User's Guide Rev. 2, 12 July 2022

EdgeFast BT PAL Documentation



Document identifier: EFBTPALAPIRM

CONTENTS:

1 Bluetooth		tooth 1
	1.1	Scope
	1.2	Start Reading
		1.2.1 Pairing and Bonding
		1.2.2 Data Length and PHY Update Procedures
	1.3	Connection Management
		1.3.1 API Reference
	1.4	Data Buffers
		1.4.1 API Reference
	1.5	Generic Access Profile (GAP)
		1.5.1 API Reference
	1.6	Generic Attribute Profile (GATT)
		1.6.1 API Reference
		1.6.1.1 GATT Server
		1.6.1.2 GATT Client
	1.7	Hands Free Profile (HFP)
		1.7.1 API Reference
	1.8	Logical Link Control and Adaptation Protocol (L2CAP)
		1.8.1 API Reference
	1.9	Serial Port Emulation (RFCOMM)
		1.9.1 API Reference
	1.10	Service Discovery Protocol (SDP)
		1.10.1 API Reference
	1.11	Advance Audio Distribution Profile (A2DP)
		1.11.1 API Reference
	1.12	Serial Port Profile (SPP)
		1.12.1 API Reference
	1.13	Universal Unique Identifiers (UUIDs)
		1.13.1 API Reference
	1.14	services
		1.14.1 HTTP Proxy Service (HPS)
		1.14.1.1 API Reference
		1.14.2 Health Thermometer Service (HTS)
		1.14.2.1 API Reference
		1.14.3 Internet Protocol Support Profile (IPSP)
		1.14.3.1 API Reference
		1.14.4 Proximity Reporter (PXR)
		1.14.4.1 API Reference
Inc	lex	223

CHAPTER

ONE

BLUETOOTH

1.1 Scope

This document contains the descriptions of BLE and BR/EDR. Please ignore the BR/EDR part if the board doesn't support BR/EDR. Please check whether the board support BR/EDR based on the board package (<package>/boards/<board>/edgefast_bluetooth_examples).

1.2 Start Reading

In edgefast bluetooth stack, features are controlled by macros defined or configured in "app_config.h", so that the related features can be disabled or enabled.

1.2.1 Pairing and Bonding

Some macros can be configured to enable or disable "pairinig and bonding" feature in edgefast bluetooth stack. The macro "CONFIG_BT_SMP" configs whether the stack supports SMP. When the macro is set to 1, SMP is enabled, otherwise disabled. It is advised that these three related macros should be configured as follows:

```
Case1 when "CONFIG_BT_SMP" is set to 1:
CONFIG_BT_CONN_DISABLE_SECURITY=0;
```

CONFIG_BT_SETTINGS=1;

CONFIG_BT_HFP_HF=1.

Case2 when "CONFIG_BT_SMP" is set to 0:

CONFIG_BT_CONN_DISABLE_SECURITY=1;

CONFIG_BT_SETTINGS=0;

CONFIG_BT_HFP_HF=0.

1.2.2 Data Length and PHY Update Procedures

The data length and PHY update procedures should not be performed in the context of the connection callback. The data length and PHY update procedures could be performed in application task context after the connection is established.

The suggested process are as follows:

- 1. In connection callback, send the connection established event to the application task.
- 2. When the connection established event flag is set, perform the data length and phy update procedures in the application task.

1.3 Connection Management

The Zephyr Bluetooth stack uses an abstraction called bt_conn to represent connections to other devices. The internals of this struct are not exposed to the application, but a limited amount of information (such as the remote address) can be acquired using the $bt_conn_get_info()$ API. Connection objects are reference counted, and the application is expected to use the $bt_conn_ref()$ API whenever storing a connection pointer for a longer period of time, since this ensures that the object remains valid (even if the connection would get disconnected). Similarly the $bt_conn_unref()$ API is to be used when releasing a reference to a connection.

An application may track connections by registering a *bt_conn_cb* struct using the *bt_conn_cb_register()* API. This struct lets the application define callbacks for connection & disconnection events, as well as other events related to a connection such as a change in the security level or the connection parameters. When acting as a central the application will also get hold of the connection object through the return value of the bt_conn_create_le() API.

1.3.1 API Reference

group bt_conn

Connection management.

Defines

BT_LE_CONN_PARAM_INIT(int_min, int_max, lat, to)

Initialize connection parameters.

Parameters

- int_min Minimum Connection Interval (N * 1.25 ms)
- int_max Maximum Connection Interval (N * 1.25 ms)
- lat Connection Latency
- to Supervision Timeout (N * 10 ms)

BT_LE_CONN_PARAM(int_min, int_max, lat, to)

Helper to declare connection parameters inline

Parameters

- int_min Minimum Connection Interval (N * 1.25 ms)
- int_max Maximum Connection Interval (N * 1.25 ms)
- lat Connection Latency

• to – Supervision Timeout (N * 10 ms)

BT_LE_CONN_PARAM_DEFAULT

Default LE connection parameters: Connection Interval: 30-50 ms Latency: 0 Timeout: 4 s

BT_CONN_LE_PHY_PARAM_INIT(_pref_tx_phy, _pref_rx_phy)

Initialize PHY parameters

Parameters

- _pref_tx_phy Bitmask of preferred transmit PHYs.
- _pref_rx_phy Bitmask of preferred receive PHYs.

BT_CONN_LE_PHY_PARAM(_pref_tx_phy, _pref_rx_phy)

Helper to declare PHY parameters inline

Parameters

- _pref_tx_phy Bitmask of preferred transmit PHYs.
- _pref_rx_phy Bitmask of preferred receive PHYs.

BT_CONN_LE_PHY_PARAM_1M

Only LE 1M PHY

BT_CONN_LE_PHY_PARAM_2M

Only LE 2M PHY

BT_CONN_LE_PHY_PARAM_CODED

Only LE Coded PHY.

BT_CONN_LE_PHY_PARAM_ALL

All LE PHYs.

BT_CONN_LE_DATA_LEN_PARAM_INIT(_tx_max_len, _tx_max_time)

Initialize transmit data length parameters

Parameters

- _tx_max_len Maximum Link Layer transmission payload size in bytes.
- _tx_max_time Maximum Link Layer transmission payload time in us.

BT_CONN_LE_DATA_LEN_PARAM(_tx_max_len, _tx_max_time)

Helper to declare transmit data length parameters inline

Parameters

- _tx_max_len Maximum Link Layer transmission payload size in bytes.
- **_tx_max_time** Maximum Link Layer transmission payload time in us.

BT_LE_DATA_LEN_PARAM_DEFAULT

Default LE data length parameters.

BT_LE_DATA_LEN_PARAM_MAX

Maximum LE data length parameters.

BT_CONN_INTERVAL_TO_MS (interval)

BT_CONN_ROLE_MASTER

Connection role (central or peripheral)

BT_CONN_ROLE_SLAVE

BT_CONN_LE_CREATE_PARAM_INIT(_options, _interval, _window)

Initialize create connection parameters.

Parameters

- **_options** Create connection options.
- _interval Create connection scan interval (N * 0.625 ms).
- _window Create connection scan window (N * 0.625 ms).

BT_CONN_LE_CREATE_PARAM(_options, _interval, _window)

Helper to declare create connection parameters inline

Parameters

- **_options** Create connection options.
- **_interval** Create connection scan interval (N * 0.625 ms).
- _window Create connection scan window (N * 0.625 ms).

BT_CONN_LE_CREATE_CONN

Default LE create connection parameters. Scan continuously by setting scan interval equal to scan window.

BT_CONN_LE_CREATE_CONN_AUTO

Default LE create connection using filter accept list parameters. Scan window: 30 ms. Scan interval: 60 ms.

BT_CONN_CB_DEFINE(name)

Register a callback structure for connection events.

Parameters

• _name - Name of callback structure.

BT_PASSKEY_INVALID

Special passkey value that can be used to disable a previously set fixed passkey.

BT_BR_CONN_PARAM_INIT(role_switch)

Initialize BR/EDR connection parameters.

Parameters

• role_switch - True if role switch is allowed

BT_BR_CONN_PARAM(role_switch)

Helper to declare BR/EDR connection parameters inline

Parameters

• role_switch - True if role switch is allowed

BT_BR_CONN_PARAM_DEFAULT

Default BR/EDR connection parameters: Role switch allowed

Typedefs

```
typedef enum _bt_security bt_security_t
```

Enums

```
enum [anonymous]
```

Connection PHY options

Values:

enumerator BT_CONN_LE_PHY_OPT_NONE

Convenience value when no options are specified.

enumerator BT_CONN_LE_PHY_OPT_CODED_S2

LE Coded using S=2 coding preferred when transmitting.

enumerator $BT_CONN_LE_PHY_OPT_CODED_S8$

LE Coded using S=8 coding preferred when transmitting.

enum [anonymous]

Connection Type

Values:

enumerator BT_CONN_TYPE_LE

LE Connection Type

enumerator BT_CONN_TYPE_BR

BR/EDR Connection Type

enumerator BT_CONN_TYPE_SCO

SCO Connection Type

enumerator BT_CONN_TYPE_ISO

ISO Connection Type

```
enumerator BT_CONN_TYPE_ALL
         All Connection Type
enum [anonymous]
     Values:
     enumerator BT_CONN_ROLE_CENTRAL
     enumerator BT_CONN_ROLE_PERIPHERAL
enum bt_conn_state
     Values:
     enumerator BT_CONN_STATE_DISCONNECTED
         Channel disconnected
     enumerator BT_CONN_STATE_CONNECTING
         Channel in connecting state
     enumerator BT_CONN_STATE_CONNECTED
         Channel connected and ready for upper layer traffic on it
     enumerator BT_CONN_STATE_DISCONNECTING
         Channel in disconnecting state
enum _bt_security
    Security level.
     Values:
     enumerator BT_SECURITY_L0
         Level 0: Only for BR/EDR special cases, like SDP
     enumerator BT_SECURITY_L1
         Level 1: No encryption and no authentication.
     enumerator BT_SECURITY_L2
         Level 2: Encryption and no authentication (no MITM).
     enumerator BT_SECURITY_L3
         Level 3: Encryption and authentication (MITM).
     enumerator BT_SECURITY_L4
```

Level 4: Authenticated Secure Connections and 128-bit key.

enumerator BT_SECURITY_FORCE_PAIR

Bit to force new pairing procedure, bit-wise OR with requested security level.

enum bt_security_flag

Security Info Flags.

Values:

enumerator BT_SECURITY_FLAG_SC

Paired with Secure Connections.

enumerator BT_SECURITY_FLAG_00B

Paired with Out of Band method.

enum bt_conn_le_tx_power_phy

Values:

enumerator BT_CONN_LE_TX_POWER_PHY_NONE

Convenience macro for when no PHY is set.

enumerator BT_CONN_LE_TX_POWER_PHY_1M

LE 1M PHY

enumerator BT_CONN_LE_TX_POWER_PHY_2M

LE 2M PHY

enumerator BT_CONN_LE_TX_POWER_PHY_CODED_S8

LE Coded PHY using S=8 coding.

enumerator BT_CONN_LE_TX_POWER_PHY_CODED_S2

LE Coded PHY using S=2 coding.

enum [anonymous]

Values:

enumerator BT_CONN_LE_OPT_NONE

Convenience value when no options are specified.

enumerator BT_CONN_LE_OPT_CODED

Enable LE Coded PHY.

Enable scanning on the LE Coded PHY.

enumerator BT_CONN_LE_OPT_NO_1M

Disable LE 1M PHY.

Disable scanning on the LE 1M PHY.

@note Requires @ref BT_CONN_LE_OPT_CODED.

enum bt_security_err

Values:

enumerator BT_SECURITY_ERR_SUCCESS

Security procedure successful.

enumerator BT_SECURITY_ERR_AUTH_FAIL

Authentication failed.

enumerator BT_SECURITY_ERR_PIN_OR_KEY_MISSING

PIN or encryption key is missing.

enumerator BT_SECURITY_ERR_OOB_NOT_AVAILABLE

OOB data is not available.

enumerator BT_SECURITY_ERR_AUTH_REQUIREMENT

The requested security level could not be reached.

enumerator BT_SECURITY_ERR_PAIR_NOT_SUPPORTED

Pairing is not supported

enumerator BT_SECURITY_ERR_PAIR_NOT_ALLOWED

Pairing is not allowed.

enumerator BT_SECURITY_ERR_INVALID_PARAM

Invalid parameters.

enumerator BT_SECURITY_ERR_KEY_REJECTED

Distributed Key Rejected

enumerator BT_SECURITY_ERR_UNSPECIFIED

Pairing failed but the exact reason could not be specified.

Functions

struct bt_conn *bt_conn_ref(struct bt_conn *conn)

Increment a connection's reference count.

Increment the reference count of a connection object.

Note: Will return NULL if the reference count is zero.

Parameters

• **conn** – Connection object.

Returns

Connection object with incremented reference count, or NULL if the reference count is zero.

void bt_conn_unref(struct bt conn *conn)

Decrement a connection's reference count.

Decrement the reference count of a connection object.

Parameters

• conn – Connection object.

void bt_conn_foreach(int type, void (*func)(struct bt_conn *conn, void *data), void *data)

Iterate through all existing connections.

Parameters

- type Connection Type
- **func** Function to call for each connection.
- data Data to pass to the callback function.

```
struct bt_conn *bt_conn_lookup_addr_le(uint8_t id, const bt_addr_le_t *peer)
```

Look up an existing connection by address.

Look up an existing connection based on the remote address.

The caller gets a new reference to the connection object which must be released with $bt_conn_unref()$ once done using the object.

Parameters

- id Local identity (in most cases BT_ID_DEFAULT).
- **peer** Remote address.

Returns

Connection object or NULL if not found.

```
const bt_addr_le_t *bt_conn_get_dst(const struct bt_conn *conn)
```

Get destination (peer) address of a connection.

Parameters

• conn – Connection object.

Returns

Destination address.

```
const bt_addr_t *bt_conn_get_dst_br(const struct bt_conn *conn)
```

Get destination (peer) address of a BR connection.

Parameters

• conn – Connection object.

Returns

Destination address.

uint8_t bt_conn_index(const struct bt_conn *conn)

Get array index of a connection.

This function is used to map bt_conn to index of an array of connections. The array has CON-FIG_BT_MAX_CONN elements.

Parameters

• **conn** – Connection object.

Returns

Index of the connection object. The range of the returned value is 0..CONFIG_BT_MAX_CONN-1

int bt_conn_get_info(const struct bt_conn *conn, struct bt_conn_info *info)

Get connection info.

Parameters

- conn Connection object.
- **info** Connection info object.

Returns

Zero on success or (negative) error code on failure.

int **bt_conn_get_remote_info**(struct bt_conn *conn, struct bt_conn_remote_info *remote_info)

Get connection info for the remote device.

Note: In order to retrieve the remote version (version, manufacturer and subversion) @kconfig{CONFIG_BT_REMOTE_VERSION} must be enabled

Note: The remote information is exchanged directly after the connection has been established. The application can be notified about when the remote information is available through the remote_info_available callback.

Parameters

- conn Connection object.
- remote_info Connection remote info object.

Returns

Zero on success or (negative) error code on failure.

Returns

-EBUSY The remote information is not yet available.

int **bt_conn_le_get_tx_power_level**(struct bt_conn *conn, struct bt_conn_le_tx_power *tx_power_level)

Get connection transmit power level.

Parameters

- conn Connection object.
- tx_power_level Transmit power level descriptor.

Returns

Zero on success or (negative) error code on failure.

Returns

-ENOBUFS HCI command buffer is not available.

int **bt_conn_le_param_update**(struct bt_conn *conn, const struct bt_le_conn_param *param)

Update the connection parameters.

If the local device is in the peripheral role then updating the connection parameters will be delayed. This delay can be configured by through the @kconfig{CONFIG_BT_CONN_PARAM_UPDATE_TIMEOUT} option.

Parameters

- conn Connection object.
- param Updated connection parameters.

Returns

Zero on success or (negative) error code on failure.

int **bt_conn_le_data_len_update**(struct bt_conn *conn, const struct *bt_conn_le_data_len_param* *param)

Update the connection transmit data length parameters.

Parameters

- conn Connection object.
- param Updated data length parameters.

Returns

Zero on success or (negative) error code on failure.

int bt_conn_le_phy_update(struct bt_conn *conn, const struct bt_conn_le_phy_param *param)

Update the connection PHY parameters.

Update the preferred transmit and receive PHYs of the connection. Use *BT_GAP_LE_PHY_NONE* to indicate no preference.

Parameters

- **conn** Connection object.
- param Updated connection parameters.

Returns

Zero on success or (negative) error code on failure.

int **bt_conn_disconnect**(struct bt conn *conn, uint8 t reason)

Disconnect from a remote device or cancel pending connection.

Disconnect an active connection with the specified reason code or cancel pending outgoing connection.

The disconnect reason for a normal disconnect should be: BT_HCI_ERR_REMOTE_USER_TERM_CONN.

The following disconnect reasons are accepted:

- BT_HCI_ERR_AUTH_FAIL
- BT_HCI_ERR_REMOTE_USER_TERM_CONN
- BT_HCI_ERR_REMOTE_LOW_RESOURCES
- BT_HCI_ERR_REMOTE_POWER_OFF
- BT HCI ERR UNSUPP REMOTE FEATURE
- BT_HCI_ERR_PAIRING_NOT_SUPPORTED

• BT_HCI_ERR_UNACCEPT_CONN_PARAM

Parameters

- conn Connection to disconnect.
- **reason** Reason code for the disconnection.

Returns

Zero on success or (negative) error code on failure.

int **bt_conn_le_create**(const *bt_addr_le_t* *peer, const struct *bt_conn_le_create_param* *create_param, const struct *bt_le_conn_param* *conn_param, struct bt_conn **conn)

Initiate an LE connection to a remote device.

Allows initiate new LE link to remote peer using its address.

The caller gets a new reference to the connection object which must be released with $bt_conn_unref()$ once done using the object.

This uses the General Connection Establishment procedure.

The application must disable explicit scanning before initiating a new LE connection.

When @kconfig{CONFIG_BT_PRIVACY} enabled and peer is an identity address from a local bond, this API will connect to an advertisement with either:

- the address being an RPA resolved from the IRK obtained during bonding.
- the passed identity address, if the local identity is not in Network Privacy Mode.

Parameters

- **peer** [in] Remote address.
- **create_param [in]** Create connection parameters.
- **conn_param [in]** Initial connection parameters.
- conn [out] Valid connection object on success.

Returns

Zero on success or (negative) error code on failure.

```
int bt_conn_le_create_auto(const struct bt_conn_le_create_param *create_param, const struct bt_le_conn_param *conn_param)
```

Automatically connect to remote devices in the filter accept list..

This uses the Auto Connection Establishment procedure. The procedure will continue until a single connection is established or the procedure is stopped through $bt_conn_create_auto_stop$. To establish connections to all devices in the the filter accept list the procedure should be started again in the connected callback after a new connection has been established.

Parameters

- **create_param** Create connection parameters
- **conn_param** Initial connection parameters.

Returns

Zero on success or (negative) error code on failure.

Returns

-ENOMEM No free connection object available.

int bt_conn_create_auto_stop(void)

Stop automatic connect creation.

Returns

Zero on success or (negative) error code on failure.

int **bt_le_set_auto_conn**(const *bt_addr_le_t* *addr, const struct *bt_le_conn_param* *param)

Automatically connect to remote device if it's in range.

This function enables/disables automatic connection initiation. Every time the device loses the connection with peer, this connection will be re-established if connectable advertisement from peer is received.

Note: Auto connect is disabled during explicit scanning.

Parameters

- addr Remote Bluetooth address.
- param If non-NULL, auto connect is enabled with the given parameters. If NULL, auto connect is disabled.

Returns

Zero on success or error code otherwise.

int **bt_conn_set_security**(struct bt_conn *conn, bt_security_t sec)

Set security level for a connection.

This function enable security (encryption) for a connection. If the device has bond information for the peer with sufficiently strong key encryption will be enabled. If the connection is already encrypted with sufficiently strong key this function does nothing.

If the device has no bond information for the peer and is not already paired then the pairing procedure will be initiated. If the device has bond information or is already paired and the keys are too weak then the pairing procedure will be initiated.

This function may return error if required level of security is not possible to achieve due to local or remote device limitation (e.g., input output capabilities), or if the maximum number of paired devices has been reached.

This function may return error if the pairing procedure has already been initiated by the local device or the peer device.

Note: When @kconfig{CONFIG_BT_SMP_SC_ONLY} is enabled then the security level will always be level 4.

Note: When @kconfig{CONFIG_BT_SMP_OOB_LEGACY_PAIR_ONLY} is enabled then the security level will always be level 3.

Parameters

- **conn** Connection object.
- **sec** Requested security level.

Returns

0 on success or negative error

bt_security_t bt_conn_get_security(const struct bt_conn *conn)

Get security level for a connection.

Returns

Connection security level

uint8_t bt_conn_enc_key_size(const struct bt_conn *conn)

Get encryption key size.

This function gets encryption key size. If there is no security (encryption) enabled 0 will be returned.

Parameters

• conn – Existing connection object.

Returns

Encryption key size.

void bt_conn_cb_register(struct bt_conn_cb *cb)

Register connection callbacks.

Register callbacks to monitor the state of connections.

Parameters

• cb – Callback struct. Must point to memory that remains valid.

void bt_set_bondable(bool enable)

Enable/disable bonding.

