otherwise it should set `err_code` to one of the AVDTP_ERROR_ constants.

4.135 bt_avdtp_mgr_t Struct Reference

AVDTP manager.

`#include <avdtp.h>`

Data Fields

- bt_byte state

  Manager state.
- bt_byte flags

  Additional manager state.

- bt_avdtp_sep_t ∗ seps

  Pointer to a buffer of SEPs available for allocating with `bt_avdtp_register_sep`.
- bt_byte next_sep_id

  Holds ID of the next SEP to be allocated. Every time `bt_avdtp_register_sep` is called this value is increased by 1.
- bt_avdtp_stream_t ∗ streams

  Pointer to a buffer of streams available for allocating with `bt_avdtp_create_stream`.
- bt_byte next_stream_handle

  Holds ID of the next stream to be allocated. Every time `bt_avdtp_create_stream` is called this value is increased by 1.
- bt_avdtp_control_channel_t ∗ control_channels

  Pointer to a buffer of available control channles.
- bt_avdtp_transport_channel_t ∗ transport_channels

  Pointer to a buffer of available transport channles.
- bt_avdtp_codec_t ∗ codecs

  A list of supported codecs.
- bt_avdtp_mgr_callback_fp callback

  Pointer to a function which a AVDTP consumer must register in order to be notified of various events.
- void ∗ callback_param

  Pointer to arbitrary data to be passed to the `callback`.
- bt_avdtp_control_cmd_t ∗ pending_command

  If local device wants to send a command to a remote device but control channel does not exists this member holds a pointer to the command until the channel is created.
- bt_avdtp_stream_t ∗ opening_strm

  Holds a pointer to a stream being opened by a remote device.

4.135.1 Detailed Description

AVDTP manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with `bt_avdtp_get_mgr()`.

4.135.2 Field Documentation

4.135.2.1 bt_byte flags

Additional manager state.

This value can be a combination of the following values:

- AVDTP_MANAGER_FLAG_SENDING_MEDIA_PACKET

4.135.2.2 bt_byte state

Manager state.

This value can be one of the following values:

- AVDTP_MANAGER_STATE_IDLE
- AVDTP_MANAGER_STATE_CONNECTING

4.136 bt_avdtp_sep_capabilities_t Struct Reference

SEP capabilities.

```
#include <avdtp.h>
```

Data Fields

- bt_byte categories

  Defines service capabilities exposed by a SEP to a remote party.
- bt_byte media_type

  Type of media supported by this SEP.
- bt_byte codec_type

  Codec type supported by this SEP.
- void ∗ codec_config

  Pointer to a buffer that holds codec specific capabilities.
- bt_uint cp_type

  Type of content protection supported by this SEP.
- bt_byte ∗ cp_specific

  Pointer to a buffer holding content protection specific data.
- bt_byte cp_specific_len

  Length of the content protection specific data.
- bt_byte recovery_type

  Type of recovery supported by this SEP.
- bt_byte max_recovery_window

  Recovery window size.
- bt_byte max_parity_code_packets

  Maximum number of parity codec packets.
- bt_byte header_compression_options

  Header compression configuration.
- bt_byte multiplexing_options

  Multiplexing configuration.
- bt_byte tsid_media

  ID of the media transport session.
- bt_byte tcid_media

  ID of the media transport channel.
- bt_byte tsid_reporting

  ID of the reporting transport session.
- bt_byte tcid_reporting

  ID of the reporting transport channel.
- bt_byte tsid_recovery

  ID of the recovery transport session.
- bt_byte tcid_recovery

  ID of the recovery transport channel.

4.136.1 Detailed Description

SEP capabilities.

This structure is used to hold SEP capabilities.

4.136.2 Field Documentation

4.136.2.1 bt_byte categories

Defines service capabilities exposed by a SEP to a remote party.

This can be a combination of the following values:

- AVDTP_SEP_CAPABILITY_FLAG_MEDIA_TRANSPORT
- AVDTP_SEP_CAPABILITY_FLAG_REPORTING
- AVDTP_SEP_CAPABILITY_FLAG_RECOVERY
- AVDTP_SEP_CAPABILITY_FLAG_CONTENT_PROTECTION
- AVDTP_SEP_CAPABILITY_FLAG_HEADER_COMPRESSION
- AVDTP_SEP_CAPABILITY_FLAG_MULTIPLEXING
- AVDTP_SEP_CAPABILITY_FLAG_MEDIA_CODEC

4.136.2.2 bt_byte codec_type

Codec type supported by this SEP.

This can be on of the following values:

- AVDTP_CODEC_TYPE_SBC
- AVDTP_CODEC_TYPE_MPEG1_2_AUDIO
- AVDTP_CODEC_TYPE_MPEG2_4_AAC
- AVDTP_CODEC_TYPE_ATRAC
- AVDTP_CODEC_TYPE_NON_A2DP

4.136.2.3 bt_byte* cp_specific

Pointer to a buffer holding content protection specific data.

Note

Content protection is currently not supported by dotstack.

4.136.2.4 bt_byte cp_specific_len

Length of the content protection specific data.

Note

Content protection is currently not supported by dotstack.

4.136.2.5 bt_uint cp_type

Type of content protection supported by this SEP.

Note

Content protection is currently not supported by dotstack.

4.136.2.6 bt_byte header_compression_options

Header compression configuration.

Note

Header compression is currently not supported by dotstack.

4.136.2.7 bt_byte max_parity_code_packets

Maximum number of parity codec packets.

Note

Recovery is currently not supported by dotstack.

4.136.2.8 bt_byte max_recovery_window

Recovery window size.

Note

Recovery is currently not supported by dotstack.

4.136.2.9 bt_byte media_type

Type of media supported by this SEP.

This can be on of the following values:

- AVDTP_MEDIA_TYPE_AUDIO
- AVDTP_MEDIA_TYPE_VIDEO
- AVDTP_MEDIA_TYPE_MULTIMEDIA

4.136.2.10 bt_byte multiplexing_options

Multiplexing configuration.

Note

Multiplexing is currently not supported by dotstack.

4.136.2.11 bt_byte recovery_type

Type of recovery supported by this SEP.

Note

Recovery is currently not supported by dotstack.

4.137 bt_avdtp_sep_t Struct Reference

SEP description.

`#include <avdtp.h>`

Data Fields

- bt_avdtp_sep_t ∗ next_sep

    Pointer to next SEP.
- bt_byte id

    ID of the SEP.
- bt_byte type

    Type of the SEP.
- const bt_avdtp_sep_capabilities_t ∗ caps

    SEP capabilities.
- bt_byte state

    State of the SEP buffer.

4.137.1 Detailed Description

SEP description.

This structure is used to hold information about SEPs available on a local device.

4.137.2 Field Documentation

4.137.2.1 bt_byte state

State of the SEP buffer.

This can be one of the following values:

- AVDTP_SEP_STATE_FREE
- AVDTP_SEP_STATE_IDLE

4.137.2.2 bt_byte type

Type of the SEP.

This can be one of the following values:

- AVDTP_SEP_TYPE_SOURCE
- AVDTP_SEP_TYPE_SINK

4.138 bt_avdtp_stream_t Struct Reference

Stream description.