Set/clear the Bonding flag in the Authentication Requirements of SMP Pairing Request/Response data. The initial value of this flag depends on BT_BONDABLE Kconfig setting. For the vast majority of applications calling this function shouldn't be needed.

Parameters

• enable – Value allowing/disallowing to be bondable.

void bt_set_oob_data_flag(bool enable)

Allow/disallow remote OOB data to be used for pairing.

Set/clear the OOB data flag for SMP Pairing Request/Response data. The initial value of this flag depends on BT_OOB_DATA_PRESENT Kconfig setting.

Parameters

• enable – Value allowing/disallowing remote OOB data.

int bt_le_oob_set_legacy_tk(struct bt_conn *conn, const uint8_t *tk)

Set OOB Temporary Key to be used for pairing.

This function allows to set OOB data for the LE legacy pairing procedure. The function should only be called in response to the oob_data_request() callback provided that the legacy method is user pairing.

Parameters

- conn Connection object
- **tk** Pointer to 16 byte long TK array

Returns

Zero on success or -EINVAL if NULL

int **bt_le_oob_set_sc_data**(struct bt_conn *conn, const struct bt_le_oob_sc_data *oobd_local, const struct bt_le_oob_sc_data *oobd_remote)

Set OOB data during LE Secure Connections (SC) pairing procedure.

This function allows to set OOB data during the LE SC pairing procedure. The function should only be called in response to the oob_data_request() callback provided that LE SC method is used for pairing.

The user should submit OOB data according to the information received in the callback. This may yield three different configurations: with only local OOB data present, with only remote OOB data present or with both local and remote OOB data present.

Parameters

- conn Connection object
- oobd_local Local OOB data or NULL if not present
- **oobd_remote** Remote OOB data or NULL if not present

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

int **bt_le_oob_get_sc_data**(struct bt_conn *conn, const struct *bt_le_oob_sc_data* **oobd_local, const struct *bt_le_oob_sc_data* **oobd_remote)

Get OOB data used for LE Secure Connections (SC) pairing procedure.

This function allows to get OOB data during the LE SC pairing procedure that were set by the $bt_le_oob_set_sc_data()$ API.

Note: The OOB data will only be available as long as the connection object associated with it is valid.

Parameters

- conn Connection object
- oobd local Local OOB data or NULL if not set
- **oobd remote** Remote OOB data or NULL if not set

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

int bt_passkey_set(unsigned int passkey)

Set a fixed passkey to be used for pairing.

This API is only available when the CONFIG_BT_FIXED_PASSKEY configuration option has been enabled.

Sets a fixed passkey to be used for pairing. If set, the pairing_confirm() callback will be called for all incoming pairings.

Parameters

• **passkey** – A valid passkey (0 - 999999) or BT_PASSKEY_INVALID to disable a previously set fixed passkey.

Returns

0 on success or a negative error code on failure.

int bt_conn_auth_cb_register(const struct bt_conn_auth_cb *cb)

Register authentication callbacks.

Register callbacks to handle authenticated pairing. Passing NULL unregisters a previous callbacks structure.

Parameters

• cb - Callback struct.

Returns

Zero on success or negative error code otherwise

int **bt_conn_auth_cb_overlay**(struct bt_conn *conn, const struct bt_conn_auth_cb *cb)

Overlay authentication callbacks used for a given connection.

This function can be used only for Bluetooth LE connections. The @kconfig{CONFIG_BT_SMP} must be enabled for this function.

The authentication callbacks for a given connection cannot be overlaid if security procedures in the SMP module have already started. This function can be called only once per connection.

Parameters

- **conn** Connection object.
- **cb** Callback struct.

Returns

Zero on success or negative error code otherwise

int bt_conn_auth_info_cb_register(struct bt_conn_auth_info_cb *cb)

Register authentication information callbacks.

Register callbacks to get authenticated pairing information. Multiple registrations can be done.

Parameters

• **cb** – Callback struct.

Returns

Zero on success or negative error code otherwise

int bt_conn_auth_info_cb_unregister(struct bt_conn_auth_info_cb *cb)

Unregister authentication information callbacks.

Unregister callbacks to stop getting authenticated pairing information.

Parameters

• cb - Callback struct.

Returns

Zero on success or negative error code otherwise

$int \ \textbf{bt_conn_auth_passkey_entry} (struct \ bt_conn \ *conn, unsigned \ int \ passkey)$

Reply with entered passkey.

This function should be called only after passkey_entry callback from *bt_conn_auth_cb* structure was called.

Parameters

- conn Connection object.
- passkey Entered passkey.

Returns

Zero on success or negative error code otherwise

int bt_conn_auth_cancel(struct bt_conn *conn)

Cancel ongoing authenticated pairing.

This function allows to cancel ongoing authenticated pairing.

Parameters

• conn – Connection object.

Returns

Zero on success or negative error code otherwise

int bt_conn_auth_passkey_confirm(struct bt_conn *conn)

Reply if passkey was confirmed to match by user.

This function should be called only after passkey_confirm callback from bt_conn_auth_cb structure was called.

Parameters

• **conn** – Connection object.

Returns

Zero on success or negative error code otherwise

int bt_conn_auth_pairing_confirm(struct bt conn *conn)

Reply if incoming pairing was confirmed by user.

This function should be called only after pairing_confirm callback from *bt_conn_auth_cb* structure was called if user confirmed incoming pairing.

Parameters

• conn – Connection object.

Returns

Zero on success or negative error code otherwise

int bt_conn_auth_pincode_entry(struct bt_conn *conn, const char *pin)

Reply with entered PIN code.

This function should be called only after PIN code callback from *bt_conn_auth_cb* structure was called. It's for legacy 2.0 devices.

Parameters

- conn Connection object.
- pin Entered PIN code.

Returns

Zero on success or negative error code otherwise

struct bt_conn *bt_conn_create_br(const bt_addr_t *peer, const struct bt_br_conn_param *param)

Initiate an BR/EDR connection to a remote device.

Allows initiate new BR/EDR link to remote peer using its address.

The caller gets a new reference to the connection object which must be released with $bt_conn_unref()$ once done using the object.

Parameters

- **peer** Remote address.
- param Initial connection parameters.

Returns

Valid connection object on success or NULL otherwise.

```
struct bt_conn *bt_conn_create_sco(const bt_addr_t *peer)
```

Initiate an SCO connection to a remote device.

Allows initiate new SCO link to remote peer using its address.

The caller gets a new reference to the connection object which must be released with $bt_conn_unref()$ once done using the object.

Parameters

• **peer** – Remote address.

Returns

Valid connection object on success or NULL otherwise.

struct bt_le_conn_param

#include <conn.h> Connection parameters for LE connections

struct bt_conn_le_phy_info

#include <conn.h> Connection PHY information for LE connections

Public Members

```
uint8_t rx_phy
```

Connection transmit PHY

struct bt_conn_le_phy_param

#include <conn.h> Preferred PHY parameters for LE connections

Public Members

```
uint8_t pref_tx_phy
```

Connection PHY options.

```
uint8_t pref_rx_phy
```

Bitmask of preferred transmit PHYs

struct bt_conn_le_data_len_info

#include <conn.h> Connection data length information for LE connections

Public Members

uint16_t tx_max_len

Maximum Link Layer transmission payload size in bytes.

uint16_t tx_max_time

Maximum Link Layer transmission payload time in us.

uint16_t rx_max_len

Maximum Link Layer reception payload size in bytes.

uint16_t rx_max_time

Maximum Link Layer reception payload time in us.

struct bt_conn_le_data_len_param

#include <conn.h> Connection data length parameters for LE connections

Public Members

uint16_t tx_max_len

Maximum Link Layer transmission payload size in bytes.

uint16 t tx_max_time

Maximum Link Layer transmission payload time in us.

struct bt_conn_le_info

#include <conn.h> LE Connection Info Structure

Public Members

```
const bt_addr_le_t *src
```

Source (Local) Identity Address

```
const bt_addr_le_t *dst
```

Destination (Remote) Identity Address or remote Resolvable Private Address (RPA) before identity has been resolved.

```
const bt_addr_le_t *local
```

Local device address used during connection setup.

```
const bt_addr_le_t *remote
```

Remote device address used during connection setup.

```
uint16_t latency
         Connection interval
     uint16_t timeout
         Connection peripheral latency
     const struct bt_conn_le_phy_info *phy
         Connection supervision timeout
struct bt_conn_br_info
     #include <conn.h> BR/EDR Connection Info Structure
struct bt_security_info
     #include <conn.h> Security Info Structure.
     Public Members
     bt_security_t level
         Security Level.
     uint8_t enc_key_size
         Encryption Key Size.
     enum bt_security_flag flags
         Flags.
struct bt_conn_info
     #include <conn.h> Connection Info Structure
     Public Members
     uint8_t type
         Connection Type.
     uint8_t role
         Connection Role.
     uint8 t id
         Which local identity the connection was created with
     union bt_conn_info.[anonymous] [anonymous]
         Connection Type specific Info.
```

```
enum bt_conn_state state
```

Connection state.

struct bt_security_info security

Security specific info.

struct bt_conn_le_remote_info

#include <conn.h> LE Connection Remote Info Structure

Public Members

```
const uint8_t *features
```

Remote LE feature set (bitmask).

struct bt_conn_br_remote_info

#include <conn.h> BR/EDR Connection Remote Info structure

Public Members

```
const uint8_t *features
```

Remote feature set (pages of bitmasks).

uint8_t num_pages

Number of pages in the remote feature set.

struct bt_conn_remote_info

#include <conn.h> Connection Remote Info Structure.

Note: The version, manufacturer and subversion fields will only contain valid data if @kconfig{CONFIG_BT_REMOTE_VERSION} is enabled.

Public Members

uint8_t type

Connection Type

uint8_t version

Remote Link Layer version

uint16 t manufacturer

Remote manufacturer identifier

uint16_t subversion

Per-manufacturer unique revision

struct bt_conn_le_tx_power

#include <conn.h> LE Transmit Power Level Structure

Public Members

```
uint8_t phy
```

Input: 1M, 2M, Coded S2 or Coded S8

int8_t current_level

Output: current transmit power level

int8_t max_level

Output: maximum transmit power level

struct bt_conn_le_create_param

#include <conn.h>

Public Members

uint32_t options

Bit-field of create connection options.

uint16_t interval

Scan interval (N * 0.625 ms)

uint16 t window

Scan window (N * 0.625 ms)

uint16_t interval_coded

Scan interval LE Coded PHY (N * 0.625 MS)

Set zero to use same as LE 1M PHY scan interval

uint16_t window_coded

Scan window LE Coded PHY (N * 0.625 MS)

Set zero to use same as LE 1M PHY scan window.

uint16_t timeout

Connection initiation timeout (N * 10 MS)

Set zero to use the default @kconfig{CONFIG_BT_CREATE_CONN_TIMEOUT} timeout.

Note: Unused in *bt_conn_le_create_auto*

struct bt_conn_cb

#include <conn.h> Connection callback structure.

This structure is used for tracking the state of a connection. It is registered with the help of the $bt_conn_cb_register()$ API. It's permissible to register multiple instances of this bt_conn_cb type, in case different modules of an application are interested in tracking the connection state. If a callback is not of interest for an instance, it may be set to NULL and will as a consequence not be used for that instance.

Public Members

void (*connected)(struct bt_conn *conn, uint8_t err)

A new connection has been established.

This callback notifies the application of a new connection. In case the err parameter is non-zero it means that the connection establishment failed.

err can mean either of the following:

- BT_HCI_ERR_UNKNOWN_CONN_ID Creating the connection started by bt_conn_le_create was canceled either by the user through bt_conn_disconnect or by the timeout in the host through bt_conn_le_create_param timeout parameter, which defaults to @kcon-fig{CONFIG_BT_CREATE_CONN_TIMEOUT} seconds.
- BT_HCI_ERR_ADV_TIMEOUT High duty cycle directed connectable advertiser started by bt_le_adv_start failed to be connected within the timeout.

Note: If the connection was established from an advertising set then the advertising set cannot be restarted directly from this callback. Instead use the connected callback of the advertising set.

Param conn

New connection object.

Param err

HCI error. Zero for success, non-zero otherwise.

void (*disconnected)(struct bt_conn *conn, uint8_t reason)

A connection has been disconnected.

This callback notifies the application that a connection has been disconnected.

When this callback is called the stack still has one reference to the connection object. If the application in this callback tries to start either a connectable advertiser or create a new connection this might fail because there are no free connection objects available. To avoid this issue it is recommended to either start connectable advertise or create a new connection using k_{work_submit} or increase @kconfig{CONFIG_BT_MAX_CONN}.

Param conn

Connection object.

Param reason

BT_HCI_ERR_* reason for the disconnection.

bool (*le_param_req)(struct bt conn *conn, struct bt le conn param *param)

LE connection parameter update request.

This callback notifies the application that a remote device is requesting to update the connection parameters. The application accepts the parameters by returning true, or rejects them by returning false. Before accepting, the application may also adjust the parameters to better suit its needs.

It is recommended for an application to have just one of these callbacks for simplicity. However, if an application registers multiple it needs to manage the potentially different requirements for each callback. Each callback gets the parameters as returned by previous callbacks, i.e. they are not necessarily the same ones as the remote originally sent.

If the application does not have this callback then the default is to accept the parameters.

Param conn

Connection object.

Param param

Proposed connection parameters.

Return

true to accept the parameters, or false to reject them.

void (*le_param_updated)(struct bt_conn *conn, uint16_t interval, uint16_t latency, uint16_t timeout)

The parameters for an LE connection have been updated.

This callback notifies the application that the connection parameters for an LE connection have been updated.

Param conn

Connection object.

Param interval

Connection interval.

Param latency

Connection latency.

Param timeout

Connection supervision timeout.

void (*identity_resolved)(struct bt_conn *conn, const bt_addr_le_t *rpa, const bt_addr_le_t
*identity)

Remote Identity Address has been resolved.

This callback notifies the application that a remote Identity Address has been resolved

Param conn

Connection object.

Param rpa

Resolvable Private Address.

Param identity

Identity Address.

void (*security_changed)(struct bt_conn *conn, bt_security_t level, enum bt_security_err err)

The security level of a connection has changed.

This callback notifies the application that the security of a connection has changed.

The security level of the connection can either have been increased or remain unchanged. An increased security level means that the pairing procedure has been performed or the bond information from a previous connection has been applied. If the security level remains unchanged this means that the encryption key has been refreshed for the connection.

Param conn

Connection object.

Param level

New security level of the connection.

Param err

Security error. Zero for success, non-zero otherwise.

```
void (*remote_info_available)(struct bt_conn *conn, struct bt_conn_remote_info *remote_info)
```

Remote information procedures has completed.

This callback notifies the application that the remote information has been retrieved from the remote peer.

Param conn

Connection object.

Param remote info

Connection information of remote device.

```
void (*le_phy_updated)(struct bt_conn *conn, struct bt_conn_le_phy_info *param)
```

The PHY of the connection has changed.

This callback notifies the application that the PHY of the connection has changed.

Param conn

Connection object.

Param info

Connection LE PHY information.

```
void (*le_data_len_updated)(struct bt_conn *conn, struct bt_conn_le_data_len_info *info)
```

The data length parameters of the connection has changed.

This callback notifies the application that the maximum Link Layer payload length or transmission time has changed.

Param conn

Connection object.

Param info

Connection data length information.

struct bt_conn_oob_info

#include <conn.h> Info Structure for OOB pairing

Public Types

enum [anonymous]

Type of OOB pairing method

Values:

enumerator BT_CONN_OOB_LE_LEGACY

LE legacy pairing

enumerator BT_CONN_OOB_LE_SC

LE SC pairing

Public Members

```
enum bt_conn_oob_info.[anonymous] type
```

Type of OOB pairing method

struct bt_conn_pairing_feat

#include <conn.h> Pairing request and pairing response info structure.

This structure is the same for both smp_pairing_req and smp_pairing_rsp and a subset of the packet data, except for the initial Code octet. It is documented in Core Spec. Vol. 3, Part H, 3.5.1 and 3.5.2.

Public Members

```
uint8_t io_capability
IO Capability, Core Spec. Vol 3, Part H, 3.5.1, Table 3.4

uint8_t oob_data_flag
OOB data flag, Core Spec. Vol 3, Part H, 3.5.1, Table 3.5

uint8_t auth_req
AuthReq, Core Spec. Vol 3, Part H, 3.5.1, Fig. 3.3

uint8_t max_enc_key_size
Maximum Encryption Key Size, Core Spec. Vol 3, Part H, 3.5.1

uint8_t init_key_dist
Initiator Key Distribution/Generation, Core Spec. Vol 3, Part H, 3.6.1, Fig. 3.11

uint8_t resp_key_dist
Responder Key Distribution/Generation, Core Spec. Vol 3, Part H 3.6.1, Fig. 3.11
```

struct bt_conn_auth_cb

#include <conn.h> Authenticated pairing callback structure

Public Members

```
enum bt_security_err (*pairing_accept)(struct bt_conn *conn, const struct bt_conn_pairing_feat *const feat)
```

Query to proceed incoming pairing or not.

On any incoming pairing req/rsp this callback will be called for the application to decide whether to allow for the pairing to continue.

The pairing info received from the peer is passed to assist making the decision.

As this callback is synchronous the application should return a response value immediately. Otherwise it may affect the timing during pairing. Hence, this information should not be conveyed to the user to take action.

The remaining callbacks are not affected by this, but do notice that other callbacks can be called during the pairing. Eg. if pairing_confirm is registered both will be called for Just-Works pairings.

This callback may be unregistered in which case pairing continues as if the Kconfig flag was not set.

This callback is not called for BR/EDR Secure Simple Pairing (SSP).

Param conn

Connection where pairing is initiated.

Param feat

Pairing req/resp info.

void (*passkey_display)(struct bt_conn *conn, unsigned int passkey)

Display a passkey to the user.

When called the application is expected to display the given passkey to the user, with the expectation that the passkey will then be entered on the peer device. The passkey will be in the range of 0 - 999999, and is expected to be padded with zeroes so that six digits are always shown. E.g. the value 37 should be shown as 000037.

This callback may be set to NULL, which means that the local device lacks the ability do display a passkey. If set to non-NULL the cancel callback must also be provided, since this is the only way the application can find out that it should stop displaying the passkey.

Param conn

Connection where pairing is currently active.

Param passkey

Passkey to show to the user.

void (*passkey_entry)(struct bt_conn *conn)

Request the user to enter a passkey.

When called the user is expected to enter a passkey. The passkey must be in the range of 0 - 999999, and should be expected to be zero-padded, as that's how the peer device will typically be showing it (e.g. 37 would be shown as 000037).

Once the user has entered the passkey its value should be given to the stack using the bt_conn_auth_passkey_entry() API.

This callback may be set to NULL, which means that the local device lacks the ability to enter a passkey. If set to non-NULL the cancel callback must also be provided, since this is the only way the application can find out that it should stop requesting the user to enter a passkey.

Param conn

Connection where pairing is currently active.

void (*passkey_confirm)(struct bt_conn *conn, unsigned int passkey)

Request the user to confirm a passkey.

When called the user is expected to confirm that the given passkey is also shown on the peer device.. The passkey will be in the range of 0 - 999999, and should be zero-padded to always be six digits (e.g. 37 would be shown as 000037).

Once the user has confirmed the passkey to match, the *bt_conn_auth_passkey_confirm()* API should be called. If the user concluded that the passkey doesn't match the *bt_conn_auth_cancel()* API should be called.

This callback may be set to NULL, which means that the local device lacks the ability to confirm a passkey. If set to non-NULL the cancel callback must also be provided, since this is the only way the application can find out that it should stop requesting the user to confirm a passkey.

Param conn

Connection where pairing is currently active.

Param passkey

Passkey to be confirmed.

void (*oob_data_request)(struct bt_conn *conn, struct bt_conn_oob_info *info)

Request the user to provide Out of Band (OOB) data.

When called the user is expected to provide OOB data. The required data are indicated by the information structure.

For LE Secure Connections OOB pairing, the user should provide local OOB data, remote OOB data or both depending on their availability. Their value should be given to the stack using the $bt_le_oob_set_sc_data()$ API.

This callback must be set to non-NULL in order to support OOB pairing.

Param conn

Connection where pairing is currently active.

Param info

OOB pairing information.

void (*cancel)(struct bt_conn *conn)

Cancel the ongoing user request.

This callback will be called to notify the application that it should cancel any previous user request (passkey display, entry or confirmation).

This may be set to NULL, but must always be provided whenever the passkey_display, passkey_entry passkey_confirm or pairing_confirm callback has been provided.

Param conn

Connection where pairing is currently active.

void (*pairing_confirm)(struct bt_conn *conn)

Request confirmation for an incoming pairing.

This callback will be called to confirm an incoming pairing request where none of the other user callbacks is applicable.

If the user decides to accept the pairing the $bt_conn_auth_pairing_confirm()$ API should be called. If the user decides to reject the pairing the $bt_conn_auth_cancel()$ API should be called.

This callback may be set to NULL, which means that the local device lacks the ability to confirm a pairing request. If set to non-NULL the cancel callback must also be provided, since this is the only way the application can find out that it should stop requesting the user to confirm a pairing request.

Param conn

Connection where pairing is currently active.

void (*pincode_entry)(struct bt_conn *conn, bool highsec)

Request the user to enter a passkey.

This callback will be called for a BR/EDR (Bluetooth Classic) connection where pairing is being performed. Once called the user is expected to enter a PIN code with a length between 1 and 16 digits. If the *highsec* parameter is set to true the PIN code must be 16 digits long.

Once entered, the PIN code should be given to the stack using the bt_conn_auth_pincode_entry() API.

This callback may be set to NULL, however in that case pairing over BR/EDR will not be possible. If provided, the cancel callback must be provided as well.

Param conn

Connection where pairing is currently active.

Param highsec

true if 16 digit PIN is required.

struct bt_conn_auth_info_cb

#include <conn.h> Authenticated pairing information callback structure

Public Members

```
void (*pairing_complete)(struct bt_conn *conn, bool bonded)
```

notify that pairing procedure was complete.

This callback notifies the application that the pairing procedure has been completed.

Param conn

Connection object.

Param bonded

Bond information has been distributed during the pairing procedure.

```
void (*pairing_failed)(struct bt_conn *conn, enum bt_security_err reason)
```

notify that pairing process has failed.

Param conn

Connection object.

Param reason

Pairing failed reason

```
void (*bond_deleted)(uint8_t id, const bt_addr_le_t *peer)
```

Notify that bond has been deleted.

This callback notifies the application that the bond information for the remote peer has been deleted

Param id

Which local identity had the bond.

Param peer

Remote address.

```
sys_snode_t node
```

Internally used field for list handling

struct bt_br_conn_param

#include <conn.h> Connection parameters for BR/EDR connections

```
union __unnamed39__
```

Connection Type specific Info.

Public Members

```
struct bt_conn_le_info le

LE Connection specific Info.

struct bt_conn_br_info br

BR/EDR Connection specific Info.

union __unnamed41__

Public Members
```

struct bt_conn_br_remote_info br

struct *bt_conn_le_remote_info* **1e**LE connection remote info

BR/EDR connection remote info

union __unnamed44__

Public Members

struct *bt_conn_oob_info*.[anonymous].[anonymous] **lesc**LE Secure Connections OOB pairing parameters

struct lesc

LE Secure Connections OOB pairing parameters

Public Members

enum bt_config (anonymous).[anonymous] (anonymous) oob_config OOB data configuration

1.4 Data Buffers

1.4.1 API Reference

group bt_buf

Data buffers.

Defines

BT_BUF_RESERVE

BT_BUF_SIZE(size)

Helper to include reserved HCI data in buffer calculations

BT_BUF_ACL_SIZE(size)

Helper to calculate needed buffer size for HCI ACL packets

BT_BUF_EVT_SIZE(size)

Helper to calculate needed buffer size for HCI Event packets.

BT_BUF_CMD_SIZE(size)

Helper to calculate needed buffer size for HCI Command packets.

BT_BUF_ISO_SIZE(size)

Helper to calculate needed buffer size for HCI ISO packets.

BT_BUF_ACL_RX_SIZE

Data size needed for HCI ACL RX buffers

BT_BUF_EVT_RX_SIZE

Data size needed for HCI Event RX buffers

BT_BUF_ISO_RX_SIZE

BT_BUF_ISO_RX_COUNT

BT_BUF_RX_SIZE

Data size needed for HCI ACL, HCI ISO or Event RX buffers

BT_BUF_RX_COUNT

Buffer count needed for HCI ACL, HCI ISO or Event RX buffers

BT_BUF_CMD_TX_SIZE

Data size needed for HCI Command buffers.

Enums

enum bt_buf_type

Possible types of buffers passed around the Bluetooth stack

Values:

enumerator BT_BUF_CMD

HCI command

1.4. Data Buffers 31

```
enumerator BT_BUF_EVT
```

HCI event

enumerator BT_BUF_ACL_OUT

Outgoing ACL data

enumerator BT_BUF_ACL_IN

Incoming ACL data

enumerator BT_BUF_ISO_OUT

Outgoing ISO data

enumerator BT_BUF_ISO_IN

Incoming ISO data

enumerator BT_BUF_H4

H:4 data

Functions

struct net_buf *bt_buf_get_rx(enum bt_buf_type type, k_timeout_t timeout)

Allocate a buffer for incoming data

This will set the buffer type so bt buf set type() does not need to be explicitly called before bt recv prio().

Parameters

- **type** Type of buffer. Only BT_BUF_EVT, BT_BUF_ACL_IN and BT_BUF_ISO_IN are allowed.
- **timeout** Non-negative waiting period to obtain a buffer or one of the special values K_NO_WAIT and K_FOREVER.

Returns

A new buffer.

struct net_buf *bt_buf_get_tx(enum bt_buf_type type, k_timeout_t timeout, const void *data, size_t size)

Allocate a buffer for outgoing data

This will set the buffer type so $bt_buf_set_type()$ does not need to be explicitly called before bt_send().

Parameters

- **type** Type of buffer. Only BT_BUF_CMD, BT_BUF_ACL_OUT or BT_BUF_H4, when operating on H:4 mode, are allowed.
- **timeout** Non-negative waiting period to obtain a buffer or one of the special values K_NO_WAIT and K_FOREVER.
- data Initial data to append to buffer.
- size Initial data size.

Returns

A new buffer.

```
struct net_buf *bt_buf_get_cmd_complete(k_timeout_t timeout)
```

Allocate a buffer for an HCI Command Complete/Status Event

This will set the buffer type so $bt_buf_set_type()$ does not need to be explicitly called before bt_recv_prio().

Parameters

• **timeout** – Non-negative waiting period to obtain a buffer or one of the special values K_NO_WAIT and K_FOREVER.

Returns

A new buffer.

```
struct net_buf *bt_buf_get_evt(uint8_t evt, bool discardable, k_timeout_t timeout)
```

Allocate a buffer for an HCI Event

This will set the buffer type so $bt_buf_set_type()$ does not need to be explicitly called before $bt_recv_prio()$ or $bt_recv()$.

Parameters

- evt HCI event code
- **discardable** Whether the driver considers the event discardable.
- **timeout** Non-negative waiting period to obtain a buffer or one of the special values K_NO_WAIT and K_FOREVER.

Returns

A new buffer.

static inline void **bt_buf_set_type**(struct net_buf *buf, enum *bt_buf_type* type)

Set the buffer type

Parameters

- **buf** Bluetooth buffer
- type The BT_* type to set the buffer to

static inline enum bt_buf_type bt_buf_get_type(struct net_buf *buf)

Get the buffer type

Parameters

• **buf** – Bluetooth buffer

Returns

The BT * type to of the buffer

struct bt_buf_data

#include <buf.h> This is a base type for bt_buf user data.

1.4. Data Buffers 33

1.5 Generic Access Profile (GAP)

1.5.1 API Reference

group bt_gap

Generic Access Profile.

Defines

BT_ID_DEFAULT

Convenience macro for specifying the default identity. This helps make the code more readable, especially when only one identity is supported.

BT_DATA(_type, _data, _data_len)

Helper to declare elements of bt_data arrays.

This macro is mainly for creating an array of struct bt_data elements which is then passed to e.g. $bt_le_adv_start()$.

Parameters

- **_type** Type of advertising data field
- _data Pointer to the data field payload
- _data_len Number of bytes behind the _data pointer

BT_DATA_BYTES(_type, _bytes...)

Helper to declare elements of *bt_data* arrays.

This macro is mainly for creating an array of struct bt_data elements which is then passed to e.g. $bt_le_adv_start()$.

Parameters

- _type Type of advertising data field
- _bytes Variable number of single-byte parameters

BT_LE_ADV_PARAM_INIT(_options, _int_min, _int_max, _peer)

Initialize advertising parameters.

Parameters

- **_options** Advertising Options
- _int_min Minimum advertising interval
- _int_max Maximum advertising interval
- _peer Peer address, set to NULL for undirected advertising or address of peer for directed advertising.

BT_LE_ADV_PARAM(_options, _int_min, _int_max, _peer)

Helper to declare advertising parameters inline.

Parameters

• **_options** – Advertising Options

- _int_min Minimum advertising interval
- _int_max Maximum advertising interval
- **_peer** Peer address, set to NULL for undirected advertising or address of peer for directed advertising.

BT_LE_ADV_CONN_DIR(_peer)

BT_LE_ADV_CONN

BT_LE_ADV_CONN_NAME

BT_LE_ADV_CONN_NAME_AD

BT_LE_ADV_CONN_DIR_LOW_DUTY(_peer)

BT_LE_ADV_NCONN

Non-connectable advertising with private address

BT_LE_ADV_NCONN_NAME

Non-connectable advertising with BT_LE_ADV_OPT_USE_NAME

BT_LE_ADV_NCONN_IDENTITY

Non-connectable advertising with BT LE ADV OPT USE IDENTITY

BT_LE_EXT_ADV_CONN_NAME

Connectable extended advertising with BT_LE_ADV_OPT_USE_NAME

BT_LE_EXT_ADV_SCAN_NAME

Scannable extended advertising with BT_LE_ADV_OPT_USE_NAME

BT_LE_EXT_ADV_NCONN

Non-connectable extended advertising with private address

BT_LE_EXT_ADV_NCONN_NAME

Non-connectable extended advertising with BT_LE_ADV_OPT_USE_NAME

BT_LE_EXT_ADV_NCONN_IDENTITY

Non-connectable extended advertising with BT_LE_ADV_OPT_USE_IDENTITY

BT_LE_EXT_ADV_CODED_NCONN

Non-connectable extended advertising on coded PHY with private address

BT_LE_EXT_ADV_CODED_NCONN_NAME

Non-connectable extended advertising on coded PHY with BT_LE_ADV_OPT_USE_NAME

BT_LE_EXT_ADV_CODED_NCONN_IDENTITY

Non-connectable extended advertising on coded PHY with BT_LE_ADV_OPT_USE_IDENTITY

BT_LE_EXT_ADV_START_PARAM_INIT(_timeout, _n_evts)

Helper to initialize extended advertising start parameters inline

Parameters

- _timeout Advertiser timeout
- _n_evts Number of advertising events

BT_LE_EXT_ADV_START_PARAM(_timeout, _n_evts)

Helper to declare extended advertising start parameters inline

Parameters

- _timeout Advertiser timeout
- _n_evts Number of advertising events

BT_LE_EXT_ADV_START_DEFAULT

BT_LE_PER_ADV_PARAM_INIT(_int_min, _int_max, _options)

Helper to declare periodic advertising parameters inline

Parameters

- _int_min Minimum periodic advertising interval
- _int_max Maximum periodic advertising interval
- **_options** Periodic advertising properties bitfield.

BT_LE_PER_ADV_PARAM(_int_min, _int_max, _options)

Helper to declare periodic advertising parameters inline

Parameters

- _int_min Minimum periodic advertising interval
- _int_max Maximum periodic advertising interval
- **_options** Periodic advertising properties bitfield.

BT_LE_PER_ADV_DEFAULT

BT_LE_SCAN_OPT_FILTER_WHITELIST

BT_LE_SCAN_PARAM_INIT(_type, _options, _interval, _window)

Initialize scan parameters.

Parameters

- _type Scan Type, BT_LE_SCAN_TYPE_ACTIVE or BT_LE_SCAN_TYPE_PASSIVE.
- **_options** Scan options
- **_interval** Scan Interval (N * 0.625 ms)
- **_window** Scan Window (N * 0.625 ms)

BT_LE_SCAN_PARAM(_type, _options, _interval, _window)

Helper to declare scan parameters inline.

Parameters

- _type Scan Type, BT_LE_SCAN_TYPE_ACTIVE or BT_LE_SCAN_TYPE_PASSIVE.
- **_options** Scan options
- _interval Scan Interval (N * 0.625 ms)
- _window Scan Window (N * 0.625 ms)

BT_LE_SCAN_ACTIVE

Helper macro to enable active scanning to discover new devices.

BT_LE_SCAN_PASSIVE

Helper macro to enable passive scanning to discover new devices.

This macro should be used if information required for device identification (e.g., UUID) are known to be placed in Advertising Data.

BT_LE_SCAN_CODED_ACTIVE

Helper macro to enable active scanning to discover new devices. Include scanning on Coded PHY in addition to 1M PHY.

BT_LE_SCAN_CODED_PASSIVE

Helper macro to enable passive scanning to discover new devices. Include scanning on Coded PHY in addition to 1M PHY.

This macro should be used if information required for device identification (e.g., UUID) are known to be placed in Advertising Data.

Typedefs

```
typedef void (*bt_ready_cb_t)(int err)
```

Callback for notifying that Bluetooth has been enabled.

Param err

zero on success or (negative) error code otherwise.

```
typedef void bt_le_scan_cb_t(const bt_addr_le_t *addr, int8_t rssi, uint8_t adv_type, struct net_buf_simple *buf)
```

Callback type for reporting LE scan results.

A function of this type is given to the $bt_le_scan_start()$ function and will be called for any discovered LE device.

Param addr

Advertiser LE address and type.

Param rssi

Strength of advertiser signal.

Param adv type

Type of advertising response from advertiser. Uses the BT_GAP_ADV_TYPE_* values.

Param buf

Buffer containing advertiser data.

```
typedef void bt_br_discovery_cb_t(struct bt_br_discovery_result *results, size_t count)
```

Callback type for reporting BR/EDR discovery (inquiry) results.

A callback of this type is given to the *bt_br_discovery_start()* function and will be called at the end of the discovery with information about found devices populated in the results array.

Param results

Storage used for discovery results

Param count

Number of valid discovery results.

Enums

enum [anonymous]

Advertising options

Values:

enumerator BT_LE_ADV_OPT_NONE

Convenience value when no options are specified.

enumerator BT_LE_ADV_OPT_CONNECTABLE

Advertise as connectable.

Advertise as connectable. If not connectable then the type of advertising is determined by providing scan response data. The advertiser address is determined by the type of advertising and/or enabling privacy @kconfig{CONFIG_BT_PRIVACY}.

enumerator BT_LE_ADV_OPT_ONE_TIME

Advertise one time.

Don't try to resume connectable advertising after a connection. This option is only meaningful when used together with BT_LE_ADV_OPT_CONNECTABLE. If set the advertising will be stopped when $bt_le_adv_stop()$ is called or when an incoming (peripheral) connection happens. If this option is not set the stack will take care of keeping advertising enabled even as connections occur. If Advertising directed or the advertiser was started with $bt_le_ext_adv_start$ then this behavior is the default behavior and this flag has no effect.

enumerator BT_LE_ADV_OPT_USE_IDENTITY

Advertise using identity address.

Advertise using the identity address as the advertiser address.

Note: The address used for advertising will not be the same as returned by $bt_le_oob_get_local$, instead bt_id_get should be used to get the LE address.

Warning: This will compromise the privacy of the device, so care must be taken when using this option.

enumerator BT_LE_ADV_OPT_USE_NAME

Advertise using GAP device name.

Include the GAP device name automatically when advertising. By default the GAP device name is put at the end of the scan response data.

When advertising using @ref BT_LE_ADV_OPT_EXT_ADV and not @ref BT_LE_ADV_OPT_SCANNABLE then it will be put at the end of the advertising data.

If the GAP device name does not fit into advertising data it will be converted to a shortened name if possible.

@ref BT_LE_ADV_OPT_FORCE_NAME_IN_AD can be used to force the device name to appear in the advertising data of an advert with scan response data.

The application can set the device name itself by including the following in the advertising data.

@code

BT_DATA(BT_DATA_NAME_COMPLETE, name, sizeof(name) - 1)
@endcode

enumerator BT_LE_ADV_OPT_DIR_MODE_LOW_DUTY

Low duty cycle directed advertising.

Use low duty directed advertising mode, otherwise high duty mode will be used.

enumerator BT_LE_ADV_OPT_DIR_ADDR_RPA

Directed advertising to privacy-enabled peer.

Enable use of Resolvable Private Address (RPA) as the target address in directed advertisements. This is required if the remote device is privacy-enabled and supports address resolution of the target address in directed advertisement. It is the responsibility of the application to check that the remote device supports address resolution of directed advertisements by reading its Central Address Resolution characteristic.

enumerator BT_LE_ADV_OPT_FILTER_SCAN_REQ

Use filter accept list to filter devices that can request scan response data.

enumerator BT_LE_ADV_OPT_FILTER_CONN

Use filter accept list to filter devices that can connect.

enumerator BT_LE_ADV_OPT_NOTIFY_SCAN_REQ

Notify the application when a scan response data has been sent to an active scanner.

enumerator BT_LE_ADV_OPT_SCANNABLE

Support scan response data.

When used together with $BT_LE_ADV_OPT_EXT_ADV$ then this option cannot be used together with the $BT_LE_ADV_OPT_CONNECTABLE$ option. When used together with $BT_LE_ADV_OPT_EXT_ADV$ then scan response data must be set.

enumerator BT_LE_ADV_OPT_EXT_ADV

Advertise with extended advertising.

This options enables extended advertising in the advertising set. In extended advertising the advertising set will send a small header packet on the three primary advertising channels. This small header points to the advertising data packet that will be sent on one of the 37 secondary advertising channels. The advertiser will send primary advertising on LE 1M PHY, and secondary advertising on LE 2M PHY. Connections will be established on LE 2M PHY.

Without this option the advertiser will send advertising data on the three primary advertising channels.

Note: Enabling this option requires extended advertising support in the peer devices scanning for advertisement packets.

enumerator BT_LE_ADV_OPT_NO_2M

Disable use of LE 2M PHY on the secondary advertising channel.

Disabling the use of LE 2M PHY could be necessary if scanners don't support the LE 2M PHY. The advertiser will send primary advertising on LE 1M PHY, and secondary advertising on LE 1M PHY. Connections will be established on LE 1M PHY.

Note: Cannot be set if BT_LE_ADV_OPT_CODED is set.

Note: Requires *BT_LE_ADV_OPT_EXT_ADV*.

enumerator BT_LE_ADV_OPT_CODED

Advertise on the LE Coded PHY (Long Range).

The advertiser will send both primary and secondary advertising on the LE Coded PHY. This gives the advertiser increased range with the trade-off of lower data rate and higher power consumption. Connections will be established on LE Coded PHY.

Note: Requires *BT_LE_ADV_OPT_EXT_ADV*

enumerator BT_LE_ADV_OPT_ANONYMOUS

Advertise without a device address (identity or RPA).

Note: Requires *BT_LE_ADV_OPT_EXT_ADV*

enumerator BT_LE_ADV_OPT_USE_TX_POWER

Advertise with transmit power.

Note: Requires *BT_LE_ADV_OPT_EXT_ADV*

enumerator BT_LE_ADV_OPT_DISABLE_CHAN_37

Disable advertising on channel index 37.

enumerator BT_LE_ADV_OPT_DISABLE_CHAN_38

Disable advertising on channel index 38.

enumerator BT_LE_ADV_OPT_DISABLE_CHAN_39

Disable advertising on channel index 39.

enumerator BT_LE_ADV_OPT_FORCE_NAME_IN_AD

Put GAP device name into advert data.

Will place the GAP device name into the advertising data rather than the scan response data.

Note: Requires *BT_LE_ADV_OPT_USE_NAME*

enum [anonymous]

Periodic Advertising options

Values:

enumerator BT_LE_PER_ADV_OPT_NONE

Convenience value when no options are specified.

enumerator BT_LE_PER_ADV_OPT_USE_TX_POWER

Advertise with transmit power.

Note: Requires *BT_LE_ADV_OPT_EXT_ADV*

enumerator BT_LE_PER_ADV_OPT_INCLUDE_ADI

Advertise with included AdvDataInfo (ADI).

Note: Requires *BT_LE_ADV_OPT_EXT_ADV*

enum [anonymous]

Periodic advertising sync options

Values:

enumerator BT_LE_PER_ADV_SYNC_OPT_NONE

Convenience value when no options are specified.

enumerator BT_LE_PER_ADV_SYNC_OPT_USE_PER_ADV_LIST

Use the periodic advertising list to sync with advertiser.

When this option is set, the address and SID of the parameters are ignored.

enumerator BT_LE_PER_ADV_SYNC_OPT_REPORTING_INITIALLY_DISABLED

Disables periodic advertising reports.

No advertisement reports will be handled until enabled.

enumerator BT_LE_PER_ADV_SYNC_OPT_FILTER_DUPLICATE

Filter duplicate Periodic Advertising reports

enumerator BT_LE_PER_ADV_SYNC_OPT_DONT_SYNC_AOA

Sync with Angle of Arrival (AoA) constant tone extension

enumerator BT_LE_PER_ADV_SYNC_OPT_DONT_SYNC_AOD_1US

Sync with Angle of Departure (AoD) 1 us constant tone extension

enumerator BT_LE_PER_ADV_SYNC_OPT_DONT_SYNC_AOD_2US

Sync with Angle of Departure (AoD) 2 us constant tone extension

enumerator BT_LE_PER_ADV_SYNC_OPT_SYNC_ONLY_CONST_TONE_EXT

Do not sync to packets without a constant tone extension

enum [anonymous]

Periodic Advertising Sync Transfer options

Values:

enumerator BT_LE_PER_ADV_SYNC_TRANSFER_OPT_NONE

Convenience value when no options are specified.

enumerator BT_LE_PER_ADV_SYNC_TRANSFER_OPT_SYNC_NO_AOA

No Angle of Arrival (AoA)

Do not sync with Angle of Arrival (AoA) constant tone extension

enumerator BT_LE_PER_ADV_SYNC_TRANSFER_OPT_SYNC_NO_AOD_1US

No Angle of Departure (AoD) 1 us.