`#include <avdtp.h>`

Data Fields

- struct _bt_avdtp_stream_t ∗ next_stream

  Pointer to next stream.
- bt_byte handle

  Stream handle. This values is allocated by dotstack and is used to manipulate the stream by the AVDTP consumer.
- bt_byte state

  State of the stream.
- bt_byte flags

  Additional stream state.
- struct _bt_avdtp_sep_t ∗ sep

  Local SEP this stream is connected to.
- bt_avdtp_sep_capabilities_t ∗ config

  Current SEP configuration.
- bt_byte remote_seid

  ID of the remote SEP.
- bt_bdaddr_t remote_addr

  BT address of the remote device.
- bt_avdtp_transport_session_t sessions [AVDTP_MAX_STREAM_TRANSPORT_SESSION]

  List of transport session available/active on this stream.
- struct _bt_avdtp_mgr_t ∗ mgr

  AVDTP manager this stream belongs to.
- bt_byte cur_channel_index

  This value is currenlty not used.
- bt_byte ∗ listen_sep_list

  A list of SEPs this channel this channel is listening on (i.e. can accept incoming connection requests).
- bt_byte listen_sep_count

  The number of SEPs in `listen_sep_list`.
- bt_queue_element_t ∗ media_rx_queue

  A list of media packet buffer for receiving incoming packets.
- bt_queue_element_t ∗ media_tx_queue

  A list of media packet buffer to be sent to a remote device.

4.138.1 Detailed Description

Stream description.

This structure is used to hold information about streams available on a local device.

4.138.2 Field Documentation

4.138.2.1 bt_byte flags

Additional stream state.

This value can be one of the following values:

- AVDTP_STREAM_FLAG_LISTENING

4.138.2.2 bt_avdtp_transport_session_t sessions[AVDTP_MAX_STREAM_TRANSPORT_SESSION]

List of transport session available/active on this stream.

Note

There can be up to 3 (AVDTP_MAX_STREAM_TRANSPORT_SESSION == 3) transport sessions on a stream - media, reporting and recovery. dotstack supports only media sessions so the other two are never used.

4.138.2.3 bt_byte state

State of the stream.

This value can be one of the following values:

- AVDTP_STREAM_STATE_IDLE
- AVDTP_STREAM_OPENING_TRANSPORT_CHANNELS
- AVDTP_STREAM_CLOSING_TRANSPORT_CHANNELS
- AVDTP_STREAM_STATE_CONFIGURED
- AVDTP_STREAM_STATE_OPEN
- AVDTP_STREAM_STATE_STREAMING
- AVDTP_STREAM_STATE_CLOSING
- AVDTP_STREAM_STATE_ABORTING

4.139 bt_avdtp_transport_channel_t Struct Reference

Transport channel description.

`#include <avdtp.h>`

Data Fields

- struct _bt_avdtp_transport_channel_t ∗ next_channel

  Pointer to next channel.
- bt_byte id

  ID of the channel.
- bt_byte type

  Type of the channel.
- bt_l2cap_channel_t ∗ l2cap_channel

  L2CAp channel used to transfer this AVDTP channel's data.
- bt_byte ref_count

  Channel's reference count.
- bt_byte connect_ref_count

  This value is currently not used.

4.139.1 Detailed Description

Transport channel description.

This structure is used to hold information about transport channels available on a local device.

4.139.2 Field Documentation

4.139.2.1 bt_byte ref_count

Channel's reference count.

This value is intended for use with shared channels. When ref count reaches 0 the channel is closed.

Note

Shared channles (i.e. multiplexing) is currently not supported by dotstack

4.139.2.2 bt_byte type

Type of the channel.

This can be one of the following values:

- AVDTP_TRANSPORT_CHANNEL_TYPE_DEDICATED
- AVDTP_TRANSPORT_CHANNEL_TYPE_SHARED

  Note

  Shared channles (i.e. multiplexing) is currently not supported by dotstack

4.140 bt_avdtp_transport_session_t Struct Reference

Transport session description.

`#include <avdtp.h>`

Data Fields

- bt_byte id

  ID of the transport session.
- bt_byte type

  Type of the transport session.
- bt_avdtp_transport_channel_t ∗ transport_channel

  Transport channel used for this transport session.

4.140.1 Detailed Description

Transport session description.

This structure is used to hold information about transport sessions available on a local device.

4.140.2 Field Documentation

4.140.2.1 bt_byte type

Type of the transport session.

This can be one of the following values:

- AVDTP_TRANSPORT_SESSION_TYPE_MEDIA

- AVDTP_TRANSPORT_SESSION_TYPE_REPORTING
- AVDTP_TRANSPORT_SESSION_TYPE_RECOVERY

4.141 bt_avrcp_channel_t Struct Reference

AVRCP channel description.

`#include <avrcp.h>`

4.141.1 Detailed Description

AVRCP channel description.

This structure is used to hold information about an AVRCP channel.

4.142 bt_avrcp_device_t Struct Reference

4.143 bt_avrcp_event_t Union Reference

Parameter to an application callback.

`#include <avrcp.h>`

Data Fields

- bt_avrcp_evt_channel_connected_t channel_connected

  Valid if event is AVRCP_EVT_CONTROL_CHANNEL_CONNECTED.
- bt_avrcp_evt_channel_disconnected_t channel_disconnected

  Valid if event is AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED.
- bt_avrcp_evt_connection_failed_t connection_failed

  Valid if event is AVRCP_EVT_CONTROL_CONNECTION_FAILED.
- bt_avrcp_evt_panel_response_received_t panel_response_received

  Valid if event is AVRCP_EVT_PANEL_RESPONSE_RECEIVED.
- bt_avrcp_evt_search_completed_t device_search

  Valid if event is AVRCP_EVT_SEARCH_COMPLETED.
- bt_av_capability_company_id_t company_id

  Valid if event is AVRCP_EVT_COMPANY_ID_LIST_RECEIVED.
- bt_av_capability_event_id_t supported_event_id

  Valid if event is AVRCP_EVT_EVENT_ID_LIST_RECEIVED.
- bt_av_player_settings_t player_settings

  Valid if event is AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED.
- bt_av_player_setting_values_t player_setting_values

  Valid if event is AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED.
- bt_av_player_setting_current_values_t player_setting_current_values

  Valid if event is AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED.
- bt_av_player_settings_text_t player_settings_text

  Valid if event is AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED.
- bt_av_player_setting_values_text_t player_setting_values_text

  Valid if event is AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED.

- bt_av_element_attributes_t element_attributes

  Valid if event is AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED.
- bt_av_play_status_t play_status

  Valid if event is AVRCP_EVT_GET_PLAY_STATUS_RECEIVED.
- bt_av_set_absolute_volume_t absolute_volume

  Valid if event is AVRCP_EVT_SET_ABSOLUTE_VOLUME_COMPLETED.
- bt_av_set_addressed_player_t addressed_player

  Valid if event is AVRCP_EVT_SET_ADDRESSED_PLAYER_COMPLETED.
- bt_av_play_item_t play_item_status

  Valid if event is AVRCP_EVT_PLAY_ITEM_COMPLETED.
- bt_av_add_to_now_playing_t add_to_now_playing_status

  Valid if event is AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED.
- bt_av_notification_t notification

  Valid if event is on of the following:
- bt_avrcp_evt_register_events_completed_t register_events

  Valid if event AVRCP_EVT_REGISTER_NOTIFICATIONS_COMPLETED.
- bt_avrcp_evt_panel_command_received_t panel_command_received

  Valid if event is AVRCP_EVT_PANEL_COMMAND_RECEIVED.
- bt_av_battery_status_of_ct_t battery_status_of_ct

  Valid if event is AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED.
- bt_av_displayable_character_set_t displayable_character_set

  Valid if event is AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED.
- bt_av_get_element_attributes_t get_element_attributes

  Valid if event is AVRCP_EVT_ELEMENT_ATTRIBUTES_REQUESTED.
- bt_av_register_notification_t register_notification

  Valid if event is AVRCP_EVT_REGISTER_NOTIFICATION_REQUESTED.