Do not sync with Angle of Departure (AoD) 1 us constant tone extension

enumerator BT_LE_PER_ADV_SYNC_TRANSFER_OPT_SYNC_NO_AOD_2US

No Angle of Departure (AoD) 2.

Do not sync with Angle of Departure (AoD) 2 us constant tone extension

enumerator BT_LE_PER_ADV_SYNC_TRANSFER_OPT_SYNC_ONLY_CTE

Only sync to packets with constant tone extension

enum [anonymous]

Values:

enumerator BT_LE_SCAN_OPT_NONE

Convenience value when no options are specified.

enumerator BT_LE_SCAN_OPT_FILTER_DUPLICATE

Filter duplicates.

enumerator BT_LE_SCAN_OPT_FILTER_ACCEPT_LIST

Filter using filter accept list.

enumerator BT_LE_SCAN_OPT_CODED

Enable scan on coded PHY (Long Range).

enumerator BT_LE_SCAN_OPT_NO_1M

Disable scan on 1M phy.

Note: Requires *BT_LE_SCAN_OPT_CODED*.

enum [anonymous]

Values:

enumerator BT_LE_SCAN_TYPE_PASSIVE

Scan without requesting additional information from advertisers.

enumerator BT_LE_SCAN_TYPE_ACTIVE

Scan and request additional information from advertisers.

Using this scan type will automatically send scan requests to all devices. Scan responses are received in the same manner and using the same callbacks as advertising reports.

Functions

```
int bt_enable(bt_ready_cb_t cb)
```

Enable Bluetooth.

Enable Bluetooth. Must be the called before any calls that require communication with the local Bluetooth hardware.

When @kconfig{CONFIG_BT_SETTINGS} is enabled, the application must load the Bluetooth settings after this API call successfully completes before Bluetooth APIs can be used. Loading the settings before calling this function is insufficient. Bluetooth settings can be loaded with settings_load() or settings_load_subtree() with argument "bt". The latter selectively loads only Bluetooth settings and is recommended if settings_load() has been called earlier.

Parameters

• cb – Callback to notify completion or NULL to perform the enabling synchronously.

Returns

Zero on success or (negative) error code otherwise.

bool bt_is_ready(void)

Check if Bluetooth is ready.

Returns

true when Bluetooth is ready, false otherwise

int bt_set_name(const char *name)

Set Bluetooth Device Name.

Set Bluetooth GAP Device Name.

When advertising with device name in the advertising data the name should be updated by calling $bt_le_adv_update_data$ or $bt_le_ext_adv_set_data$.

Parameters

• name – New name

Returns

Zero on success or (negative) error code otherwise.

const char *bt_get_name(void)

Get Bluetooth Device Name.

Get Bluetooth GAP Device Name.

Returns

Bluetooth Device Name

uint16_t bt_get_appearance(void)

Get local Bluetooth appearance.

Bluetooth Appearance is a description of the external appearance of a device in terms of an Appearance Value.

See also:

https://specificationrefs.bluetooth.com/assigned-values/Appearance%20Values.pdf

Returns

Appearance Value of local Bluetooth host.

int bt_set_appearance(uint16_t new_appearance)

Set local Bluetooth appearance.

Automatically preserves the new appearance across reboots if @kconfig{CONFIG_BT_SETTINGS} is enabled.

 $This \ symbol \ is \ linkable \ if \ @kconfig\{CONFIG_BT_DEVICE_APPEARANCE_DYNAMIC\} \ is \ enabled.$

Parameters

• new_appearance - Appearance Value

Return values

- **0** Success.
- other Persistent storage failed. Appearance was not updated.

```
void bt_id_get(bt_addr_le_t *addrs, size_t *count)
```

Get the currently configured identities.

Returns an array of the currently configured identity addresses. To make sure all available identities can be retrieved, the number of elements in the *addrs* array should be CONFIG_BT_ID_MAX. The identity identifier that some APIs expect (such as advertising parameters) is simply the index of the identity in the *addrs* array.

If *addrs* is passed as NULL, then returned *count* contains the count of all available identities that can be retrieved with a subsequent call to this function with non-NULL *addrs* parameter.

Note: Deleted identities may show up as *BT_ADDR_LE_ANY* in the returned array.

Parameters

- addrs Array where to store the configured identities.
- **count** Should be initialized to the array size. Once the function returns it will contain the number of returned identities.

```
int bt_id_create(bt_addr_le_t *addr, uint8_t *irk)
```

Create a new identity.

Create a new identity using the given address and IRK. This function can be called before calling $bt_enable()$, in which case it can be used to override the controller's public address (in case it has one). However, the new identity will only be stored persistently in flash when this API is used after $bt_enable()$. The reason is that the persistent settings are loaded after $bt_enable()$ and would therefore cause potential conflicts with the stack blindly overwriting what's stored in flash. The identity will also not be written to flash in case a pre-defined address is provided, since in such a situation the app clearly has some place it got the address from and will be able to repeat the procedure on every power cycle, i.e. it would be redundant to also store the information in flash.

Generating random static address or random IRK is not supported when calling this function before *bt_enable()*.

If the application wants to have the stack randomly generate identities and store them in flash for later recovery, the way to do it would be to first initialize the stack (using bt_enable), then call settings_load(), and after that check with $bt_id_get()$ how many identities were recovered. If an insufficient amount of identities were recovered the app may then call $bt_id_create()$ to create new ones.

Parameters

- addr Address to use for the new identity. If NULL or initialized to BT_ADDR_LE_ANY the stack will generate a new random static address for the identity and copy it to the given parameter upon return from this function (in case the parameter was non-NULL).
- **irk** Identity Resolving Key (16 bytes) to be used with this identity. If set to all zeroes or NULL, the stack will generate a random IRK for the identity and copy it back to the parameter upon return from this function (in case the parameter was non-NULL). If privacy @kconfig{CONFIG_BT_PRIVACY} is not enabled this parameter must be NULL.

Returns

Identity identifier (>= 0) in case of success, or a negative error code on failure.

```
int bt_id_reset(uint8_t id, bt_addr_le_t *addr, uint8_t *irk)
```

Reset/reclaim an identity for reuse.

The semantics of the addr and irk parameters of this function are the same as with $bt_id_create()$. The difference is the first id parameter that needs to be an existing identity (if it doesn't exist this function will return an error). When given an existing identity this function will disconnect any connections created using it, remove any pairing keys or other data associated with it, and then create a new identity in the same slot, based on the addr and irk parameters.

Note: the default identity (BT_ID_DEFAULT) cannot be reset, i.e. this API will return an error if asked to do that.

Parameters

- **id** Existing identity identifier.
- addr Address to use for the new identity. If NULL or initialized to BT_ADDR_LE_ANY the stack will generate a new static random address for the identity and copy it to the given parameter upon return from this function (in case the parameter was non-NULL).
- **irk** Identity Resolving Key (16 bytes) to be used with this identity. If set to all zeroes or NULL, the stack will generate a random IRK for the identity and copy it back to the parameter upon return from this function (in case the parameter was non-NULL). If privacy @kconfig{CONFIG_BT_PRIVACY} is not enabled this parameter must be NULL.

Returns

Identity identifier (\geq 0) in case of success, or a negative error code on failure.

int **bt_id_delete**(uint8 t id)

Delete an identity.

When given a valid identity this function will disconnect any connections created using it, remove any pairing keys or other data associated with it, and then flag is as deleted, so that it can not be used for any operations. To take back into use the slot the identity was occupying the *bt_id_reset()* API needs to be used.

Note: the default identity (BT_ID_DEFAULT) cannot be deleted, i.e. this API will return an error if asked to do that.

Parameters

• **id** – Existing identity identifier.

Returns

0 in case of success, or a negative error code on failure.

int **bt_le_adv_start**(const struct *bt_le_adv_param* *param, const struct *bt_data* *ad, size_t ad_len, const struct *bt_data* *sd, size_t sd_len)

Start advertising.

Set advertisement data, scan response data, advertisement parameters and start advertising.

When the advertisement parameter peer address has been set the advertising will be directed to the peer. In this case advertisement data and scan response data parameters are ignored. If the mode is high duty cycle the timeout will be $BT_GAP_ADV_HIGH_DUTY_CYCLE_MAX_TIMEOUT$.

Parameters

• **param** – Advertising parameters.

- ad Data to be used in advertisement packets.
- ad_len Number of elements in ad
- sd Data to be used in scan response packets.
- sd_len Number of elements in sd

Returns

Zero on success or (negative) error code otherwise.

Returns

-ENOMEM No free connection objects available for connectable advertiser.

Returns

-ECONNREFUSED When connectable advertising is requested and there is already maximum number of connections established in the controller. This error code is only guaranteed when using Zephyr controller, for other controllers code returned in this case may be -EIO.

int **bt_le_adv_update_data**(const struct *bt_data* *ad, size_t ad_len, const struct *bt_data* *sd, size_t sd_len) Update advertising.

Update advertisement and scan response data.

Parameters

- ad Data to be used in advertisement packets.
- ad_len Number of elements in ad
- sd Data to be used in scan response packets.
- sd_len Number of elements in sd

Returns

Zero on success or (negative) error code otherwise.

int bt_le_adv_stop(void)

Stop advertising.

Stops ongoing advertising.

Returns

Zero on success or (negative) error code otherwise.

int **bt_le_ext_adv_create**(const struct *bt_le_adv_param* *param, const struct *bt_le_ext_adv_cb* *cb, struct bt_le_ext_adv **adv)

Create advertising set.

Create a new advertising set and set advertising parameters. Advertising parameters can be updated with $bt_le_ext_adv_update_param$.

Parameters

- param [in] Advertising parameters.
- **cb [in]** Callback struct to notify about advertiser activity. Can be NULL. Must point to valid memory during the lifetime of the advertising set.
- adv [out] Valid advertising set object on success.

Returns

Zero on success or (negative) error code otherwise.

int **bt_le_ext_adv_start**(struct bt_le_ext_adv *adv, struct bt_le_ext_adv_start_param *param)

Start advertising with the given advertising set.

If the advertiser is limited by either the timeout or number of advertising events the application will be notified by the advertiser sent callback once the limit is reached. If the advertiser is limited by both the timeout and the number of advertising events then the limit that is reached first will stop the advertiser.

Parameters

- adv Advertising set object.
- param Advertise start parameters.

int bt_le_ext_adv_stop(struct bt_le_ext_adv *adv)

Stop advertising with the given advertising set.

Stop advertising with a specific advertising set. When using this function the advertising sent callback will not be called.

Parameters

• adv – Advertising set object.

Returns

Zero on success or (negative) error code otherwise.

int **bt_le_ext_adv_set_data**(struct bt_le_ext_adv *adv, const struct *bt_data* *ad, size_t ad_len, const struct *bt_data* *sd, size_t ad_len, const struct *bt_data* *sd, size_t ad_len)

Set an advertising set's advertising or scan response data.

Set advertisement data or scan response data. If the advertising set is currently advertising then the advertising data will be updated in subsequent advertising events.

When both BT_LE_ADV_OPT_EXT_ADV and BT_LE_ADV_OPT_SCANNABLE are enabled then advertising data is ignored. When BT_LE_ADV_OPT_SCANNABLE is not enabled then scan response data is ignored.

If the advertising set has been configured to send advertising data on the primary advertising channels then the maximum data length is $BT_GAP_ADV_MAX_ADV_DATA_LEN$ bytes. If the advertising set has been configured for extended advertising, then the maximum data length is defined by the controller with the maximum possible of $BT_GAP_ADV_MAX_EXT_ADV_DATA_LEN$ bytes.

Note: Not all scanners support extended data length advertising data.

Note: When updating the advertising data while advertising the advertising data and scan response data length must be smaller or equal to what can be fit in a single advertising packet. Otherwise the advertiser must be stopped.

Parameters

- **adv** Advertising set object.
- ad Data to be used in advertisement packets.
- ad_len Number of elements in ad
- sd Data to be used in scan response packets.
- sd_len Number of elements in sd

Returns

Zero on success or (negative) error code otherwise.

int **bt_le_ext_adv_update_param**(struct bt_le_ext_adv *adv, const struct bt_le_adv_param *param)

Update advertising parameters.

Update the advertising parameters. The function will return an error if the advertiser set is currently advertising. Stop the advertising set before calling this function.

Note: When changing the option $BT_LE_ADV_OPT_USE_NAME$ then $bt_le_ext_adv_set_data$ needs to be called in order to update the advertising data and scan response data.

Parameters

- adv Advertising set object.
- **param** Advertising parameters.

Returns

Zero on success or (negative) error code otherwise.

int bt_le_ext_adv_delete(struct bt le ext adv *adv)

Delete advertising set.

Delete advertising set. This will free up the advertising set and make it possible to create a new advertising set.

Returns

Zero on success or (negative) error code otherwise.

uint8_t bt_le_ext_adv_get_index(struct bt_le_ext_adv *adv)

Get array index of an advertising set.

This function is used to map bt_adv to index of an array of advertising sets. The array has CON-FIG_BT_EXT_ADV_MAX_ADV_SET elements.

Parameters

• adv – Advertising set.

Returns

Index of the advertising set object. The range of the returned value is 0..CONFIG_BT_EXT_ADV_MAX_ADV_SET-1

int **bt_le_ext_adv_get_info** (const struct bt_le_ext_adv *adv, struct bt_le_ext_adv_info *info)

Get advertising set info.

Parameters

- adv Advertising set object
- **info** Advertising set info object

Returns

Zero on success or (negative) error code on failure.

int **bt_le_per_adv_set_param**(struct bt_le_ext_adv *adv, const struct bt_le_per_adv_param *param)

Set or update the periodic advertising parameters.

The periodic advertising parameters can only be set or updated on an extended advertisement set which is neither scannable, connectable nor anonymous.

Parameters

- adv Advertising set object.
- param Advertising parameters.

Returns

Zero on success or (negative) error code otherwise.

int bt_le_per_adv_set_data(const struct bt_le_ext_adv *adv, const struct bt_data *ad, size_t ad_len)

Set or update the periodic advertising data.

The periodic advertisement data can only be set or updated on an extended advertisement set which is neither scannable, connectable nor anonymous.

Parameters

- **adv** Advertising set object.
- ad Advertising data.
- ad_len Advertising data length.

Returns

Zero on success or (negative) error code otherwise.

```
int bt_le_per_adv_start(struct bt_le_ext_adv *adv)
```

Starts periodic advertising.

Enabling the periodic advertising can be done independently of extended advertising, but both periodic advertising and extended advertising shall be enabled before any periodic advertising data is sent. The periodic advertising and extended advertising can be enabled in any order.

Once periodic advertising has been enabled, it will continue advertising until $bt_le_per_adv_stop()$ has been called, or if the advertising set is deleted by $bt_le_ext_adv_delete()$. Calling $bt_le_ext_adv_stop()$ will not stop the periodic advertising.

Parameters

• **adv** – Advertising set object.

Returns

Zero on success or (negative) error code otherwise.

```
int bt_le_per_adv_stop(struct bt_le_ext_adv *adv)
```

Stops periodic advertising.

Disabling the periodic advertising can be done independently of extended advertising. Disabling periodic advertising will not disable extended advertising.

Parameters

• adv – Advertising set object.

Returns

Zero on success or (negative) error code otherwise.

uint8_t bt_le_per_adv_sync_get_index(struct bt_le_per_adv_sync *per_adv_sync)

Get array index of an periodic advertising sync object.

This function is get the index of an array of periodic advertising sync objects. The array has CON-FIG_BT_PER_ADV_SYNC_MAX elements.

Parameters

• per_adv_sync - The periodic advertising sync object.

Returns

Index of the periodic advertising sync object. The range of the returned value is 0..CONFIG_BT_PER_ADV_SYNC_MAX-1

Get periodic adv sync information.

Parameters

- **per_adv_sync** Periodic advertising sync object.
- info Periodic advertising sync info object

Returns

Zero on success or (negative) error code on failure.

Look up an existing periodic advertising sync object by advertiser address.

Parameters

- adv_addr Advertiser address.
- **sid** The advertising set ID.

Returns

Periodic advertising sync object or NULL if not found.

Create a periodic advertising sync object.

Create a periodic advertising sync object that can try to synchronize to periodic advertising reports from an advertiser. Scan shall either be disabled or extended scan shall be enabled.

Parameters

- param [in] Periodic advertising sync parameters.
- out_sync [out] Periodic advertising sync object on.

Returns

Zero on success or (negative) error code otherwise.

int bt_le_per_adv_sync_delete(struct bt_le_per_adv_sync *per_adv_sync)

Delete periodic advertising sync.

Delete the periodic advertising sync object. Can be called regardless of the state of the sync. If the syncing is currently syncing, the syncing is cancelled. If the sync has been established, it is terminated. The periodic advertising sync object will be invalidated afterwards.

If the state of the sync object is syncing, then a new periodic advertising sync object may not be created until the controller has finished canceling this object.

Parameters

• **per_adv_sync** – The periodic advertising sync object.

Returns

Zero on success or (negative) error code otherwise.

void bt_le_per_adv_sync_cb_register(struct bt_le_per_adv_sync_cb *cb)

Register periodic advertising sync callbacks.

Adds the callback structure to the list of callback structures for periodic advertising syncs.

This callback will be called for all periodic advertising sync activity, such as synced, terminated and when data is received.

Parameters

• **cb** – Callback struct. Must point to memory that remains valid.

int bt_le_per_adv_sync_recv_enable(struct bt_le_per_adv_sync *per_adv_sync)

Enables receiving periodic advertising reports for a sync.

If the sync is already receiving the reports, -EALREADY is returned.

Parameters

• **per_adv_sync** – The periodic advertising sync object.

Returns

Zero on success or (negative) error code otherwise.

int bt_le_per_adv_sync_recv_disable(struct bt_le_per_adv_sync *per_adv_sync)

Disables receiving periodic advertising reports for a sync.

If the sync report receiving is already disabled, -EALREADY is returned.

Parameters

• **per_adv_sync** – The periodic advertising sync object.

Returns

Zero on success or (negative) error code otherwise.

int **bt_le_per_adv_sync_transfer**(const struct bt_le_per_adv_sync *per_adv_sync, const struct bt_conn *conn, uint16 t service data)

Transfer the periodic advertising sync information to a peer device.

This will allow another device to quickly synchronize to the same periodic advertising train that this device is currently synced to.

Parameters

- **per_adv_sync** The periodic advertising sync to transfer.
- **conn** The peer device that will receive the sync information.
- **service_data** Application service data provided to the remote host.

Returns

Zero on success or (negative) error code otherwise.

int **bt_le_per_adv_set_info_transfer**(const struct bt_le_ext_adv *adv, const struct bt_conn *conn, uint16_t service_data)

Transfer the information about a periodic advertising set.

This will allow another device to quickly synchronize to periodic advertising set from this device.

Parameters

- adv The periodic advertising set to transfer info of.
- **conn** The peer device that will receive the information.
- **service_data** Application service data provided to the remote host.

Returns

Zero on success or (negative) error code otherwise.

```
int bt_le_per_adv_sync_transfer_subscribe(const struct bt_conn *conn, const struct bt_le_per_adv_sync_transfer_param *param)
```

Subscribe to periodic advertising sync transfers (PASTs).

Sets the parameters and allow other devices to transfer periodic advertising syncs.

Parameters

- **conn** The connection to set the parameters for. If NULL default parameters for all connections will be set. Parameters set for specific connection will always have precedence.
- param The periodic advertising sync transfer parameters.

Returns

Zero on success or (negative) error code otherwise.

int bt_le_per_adv_sync_transfer_unsubscribe(const struct bt conn *conn)

Unsubscribe from periodic advertising sync transfers (PASTs).

Remove the parameters that allow other devices to transfer periodic advertising syncs.

Parameters

• **conn** – The connection to remove the parameters for. If NULL default parameters for all connections will be removed. Unsubscribing for a specific device, will still allow other devices to transfer periodic advertising syncs.

Returns

Zero on success or (negative) error code otherwise.

```
int bt_le_per_adv_list_add(const bt_addr_le_t *addr, uint8_t sid)
```

Add a device to the periodic advertising list.

Add peer device LE address to the periodic advertising list. This will make it possibly to automatically create a periodic advertising sync to this device.

Parameters

- **addr** Bluetooth LE identity address.
- **sid** The advertising set ID. This value is obtained from the *bt_le_scan_recv_info* in the scan callback.

Returns

Zero on success or (negative) error code otherwise.

int **bt_le_per_adv_list_remove**(const *bt_addr_le_t* *addr, uint8_t sid)

Remove a device from the periodic advertising list.

Removes peer device LE address from the periodic advertising list.

Parameters

- addr Bluetooth LE identity address.
- **sid** The advertising set ID. This value is obtained from the *bt_le_scan_recv_info* in the scan callback.

Returns

Zero on success or (negative) error code otherwise.

int bt_le_per_adv_list_clear(void)

Clear the periodic advertising list.

Clears the entire periodic advertising list.

Returns

Zero on success or (negative) error code otherwise.

```
int bt_le_scan_start(const struct bt_le_scan_param *param, bt_le_scan_cb_t cb)
```

Start (LE) scanning.

Start LE scanning with given parameters and provide results through the specified callback.

Note: The LE scanner by default does not use the Identity Address of the local device when @kconfig{CONFIG_BT_PRIVACY} is disabled. This is to prevent the active scanner from disclosing the identity information when requesting additional information from advertisers. In order to enable directed advertiser reports then @kconfig{CONFIG_BT_SCAN_WITH_IDENTITY} must be enabled.