4.143.1 Detailed Description

Parameter to an application callback.

This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

4.143.2 Field Documentation

4.143.2.1 bt_av_notification_t notification

Valid if event is on of the following:

- AVRCP_EVT_PLAYBACK_STATUS_CHANGED
- AVRCP_EVT_TRACK_CHANGED
- AVRCP_EVT_TRACK_REACHED_END
- AVRCP_EVT_TRACK_REACHED_START
- AVRCP_EVT_PLAYBACK_POS_CHANGED
- AVRCP_EVT_BATT_STATUS_CHANGED
- AVRCP_EVT_SYSTEM_STATUS_CHANGED

- AVRCP_EVT_PLAYER_APPLICATION_SETTING_CHANGED
- AVRCP_EVT_NOW_PLAYING_CONTENT_CHANGED
- AVRCP_EVT_AVAILABLE_PLAYERS_CHANGED
- AVRCP_EVT_ADDRESSED_PLAYER_CHANGED
- AVRCP_EVT_UIDS_CHANGED
- AVRCP_EVT_VOLUME_CHANGED

4.144 bt_avrcp_evt_channel_connected_t Struct Reference

Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event.

`#include <avrcp.h>`

Data Fields

- bt_avrcp_channel_t ∗ channel

  AVRCP channel.

4.144.1 Detailed Description

Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_connected - when a control channel between two devices has been established.

4.145 bt_avrcp_evt_channel_disconnected_t Struct Reference

Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event.

`#include <avrcp.h>`

Data Fields

- bt_avrcp_channel_t ∗ channel

  AVRCP channel.

4.145.1 Detailed Description

Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::channel_disconnected - when a control channel between two devices has been terminated.

4.146 bt_avrcp_evt_connection_failed_t Struct Reference

Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event.

`#include <avrcp.h>`

Data Fields

- bt_avrcp_channel_t ∗ channel

  AVRCP channel.

4.146.1 Detailed Description

Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::connection_failed - when a local device failed to create a control channel between two AVRCP entities.

4.147 bt_avrcp_evt_panel_command_received_t Struct Reference

Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event.

```
#include <avrcp.h>
```

Data Fields

- bt_avrcp_channel_t ∗ channel

  AVRCP channel.
- bt_byte ctype

  Command type.
- bt_byte button_status

  Button status.
- bt_byte opcode

  Operation Id.
- bt_byte ∗ params

  Operation parameters.
- bt_byte params_len

  Length of the operation parameters.

4.147.1 Detailed Description

Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_command_received - when a local device received a PASS THROUGH command.

4.148 bt_avrcp_evt_panel_response_received_t Struct Reference

Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event.

```
#include <avrcp.h>
```

Data Fields

- bt_avrcp_channel_t ∗ channel

  AVRCP channel.
- bt_byte ctype

  Response type.
- bt_byte button_status

  Button status.
- bt_byte opcode

  Operation Id.
- bt_byte ∗ params

  Operation parameters.
- bt_byte params_len

  Length of the operation parameters.

4.148.1 Detailed Description

Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::panel_response_received - when a local device received a response to a PASS THRO↩ UGH command.

4.149 bt_avrcp_evt_register_events_completed_t Struct Reference

Data Fields

- bt_avrcp_channel_t ∗ channel

  AVRCP channel.

4.150 bt_avrcp_evt_search_completed_t Struct Reference

Parameter to AVRCP_EVT_SEARCH_COMPLETED event.

```
#include <avrcp.h>
```

Data Fields

- bt_avrcp_device_t ∗ devices

  list of found devices
- bt_byte count

  the number of found devices

4.150.1 Detailed Description

Parameter to AVRCP_EVT_SEARCH_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt_avrcp_event_t union - bt_avrcp_event_t::device_search - when searching for nearby devices has finished.

4.151 bt_avrcp_mgr_t Struct Reference

AVRCP manager.

`#include <avrcp.h>`

Data Fields

- bt_byte state

  Manager state.
- bt_byte flags

  Additional manager state.
- bt_ulong company_id

  The 24-bit unique ID obtained from the IEEE Registration Authority Committee.
- bt_uint supported_events

  Events supported by the target.
- bt_ulong song_length

  Current song length.
- bt_ulong song_position

  Current song poistion.
- bt_byte play_status

  Current playback status.
- bt_byte volume

  Current volumne.
- bt_byte battery_status

  Current battery statusThis can be one of the following values:
- bt_byte system_status

  Current system status.
- bt_av_element_id_t current_track_id

  Current track id.
- bt_uint uid_counter

  The number of media items in the target.
- bt_avrcp_channel_t ∗ channels

  List of available AVRCP channels.
- bt_avrcp_mgr_callback_fp callback

  Pointer to a function used to notify the AVRCP consumer about various events.
- void ∗ callback_param

  Pointer to arbitrary data to be passed to the `callback`.

4.151.1 Detailed Description

AVRCP manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with `bt_avrcp_get_mgr()`.

4.151.2 Field Documentation

4.151.2.1 bt_byte battery_status

Current battery statusThis can be one of the following values:

- AVC_BATTERY_STATUS_NORMAL
- AVC_BATTERY_STATUS_WARNING
- AVC_BATTERY_STATUS_CRITICAL
- AVC_BATTERY_STATUS_EXTERNAL
- AVC_BATTERY_STATUS_FULL_CHARGE

4.151.2.2 bt_ulong company_id

The 24-bit unique ID obtained from the IEEE Registration Authority Committee.

If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used.

4.151.2.3 bt_byte flags

Additional manager state.

This value can be a combination of the following values:

- AVRCP_MANAGER_FLAG_SEARCHING

4.151.2.4 bt_byte state

Manager state.

This value can be one of the following values:

- AVRCP_MANAGER_STATE_IDLE
- AVRCP_MANAGER_STATE_CONNECTING
- AVRCP_MANAGER_STATE_DISCONNECTING

4.151.2.5 bt_uint supported_events

Events supported by the target.

This value can be a combination of the following values:

- AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED
- AVC_EVENT_FLAG_TRACK_CHANGED
- AVC_EVENT_FLAG_TRACK_REACHED_END
- AVC_EVENT_FLAG_TRACK_REACHED_START
- AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED
- AVC_EVENT_FLAG_BATT_STATUS_CHANGED

- AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED
- AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED
- AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED
- AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED
- AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED
- AVC_EVENT_FLAG_UIDS_CHANGED
- AVC_EVENT_FLAG_VOLUME_CHANGED

4.152 bt_cp_header_t Struct Reference

4.153 bt_gatt_client_char_declaration_t Struct Reference

Characteristic Declaration.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte props
  Characteristic properties.
- bt_uint start_handle
  First characteristic attribute handle.
- bt_uint end_handle
  Last characteristic attribute handle.
- bt_uint value_handle
  Characteristic value attribute handle.
- bt_att_client_uuid_t uuid
  Characteristic UUID.

4.153.1 Detailed Description

Characteristic Declaration.

This structure is used to hold an characteristic declaration.

4.154 bt_gatt_client_char_descriptor_t Struct Reference

Characteristic Descriptor Declaration.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint handle
  Characteristic descriptor attribute handle.
- bt_att_client_uuid_t uuid
  Characteristic descriptor UUID.

4.154.1 Detailed Description

Characteristic Descriptor Declaration.

This structure is used to hold an characteristic descriptor declaration.

4.155 bt_gatt_client_char_value_t Struct Reference

Characteristic Value.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint handle

  Characteristic value attribute handle.
- bt_byte ∗ value

  Characteristic value.
- bt_uint len

  Characteristic value length.