Parameters

- param Scan parameters.
- cb Callback to notify scan results. May be NULL if callback registration through bt_le_scan_cb_register is preferred.

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

int bt_le_scan_stop(void)

Stop (LE) scanning.

Stops ongoing LE scanning.

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

void bt_le_scan_cb_register(struct bt_le_scan_cb *cb)

Register scanner packet callbacks.

Adds the callback structure to the list of callback structures that monitors scanner activity.

This callback will be called for all scanner activity, regardless of what API was used to start the scanner.

Parameters

• **cb** – Callback struct. Must point to memory that remains valid.

void bt_le_scan_cb_unregister(struct bt_le_scan_cb *cb)

Unregister scanner packet callbacks.

Remove the callback structure from the list of scanner callbacks.

Parameters

• cb – Callback struct. Must point to memory that remains valid.

```
int bt_le_filter_accept_list_add(const bt_addr_le_t *addr)
```

Add device (LE) to filter accept list.

Add peer device LE address to the filter accept list.

Note: The filter accept list cannot be modified when an LE role is using the filter accept list, i.e advertiser or scanner using a filter accept list or automatic connecting to devices using filter accept list.

Parameters

• addr – Bluetooth LE identity address.

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

static inline int **bt_le_whitelist_add**(const *bt_addr_le_t* *addr)

int bt_le_filter_accept_list_remove(const bt_addr_le_t *addr)

Remove device (LE) from filter accept list.

Remove peer device LE address from the filter accept list.

Note: The filter accept list cannot be modified when an LE role is using the filter accept list, i.e advertiser or scanner using a filter accept list or automatic connecting to devices using filter accept list.

Parameters

• addr – Bluetooth LE identity address.

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

static inline int **bt_le_whitelist_rem**(const *bt_addr_le_t* *addr)

int bt_le_filter_accept_list_clear(void)

Clear filter accept list.

Clear all devices from the filter accept list.

Note: The filter accept list cannot be modified when an LE role is using the filter accept list, i.e advertiser or scanner using a filter accept list or automatic connecting to devices using filter accept list.

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

static inline int bt_le_whitelist_clear(void)

int bt_le_set_chan_map(uint8_t chan_map[5])

Set (LE) channel map.

Parameters

• **chan_map** – Channel map.

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

int bt_le_set_rpa_timeout(uint16_t new_rpa_timeout)

Set the Resolvable Private Address timeout in runtime.

The new RPA timeout value will be used for the next RPA rotation and all subsequent rotations until another override is scheduled with this API.

Initially, the if @kconfig{CONFIG BT RPA TIMEOUT} is used as the RPA timeout.

This symbol is linkable if @kconfig{CONFIG_BT_RPA_TIMEOUT_DYNAMIC} is enabled.

Parameters

• new_rpa_timeout - Resolvable Private Address timeout in seconds

Return values

- 0 Success.
- **-EINVAL** RPA timeout value is invalid. Valid range is 1s 3600s.

void **bt_data_parse**(struct net_buf_simple *ad, bool (*func)(struct *bt_data* *data, void *user_data), void *user_data)

Helper for parsing advertising (or EIR or OOB) data.

A helper for parsing the basic data types used for Extended Inquiry Response (EIR), Advertising Data (AD), and OOB data blocks. The most common scenario is to call this helper on the advertising data received in the callback that was given to $bt_le_scan_start()$.

Warning: This helper function will consume ad when parsing. The user should make a copy if the original data is to be used afterwards

Parameters

- **ad** Advertising data as given to the bt_le_scan_cb_t callback.
- **func** Callback function which will be called for each element that's found in the data. The callback should return true to continue parsing, or false to stop parsing.
- **user_data** User data to be passed to the callback.

int **bt_le_oob_get_local**(uint8_t id, struct *bt_le_oob* *oob)

Get local LE Out of Band (OOB) information.

This function allows to get local information that are useful for Out of Band pairing or connection creation.

If privacy @kconfig{CONFIG_BT_PRIVACY} is enabled this will result in generating new Resolvable Private Address (RPA) that is valid for @kconfig{CONFIG_BT_RPA_TIMEOUT} seconds. This address will be used for advertising started by $bt_le_adv_start$, active scanning and connection creation.

Note: If privacy is enabled the RPA cannot be refreshed in the following cases:

- Creating a connection in progress, wait for the connected callback. In addition when extended advertising @kconfig{CONFIG_BT_EXT_ADV} is not enabled or not supported by the controller:
- Advertiser is enabled using a Random Static Identity Address for a different local identity.
- The local identity conflicts with the local identity used by other roles.

Parameters

- id [in] Local identity, in most cases BT_ID_DEFAULT.
- oob [out] LE OOB information

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

int **bt_le_ext_adv_oob_get_local**(struct bt_le_ext_adv *adv, struct bt_le_oob *oob)

Get local LE Out of Band (OOB) information.

This function allows to get local information that are useful for Out of Band pairing or connection creation.

If privacy @kconfig{CONFIG_BT_PRIVACY} is enabled this will result in generating new Resolvable Private Address (RPA) that is valid for @kconfig{CONFIG_BT_RPA_TIMEOUT} seconds. This address will be used by the advertising set.

Note: When generating OOB information for multiple advertising set all OOB information needs to be generated at the same time.

Note: If privacy is enabled the RPA cannot be refreshed in the following cases:

• Creating a connection in progress, wait for the connected callback.

Parameters

- adv [in] The advertising set object
- oob [out] LE OOB information

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

int **bt_br_discovery_start**(const struct *bt_br_discovery_param* *param, struct *bt_br_discovery_result* *results, size t count, *bt_br_discovery_cb_t* cb)

Start BR/EDR discovery.

Start BR/EDR discovery (inquiry) and provide results through the specified callback. When bt_br_discovery_cb_t is called it indicates that discovery has completed. If more inquiry results were received during session than fits in provided result storage, only ones with highest RSSI will be reported.

Parameters

- param Discovery parameters.
- results Storage for discovery results.
- count Number of results in storage. Valid range: 1-255.
- **cb** Callback to notify discovery results.

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error

int bt_br_discovery_stop(void)

Stop BR/EDR discovery.

Stops ongoing BR/EDR discovery. If discovery was stopped by this call results won't be reported

Returns

Zero on success or error code otherwise, positive in case of protocol error or negative (POSIX) in case of stack internal error.

int bt_br_oob_get_local(struct bt br oob *oob)

Get BR/EDR local Out Of Band information.

This function allows to get local controller information that are useful for Out Of Band pairing or connection creation process.

Parameters

• oob – Out Of Band information

int bt_br_set_discoverable(bool enable)

Enable/disable set controller in discoverable state.

Allows make local controller to listen on INQUIRY SCAN channel and responds to devices making general inquiry. To enable this state it's mandatory to first be in connectable state.

Parameters

• **enable** – Value allowing/disallowing controller to become discoverable.

Returns

Negative if fail set to requested state or requested state has been already set. Zero if done successfully.

int bt_br_set_connectable(bool enable)

Enable/disable set controller in connectable state.

Allows make local controller to be connectable. It means the controller start listen to devices requests on PAGE SCAN channel. If disabled also resets discoverability if was set.

Parameters

• **enable** – Value allowing/disallowing controller to be connectable.

Returns

Negative if fail set to requested state or requested state has been already set. Zero if done successfully.

int **bt_unpair**(uint8_t id, const *bt_addr_le_t* *addr)

Clear pairing information.

Parameters

- id Local identity (mostly just BT ID DEFAULT).
- addr Remote address, NULL or BT_ADDR_LE_ANY to clear all remote devices.

Returns

0 on success or negative error value on failure.

void **bt_foreach_bond**(uint8_t id, void (*func)(const struct *bt_bond_info* *info, void *user_data), void *user_data)

Iterate through all existing bonds.

Parameters

- id Local identity (mostly just BT_ID_DEFAULT).
- **func** Function to call for each bond.
- **user_data** Data to pass to the callback function.

int **bt_configure_data_path**(uint8_t dir, uint8_t id, uint8_t vs_config_len, const uint8_t *vs_config) Configure vendor data path.

Request the Controller to configure the data transport path in a given direction between the Controller and the Host.

Parameters

- **dir** Direction to be configured, BT_HCI_DATAPATH_DIR_HOST_TO_CTLR or BT_HCI_DATAPATH_DIR_CTLR_TO_HOST
- id Vendor specific logical transport channel ID, range [BT_HCI_DATAPATH_ID_VS..BT_HCI_DATAPATH_ID_VS_END]
- vs_config_len Length of additional vendor specific configuration data
- vs_config Pointer to additional vendor specific configuration data

Returns

0 in case of success or negative value in case of error.

struct bt_le_ext_adv_sent_info

#include <bluetooth.h>

Public Members

uint8_t num_sent

The number of advertising events completed.

struct bt_le_ext_adv_connected_info

#include <bluetooth.h>

Public Members

```
struct bt_conn *conn
```

Connection object of the new connection

struct bt_le_ext_adv_scanned_info

#include <bluetooth.h>

Public Members

```
bt\_addr\_le\_t *addr
```

Active scanner LE address and type

struct bt_le_ext_adv_cb

#include <bluetooth.h>

Public Members

```
void (*sent)(struct bt_le_ext_adv *adv, struct bt_le_ext_adv_sent_info *info)
```

The advertising set has finished sending adv data.

This callback notifies the application that the advertising set has finished sending advertising data. The advertising set can either have been stopped by a timeout or because the specified number of advertising events has been reached.

Param adv

The advertising set object.

Param info

Information about the sent event.

```
void (*connected)(struct bt_le_ext_adv *adv, struct bt_le_ext_adv_connected_info *info)
```

The advertising set has accepted a new connection.

This callback notifies the application that the advertising set has accepted a new connection.

Param adv

The advertising set object.

Param info

Information about the connected event.

void (*scanned)(struct bt_le_ext_adv *adv, struct bt_le_ext_adv_scanned_info *info)

The advertising set has sent scan response data.

This callback notifies the application that the advertising set has has received a Scan Request packet, and has sent a Scan Response packet.

Param adv

The advertising set object.

Param addr

Information about the scanned event.

struct bt_data

#include <bluetooth.h> Bluetooth data.

Description of different data types that can be encoded into advertising data. Used to form arrays that are passed to the $bt_le_adv_start()$ function.

struct bt_le_adv_param

#include <bluetooth.h> LE Advertising Parameters.

Public Members

uint8 t id

Local identity.

Note: When extended advertising @kconfig{CONFIG_BT_EXT_ADV} is not enabled or not supported by the controller it is not possible to scan and advertise simultaneously using two different random addresses.

uint8_t sid

Advertising Set Identifier, valid range 0x00 - 0x0f.

Note: Requires *BT_LE_ADV_OPT_EXT_ADV*

uint8 t secondary_max_skip

Secondary channel maximum skip count.

Maximum advertising events the advertiser can skip before it must send advertising data on the secondary advertising channel.

Note: Requires *BT_LE_ADV_OPT_EXT_ADV*

uint32_t options

Bit-field of advertising options

uint32_t interval_min

Minimum Advertising Interval (N * 0.625 milliseconds) Minimum Advertising Interval shall be less than or equal to the Maximum Advertising Interval. The Minimum Advertising Interval and Maximum Advertising Interval should not be the same value (as stated in Bluetooth Core Spec 5.2, section 7.8.5) Range: 0x0020 to 0x4000

uint32_t interval_max

Maximum Advertising Interval (N * 0.625 milliseconds) Minimum Advertising Interval shall be less than or equal to the Maximum Advertising Interval. The Minimum Advertising Interval and Maximum Advertising Interval should not be the same value (as stated in Bluetooth Core Spec 5.2, section 7.8.5) Range: 0x0020 to 0x4000

const bt_addr_le_t *peer

Directed advertising to peer.

When this parameter is set the advertiser will send directed advertising to the remote device.

The advertising type will either be high duty cycle, or low duty cycle if the BT_LE_ADV_OPT_DIR_MODE_LOW_DUTY option is enabled. When using BT_LE_ADV_OPT_EXT_ADV then only low duty cycle is allowed.

In case of connectable high duty cycle if the connection could not be established within the timeout the connected() callback will be called with the status set to BT_HCI_ERR_ADV_TIMEOUT.

struct bt_le_per_adv_param

#include <bluetooth.h>

Public Members

uint16_t interval_min

Minimum Periodic Advertising Interval (N * 1.25 ms)

Shall be greater or equal to BT_GAP_PER_ADV_MIN_INTERVAL and less or equal to interval_max.

uint16_t interval_max

Maximum Periodic Advertising Interval (N * 1.25 ms)

Shall be less or equal to BT_GAP_PER_ADV_MAX_INTERVAL and greater or equal to interval_min.

uint32_t options

Bit-field of periodic advertising options

struct bt_le_ext_adv_start_param

#include <bluetooth.h>

Public Members

uint16_t timeout

Advertiser timeout (N * 10 ms).

Application will be notified by the advertiser sent callback. Set to zero for no timeout.

When using high duty cycle directed connectable advertising then this parameters must be set to a non-zero value less than or equal to the maximum of BT_GAP_ADV_HIGH_DUTY_CYCLE_MAX_TIMEOUT.

If privacy @kconfig{CONFIG_BT_PRIVACY} is enabled then the timeout must be less than @kconfig{CONFIG_BT_RPA_TIMEOUT}.

uint8_t num_events

Number of advertising events.

Application will be notified by the advertiser sent callback. Set to zero for no limit.

struct bt_le_ext_adv_info

#include <bluetooth.h> Advertising set info structure.

Public Members

int8_t tx_power

Currently selected Transmit Power (dBM).

struct bt_le_per_adv_sync_synced_info

#include <bluetooth.h>

Public Members

```
const bt_addr_le_t *addr
```

Advertiser LE address and type.

uint8_t sid

Advertiser SID

uint16_t interval

Periodic advertising interval (N * 1.25 ms)

uint8_t phy

Advertiser PHY

bool recv_enabled

True if receiving periodic advertisements, false otherwise.

uint16_t service_data

```
Service Data provided by the peer when sync is transferred.
         Will always be 0 when the sync is locally created.
     struct bt_conn *conn
         Peer that transferred the periodic advertising sync.
         Will always be 0 when the sync is locally created.
struct bt_le_per_adv_sync_term_info
     #include <bluetooth.h>
     Public Members
     const bt_addr_le_t *addr
         Advertiser LE address and type.
     uint8\_t sid
         Advertiser SID
     uint8 t reason
         Cause of periodic advertising termination
struct bt_le_per_adv_sync_recv_info
     #include <bluetooth.h>
     Public Members
     const bt_addr_le_t *addr
         Advertiser LE address and type.
     uint8_t sid
         Advertiser SID
     int8_t tx_power
         The TX power of the advertisement.
     int8_t rssi
         The RSSI of the advertisement excluding any CTE.
     uint8_t cte_type
         The Constant Tone Extension (CTE) of the advertisement (bt_df_cte_type)
struct bt_le_per_adv_sync_state_info
     #include <bluetooth.h>
```

Public Members

bool recv_enabled

True if receiving periodic advertisements, false otherwise.

struct bt_le_per_adv_sync_cb

#include <bluetooth.h>

Public Members

void (*synced)(struct bt_le_per_adv_sync *sync, struct bt_le_per_adv_sync_synced_info *info)

The periodic advertising has been successfully synced.

This callback notifies the application that the periodic advertising set has been successfully synced, and will now start to receive periodic advertising reports.

Param sync

The periodic advertising sync object.

Param info

Information about the sync event.

void (*term)(struct bt_le_per_adv_sync *sync, const struct bt_le_per_adv_sync_term_info *info)

The periodic advertising sync has been terminated.

This callback notifies the application that the periodic advertising sync has been terminated, either by local request, remote request or because due to missing data, e.g. by being out of range or sync.

Param sync

The periodic advertising sync object.

void (***recv**)(struct bt_le_per_adv_sync *sync, const struct bt_le_per_adv_sync_recv_info *info, struct net_buf_simple *buf)

Periodic advertising data received.

This callback notifies the application of an periodic advertising report.

Param sync

The advertising set object.

Param info

Information about the periodic advertising event.

Param buf

Buffer containing the periodic advertising data.

void (*state_changed)(struct bt_le_per_adv_sync *sync, const struct bt_le_per_adv_sync_state_info
*info)

The periodic advertising sync state has changed.

This callback notifies the application about changes to the sync state. Initialize sync and termination is handled by their individual callbacks, and won't be notified here.

Param sync

The periodic advertising sync object.

Param info

Information about the state change.

```
void (*biginfo)(struct bt_le_per_adv_sync *sync, const struct bt_iso_biginfo *biginfo)
```

BIGInfo advertising report received.

This callback notifies the application of a BIGInfo advertising report. This is received if the advertiser is broadcasting isochronous streams in a BIG. See iso.h for more information.

Param sync

The advertising set object.

Param biginfo

The BIGInfo report.

```
void (*cte_report_cb)(struct bt_le_per_adv_sync *sync, struct
bt_df_per_adv_sync_iq_samples_report const *info)
```

Callback for IQ samples report collected when sampling CTE received with periodic advertising PDU.

Param sync

The periodic advertising sync object.

Param info

Information about the sync event.

struct bt_le_per_adv_sync_param

#include <bluetooth.h>

Public Members

bt addr le t addr

Periodic Advertiser Address.

Only valid if not using the periodic advertising list (BT_LE_PER_ADV_SYNC_OPT_USE_PER_ADV_LIST)

uint8_t sid

Advertiser SID.

Only valid if not using the periodic advertising list (BT_LE_PER_ADV_SYNC_OPT_USE_PER_ADV_LIST)

uint32_t options

Bit-field of periodic advertising sync options.

uint16 t skip

Maximum event skip.

Maximum number of periodic advertising events that can be skipped after a successful receive. Range: 0x0000 to 0x01F3

uint16 t timeout

Synchronization timeout (N * 10 ms)

Synchronization timeout for the periodic advertising sync. Range 0x000A to 0x4000 (100 ms to 163840 ms)

struct bt_le_per_adv_sync_info

#include <bluetooth.h> Advertising set info structure.

Public Members

bt_addr_le_t addr

Periodic Advertiser Address

uint8_t sid

Advertiser SID

uint16 t interval

Periodic advertising interval (N * 1.25 ms)

uint8_t phy

Advertiser PHY

struct bt_le_per_adv_sync_transfer_param

#include <bluetooth.h>

Public Members

uint16_t skip

Maximum event skip.

The number of periodic advertising packets that can be skipped after a successful receive.

uint16_t **timeout**

Synchronization timeout (N * 10 ms)

Synchronization timeout for the periodic advertising sync. Range 0x000A to 0x4000 (100 ms to 163840 ms)

uint32_t options

Periodic Advertising Sync Transfer options

struct bt_le_scan_param

#include <bluetooth.h> LE scan parameters

Public Members

uint8_t type

 $Scan\ type\ (BT_LE_SCAN_TYPE_ACTIVE\ or\ BT_LE_SCAN_TYPE_PASSIVE)$

uint32_t **options**

Bit-field of scanning options.

uint16_t interval

Scan interval (N * 0.625 ms)

uint16_t window

Scan window (N * 0.625 ms)

uint16 t timeout

Scan timeout (N * 10 ms)

Application will be notified by the scan timeout callback. Set zero to disable timeout.

uint16 t interval_coded

Scan interval LE Coded PHY (N * 0.625 MS)

Set zero to use same as LE 1M PHY scan interval.

uint16 t window_coded

Scan window LE Coded PHY (N * 0.625 MS)

Set zero to use same as LE 1M PHY scan window.

struct bt_le_scan_recv_info

#include <bluetooth.h> LE advertisement and scan response packet information

Public Members

```
const bt_addr_le_t *addr
```

Advertiser LE address and type.

If advertiser is anonymous then this address will be *BT_ADDR_LE_ANY*.

uint8_t sid

Advertising Set Identifier.

int8_t **rssi**

Strength of advertiser signal.

int8_t tx_power

Transmit power of the advertiser.

uint8_t adv_type

Advertising packet type.

Uses the BT_GAP_ADV_TYPE_* value.

May indicate that this is a scan response if the type is BT_GAP_ADV_TYPE_SCAN_RSP.

uint16_t adv_props

Advertising packet properties bitfield.

Uses the BT_GAP_ADV_PROP_* values. May indicate that this is a scan response if the value contains the BT_GAP_ADV_PROP_SCAN_RESPONSE bit.

uint16_t interval

Periodic advertising interval.

If 0 there is no periodic advertising.

uint8_t primary_phy

Primary advertising channel PHY.

uint8_t secondary_phy

Secondary advertising channel PHY.

struct bt_le_scan_cb

#include <bluetooth.h> Listener context for (LE) scanning.

Public Members

```
void (*recv)(const struct bt_le_scan_recv_info *info, struct net_buf_simple *buf)
```

Advertisement packet and scan response received callback.

Param info

Advertiser packet and scan response information.

Param buf

Buffer containing advertiser data.

```
void (*timeout)(void)
```

The scanner has stopped scanning after scan timeout.

struct bt_le_oob_sc_data

#include <bluetooth.h> LE Secure Connections pairing Out of Band data.

Public Members

```
uint8_t r[16]
```

Random Number.

uint8 t **c**[16]

Confirm Value.

struct bt_le_oob

#include <bluetooth.h> LE Out of Band information.

Public Members

```
bt_addr_le_t addr
```

LE address. If privacy is enabled this is a Resolvable Private Address.

```
struct bt_le_oob_sc_data le_sc_data
```

LE Secure Connections pairing Out of Band data.

struct bt_br_discovery_result

#include <bluetooth.h> BR/EDR discovery result structure.