4.155.1 Detailed Description

Characteristic Value.

This structure is used to hold an characteristic value.

4.156 bt_gatt_client_evt_conn_param_update_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_int hci_status

  HCI command status.

4.156.1 Detailed Description

Parameter to GATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback after the new connection parameters have been set.

4.157 bt_gatt_client_evt_conn_param_update_t Struct Reference

Parameter to GATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST event.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint min_interval

    Minimum connection interval expressed in 1.25ms units.
- bt_uint max_interval

    Maximum connection interval expressed in 1.25ms units.
- bt_uint slave_latency

    Slave latency expressed as number of connection events.
- bt_uint supervision_timeout

    Link supervision timeout expressed in 10ms units.

4.157.1 Detailed Description

Parameter to GATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST event.

A pointer to this structure is passed to the GATT client application callback when client received update connection parameters request.

4.158 bt_gatt_client_evt_discover_all_chars_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_DISCOVER_ALL_CHARS_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint count

    The number of characteristic declarations found.
- bt_gatt_client_char_declaration_t ∗ chars

    An array of characteristic declarations.

4.158.1 Detailed Description

Parameter to GATT_CLIENT_EVT_DISCOVER_ALL_CHARS_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover all characteristics of a service" operation has completed.

4.158.2 Field Documentation

4.158.2.1 bt_gatt_client_char_declaration_t∗ chars

An array of characteristic declarations.

This is the same array that is passed to bt_gatt_client_discover_all_chars_ex().

4.159 bt_gatt_client_evt_discover_all_services_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_DISCOVER_ALL_SERVICES_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint count

  The number of found services.
- bt_gatt_client_service_definition_t ∗ services

  An array of service definitions.

4.159.1 Detailed Description

Parameter to GATT_CLIENT_EVT_DISCOVER_ALL_SERVICES_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover all services" operation has completed.

4.159.2 Field Documentation

4.159.2.1 bt_gatt_client_service_definition_t∗ services

An array of service definitions.

This is the same array that is passed to bt_gatt_client_discover_all_services_ex().

4.160 bt_gatt_client_evt_discover_char_by_uuid_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_DISCOVER_CHAR_BY_UUID_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint count

  The number of characteristic declarations found.
- bt_gatt_client_char_declaration_t ∗ chars

  An array of characteristic declarations.

4.160.1 Detailed Description

Parameter to GATT_CLIENT_EVT_DISCOVER_CHAR_BY_UUID_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover characteristics by UUID" operation has completed.

4.160.2 Field Documentation

4.160.2.1 bt_gatt_client_char_declaration_t∗ chars

An array of characteristic declarations.

This is the same array that is passed to bt_gatt_client_discover_char_by_uuid() or bt_gatt_client_discover_char_↩ by_uuid_80().

4.161 bt_gatt_client_evt_discover_descriptors_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_DISCOVER_CHAR_DESCRIPTORS_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint count

  The number of characteristic descriptors found.
- bt_gatt_client_char_descriptor_t ∗ descriptors

  An array of characteristic descriptors.

4.161.1 Detailed Description

Parameter to GATT_CLIENT_EVT_DISCOVER_CHAR_DESCRIPTORS_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover all characteristic descriptors" operation has completed.

4.161.2 Field Documentation

4.161.2.1 bt_gatt_client_char_descriptor_t∗ descriptors

An array of characteristic descriptors.

This is the same array that is passed to bt_gatt_client_discover_char_descriptors().

4.162 bt_gatt_client_evt_discover_service_by_uuid_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_DISCOVER_SERVICE_BY_UUID_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint count

  The number of found services.
- bt_gatt_client_service_definition_t ∗ services

  An array of service definitions.

4.162.1 Detailed Description

Parameter to GATT_CLIENT_EVT_DISCOVER_SERVICE_BY_UUID_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover services by UUID" operation has completed.

4.162.2 Field Documentation

4.162.2.1 bt_gatt_client_service_definition_t∗ services

An array of service definitions.

This is the same array that is passed to bt_gatt_client_discover_by_service_uuid().

4.163 bt_gatt_client_evt_exchange_mtu_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_EXCHANGE_MTU_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

    Operation status.
- bt_uint mtu

    Server's MTU.

4.163.1 Detailed Description

Parameter to GATT_CLIENT_EVT_EXCHANGE_MTU_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when the client received a response (either positive or negative) to a "exchange MTU" request.

4.164 bt_gatt_client_evt_find_included_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_FIND_INCLUDED_SERVICES_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint count

    The number of included service declarations found.
- bt_gatt_client_inc_service_declaration_t ∗ services

    An array of included service declarations.

4.164.1 Detailed Description

Parameter to GATT_CLIENT_EVT_FIND_INCLUDED_SERVICES_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "find included services" operation has completed.

4.164.2 Field Documentation

4.164.2.1 bt_gatt_client_inc_service_declaration_t∗ services

An array of included service declarations.

This is the same array that is passed to bt_gatt_client_find_included_services_ex().

4.165 bt_gatt_client_evt_profile_found_t Struct Reference

4.166 bt_gatt_client_evt_read_by_char_uuid_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_READ_USING_CHAR_UUID_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_att_client_uuid_t ∗ uuid

  Characteristic UUID.
- bt_uint count

  The number of characteristic values read.
- bt_gatt_client_char_value_t ∗ values

  An array of characteristic values.

4.166.1 Detailed Description

Parameter to GATT_CLIENT_EVT_READ_USING_CHAR_UUID_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read using characteristic UUID" operation has completed.

4.166.2 Field Documentation

4.166.2.1 bt_gatt_client_char_value_t∗ values

An array of characteristic values.

This is the same array that is passed to bt_gatt_client_read_by_char_uuid() or bt_gatt_client_read_by_char_uuid←_80().

4.167 bt_gatt_client_evt_read_char_descriptor_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_READ_CHAR_DESCRIPTOR_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- const bt_byte ∗ value

  Characteristic descriptor. This is the same buffer that is passed to bt_gatt_client_read_char_descriptor().
- bt_uint len

  The length of the characteristic descriptor.

4.167.1 Detailed Description

Parameter to GATT_CLIENT_EVT_READ_CHAR_DESCRIPTOR_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read characteristic descriptor" operation has completed.

4.168 bt_gatt_client_evt_read_char_long_descriptor_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_READ_CHAR_LONG_DESCRIPTOR_COMPLETED event.

`#include <gatt_client.h>`

Data Fields

- bt_byte status

  Operation status.
- bt_uint offset

  The offset of the first octet read.
- const bt_byte ∗ value

  Characteristic descriptor. This is the same buffer that is passed to bt_gatt_client_read_char_long_descriptor().
- bt_uint len

  The length of the characteristic descriptor.

4.168.1 Detailed Description

Parameter to GATT_CLIENT_EVT_READ_CHAR_LONG_DESCRIPTOR_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read long characteristic descriptor" operation has completed.

4.169 bt_gatt_client_evt_read_char_long_value_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_READ_CHAR_LONG_VALUE_COMPLETED event.

`#include <gatt_client.h>`

Data Fields

- bt_byte status

  Operation status.
- bt_uint offset

  The offset of the first octet read.
- const bt_byte ∗ value

  Characteristic value. This is the same buffer that is passed to bt_gatt_client_read_char_long_value().
- bt_uint len

  The length of the characteristic value.
- bt_att_client_uuid_t ∗ uuid

  Characteristic UUID;.

4.169.1 Detailed Description

Parameter to GATT_CLIENT_EVT_READ_CHAR_LONG_VALUE_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read long characteristic value" operation has completed.