Public Members

```
uint8_t _priv[4] private
```

 bt_addr_t addr

Remote device address

int8_t rssi

RSSI from inquiry

uint8 t cod[3]

Class of Device

uint8_t eir[240]

Extended Inquiry Response

struct bt_br_discovery_param

#include <bluetooth.h> BR/EDR discovery parameters

Public Members

```
uint8_t length
```

Maximum length of the discovery in units of 1.28 seconds. Valid range is 0x01 - 0x30.

bool limited

True if limited discovery procedure is to be used.

struct bt_br_oob

#include <bluetooth.h>

Public Members

bt_addr_t addr

BR/EDR address.

struct bt_bond_info

#include <bluetooth.h> Information about a bond with a remote device.

Public Members

```
bt_addr_le_t addr
```

Address of the remote device.

group bt_addr

Bluetooth device address definitions and utilities.

Defines

BT_ADDR_LE_PUBLIC

BT_ADDR_LE_RANDOM

BT_ADDR_LE_PUBLIC_ID

BT_ADDR_LE_RANDOM_ID

BT_ADDR_LE_UNRESOLVED

BT_ADDR_LE_ANONYMOUS

BT_ADDR_SIZE

Length in bytes of a standard Bluetooth address

BT_ADDR_LE_SIZE

Length in bytes of an LE Bluetooth address. Not packed, so no sizeof()

BT_ADDR_ANY

Bluetooth device "any" address, not a valid address

BT_ADDR_NONE

Bluetooth device "none" address, not a valid address

BT_ADDR_LE_ANY

Bluetooth LE device "any" address, not a valid address

BT_ADDR_LE_NONE

Bluetooth LE device "none" address, not a valid address

BT_ADDR_IS_RPA(a)

Check if a Bluetooth LE random address is resolvable private address.

BT_ADDR_IS_NRPA(a)

Check if a Bluetooth LE random address is a non-resolvable private address.

BT_ADDR_IS_STATIC(a)

Check if a Bluetooth LE random address is a static address.

BT_ADDR_SET_RPA(a)

Set a Bluetooth LE random address as a resolvable private address.

BT_ADDR_SET_NRPA(a)

Set a Bluetooth LE random address as a non-resolvable private address.

BT ADDR SET STATIC(a)

Set a Bluetooth LE random address as a static address.

BT_ADDR_STR_LEN

Recommended length of user string buffer for Bluetooth address.

The recommended length guarantee the output of address conversion will not lose valuable information about address being processed.

BT_ADDR_LE_STR_LEN

Recommended length of user string buffer for Bluetooth LE address.

The recommended length guarantee the output of address conversion will not lose valuable information about address being processed.

Functions

static inline int **bt_addr_cmp**(const *bt_addr_t* *a, const *bt_addr_t* *b)

Compare Bluetooth device addresses.

Parameters

- a First Bluetooth device address to compare
- **b** Second Bluetooth device address to compare

Returns

negative value if a < b, 0 if a == b, else positive

static inline int **bt_addr_le_cmp**(const *bt_addr_le_t* *a, const *bt_addr_le_t* *b)

Compare Bluetooth LE device addresses.

Parameters

• a – First Bluetooth LE device address to compare

• **b** – Second Bluetooth LE device address to compare

Returns

negative value if a < b, 0 if a == b, else positive

static inline void **bt_addr_copy**(*bt_addr_t* *dst, const *bt_addr_t* *src)

Copy Bluetooth device address.

Parameters

- **dst** Bluetooth device address destination buffer.
- **src** Bluetooth device address source buffer.

static inline void **bt_addr_le_copy**(*bt_addr_le_t* *dst, const *bt_addr_le_t* *src)

Copy Bluetooth LE device address.

Parameters

- dst Bluetooth LE device address destination buffer.
- src Bluetooth LE device address source buffer.

```
int bt_addr_le_create_nrpa(bt_addr_le_t *addr)
```

Create a Bluetooth LE random non-resolvable private address.

```
int bt_addr_le_create_static(bt addr le t *addr)
```

Create a Bluetooth LE random static address.

```
static inline bool bt_addr_le_is_rpa(const bt_addr_le_t *addr)
```

Check if a Bluetooth LE address is a random private resolvable address.

Parameters

• addr – Bluetooth LE device address.

Returns

true if address is a random private resolvable address.

```
static inline bool bt_addr_le_is_identity(const bt_addr_le_t *addr)
```

Check if a Bluetooth LE address is valid identity address.

Valid Bluetooth LE identity addresses are either public address or random static address.

Parameters

• addr – Bluetooth LE device address.

Returns

true if address is a valid identity address.

```
static inline int bt_addr_to_str(const bt_addr_t *addr, char *str, size_t len)
```

Converts binary Bluetooth address to string.

Parameters

- addr Address of buffer containing binary Bluetooth address.
- str Address of user buffer with enough room to store formatted string containing binary address.
- 1en Length of data to be copied to user string buffer. Refer to BT_ADDR_STR_LEN about recommended value.

Returns

Number of successfully formatted bytes from binary address.

```
static inline int bt_addr_le_to_str(const bt_addr_le_t *addr, char *str, size_t len)
```

Converts binary LE Bluetooth address to string.

Parameters

- addr Address of buffer containing binary LE Bluetooth address.
- **str** Address of user buffer with enough room to store formatted string containing binary LE address.
- **len** Length of data to be copied to user string buffer. Refer to BT_ADDR_LE_STR_LEN about recommended value.

Returns

Number of successfully formatted bytes from binary address.

```
int bt_addr_from_str(const char *str, bt_addr_t *addr)
```

Convert Bluetooth address from string to binary.

Parameters

- **str [in]** The string representation of a Bluetooth address.
- addr [out] Address of buffer to store the Bluetooth address

Returns

Zero on success or (negative) error code otherwise.

```
int bt_addr_le_from_str(const char *str, const char *type, bt_addr_le_t *addr)
```

Convert LE Bluetooth address from string to binary.

Parameters

- **str [in]** The string representation of an LE Bluetooth address.
- type [in] The string representation of the LE Bluetooth address type.
- addr [out] Address of buffer to store the LE Bluetooth address

Returns

Zero on success or (negative) error code otherwise.

Variables

```
const bt_addr_t bt_addr_any
const bt_addr_t bt_addr_none
const bt_addr_le_t bt_addr_le_any
const bt_addr_le_t bt_addr_le_none
struct bt_addr_t
#include <addr.h> Bluetooth Device Address
```

struct bt_addr_le_t

#include <addr.h> Bluetooth LE Device Address

group bt_gap_defines

Bluetooth Generic Access Profile defines and Assigned Numbers.

Defines

BT_COMP_ID_LF

Company Identifiers (see Bluetooth Assigned Numbers)

BT_DATA_FLAGS

EIR/AD data type definitions

BT_DATA_UUID16_SOME

BT_DATA_UUID16_ALL

BT_DATA_UUID32_SOME

BT_DATA_UUID32_ALL

BT_DATA_UUID128_SOME

BT_DATA_UUID128_ALL

BT_DATA_NAME_SHORTENED

BT_DATA_NAME_COMPLETE

BT_DATA_TX_POWER

BT_DATA_SM_TK_VALUE

BT_DATA_SM_OOB_FLAGS

BT_DATA_SOLICIT16

BT_DATA_SOLICIT128

BT_DATA_SVC_DATA16

BT_DATA_GAP_APPEARANCE BT_DATA_LE_BT_DEVICE_ADDRESS BT_DATA_LE_ROLE BT_DATA_SOLICIT32 BT_DATA_SVC_DATA32 BT_DATA_SVC_DATA128 BT_DATA_LE_SC_CONFIRM_VALUE BT_DATA_LE_SC_RANDOM_VALUE BT_DATA_URI BT_DATA_LE_SUPPORTED_FEATURES BT_DATA_CHANNEL_MAP_UPDATE_IND BT_DATA_MESH_PROV BT_DATA_MESH_MESSAGE BT_DATA_MESH_BEACON BT_DATA_BIG_INFO BT_DATA_BROADCAST_CODE BT_DATA_CSIS_RSI BT_DATA_MANUFACTURER_DATA BT_LE_AD_LIMITED BT_LE_AD_GENERAL BT_LE_AD_NO_BREDR

BT_GAP_SCAN_FAST_INTERVAL

BT_GAP_SCAN_FAST_WINDOW

BT_GAP_SCAN_SLOW_INTERVAL_1

BT_GAP_SCAN_SLOW_WINDOW_1

BT_GAP_SCAN_SLOW_INTERVAL_2

BT_GAP_SCAN_SLOW_WINDOW_2

BT_GAP_ADV_FAST_INT_MIN_1

BT_GAP_ADV_FAST_INT_MAX_1

BT_GAP_ADV_FAST_INT_MIN_2

BT_GAP_ADV_FAST_INT_MAX_2

BT_GAP_ADV_SLOW_INT_MIN

BT_GAP_ADV_SLOW_INT_MAX

BT_GAP_PER_ADV_FAST_INT_MIN_1

BT_GAP_PER_ADV_FAST_INT_MAX_1

BT_GAP_PER_ADV_FAST_INT_MIN_2

BT_GAP_PER_ADV_FAST_INT_MAX_2

BT_GAP_PER_ADV_SLOW_INT_MIN

BT_GAP_PER_ADV_SLOW_INT_MAX

BT_GAP_INIT_CONN_INT_MIN

BT_GAP_INIT_CONN_INT_MAX

BT_GAP_ADV_MAX_ADV_DATA_LEN

Maximum advertising data length.

BT_GAP_ADV_MAX_EXT_ADV_DATA_LEN

Maximum extended advertising data length.

Note: The maximum advertising data length that can be sent by an extended advertiser is defined by the controller.

BT_GAP_TX_POWER_INVALID

BT_GAP_RSSI_INVALID

BT_GAP_SID_INVALID

BT_GAP_NO_TIMEOUT

BT_GAP_ADV_HIGH_DUTY_CYCLE_MAX_TIMEOUT

BT_GAP_DATA_LEN_DEFAULT

BT_GAP_DATA_LEN_MAX

BT_GAP_DATA_TIME_DEFAULT

BT_GAP_DATA_TIME_MAX

BT_GAP_SID_MAX

BT_GAP_PER_ADV_MAX_SKIP

BT_GAP_PER_ADV_MIN_TIMEOUT

BT_GAP_PER_ADV_MAX_TIMEOUT

BT_GAP_PER_ADV_MIN_INTERVAL

Minimum Periodic Advertising Interval (N * 1.25 ms)

BT_GAP_PER_ADV_MAX_INTERVAL

Maximum Periodic Advertising Interval (N * 1.25 ms)

BT_GAP_PER_ADV_INTERVAL_TO_MS(interval)

Convert periodic advertising interval (N * 1.25 ms) to milliseconds.

5 / 4 represents 1.25 ms unit.

78

BT_LE_SUPP_FEAT_40_ENCODE(w64)

Encode 40 least significant bits of 64-bit LE Supported Features into array values in little-endian format.

Helper macro to encode 40 least significant bits of 64-bit LE Supported Features value into advertising data. The number of bits that are encoded is a number of LE Supported Features defined by BT 5.3 Core specification.

Example of how to encode the **0x000000DFF00DF00D** into advertising data.

```
BT_DATA_BYTES(BT_DATA_LE_SUPPORTED_FEATURES, BT_LE_SUPP_FEAT_40_

→ENCODE(0x000000DFF00DF00D))
```

Parameters

• w64 – LE Supported Features value (64-bits)

Returns

The comma separated values for LE Supported Features value that may be used directly as an argument for *BT_DATA_BYTES*.

BT_LE_SUPP_FEAT_32_ENCODE(w64)

Encode 4 least significant bytes of 64-bit LE Supported Features into 4 bytes long array of values in little-endian format.

Helper macro to encode 64-bit LE Supported Features value into advertising data. The macro encodes 4 least significant bytes into advertising data. Other 4 bytes are not encoded.

Example of how to encode the 0x000000DFF00DF00D into advertising data.

```
BT_DATA_BYTES(BT_DATA_LE_SUPPORTED_FEATURES, BT_LE_SUPP_FEAT_32_

→ENCODE(0x000000DFF00DF00D))
```

Parameters

• w64 – LE Supported Features value (64-bits)

Returns

The comma separated values for LE Supported Features value that may be used directly as an argument for BT_DATA_BYTES .

BT_LE_SUPP_FEAT_24_ENCODE(w64)

Encode 3 least significant bytes of 64-bit LE Supported Features into 3 bytes long array of values in little-endian format.

Helper macro to encode 64-bit LE Supported Features value into advertising data. The macro encodes 3 least significant bytes into advertising data. Other 5 bytes are not encoded.

Example of how to encode the 0x000000DFF00DF00D into advertising data.

```
BT_DATA_BYTES(BT_DATA_LE_SUPPORTED_FEATURES, BT_LE_SUPP_FEAT_24_

ENCODE(0x000000DFF00DF00D))
```

Parameters

• w64 – LE Supported Features value (64-bits)

Returns

The comma separated values for LE Supported Features value that may be used directly as an argument for *BT_DATA_BYTES*.

BT_LE_SUPP_FEAT_16_ENCODE(w64)

Encode 2 least significant bytes of 64-bit LE Supported Features into 2 bytes long array of values in little-endian format.

Helper macro to encode 64-bit LE Supported Features value into advertising data. The macro encodes 3 least significant bytes into advertising data. Other 6 bytes are not encoded.

Example of how to encode the 0x000000DFF00DF00D into advertising data.

```
BT_DATA_BYTES(BT_DATA_LE_SUPPORTED_FEATURES, BT_LE_SUPP_FEAT_16_

SENCODE(0x000000DFF00DF00D))
```

Parameters

• **w64** – LE Supported Features value (64-bits)

Returns

The comma separated values for LE Supported Features value that may be used directly as an argument for *BT_DATA_BYTES*.

BT_LE_SUPP_FEAT_8_ENCODE (w64)

Encode the least significant byte of 64-bit LE Supported Features into single byte long array.

Helper macro to encode 64-bit LE Supported Features value into advertising data. The macro encodes the least significant byte into advertising data. Other 7 bytes are not encoded.

Example of how to encode the 0x000000DFF00DF00D into advertising data.

```
BT_DATA_BYTES(BT_DATA_LE_SUPPORTED_FEATURES, BT_LE_SUPP_FEAT_8_

SENCODE(0x000000DFF00DF00D))
```

Parameters

• w64 – LE Supported Features value (64-bits)

Returns

The value of least significant byte of LE Supported Features value that may be used directly as an argument for *BT_DATA_BYTES*.

BT_LE_SUPP_FEAT_VALIDATE (w64)

Validate whether LE Supported Features value does not use bits that are reserved for future use.

Helper macro to check if w64 has zeros as bits 40-63. The macro is compliant with BT 5.3 Core Specification where bits 0-40 has assigned values. In case of invalid value, build time error is reported.

Enums

enum [anonymous] LE PHY types Values: enumerator BT_GAP_LE_PHY_NONE Convenience macro for when no PHY is set. enumerator BT_GAP_LE_PHY_1M LE 1M PHY enumerator BT_GAP_LE_PHY_2M LE 2M PHY enumerator BT_GAP_LE_PHY_CODED LE Coded PHY enum [anonymous] Advertising PDU types Values: enumerator BT_GAP_ADV_TYPE_ADV_IND Scannable and connectable advertising. enumerator BT_GAP_ADV_TYPE_ADV_DIRECT_IND Directed connectable advertising. enumerator BT_GAP_ADV_TYPE_ADV_SCAN_IND Non-connectable and scannable advertising. enumerator BT_GAP_ADV_TYPE_ADV_NONCONN_IND Non-connectable and non-scannable advertising. enumerator BT_GAP_ADV_TYPE_SCAN_RSP Additional advertising data requested by an active scanner. enumerator BT_GAP_ADV_TYPE_EXT_ADV Extended advertising, see advertising properties.

1.5. Generic Access Profile (GAP)

Advertising PDU properties

enum [anonymous]

Values:

```
enumerator BT_GAP_ADV_PROP_CONNECTABLE
         Connectable advertising.
     enumerator BT_GAP_ADV_PROP_SCANNABLE
         Scannable advertising.
     enumerator BT_GAP_ADV_PROP_DIRECTED
         Directed advertising.
     enumerator BT_GAP_ADV_PROP_SCAN_RESPONSE
         Additional advertising data requested by an active scanner.
     enumerator BT_GAP_ADV_PROP_EXT_ADV
         Extended advertising.
enum [anonymous]
    Constant Tone Extension (CTE) types
     Values:
     enumerator BT_GAP_CTE_AOA
         Angle of Arrival
     enumerator BT_GAP_CTE_AOD_1US
         Angle of Departure with 1 us slots
     enumerator BT_GAP_CTE_AOD_2US
         Angle of Departure with 2 us slots
     enumerator BT_GAP_CTE_NONE
         No extensions
enum [anonymous]
     Peripheral sleep clock accuracy (SCA) in ppm (parts per million)
     Values:
     enumerator BT_GAP_SCA_UNKNOWN
     enumerator BT_GAP_SCA_251_500
     enumerator BT_GAP_SCA_151_250
     enumerator BT_GAP_SCA_101_150
     enumerator BT_GAP_SCA_76_100
```

```
enumerator BT_GAP_SCA_51_75
enumerator BT_GAP_SCA_31_50
enumerator BT_GAP_SCA_21_30
enumerator BT_GAP_SCA_0_20
```

1.6 Generic Attribute Profile (GATT)

GATT layer manages the service database providing APIs for service registration and attribute declaration.

Services can be registered using $bt_gatt_service_register()$ API which takes the $bt_gatt_service$ struct that provides the list of attributes the service contains. The helper macro $BT_GATT_SERVICE()$ can be used to declare a service.

Attributes can be declared using the *bt_gatt_attr* struct or using one of the helper macros:

```
BT GATT PRIMARY SERVICE
```

Declares a Primary Service.

BT_GATT_SECONDARY_SERVICE

Declares a Secondary Service.

BT_GATT_INCLUDE_SERVICE

Declares a Include Service.

BT_GATT_CHARACTERISTIC

Declares a Characteristic.

BT_GATT_DESCRIPTOR

Declares a Descriptor.

BT_GATT_ATTRIBUTE

Declares an Attribute.

BT_GATT_CCC

Declares a Client Characteristic Configuration.

BT GATT CEP

Declares a Characteristic Extended Properties.

BT_GATT_CUD

Declares a Characteristic User Format.

Each attribute contain a uuid, which describes their type, a read callback, a write callback and a set of permission. Both read and write callbacks can be set to NULL if the attribute permission don't allow their respective operations.

Note: Attribute read and write callbacks are called directly from RX Thread thus it is not recommended to block for long periods of time in them.

Attribute value changes can be notified using $bt_gatt_notify()$ API, alternatively there is $bt_gatt_notify_cb()$ where is is possible to pass a callback to be called when it is necessary to know the exact instant when the data has been transmitted over the air. Indications are supported by $bt_gatt_indicate()$ API.

Client procedures can be enabled with the configuration option: CONFIG_BT_GATT_CLIENT

Discover procedures can be initiated with the use of bt_gatt_discover() API which takes the bt_gatt_discover_params struct which describes the type of discovery. The parameters also serves as a filter when setting the uuid field only attributes which matches will be discovered, in contrast setting it to NULL allows all attributes to be discovered.

Note: Caching discovered attributes is not supported.

Read procedures are supported by $bt_gatt_read()$ API which takes the $bt_gatt_read_params$ struct as parameters. In the parameters one or more attributes can be set, though setting multiple handles requires the option: CONFIG_BT_GATT_READ_MULTIPLE

Write procedures are supported by $bt_gatt_write()$ API and takes $bt_gatt_write_params$ struct as parameters. In case the write operation don't require a response $bt_gatt_write_without_response()$ or $bt_gatt_write_without_response_cb()$ APIs can be used, with the later working similarly to $bt_gatt_notify_cb()$.

Subscriptions to notification and indication can be initiated with use of $bt_gatt_subscribe()$ API which takes $bt_gatt_subscribe_params$ as parameters. Multiple subscriptions to the same attribute are supported so there could be multiple notify callback being triggered for the same attribute. Subscriptions can be removed with use of $bt_gatt_unsubscribe()$ API.

Note: When subscriptions are removed notify callback is called with the data set to NULL.

1.6.1 API Reference

group bt_gatt

Generic Attribute Profile (GATT)

Defines

BT_GATT_ERR(att err)

Construct error return value for attribute read and write callbacks.

Parameters

• _att_err - ATT error code

Returns

Appropriate error code for the attribute callbacks.

BT_GATT_CHRC_BROADCAST

Characteristic broadcast property.

Characteristic Properties Bit field values

If set, permits broadcasts of the Characteristic Value using Server Characteristic Configuration Descriptor.

BT_GATT_CHRC_READ

Characteristic read property.

If set, permits reads of the Characteristic Value.

BT_GATT_CHRC_WRITE_WITHOUT_RESP

Characteristic write without response property.

If set, permits write of the Characteristic Value without response.

BT_GATT_CHRC_WRITE

Characteristic write with response property.

If set, permits write of the Characteristic Value with response.

BT_GATT_CHRC_NOTIFY

Characteristic notify property.

If set, permits notifications of a Characteristic Value without acknowledgment.