4.170 bt_gatt_client_evt_read_char_value_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_READ_CHAR_VALUE_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- const bt_byte ∗ value

  Characteristic value. This is the same buffer that is passed to bt_gatt_client_read_char_value().
- bt_uint len

  The length of the characteristic value.
- bt_att_client_uuid_t ∗ uuid

  Characteristic UUID;.

4.170.1 Detailed Description

Parameter to GATT_CLIENT_EVT_READ_CHAR_VALUE_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read characteristic value" operation has completed.

4.171 bt_gatt_client_evt_read_multiple_char_values_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_READ_MULTIPLE_CHAR_VALUES_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- const bt_byte ∗ value

  Characteristic values. This is the same buffer that is passed to bt_gatt_client_read_multiple_char_values().
- bt_uint len

  The length of the characteristic values.

4.171.1 Detailed Description

Parameter to GATT_CLIENT_EVT_READ_MULTIPLE_CHAR_VALUES_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read multiple characteristic values" operation has completed.

4.172 bt_gatt_client_evt_t Union Reference

Parameter to GATT client application callback.

```
#include <gatt_client.h>
```

Data Fields

- bt_gatt_client_evt_exchange_mtu_completed_t exchange_mtu

  Valid if event is GATT_CLIENT_EVT_EXCHANGE_MTU_COMPLETED.
- bt_gatt_client_evt_discover_all_services_completed_t discover_all_services

  Valid if event is GATT_CLIENT_EVT_DISCOVER_ALL_SERVICES_COMPLETED.
- bt_gatt_client_evt_discover_service_by_uuid_completed_t discover_service_by_uuid

  Valid if event is GATT_CLIENT_EVT_DISCOVER_SERVICE_BY_UUID_COMPLETED.
- bt_gatt_client_evt_discover_all_chars_completed_t discover_all_chars

  Valid if event is GATT_CLIENT_EVT_DISCOVER_ALL_CHARS_COMPLETED.
- bt_gatt_client_evt_discover_char_by_uuid_completed_t discover_char_by_uuid

  Valid if event is GATT_CLIENT_EVT_DISCOVER_CHAR_BY_UUID_COMPLETED.
- bt_gatt_client_evt_read_char_value_completed_t read_char_value

  Valid if event is GATT_CLIENT_EVT_READ_CHAR_VALUE_COMPLETED.
- bt_gatt_client_evt_read_char_long_value_completed_t read_char_long_value

  Valid if event is GATT_CLIENT_EVT_READ_CHAR_LONG_VALUE_COMPLETED.
- bt_gatt_client_evt_write_char_value_completed_t write_char_value

  Valid if event is GATT_CLIENT_EVT_WRITE_CHAR_VALUE_COMPLETED.
- bt_gatt_client_evt_write_char_long_value_completed_t write_char_long_value

  Valid if event is GATT_CLIENT_EVT_WRITE_CHAR_LONG_VALUE_COMPLETED.
- bt_gatt_client_evt_value_notification_t value_notification

  Valid if event is GATT_CLIENT_EVT_VALUE_NOTIFICATION.
- bt_gatt_client_evt_discover_descriptors_completed_t discover_descriptors

  Valid if event is GATT_CLIENT_EVT_DISCOVER_CHAR_DESCRIPTORS_COMPLETED.
- bt_gatt_client_evt_read_char_descriptor_completed_t read_descriptor

  Valid if event is GATT_CLIENT_EVT_READ_CHAR_DESCRIPTOR_COMPLETED.
- bt_gatt_client_evt_read_char_long_descriptor_completed_t read_long_descriptor

  Valid if event is GATT_CLIENT_EVT_READ_CHAR_LONG_DESCRIPTOR_COMPLETED.
- bt_gatt_client_evt_write_char_descriptor_completed_t write_descriptor

  Valid if event is GATT_CLIENT_EVT_WRITE_CHAR_DESCRIPTOR_COMPLETED.
- bt_gatt_client_evt_write_char_long_descriptor_completed_t write_long_descriptor

  Valid if event is GATT_CLIENT_EVT_WRITE_CHAR_LONG_DESCRIPTOR_COMPLETED.
- bt_gatt_client_evt_find_included_completed_t find_included

  Valid if event is GATT_CLIENT_EVT_FIND_INCLUDED_SERVICES_COMPLETED.
- bt_gatt_client_evt_read_by_char_uuid_completed_t read_by_char_uuid

  Valid if event is GATT_CLIENT_EVT_READ_USING_CHAR_UUID_COMPLETED.
- bt_gatt_client_evt_read_multiple_char_values_completed_t read_multiple_char_values

  Valid if event is GATT_CLIENT_EVT_READ_MULTIPLE_CHAR_VALUES_COMPLETED.
- bt_gatt_client_evt_conn_param_update_t conn_param_update

  Valid if event is GATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST.
- bt_gatt_client_evt_conn_param_update_completed_t conn_param_update_completed

  Valid if event is GATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED.
- bt_gatt_client_evt_profile_found_t profile_found

  Valid if event is GATT_CLIENT_EVT_PROFILE_FOUND.

4.172.1 Detailed Description

Parameter to GATT client application callback.

This union is used to pass event specific data to the GATT client consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

4.173 bt_gatt_client_evt_value_notification_t Struct Reference

Parameter to GATT_CLIENT_EVT_VALUE_NOTIFICATION event.

```
#include <gatt_client.h>
```

Data Fields

- bt_uint handle

  Characteristic value attribute handle.
- const bt_byte ∗ value

  Characteristic value.
- bt_uint len

  Characteristic value length.
- bt_bool indication

  If non-0 a value indication has been received.

4.173.1 Detailed Description

Parameter to GATT_CLIENT_EVT_VALUE_NOTIFICATION event.

A pointer to this structure is passed to the GATT client application callback when the client received a value notification or indication.

4.174 bt_gatt_client_evt_write_char_descriptor_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_WRITE_CHAR_DESCRIPTOR_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

  Operation status.
- bt_uint handle

  Characteristic descriptor attribute handle.

4.174.1 Detailed Description

Parameter to GATT_CLIENT_EVT_WRITE_CHAR_DESCRIPTOR_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "write characteristic descriptor" operation has completed.

4.175 bt_gatt_client_evt_write_char_long_descriptor_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_WRITE_CHAR_LONG_DESCRIPTOR_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

    Operation status.

4.175.1 Detailed Description

Parameter to GATT_CLIENT_EVT_WRITE_CHAR_LONG_DESCRIPTOR_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "write long characteristic descriptor" operation has completed.

4.176 bt_gatt_client_evt_write_char_long_value_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_WRITE_CHAR_LONG_VALUE_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

    Operation status.

4.176.1 Detailed Description

Parameter to GATT_CLIENT_EVT_WRITE_CHAR_LONG_VALUE_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "write long characteristic value" operation has completed.

4.177 bt_gatt_client_evt_write_char_value_completed_t Struct Reference

Parameter to GATT_CLIENT_EVT_WRITE_CHAR_VALUE_COMPLETED event.

```
#include <gatt_client.h>
```

Data Fields

- bt_byte status

    Operation status.
- bt_uint handle

    Characteristic value attribute handle.

4.177.1 Detailed Description

Parameter to GATT_CLIENT_EVT_WRITE_CHAR_VALUE_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "write characteristic value" or "write without response" operation has completed.

4.178 bt_gatt_client_inc_service_declaration_t Struct Reference

Included Service Declaration.

`#include <gatt_client.h>`

Data Fields

- bt_uint uuid

  Service UUID.
- bt_uint decl_handle

  Included service declaration attribute handle.
- bt_uint start_handle

  First service attribute handle.
- bt_uint end_handle

  Last service attribute handle.

4.178.1 Detailed Description

Included Service Declaration.

This structure is used to hold an include service declaration.

4.179 bt_gatt_client_listener_t Struct Reference

4.180 bt_gatt_client_mgr_t Struct Reference

4.181 bt_gatt_client_profile_finder_state_t Struct Reference

4.182 bt_gatt_client_profile_finder_t Struct Reference

4.183 bt_gatt_client_profile_header_t Struct Reference

4.184 bt_gatt_client_service_definition_t Struct Reference

Service Definition.

`#include <gatt_client.h>`

Data Fields

- bt_att_client_uuid_t uuid

  Service UUID.
- bt_uint start_handle

  First attribute handle.
- bt_uint end_handle

  Last attribute handle.

4.184.1 Detailed Description

Service Definition.

This structure is used to hold a service definition.

4.185 bt_gatt_client_session_t Struct Reference

4.186 bt_gatt_evt_client_config_changed_t Struct Reference

Parameter to GATT_SERVER_EVT_CLIENT_CONFIG_CHANGED event.

```
#include <gatt_server.h>
```

Data Fields

- bt_gatt_evt_header_t header

  Common header.
- bt_byte client_config

  New client configuration.

4.186.1 Detailed Description

Parameter to GATT_SERVER_EVT_CLIENT_CONFIG_CHANGED event.

A pointer to this structure is passed to the GATT application callback when the characteristic's client configuration changed.

4.187 bt_gatt_evt_ext_properties_changed_t Struct Reference

Parameter to GATT_SERVER_EVT_EXTENDED_PROPERTIES_CHANGED event.

```
#include <gatt_server.h>
```

Data Fields

- bt_gatt_evt_header_t header

  Common header.
- bt_byte ext_properties

  New extended properties.

4.187.1 Detailed Description

Parameter to GATT_SERVER_EVT_EXTENDED_PROPERTIES_CHANGED event.

A pointer to this structure is passed to the GATT application callback when the characteristic's extended properties changed.

4.188 bt_gatt_evt_header_t Struct Reference

Common to all event parameters header.

```
#include <gatt_server.h>
```

Data Fields

- bt_uuid_t service_type

  128-bit service type UUID.
- bt_uuid_t service_id

  128-bit service id UUID.
- bt_uuid_t characteristic_id

  128-bit characteristic id UUID.

4.188.1 Detailed Description

Common to all event parameters header.

This structure holds characteristic's identifier - service type, service id, and characteristic id.

4.189 bt_gatt_evt_server_config_changed_t Struct Reference

Parameter to GATT_SERVER_EVT_SERVER_CONFIG_CHANGED event.

```
#include <gatt_server.h>
```

Data Fields

- bt_gatt_evt_header_t header

  Common header.
- bt_byte server_config

  New server configuration.

4.189.1 Detailed Description

Parameter to GATT_SERVER_EVT_SERVER_CONFIG_CHANGED event.

A pointer to this structure is passed to the GATT application callback when the characteristic's server configuration changed.

4.190 bt_gatt_evt_value_changed_t Struct Reference

Parameter to GATT_SERVER_EVT_VALUE_CHANGED event.

```
#include <gatt_server.h>
```

Data Fields

- bt_gatt_evt_header_t header

  Common header.
- const bt_byte ∗ value

  New value.
- bt_uint len

  New value length.

4.190.1 Detailed Description

Parameter to GATT_SERVER_EVT_VALUE_CHANGED event.

A pointer to this structure is passed to the GATT application callback when the characteristic's value changed.

4.191 bt_gatt_evt_value_read_t Struct Reference

Parameter to GATT_SERVER_EVT_VALUE_READ event.

```
#include <gatt_server.h>
```

Data Fields

- bt_gatt_evt_header_t header

  Common header.
- const bt_byte ∗ value

  Value.
- bt_uint len

  Value length.
- bt_uint offset

  Offset from which the value has been read.

4.191.1 Detailed Description

Parameter to GATT_SERVER_EVT_VALUE_READ event.

A pointer to this structure is passed to the GATT application callback when the characteristic's value has been read by a client.

4.192 bt_gatt_listener_t Struct Reference

4.193 bt_hci_command_p Struct Reference

4.194 bt_hci_data_p Struct Reference

4.195 bt_hci_event_p Struct Reference

4.196 bt_hci_inquiry_response_t Struct Reference

4.197 bt_hf_subscriber_number_t Struct Reference

Data Fields

- bt_char number [HFP_MAX_NUMBER_LENGTH+1]

  Phone number.
- bt_byte type

  Number type.
- bt_byte service

  Service type.

4.198 bt_hfp_ag_evt_search_completed_t Struct Reference

4.199 bt_hfp_audio_packet_t Struct Reference

Parameter to HFP_EVENT_AUDIO_DATA_RECEIVED event.

`#include <hfp.h>`

Data Fields

- bt_uint data_len

  The length of the audio data.
- bt_uint max_data_len

  Maximum length of the media data that can be stored in the buffer pointed by the `data` member.
- bt_byte ∗ data

  Pointer to a buffer to store audio data.

4.199.1 Detailed Description

Parameter to HFP_EVENT_AUDIO_DATA_RECEIVED event.

HFP layer calls the session callback when it has received a chunk of audio data from the remote device. The callback is passed a pointer to `bt_hfp_evt_audio_data_t` which contains a pointer to this structure.

4.199.2 Field Documentation

4.199.2.1 bt_byte∗ data

Pointer to a buffer to store audio data.

This pointer must be allocated by the HFP consumer before adding this structure to receive (::bt_hfp_add_audio←_rx_buffer) or send queue (::bt_hfp_add_audio_tx_buffer).

4.200 bt_hfp_call_t Struct Reference

Stores information about a call.

`#include <hfp.h>`

Data Fields

- bt_byte index

  Call index inside AG.
- bt_byte direction

  Call direction. HFP_CALL_DIR_OUTGOING or HFP_CALL_DIR_INCOMING.
- bt_byte status

  Call status.
- bt_byte mode

  Call mode.
- bt_byte multiparty

  Number or parties in the call.
- bt_char number [HFP_MAX_NUMBER_LENGTH+1]

  Phone number.
- bt_int type

  Number type.

4.200.1 Detailed Description

Stores information about a call.

4.200.2 Field Documentation

4.200.2.1 bt_byte mode

Call mode.

- HFP_CALL_MODE_VOICE
- HFP_CALL_MODE_DATA
- HFP_CALL_MODE_FAX

4.200.2.2 bt_byte multiparty

Number or parties in the call.

- HFP_CALL_MPTY_NOT_MULTIPARTY
- HFP_CALL_MPTY_MULTIPARTY

4.200.2.3 bt_byte status

Call status.

- HFP_CALL_STATUS_MASK_IDLE
- HFP_CALL_STATUS_MASK_ACTIVE
- HFP_CALL_STATUS_MASK_HELD
- HFP_CALL_STATUS_MASK_DIALING
- HFP_CALL_STATUS_MASK_ALERTING
- HFP_CALL_STATUS_MASK_INCOMING
- HFP_CALL_STATUS_MASK_WAITING

4.201 bt_hfp_event_register_t Struct Reference

Stores value of HF registrations.

`#include <hfp.h>`

Data Fields

- bt_byte mode

  The registration mode. Only HFP_REG_MODE_FORWARD is supported by the current BT spec.
- bt_byte ind

  The indicator's value. The possible values are defined by HFP_REG_IND_ constants.

4.201.1 Detailed Description

Stores value of HF registrations.