BT_GATT_CHRC_INDICATE

Characteristic indicate property.

If set, permits indications of a Characteristic Value with acknowledgment.

BT_GATT_CHRC_AUTH

Characteristic Authenticated Signed Writes property.

If set, permits signed writes to the Characteristic Value.

BT_GATT_CHRC_EXT_PROP

Characteristic Extended Properties property.

If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor.

BT_GATT_CEP_RELIABLE_WRITE

Characteristic Extended Properties Bit field values

BT_GATT_CEP_WRITABLE_AUX

BT_GATT_CCC_NOTIFY

Client Characteristic Configuration Notification.

Client Characteristic Configuration Values

If set, changes to Characteristic Value shall be notified.

BT_GATT_CCC_INDICATE

Client Characteristic Configuration Indication.

If set, changes to Characteristic Value shall be indicated.

BT_GATT_SCC_BROADCAST

Server Characteristic Configuration Broadcast.

Server Characteristic Configuration Values

If set, the characteristic value shall be broadcast in the advertising data when the server is advertising.

Typedefs

typedef ssize_t (***bt_gatt_attr_read_func_t**)(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t len, uint16_t offset)

Attribute read callback.

The callback can also be used locally to read the contents of the attribute in which case no connection will be set.

Param conn

The connection that is requesting to read

Param attr

The attribute that's being read

Param buf

Buffer to place the read result in

Param len

Length of data to read

Param offset

Offset to start reading from

Return

Number of bytes read, or in case of an error *BT_GATT_ERR()* with a specific BT_ATT_ERR_* error code.

typedef ssize_t (***bt_gatt_attr_write_func_t**)(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, const void *buf, uint16_t len, uint16_t offset, uint8_t flags)

Attribute write callback.

Param conn

The connection that is requesting to write

Param attr

The attribute that's being written

Param buf

Buffer with the data to write

Param len

Number of bytes in the buffer

Param offset

Offset to start writing from

Param flags

Flags (BT_GATT_WRITE_FLAG_*)

Return

Number of bytes written, or in case of an error $BT_GATT_ERR()$ with a specific BT ATT ERR * error code.

Enums

enum bt_gatt_perm

GATT attribute permission bit field values

Values:

enumerator BT_GATT_PERM_NONE

No operations supported, e.g. for notify-only

enumerator BT_GATT_PERM_READ

Attribute read permission.

enumerator BT_GATT_PERM_WRITE

Attribute write permission.

enumerator BT_GATT_PERM_READ_ENCRYPT

Attribute read permission with encryption.

If set, requires encryption for read access.

enumerator BT_GATT_PERM_WRITE_ENCRYPT

Attribute write permission with encryption.

If set, requires encryption for write access.

enumerator BT_GATT_PERM_READ_AUTHEN

Attribute read permission with authentication.

If set, **requires** encryption **using** authenticated link-key **for** read access.

enumerator BT_GATT_PERM_WRITE_AUTHEN

Attribute write permission with authentication.

If set, **requires** encryption **using** authenticated link-key **for** write access.

enumerator BT_GATT_PERM_PREPARE_WRITE

Attribute prepare write permission.

If set, allows prepare writes with use of BT_GATT_WRITE_FLAG_PREPARE passed to write callback.

enumerator BT_GATT_PERM_READ_LESC

Attribute read permission with LE Secure Connection encryption.

If set, requires that LE Secure Connections is used for read access.

enumerator BT_GATT_PERM_WRITE_LESC

Attribute write permission with LE Secure Connection encryption.

If set, requires that LE Secure Connections is used for write access.

enum [anonymous]

GATT attribute write flags

Values:

enumerator BT_GATT_WRITE_FLAG_PREPARE

Attribute prepare write flag.

If set, write callback should only check if the device is authorized but no data shall be written.

enumerator BT_GATT_WRITE_FLAG_CMD

Attribute write command flag.

If set, indicates that write operation is a command (Write without response) which doesn't generate any response.

enumerator BT_GATT_WRITE_FLAG_EXECUTE

Attribute write execute flag.

If set, indicates that write operation is a execute, which indicates the end of a long write, and will come after 1 or more @ref BT_GATT_WRITE_FLAG_PREPARE.

struct bt_gatt_attr

#include <gatt.h> GATT Attribute structure.

Public Members

const struct bt_uuid *uuid

Attribute UUID

bt_gatt_attr_write_func_t write

Attribute write callback

```
void *user_data
         Attribute user data
     uint16_t handle
         Attribute handle
     uint16_t perm
         Attribute permissions.
         Will be 0 if returned from bt_gatt_discover().
struct bt_gatt_service_static
     #include <gatt.h> GATT Service structure.
     Public Members
     const struct bt_gatt_attr *attrs
         Service Attributes
     size_t attr_count
         Service Attribute count
struct bt_gatt_service
     #include <gatt.h> GATT Service structure.
     Public Members
     struct bt_gatt_attr *attrs
         Service Attributes
     size t attr_count
         Service Attribute count
struct bt_gatt_service_val
     #include <gatt.h> Service Attribute Value.
     Public Members
     const struct bt_uuid *uuid
         Service UUID.
     uint16_t end_handle
```

Service end handle.

struct bt_gatt_include

#include <gatt.h> Include Attribute Value.

Public Members

```
const struct bt_uuid *uuid
```

Service UUID.

uint16_t start_handle

Service start handle.

uint16_t end_handle

Service end handle.

struct bt_gatt_cb

#include <gatt.h> GATT callback structure.

Public Members

```
void (*att_mtu_updated)(struct bt_conn *conn, uint16_t tx, uint16_t rx)
```

The maximum ATT MTU on a connection has changed.

This callback notifies the application that the maximum TX or RX ATT MTU has increased.

Param conn

Connection object.

Param tx

Updated TX ATT MTU.

Param rx

Updated RX ATT MTU.

struct bt_gatt_chrc

#include <gatt.h> Characteristic Attribute Value.

Public Members

```
const struct bt_uuid *uuid
```

Characteristic UUID.

uint16 t value_handle

Characteristic Value handle.

uint8_t properties

Characteristic properties.

struct bt_gatt_cep

#include <gatt.h> Characteristic Extended Properties Attribute Value.

Public Members

uint16_t properties

Characteristic Extended properties

struct bt_gatt_ccc

#include <gatt.h> Client Characteristic Configuration Attribute Value

Public Members

uint16_t flags

Client Characteristic Configuration flags

struct bt_gatt_scc

#include <gatt.h> Server Characteristic Configuration Attribute Value

Public Members

uint16_t flags

Server Characteristic Configuration flags

struct bt_gatt_cpf

#include <gatt.h> GATT Characteristic Presentation Format Attribute Value.

Public Members

uint8 t format

Format of the value of the characteristic

int8_t exponent

Exponent field to determine how the value of this characteristic is further formatted

uint16_t unit

Unit of the characteristic

uint8 t name_space

Name space of the description

uint16_t description

Description of the characteristic as defined in a higher layer profile

1.6.1.1 GATT Server

group bt_gatt_server

Defines

BT_GATT_SERVICE_DEFINE(_name, ...)

Statically define and register a service.

Helper macro to statically define and register a service.

Parameters

• _name - Service name.

```
_BT_GATT_ATTRS_ARRAY_DEFINE(n, _instances, _attrs_def)
```

```
_BT_GATT_SERVICE_ARRAY_ITEM(_n, _)
```

BT_GATT_SERVICE_INSTANCE_DEFINE(_name, _instances, _instance_num, _attrs_def)

Statically define service structure array.

Helper macro to statically define service structure array. Each element of the array is linked to the service attribute array which is also defined in this scope using _attrs_def macro.

Parameters

- _name Name of service structure array.
- _instances Array of instances to pass as user context to the attribute callbacks.
- _instance_num Number of elements in instance array.
- _attrs_def Macro provided by the user that defines attribute array for the service. This macro should accept single parameter which is the instance context.

BT_GATT_SERVICE(attrs)

Service Structure Declaration Macro.

Helper macro to declare a service structure.

Parameters

• _attrs - Service attributes.

BT_GATT_PRIMARY_SERVICE(_service)

Primary Service Declaration Macro.

Helper macro to declare a primary service attribute.

Parameters

• _service - Service attribute value.

BT_GATT_SECONDARY_SERVICE(_service)

Secondary Service Declaration Macro.

Helper macro to declare a secondary service attribute.

Note: A secondary service is only intended to be included from a primary service or another secondary service or other higher layer specification.

Parameters

• _service - Service attribute value.

BT_GATT_INCLUDE_SERVICE(_service_incl)

Include Service Declaration Macro.

Helper macro to declare database internal include service attribute.

Parameters

• **_service_incl** – the first service attribute of service to include

BT_GATT_CHRC_INIT(_uuid, _handle, _props)

BT_GATT_CHARACTERISTIC(_uuid, _props, _perm, _read, _write, _user_data)

Characteristic and Value Declaration Macro.

Helper macro to declare a characteristic attribute along with its attribute value.

Parameters

- _uuid Characteristic attribute uuid.
- **_props** Characteristic attribute properties, a bitmap of BT_GATT_CHRC_* macros.
- _perm Characteristic Attribute access permissions, a bitmap of bt gatt perm values.
- **_read** Characteristic Attribute read callback (*bt_gatt_attr_read_func_t*).
- _write Characteristic Attribute write callback (bt_gatt_attr_write_func_t).
- _user_data Characteristic Attribute user data.

BT_GATT_CCC_MAX

BT_GATT_CCC_INITIALIZER(_changed, _write, _match)

Initialize Client Characteristic Configuration Declaration Macro.

Helper macro to initialize a Managed CCC attribute value.

Parameters

- **_changed** Configuration changed callback.
- _write Configuration write callback.
- **_match** Configuration match callback.

BT_GATT_CCC_MANAGED(_ccc, _perm)

Managed Client Characteristic Configuration Declaration Macro.

Helper macro to declare a Managed CCC attribute.

Parameters

- **_ccc** CCC attribute user data, shall point to a _*bt*_gatt_ccc.
- **_perm** CCC access permissions, a bitmap of *bt_gatt_perm* values.

BT_GATT_CCC (changed, perm)

Client Characteristic Configuration Declaration Macro.

Helper macro to declare a CCC attribute.

Parameters

- _changed Configuration changed callback.
- **_perm** CCC access permissions, a bitmap of *bt_gatt_perm* values.

BT_GATT_CEP(value)

Characteristic Extended Properties Declaration Macro.

Helper macro to declare a CEP attribute.

Parameters

• _value – Pointer to a struct *bt_gatt_cep*.

BT_GATT_CUD(_value, _perm)

Characteristic User Format Descriptor Declaration Macro.

Helper macro to declare a CUD attribute.

Parameters

- _value User description NULL-terminated C string.
- **_perm** Descriptor attribute access permissions, a bitmap of *bt_gatt_perm* values.

BT_GATT_CPF(value)

Characteristic Presentation Format Descriptor Declaration Macro.

Helper macro to declare a CPF attribute.

Parameters

• **_value** – Pointer to a struct *bt_gatt_cpf*.

BT_GATT_DESCRIPTOR(_uuid, _perm, _read, _write, _user_data)

Descriptor Declaration Macro.

Helper macro to declare a descriptor attribute.

Parameters

- _uuid Descriptor attribute uuid.
- **_perm** Descriptor attribute access permissions, a bitmap of bt_gatt_perm values.
- **_read** Descriptor attribute read callback (*bt_gatt_attr_read_func_t*).
- _write Descriptor attribute write callback (bt_gatt_attr_write_func_t).
- $\bullet \ \ \, \textbf{_user_data} Descriptor \ attribute \ user \ data.$

BT_GATT_ATTRIBUTE(uuid, perm, read, write, user data)

Attribute Declaration Macro.

Helper macro to declare an attribute.

Parameters

- _uuid Attribute uuid.
- **_perm** Attribute access permissions, a bitmap of *bt_gatt_perm* values.
- **_read** Attribute read callback (*bt_gatt_attr_read_func_t*).
- **_write** Attribute write callback (*bt_gatt_attr_write_func_t*).
- _user_data Attribute user data.

Typedefs

```
typedef uint8_t (*bt_gatt_attr_func_t)(const struct bt_gatt_attr *attr, uint16_t handle, void *user_data)
     Attribute iterator callback.
         Param attr
             Attribute found.
         Param handle
             Attribute handle found.
         Param user data
             Data given.
         Return
             BT_GATT_ITER_CONTINUE if should continue to the next attribute.
         Return
             BT_GATT_ITER_STOP to stop.
typedef void (*bt_gatt_complete_func_t)(struct bt_conn *conn, void *user_data)
     Notification complete result callback.
         Param conn
             Connection object.
         Param user data
             Data passed in by the user.
typedef void (*bt_gatt_indicate_func_t)(struct bt_conn *conn, struct bt_gatt_indicate_params *params,
uint8_t err)
     Indication complete result callback.
         Param conn
             Connection object.
         Param params
             Indication params object.
         Param err
             ATT error code
typedef void (*bt_gatt_indicate_params_destroy_t)(struct bt_gatt_indicate_params *params)
Enums
enum [anonymous]
     Values:
     enumerator BT_GATT_ITER_STOP
     enumerator BT_GATT_ITER_CONTINUE
```

Functions

```
void bt_gatt_cb_register(struct bt_gatt_cb *cb)
```

Register GATT callbacks.

Register callbacks to monitor the state of GATT.

Parameters

• cb – Callback struct.

int bt_gatt_service_register(struct bt_gatt_service *svc)

Register GATT service.

Register GATT service. Applications can make use of macros such as BT_GATT_PRIMARY_SERVICE, BT_GATT_CHARACTERISTIC, BT_GATT_DESCRIPTOR, etc.

When using @kconfig{CONFIG_BT_SETTINGS} then all services that should have bond configuration loaded, i.e. CCC values, must be registered before calling settings_load.

When using <code>@kconfig{CONFIG_BT_GATT_CACHING}</code> and <code>@kconfig{CONFIG_BT_SETTINGS}</code> then all services that should be included in the GATT Database Hash calculation should be added before calling settings_load. All services registered after settings_load will trigger a new database hash calculation and a new hash stored.

Parameters

• svc – Service containing the available attributes

Returns

0 in case of success or negative value in case of error.

int bt_gatt_service_unregister(struct bt_gatt_service *svc)

Unregister GATT service.

Parameters

• **svc** – Service to be unregistered.

Returns

0 in case of success or negative value in case of error.

bool bt_gatt_service_is_registered(const struct bt_gatt_service *svc)

Check if GATT service is registered.

Parameters

• **svc** – Service to be checked.

Returns

true if registered or false if not register.

void **bt_gatt_foreach_attr_type**(uint16_t start_handle, uint16_t end_handle, const struct *bt_uuid* *uuid, const void *attr_data, uint16_t num_matches, *bt_gatt_attr_func_t* func, void *user_data)

Attribute iterator by type.

Iterate attributes in the given range matching given UUID and/or data.

Parameters

- start_handle Start handle.
- end_handle End handle.

- uuid UUID to match, passing NULL skips UUID matching.
- attr_data Attribute data to match, passing NULL skips data matching.
- num_matches Number matches, passing 0 makes it unlimited.
- func Callback function.
- **user_data** Data to pass to the callback.

static inline void **bt_gatt_foreach_attr**(uint16_t start_handle, uint16_t end_handle, *bt_gatt_attr_func_t* func, void *user_data)

Attribute iterator.

Iterate attributes in the given range.

Parameters

- start_handle Start handle.
- end_handle End handle.
- **func** Callback function.
- **user_data** Data to pass to the callback.

struct bt_gatt_attr *bt_gatt_attr_next(const struct bt_gatt_attr *attr)

Iterate to the next attribute.

Iterate to the next attribute following a given attribute.

Parameters

• attr - Current Attribute.

Returns

The next attribute or NULL if it cannot be found.

struct bt_gatt_attr *bt_gatt_find_by_uuid(const struct bt_gatt_attr *attr, uint16_t attr_count, const struct bt_uuid *uuid)

Find Attribute by UUID.

Find the attribute with the matching UUID. To limit the search to a service set the attr to the service attributes and the attr count to the service attribute count .

Parameters

- attr Pointer to an attribute that serves as the starting point for the search of a match for the UUID. Passing NULL will search the entire range.
- attr_count The number of attributes from the starting point to search for a match for the UUID. Set to 0 to search until the end.
- uuid UUID to match.

uint16_t bt_gatt_attr_get_handle(const struct bt_gatt_attr *attr)

Get Attribute handle.

Parameters

• attr – Attribute object.

Returns

Handle of the corresponding attribute or zero if the attribute could not be found.

uint16_t bt_gatt_attr_value_handle(const struct bt_gatt_attr *attr)

Get the handle of the characteristic value descriptor.

Note: The user data of the attribute must of type bt gatt chrc.

Parameters

• attr – A Characteristic Attribute.

Returns

the handle of the corresponding Characteristic Value. The value will be zero (the invalid handle) if attr was not a characteristic attribute.

ssize_t **bt_gatt_attr_read**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t buf_len, uint16_t offset, const void *value, uint16_t value_len)

Generic Read Attribute value helper.

Read attribute value from local database storing the result into buffer.

Parameters

- conn Connection object.
- attr Attribute to read.
- **buf** Buffer to store the value.
- buf_len Buffer length.
- offset Start offset.
- **value** Attribute value.
- value_len Length of the attribute value.

Returns

number of bytes read in case of success or negative values in case of error.

ssize_t **bt_gatt_attr_read_service**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t len, uint16_t offset)

Read Service Attribute helper.

Read service attribute value from local database storing the result into buffer after encoding it.

Note: Only use this with attributes which user_data is a *bt_uuid*.

Parameters

- conn Connection object.
- attr Attribute to read.
- **buf** Buffer to store the value read.
- len Buffer length.
- offset Start offset.

Returns

number of bytes read in case of success or negative values in case of error.

ssize_t bt_gatt_attr_read_included(struct bt_conn *conn, const struct bt_gatt_attr *attr, void *buf, uint16 t len, uint16 t offset)

Read Include Attribute helper.

Read include service attribute value from local database storing the result into buffer after encoding it.

Note: Only use this with attributes which user_data is a *bt_gatt_include*.

Parameters

- conn Connection object.
- attr Attribute to read.
- **buf** Buffer to store the value read.
- **len** Buffer length.
- offset Start offset.

Returns

number of bytes read in case of success or negative values in case of error.

ssize_t **bt_gatt_attr_read_chrc**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t len, uint16_t offset)

Read Characteristic Attribute helper.

Read characteristic attribute value from local database storing the result into buffer after encoding it.

Note: Only use this with attributes which user_data is a *bt_gatt_chrc*.

Parameters

- **conn** Connection object.
- attr Attribute to read.
- **buf** Buffer to store the value read.
- **len** Buffer length.
- **offset** Start offset.

Returns

number of bytes read in case of success or negative values in case of error.

ssize_t **bt_gatt_attr_read_ccc**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t len, uint16_t offset)

Read Client Characteristic Configuration Attribute helper.

Read CCC attribute value from local database storing the result into buffer after encoding it.

Note: Only use this with attributes which user_data is a _bt_gatt_ccc.

Parameters

- **conn** Connection object.
- attr Attribute to read.
- **buf** Buffer to store the value read.
- len Buffer length.
- offset Start offset.

Returns

number of bytes read in case of success or negative values in case of error.

ssize_t **bt_gatt_attr_write_ccc**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, const void *buf, uint16_t len, uint16_t offset, uint8_t flags)

Write Client Characteristic Configuration Attribute helper.

Write value in the buffer into CCC attribute.

Note: Only use this with attributes which user_data is a _bt_gatt_ccc.

Parameters

- **conn** Connection object.
- attr Attribute to read.
- **buf** Buffer to store the value read.
- len Buffer length.
- offset Start offset.
- **flags** Write flags.

Returns

number of bytes written in case of success or negative values in case of error.

ssize_t **bt_gatt_attr_read_cep**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t len, uint16_t offset)

Read Characteristic Extended Properties Attribute helper.

Read CEP attribute value from local database storing the result into buffer after encoding it.

Note: Only use this with attributes which user_data is a *bt_gatt_cep*.

Parameters

- conn Connection object
- attr Attribute to read
- **buf** Buffer to store the value read
- len Buffer length
- offset Start offset

Returns

number of bytes read in case of success or negative values in case of error.

ssize_t **bt_gatt_attr_read_cud**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t len, uint16_t offset)

Read Characteristic User Description Descriptor Attribute helper.

Read CUD attribute value from local database storing the result into buffer after encoding it.

Note: Only use this with attributes which user_data is a NULL-terminated C string.

Parameters

- conn Connection object
- attr Attribute to read
- **buf** Buffer to store the value read
- len Buffer length
- offset Start offset

Returns

number of bytes read in case of success or negative values in case of error.

ssize_t **bt_gatt_attr_read_cpf**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t len, uint16_t offset)

Read Characteristic Presentation format Descriptor Attribute helper.

Read CPF attribute value from local database storing the result into buffer after encoding it.

Note: Only use this with attributes which user_data is a bt_gatt_pf.

Parameters

- conn Connection object
- attr Attribute to read
- **buf** Buffer to store the value read
- len Buffer length
- offset Start offset

Returns

number of bytes read in case of success or negative values in case of error.

int **bt_gatt_notify_cb**(struct bt_conn *conn, struct bt_gatt_notify_params *params)

Notify attribute value change.

This function works in the same way as *bt_gatt_notify*. With the addition that after sending the notification the callback function will be called.