4.202 bt_hfp_evt_activate_voice_recognition_t Struct Reference

4.203 bt_hfp_evt_answer_call_t Struct Reference

4.204 bt_hfp_evt_audio_connection_state_changed_t Struct Reference

HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event parameter.

`#include <hfp.h>`

Data Fields

- bt_bool connected

  Defines the state of audio connection. TRUE - if audio is connected, FALSE - if audio is disconnected.
- bt_hci_conn_state_t ∗ hci_conn

  A pointer to `HCI` SCO connection if `connected` is TRUE, otherwise - NULL.

4.204.1 Detailed Description

HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event parameter.

HFP layer calls the session callback when the state of the audio connection has changed and passes a pointer to `bt_hfp_evt_audio_connection_state_changed_t` which contains a flag identifying the current state of the audio connection (connected or disconnected).

4.205 bt_hfp_evt_audio_data_t Struct Reference

HFP_EVENT_AUDIO_DATA_RECEIVED event parameter.

`#include <hfp.h>`

Data Fields

- bt_hfp_audio_packet_t ∗ packet

  A pointer to an audio packet buffer.

4.205.1 Detailed Description

HFP_EVENT_AUDIO_DATA_RECEIVED event parameter.

HFP layer calls the session callback when it received a chunk of audio data from the remote device and passes a pointer to `bt_hfp_evt_audio_data_t` which contains a pointer to the data and its length.

4.206 bt_hfp_evt_audio_packet_sent_t Struct Reference

4.207 bt_hfp_evt_call_waiting_t Struct Reference

HFP_EVENT_CALL_WAITING event parameter.

`#include <hfp.h>`

Data Fields

- bt_char ∗ number

  Pointer to 0-terminated string that contains the number of the incoming call.
- bt_ulong number_type

  The type of the number.

4.207.1 Detailed Description

HFP_EVENT_CALL_WAITING event parameter.

HFP layer calls the session callback when it received the number of a waiting call (another incoming call while there is already ongoing call) from the AG and passes a pointer to `bt_hfp_evt_call_waiting_t` which contains the number of the waiting call.

4.208 bt_hfp_evt_chld_t Struct Reference

4.209 bt_hfp_evt_clip_received_t Struct Reference

HFP_EVENT_CLIP_RECEIVED event parameter.

`#include <hfp.h>`

Data Fields

- bt_char ∗ number

  Pointer to 0-terminated string that contains the number of the incoming call.
- bt_ulong number_type

  The type of the number.

4.209.1 Detailed Description

HFP_EVENT_CLIP_RECEIVED event parameter.

HFP layer calls the session callback when it received the number of an incoming call from the AG and passes a pointer to `bt_hfp_evt_clip_received_t` which contains the number of the incoming call.

4.210 bt_hfp_evt_command_completed_t Struct Reference

HFP_EVENT_CMD_COMPLETED event parameter.

`#include <hfp.h>`

Data Fields

- bt_hfp_cmd_id_enum id

  The command ID.
- bt_bool success

  TRUE - if command succeeded, FALSE - if command failed.

4.210.1 Detailed Description

HFP_EVENT_CMD_COMPLETED event parameter.

HFP layer calls the session callback when a command sent by HF to AG has been processed by the AG and a result (success or error) has been received. the callback is passed a pointer to `bt_hfp_evt_command_completed_t` which contains the id of the command and result.

4.211 bt_hfp_evt_dial_request_received_t Struct Reference

HFP_AG_EVENT_DIAL_REQUEST_RECEIVED event parameter.

`#include <hfp.h>`

Data Fields

- bt_hfp_dial_request_enum request_type

  The dial request type.
- bt_char number [HFP_MAX_NUMBER_LENGTH+1]

  0-terminated string that contains the number to dial or memory location
- bt_int memory_location

  memory location to dial

4.211.1 Detailed Description

HFP_AG_EVENT_DIAL_REQUEST_RECEIVED event parameter.

HFP layer calls the session callback when it received a command from HF device with instruction to initiate a voice call. The callback is passed a pointer to `_bt_hfp_evt_dial_request_received_t` which contains the type of the call, including place a call with a phone number supplied by HF, memory dial, or last number redial.

When application is able to place a call it should call HFP layer with corresponding call information.

4.212 bt_hfp_evt_inband_ring_changed_t Struct Reference

HFP_EVENT_INBAND_RING_CHANGED event parameter.

`#include <hfp.h>`

Data Fields

- bt_bool enabled

  TRUE - if in-band ringing is enabled, FALSE - otherwise.

4.212.1 Detailed Description

HFP_EVENT_INBAND_RING_CHANGED event parameter.

HFP layer calls the session callback when in-band ringing in the AG has been turned on or off the callback is passed a pointer to `bt_hfp_evt_inband_ring_changed_t` which contains the current status of in-band ringing in the AG.

if in-band ringing is enabled the AG will provide ringing tone. However the HF may ignore this tone and generate its own upon receiving the HFP_EVENT_RING event.

if in-band ringing is disabled the HF has to generate a ringing tone on its own. upon receiving the HFP_EVENT_↩
RING event.

4.213 bt_hfp_evt_indicator_received_t Struct Reference

HFP_EVENT_INDICATOR_RECEIVED event parameter.

`#include <hfp.h>`

Data Fields

- bt_byte id

  Indicator ID. The possible values are defined by HFP_IND_ constants.

- bt_byte value

  The value of the indicator.

4.213.1 Detailed Description

HFP_EVENT_INDICATOR_RECEIVED event parameter.

HFP layer calls the session callback when it received indicator from the AG and passes a pointer to `bt_hfp_evt_indicator_received_t` which contains the id of the indicator and its value.

4.214 bt_hfp_evt_mic_volume_changed_t Struct Reference

HFP_EVENT_MIC_VOLUME_CHANGED event parameter.

`#include <hfp.h>`

Data Fields

- bt_byte volume

  The value of the microphone gain.

4.214.1 Detailed Description

HFP_EVENT_MIC_VOLUME_CHANGED event parameter.

HFP layer calls the session callback when the microphone gain has been changed in the AG and passes a pointer to `bt_hfp_evt_mic_volume_changed_t` which contains the new value for the microphone gain.

4.215 bt_hfp_evt_query_operator_completed_t Struct Reference

HFP_EVENT_QUERY_OPERATOR_COMPLETED event parameter.

`#include <hfp.h>`

Data Fields

- bt_byte mode

  Operator's mode.
- bt_char ∗ name

  Pointer to 0-terminated string that contains the name of the operator.

4.215.1 Detailed Description

HFP_EVENT_QUERY_OPERATOR_COMPLETED event parameter.

HFP layer calls the session callback when it received operator's name from the AG during SLC setup or as a result of calling `bt_hfp_hf_query_operator` and passes a pointer to `bt_hfp_evt_query_operator_completed_t` which contains the name of the operator.

4.216 bt_hfp_evt_send_dtmf_code_t Struct Reference

4.217 bt_hfp_evt_slc_connection_state_changed_t Struct Reference

HFP_EVENT_SLC_CONNECTION_STATE_CHANGED event parameter.

`#include <hfp.h>`

Data Fields

- bt_bool connected

  Defines the state of SLC. TRUE - if SLC is connected, FALSE - if SLC is disconnected.

4.217.1 Detailed Description

HFP_EVENT_SLC_CONNECTION_STATE_CHANGED event parameter.

HFP layer calls the session callback when the state of the service level connection (SLC) has changed and passes a pointer to `bt_hfp_evt_slc_connection_state_changed_t` which contains a flag identifying the current state of SLC (connected or disconnected).

4.218 bt_hfp_evt_spk_volume_changed_t Struct Reference

HFP_EVENT_SPK_VOLUME_CHANGED event parameter.

`#include <hfp.h>`

Data Fields

- bt_byte volume

  The value of the speaker volume.

4.218.1 Detailed Description

HFP_EVENT_SPK_VOLUME_CHANGED event parameter.

HFP layer calls the session callback when the speaker volume has been changed in the AG and passes a pointer to `bt_hfp_evt_spk_volume_changed_t` which contains the new value for the speaker volume.

4.219 bt_hfp_evt_terminate_call_t Struct Reference

4.220 bt_hfp_evt_voice_recognition_changed_t Struct Reference

HFP_EVENT_VOICE_RECOGNITION_CHANGED event parameter.

`#include <hfp.h>`

Data Fields

- bt_bool enabled

  TRUE - if voice dialing is enabled, FALSE - otherwise.

4.220.1 Detailed Description

HFP_EVENT_VOICE_RECOGNITION_CHANGED event parameter.

HFP layer calls the session callback when voice dialing in the AG has been turned on or off as a result of calling `bt_hfp_hf_enable_voice_recognition`. the callback is passed a pointer to `bt_hfp_evt_voice_recognition_changed_t` which contains the current status of voice dialing in the AG.

4.221 bt_hfp_hf_t Struct Reference

4.222 bt_hfp_ind Struct Reference

Stores value of an indicator.

`#include <hfp.h>`

Data Fields

- bt_byte val

  The indicator's value.
- bt_byte id

  HF indicator ID. The possible values are defined by HFP_IND_ constants.
- bt_byte enabled

  AG indicator enabled. If 0 indicator has been disabled by HF device.

4.222.1 Detailed Description

Stores value of an indicator.

4.223 bt_hfp_session Struct Reference

4.224 bt_hid_pdu_t Struct Reference

4.225 bt_hid_report_t Struct Reference

4.226 bt_hid_session_t Struct Reference

4.227 bt_media_packet_t Struct Reference

Media packet buffer.

`#include <avdtp.h>`

Data Fields

- bt_media_packet_t ∗ next_packet

  Pointer to next buffer.
- bt_byte version

Version of the RTP implementation.

- bt_byte csrc_count

  The CSRC count contains the number of CSRC identifiers that follow the fixed header.
- bt_bool marker

  The interpretation of the marker is defined by a profile.
- bt_byte payload_type

  This field identifies the format of the RTP payload and determines its interpretation by the application.
- bt_uint seq_number

  The sequence number increments by one for each media packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.
- bt_ulong timestamp

  The Time Stamp reflects the sampling instant of the first octet in the media packet.
- bt_ulong ssrc

  The SSRC field identifies the synchronization source.
- bt_ulong ∗ csrc_list

  The CSRC list identifies the contributing sources for the payload contained in this packet.
- bt_byte ∗ data

  Pointer to a buffer to store media data.
- bt_int data_len

  Lentgth of the media data.
- bt_int max_data_len

  Maximum length of the media data that can be stored in the buffer pointed by the `data` member.
- bt_cp_header_t cp_header

  Content protection header.

4.227.1 Detailed Description

Media packet buffer.

This structure is used to receive and send media packet from/to the remote device. See more information about usage of this structure in descriptions of bt_avdtp_add_media_rx_buffer and bt_avdtp_add_media_tx_buffer.

4.227.2 Field Documentation

4.227.2.1 bt_byte∗ data

Pointer to a buffer to store media data.

This pointer must be allocated by the AVDTP consumer before adding this structure to receive (bt_avdtp_add_↩ media_rx_buffer) or send queue (bt_avdtp_add_media_tx_buffer).

4.227.2.2 bt_bool marker

The interpretation of the marker is defined by a profile.

It is intended to allow significant events such as frame boundaries to be marked in the packet stream.

4.227.2.3 bt_byte payload_type

This field identifies the format of the RTP payload and determines its interpretation by the application.

A profile specifies default static mapping of payload type codes to payload formats.

4.227.2.4 bt_ulong ssrc

The SSRC field identifies the synchronization source.

This identifier is chosen randomly, with the intent that no two synchronization sources, within the same media transport session, shall have the same SSRC identifier.

4.228 bt_sdp_client_evt_connected_t Struct Reference

4.229 bt_sdp_client_evt_disconnected_t Struct Reference

4.230 bt_sdp_data_element_t Struct Reference

4.231 bt_sdp_found_attr_list_t Struct Reference

4.232 bt_sdp_sequence_t Struct Reference

4.233 bt_sdp_serialization_state_p Struct Reference

4.234 bt_sdp_service_transaction_p Struct Reference

4.235 bt_sdp_transaction_t Struct Reference

4.236 bt_security_mgr_t Struct Reference

4.237 bt_sm_event_t Union Reference

4.238 bt_sm_evt_connected_t Struct Reference

4.239 bt_sm_evt_disconnected_t Struct Reference

4.240 bt_sm_evt_generate_lkt_t Struct Reference

4.241 bt_sm_evt_ltk_generated_t Struct Reference

4.242 bt_sm_evt_master_csrk_notif_t Struct Reference

4.243 bt_sm_evt_master_irk_notif_t Struct Reference

4.244 bt_sm_evt_master_ltk_notif_t Struct Reference

4.245 bt_sm_evt_master_ltk_request_t Struct Reference

4.246 bt_sm_evt_oob_data_request_t Struct Reference

4.247 bt_sm_evt_pairing_complete_t Struct Reference

4.248 bt_sm_evt_pairing_request_t Struct Reference

4.249 bt_sm_evt_passkey_notification_t Struct Reference

4.250 bt_sm_evt_passkey_request_t Struct Reference

4.251 bt_sm_evt_peer_keys_notif_t Struct Reference

4.252 bt_sm_evt_slave_ltk_request_t Struct Reference

4.253 bt_sm_listener_t Struct Reference

4.254 bt_sm_ltk_t Struct Reference

4.255 bt_sm_pairing_features_t Struct Reference

4.256 bt_sm_session_t Struct Reference

4.257 bt_spp_port_t Struct Reference

Serial port structure.

```
#include <spp.h>
```

Data Structures

- struct _bt_spp_port_flags_t

Data Fields

- bt_byte state

  Port state.

4.257.1 Detailed Description

Serial port structure.

This structure represents a Bluetooth serial port. Application code may only use those fields that are documented. The rest of the fields are private to the SPP implementation.

4.257.2 Field Documentation

4.257.2.1 bt_byte state

Port state.

The field is set to one of the values defined in the bt_spp_port_state_e enumeration. This field must never be modified by the application.

4.258 btx_csr_autobaud_buffer_t Struct Reference

4.259 btx_csr_bccmd_header_t Struct Reference

4.260 btx_csr_bccmd_listener_t Struct Reference

4.261 btx_csr_cached_temperature_t Struct Reference

4.262 btx_csr_create_operator_c_t Struct Reference

4.263 btx_csr_exec_hq_script_buffer_t Struct Reference

4.264 btx_csr_exec_script_buffer_t Struct Reference

4.265 btx_csr_pio_direction_mask_t Struct Reference

4.266 btx_csr_pio_protection_mask_t Struct Reference

4.267 btx_csr_pio_t Struct Reference

4.268 btx_csr_rssi_acl_t Struct Reference

4.269 btx_csr_script_t Struct Reference

4.270 btx_csr_set_ps_vars_buffer_t Struct Reference

4.271 btx_csr_strm_connect_t Struct Reference

4.272 btx_csr_strm_get_sink_t Struct Reference

4.273 btx_csr_strm_get_source_t Struct Reference

4.274 btx_csr_var_t Union Reference

Index