The callback is run from System Workqueue context. When called from the System Workqueue context this API will not wait for resources for the callback but instead return an error. The number of pending callbacks can be increased with the @kconfig{CONFIG BT CONN TX MAX} option.

Alternatively it is possible to notify by UUID by setting it on the parameters, when using this method the attribute if provided is used as the start range when looking up for possible matches.

Parameters

- **conn** Connection object.
- **params** Notification parameters.

Returns

0 in case of success or negative value in case of error.

int **bt_gatt_notify_multiple**(struct bt_conn *conn, uint16_t num_params, struct *bt_gatt_notify_params* params[])

Send multiple notifications in a single PDU.

The GATT Server will send a single ATT_MULTIPLE_HANDLE_VALUE_NTF PDU containing all the notifications passed to this API.

All params must have the same func and user_data (due to implementation limitation). But func(user_data) will be invoked for each parameter.

As this API may block to wait for Bluetooth Host resources, it is not recommended to call it from a cooperative thread or a Bluetooth callback.

The peer's GATT Client must write to this device's Client Supported Features attribute and set the bit for Multiple Handle Value Notifications before this API can be used.

Only use this API to force the use of the ATT_MULTIPLE_HANDLE_VALUE_NTF PDU. For standard applications, bt_gatt_notify_cb is preferred, as it will use this PDU if supported and automatically fallback to ATT_HANDLE_VALUE_NTF when not supported by the peer.

This API has an additional limitation: it only accepts valid attribute references and not UUIDs like bt_gatt_notify and bt_gatt_notify_cb.

Parameters

- **conn** Target client. Notifying all connected clients by passing NULL is not yet supported, please use bt_gatt_notify instead.
- **num_params** Element count of params array. Has to be greater than 1.
- params Array of notification parameters. It is okay to free this after calling this function.

Return values

- **0** Success. The PDU is queued for sending.
- -EINVAL -
 - One of the attribute handles is invalid.
 - Only one parameter was passed. This API expects 2 or more.
 - Not all func were equal or not all user_data were equal.
 - One of the characteristics is not notifiable.
 - An UUID was passed in one of the parameters.
- -ERANGE -
 - The notifications cannot all fit in a single ATT MULTIPLE HANDLE VALUE NTF.
 - They exceed the MTU of all open ATT bearers.
- **-EPERM** The connection has a lower security level than required by one of the attributes.
- **-EOPNOTSUPP** The peer hasn't yet communicated that it supports this PDU type.

static inline int **bt_gatt_notify**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, const void *data, uint16 t len)

Notify attribute value change.

Send notification of attribute value change, if connection is NULL notify all peer that have notification enabled via CCC otherwise do a direct notification only the given connection.

The attribute object on the parameters can be the so called Characteristic Declaration, which is usually declared with BT_GATT_CHARACTERISTIC followed by BT_GATT_CCC, or the Characteristic Value Declaration which is automatically created after the Characteristic Declaration when using BT GATT CHARACTERISTIC.

Parameters

- conn Connection object.
- attr Characteristic or Characteristic Value attribute.
- data Pointer to Attribute data.
- **len** Attribute value length.

Returns

0 in case of success or negative value in case of error.

static inline int **bt_gatt_notify_uuid**(struct bt_conn *conn, const struct *bt_uuid* *uuid, const struct *bt_gatt_attr* *attr, const void *data, uint16_t len)

Notify attribute value change by UUID.

Send notification of attribute value change, if connection is NULL notify all peer that have notification enabled via CCC otherwise do a direct notification only on the given connection.

The attribute object is the starting point for the search of the UUID.

Parameters

- conn Connection object.
- **uuid** The UUID. If the server contains multiple services with the same UUID, then the first occurrence, starting from the attr given, is used.
- attr Pointer to an attribute that serves as the starting point for the search of a match for the LIHID
- data Pointer to Attribute data.
- **len** Attribute value length.

Returns

0 in case of success or negative value in case of error.

int **bt_gatt_indicate**(struct bt_conn *conn, struct bt_gatt_indicate_params *params)

Indicate attribute value change.

Send an indication of attribute value change. if connection is NULL indicate all peer that have notification enabled via CCC otherwise do a direct indication only the given connection.

The attribute object on the parameters can be the so called Characteristic Declaration, which is usually declared with BT_GATT_CHARACTERISTIC followed by BT_GATT_CCC, or the Characteristic Value Declaration which is automatically created after the Characteristic Declaration when using BT GATT CHARACTERISTIC.

Alternatively it is possible to indicate by UUID by setting it on the parameters, when using this method the attribute if provided is used as the start range when looking up for possible matches.

Note: This procedure is asynchronous therefore the parameters need to remains valid while it is active. The procedure is active until the destroy callback is run.

Parameters

- conn Connection object.
- params Indicate parameters.

Returns

0 in case of success or negative value in case of error.

bool bt_gatt_is_subscribed(struct bt_conn *conn, const struct bt_gatt_attr *attr, uint16_t ccc_type)

Check if connection have subscribed to attribute.

Check if connection has subscribed to attribute value change.

The attribute object can be the so called Characteristic Declaration, which is usually declared with BT_GATT_CHARACTERISTIC followed by BT_GATT_CCC, or the Characteristic Value Declaration which is automatically created after the Characteristic Declaration when using BT_GATT_CHARACTERISTIC, or the Client Characteristic Configuration Descriptor (CCCD) which is created by BT_GATT_CCC.

Parameters

- conn Connection object.
- attr Attribute object.
- ccc_type The subscription type, BT_GATT_CCC_NOTIFY and/or BT_GATT_CCC_INDICATE.

Returns

true if the attribute object has been subscribed.

uint16_t bt_gatt_get_mtu(struct bt_conn *conn)

Get ATT MTU for a connection.

Get negotiated ATT connection MTU, note that this does not equal the largest amount of attribute data that can be transferred within a single packet.

Parameters

• conn – Connection object.

Returns

MTU in bytes

struct bt_gatt_ccc_cfg

#include <gatt.h> GATT CCC configuration entry.

Public Members

```
uint8 t id
         Local identity, BT_ID_DEFAULT in most cases.
     bt_addr_le_t peer
         Remote peer address.
     uint16_t value
         Configuration value.
struct _bt_gatt_ccc
     #include <gatt.h> Internal representation of CCC value
     Public Members
     struct bt_gatt_ccc_cfg cfg[0]
         Configuration for each connection
     uint16_t value
         Highest value of all connected peer's subscriptions
     void (*cfg_changed)(const struct bt_gatt_attr *attr, uint16_t value)
         CCC attribute changed callback.
              Param attr
                The attribute that's changed value
              Param value
                New value
     ssize_t (*cfg_write)(struct bt_conn *conn, const struct bt_gatt_attr *attr, uint16_t value)
         CCC attribute write validation callback.
              Param conn
                The connection that is requesting to write
              Param attr
                The attribute that's being written
              Param value
                CCC value to write
              Return
                Number of bytes to write, or in case of an error BT\_GATT\_ERR() with a specific error code.
     bool (*cfg_match)(struct bt_conn *conn, const struct bt_gatt_attr *attr)
         CCC attribute match handler.
         Indicate if it is OK to send a notification or indication to the subscriber.
              Param conn
                The connection that is being checked
              Param attr
                The attribute that's being checked
```

Return

true if application has approved notification/indication, false if application does not approve.

struct bt_gatt_notify_params

#include <gatt.h>

Public Members

```
const struct bt_uuid *uuid
```

Notification Attribute UUID type.

Optional, use to search for an attribute with matching UUID when the attribute object pointer is not known.

```
const struct bt_gatt_attr *attr
```

Notification Attribute object.

Optional if uuid is provided, in this case it will be used as start range to search for the attribute with the given UUID.

const void *data

Notification Value data

uint16_t len

Notification Value length

```
bt_gatt_complete_func_t func
```

Notification Value callback

void *user_data

Notification Value callback user data

struct bt_gatt_indicate_params

#include < gatt.h > GATT Indicate Value parameters.

Public Members

```
const struct bt_uuid *uuid
```

Indicate Attribute UUID type.

Optional, use to search for an attribute with matching UUID when the attribute object pointer is not known.

```
const struct bt_gatt_attr *attr
```

Indicate Attribute object.

Optional if uuid is provided, in this case it will be used as start range to search for the attribute with the given UUID.

```
bt_gatt_indicate_func_t func
Indicate Value callback

bt_gatt_indicate_params_destroy_t destroy
Indicate operation complete callback

const void *data
Indicate Value data

uint16_t len
Indicate Value length

uint8_t _ref
```

1.6.1.2 GATT Client

group bt_gatt_client

Typedefs

typedef uint8_t (*bt_gatt_discover_func_t)(struct bt_conn *conn, const struct bt_gatt_attr *attr, struct bt_gatt_discover_params *params)

Discover attribute callback function.

Private reference counter

If discovery procedure has completed this callback will be called with attr set to NULL. This will not happen if procedure was stopped by returning BT_GATT_ITER_STOP.

The attribute object as well as its UUID and value objects are temporary and must be copied to in order to cache its information. Only the following fields of the attribute contains valid information:

- uuid UUID representing the type of attribute.
- handle Handle in the remote database.
- user_data The value of the attribute, if the discovery type maps to an ATT operation that provides this
 information. NULL otherwise. See below.

The effective type of attr->user_data is determined by params. Note that the fields params->type and params->uuid are left unchanged by the discovery procedure.

params->type	params->uuid	Type of attr->user_data
BT_GATT_DISCOVER_PRIMARY	any	bt_gatt_service_val
BT_GATT_DISCOVER_SECONDARY	any	bt_gatt_service_val
BT_GATT_DISCOVER_INCLUDE	any	bt_gatt_include
BT_GATT_DISCOVER_CHARACTERISTIC	any	bt_gatt_chrc
BT_GATT_DISCOVER_STD_CHAR_DESC	BT_UUID_GATT_CEP	bt_gatt_cep
BT_GATT_DISCOVER_STD_CHAR_DESC	BT_UUID_GATT_CCC	bt_gatt_ccc
BT_GATT_DISCOVER_STD_CHAR_DESC	BT_UUID_GATT_SCC	bt_gatt_scc
BT_GATT_DISCOVER_STD_CHAR_DESC	BT_UUID_GATT_CPF	bt_gatt_cpf
BT_GATT_DISCOVER_DESCRIPTOR	any	NULL
BT_GATT_DISCOVER_ATTRIBUTE	any	NULL

Also consider if using read-by-type instead of discovery is more convenient. See bt_gatt_read with $bt_gatt_read_params::handle_count$ set to 0.

Param conn

Connection object.

Param attr

Attribute found, or NULL if not found.

Param params

Discovery parameters given.

Return

BT_GATT_ITER_CONTINUE to continue discovery procedure.

Return

BT_GATT_ITER_STOP to stop discovery procedure.

typedef uint8_t (***bt_gatt_read_func_t**)(struct bt_conn *conn, uint8_t err, struct *bt_gatt_read_params* *params, const void *data, uint16_t length)

Read callback function.

Param conn

Connection object.

Param err

ATT error code.

Param params

Read parameters used.

Param data

Attribute value data. NULL means read has completed.

Param length

Attribute value length.

Return

BT_GATT_ITER_CONTINUE if should continue to the next attribute.

Return

BT_GATT_ITER_STOP to stop.

typedef void (*bt_gatt_write_func_t)(struct bt_conn *conn, uint8_t err, struct bt_gatt_write_params *params)

Write callback function.

Param conn

Connection object.

Param err

ATT error code.

Param params

Write parameters used.

typedef uint8_t (***bt_gatt_notify_func_t**)(struct bt_conn *conn, struct *bt_gatt_subscribe_params* *params, const void *data, uint16_t length)

Notification callback function.

In the case of an empty notification, the data pointer will be non-NULL while the length will be 0, which is due to the special case where a data NULL pointer means unsubscribed.

Param conn

Connection object. May be NULL, indicating that the peer is being unpaired

Param params

Subscription parameters.

Param data

Attribute value data. If NULL then subscription was removed.

Param length

Attribute value length.

Return

BT_GATT_ITER_CONTINUE to continue receiving value notifications. BT_GATT_ITER_STOP to unsubscribe from value notifications.

typedef void (*bt_gatt_subscribe_func_t)(struct bt_conn *conn, uint8_t err, struct bt_gatt_subscribe_params *params)

Subscription callback function.

Param conn

Connection object.

Param err

ATT error code.

Param params

Subscription parameters used.

Enums

enum [anonymous]

GATT Discover types

Values:

enumerator BT_GATT_DISCOVER_PRIMARY

Discover Primary Services.

enumerator BT_GATT_DISCOVER_SECONDARY

Discover Secondary Services.

enumerator BT_GATT_DISCOVER_INCLUDE

Discover Included Services.

enumerator BT_GATT_DISCOVER_CHARACTERISTIC

Discover Characteristic Values.

Discover Characteristic Value and its properties.

enumerator BT_GATT_DISCOVER_DESCRIPTOR

Discover Descriptors.

Discover Attributes which are not services or characteristics.

enumerator BT_GATT_DISCOVER_ATTRIBUTE

Discover Attributes.

Discover Attributes of any type.

enumerator BT_GATT_DISCOVER_STD_CHAR_DESC

Discover standard characteristic descriptor values.

Discover standard characteristic descriptor values and their properties.

Supported descriptors:

- Characteristic Extended Properties
- Client Characteristic Configuration
- Server Characteristic Configuration
- Characteristic Presentation Format

enum [anonymous]

Subscription flags

Values:

enumerator BT_GATT_SUBSCRIBE_FLAG_VOLATILE

Persistence flag.

If set, indicates that the subscription is not saved on the GATT server side. Therefore, upon disconnection, the subscription will be automatically removed from the client's subscriptions list and when the client reconnects, it will have to issue a new subscription.

enumerator BT_GATT_SUBSCRIBE_FLAG_NO_RESUB

No resubscribe flag.

By **default** when BT_GATT_SUBSCRIBE_FLAG_VOLATILE is unset, the subscription will be automatically renewed when the client reconnects, as a workaround **for** GATT servers that **do** not persist subscriptions.

This flag will disable the automatic resubscription. It is useful if the application layer knows that the GATT server remembers subscriptions from previous connections and wants to avoid renewing the subscriptions.

enumerator BT_GATT_SUBSCRIBE_FLAG_WRITE_PENDING

Write pending flag.

If set, indicates write operation is pending waiting remote end to respond.

enumerator BT_GATT_SUBSCRIBE_NUM_FLAGS

Functions

int **bt_gatt_exchange_mtu**(struct bt_conn *conn, struct bt_gatt_exchange_params *params)

Exchange MTU.

This client procedure can be used to set the MTU to the maximum possible size the buffers can hold.

The Response comes in callback params->func. The callback is run from the BT RX thread. params must remain valid until start of callback.

This function will block while the ATT request queue is full, except when called from the BT RX thread, as this would cause a deadlock.

Note: Shall only be used once per connection.

Parameters

- **conn** Connection object.
- params Exchange MTU parameters.

Return values

- **0** Successfully queued request. Will call params->func on resolution.
- **-ENOMEM** ATT request queue is full and blocking would cause deadlock. Allow a pending request to resolve before retrying, or call this function outside the BT RX thread to get blocking behavior. Queue size is controlled by @kconfig{CONFIG_BT_L2CAP_TX_BUF_COUNT}.
- **-EALREADY** The MTU exchange procedure has been already performed.

int **bt_gatt_discover**(struct bt_conn *conn, struct bt_gatt_discover_params *params)

GATT Discover function.

This procedure is used by a client to discover attributes on a server.

Primary Service Discovery: Procedure allows to discover primary services either by Discover All Primary Services or Discover Primary Services by Service UUID. Include Service Discovery: Procedure allows to discover all Include Services within specified range. Characteristic Discovery: Procedure allows to discover all characteristics within specified handle range as well as discover characteristics with specified UUID. Descriptors Discovery: Procedure allows to discover all characteristic descriptors within specified range.

For each attribute found the callback is called which can then decide whether to continue discovering or stop.

The Response comes in callback params->func. The callback is run from the BT RX thread. params must remain valid until start of callback where iter attr is NULL or callback will return BT_GATT_ITER_STOP.

This function will block while the ATT request queue is full, except when called from the BT RX thread, as this would cause a deadlock.

Parameters

- **conn** Connection object.
- params Discover parameters.

Return values

- **0** Successfully queued request. Will call params->func on resolution.
- **-ENOMEM** ATT request queue is full and blocking would cause deadlock. Allow a pending request to resolve before retrying, or call this function outside the BT RX thread to get blocking behavior. Queue size is controlled by @kconfig{CONFIG_BT_L2CAP_TX_BUF_COUNT}.

int **bt_gatt_read**(struct bt conn *conn, struct bt gatt read params *params)

Read Attribute Value by handle.

This procedure read the attribute value and return it to the callback.

When reading attributes by UUID the callback can be called multiple times depending on how many instances of given the UUID exists with the start_handle being updated for each instance.

If an instance does contain a long value which cannot be read entirely the caller will need to read the remaining data separately using the handle and offset.

The Response comes in callback params->func. The callback is run from the BT RX thread. params must remain valid until start of callback.

This function will block while the ATT request queue is full, except when called from the BT RX thread, as this would cause a deadlock.

- **conn** Connection object.
- params Read parameters.

Return values

- **0** Successfully queued request. Will call params->func on resolution.
- **-ENOMEM** ATT request queue is full and blocking would cause deadlock. Allow a pending request to resolve before retrying, or call this function outside the BT RX thread to get blocking behavior. Queue size is controlled by @kconfig{CONFIG_BT_L2CAP_TX_BUF_COUNT}.

int **bt_gatt_write**(struct bt_conn *conn, struct bt_gatt_write_params *params)

Write Attribute Value by handle.

The Response comes in callback params->func. The callback is run from the BT RX thread. params must remain valid until start of callback.

This function will block while the ATT request queue is full, except when called from Bluetooth event context. When called from Bluetooth context, this function will instead instead return -ENOMEM if it would block to avoid a deadlock.

Parameters

- **conn** Connection object.
- params Write parameters.

Return values

- **0** Successfully queued request. Will call params->func on resolution.
- **-ENOMEM** ATT request queue is full and blocking would cause deadlock. Allow a pending request to resolve before retrying, or call this function outside Bluetooth event context to get blocking behavior. Queue size is controlled by @kconfig{CONFIG_BT_L2CAP_TX_BUF_COUNT}.

int **bt_gatt_write_without_response_cb**(struct bt_conn *conn, uint16_t handle, const void *data, uint16_t length, bool sign, bt_gatt_complete_func_t func, void *user data)

Write Attribute Value by handle without response with callback.

This function works in the same way as *bt_gatt_write_without_response*. With the addition that after sending the write the callback function will be called.

The callback is run from System Workqueue context. When called from the System Workqueue context this API will not wait for resources for the callback but instead return an error. The number of pending callbacks can be increased with the @kconfig{CONFIG_BT_CONN_TX_MAX} option.

This function will block while the ATT request queue is full, except when called from the BT RX thread, as this would cause a deadlock.

Note: By using a callback it also disable the internal flow control which would prevent sending multiple commands without waiting for their transmissions to complete, so if that is required the caller shall not submit more data until the callback is called.

- **conn** Connection object.
- handle Attribute handle.
- data Data to be written.
- **length** Data length.
- **sign** Whether to sign data
- **func** Transmission complete callback.
- user_data User data to be passed back to callback.

Return values

- **0** Successfully queued request.
- **-ENOMEM** ATT request queue is full and blocking would cause deadlock. Allow a pending request to resolve before retrying, or call this function outside the BT RX thread to get blocking behavior. Queue size is controlled by @kconfig{CONFIG_BT_L2CAP_TX_BUF_COUNT}.

static inline int **bt_gatt_write_without_response**(struct bt_conn *conn, uint16_t handle, const void *data, uint16_t length, bool sign)

Write Attribute Value by handle without response.

This procedure write the attribute value without requiring an acknowledgment that the write was successfully performed

This function will block while the ATT request queue is full, except when called from the BT RX thread, as this would cause a deadlock.

Parameters

- conn Connection object.
- handle Attribute handle.
- data Data to be written.
- length Data length.
- sign Whether to sign data

Return values

- **0** Successfully queued request.
- **-ENOMEM** ATT request queue is full and blocking would cause deadlock. Allow a pending request to resolve before retrying, or call this function outside the BT RX thread to get blocking behavior. Queue size is controlled by @kconfig{CONFIG_BT_L2CAP_TX_BUF_COUNT}.

int **bt_gatt_subscribe**(struct bt_conn *conn, struct bt_gatt_subscribe_params *params)

Subscribe Attribute Value Notification.

This procedure subscribe to value notification using the Client Characteristic Configuration handle. If notification received subscribe value callback is called to return notified value. One may then decide whether to unsubscribe directly from this callback. Notification callback with NULL data will not be called if subscription was removed by this method.

The Response comes in callback params->func. The callback is run from the BT RX thread. params must remain valid until start of callback. The Notification callback params->notify is also called from the BT RX thread.

This function will block while the ATT request queue is full, except when called from the BT RX thread, as this would cause a deadlock.

Note: Notifications are asynchronous therefore the parameters need to remain valid while subscribed.

Parameters

- **conn** Connection object.
- params Subscribe parameters.

Return values

- **0** Successfully queued request. Will call params->write on resolution.
- **-ENOMEM** ATT request queue is full and blocking would cause deadlock. Allow a pending request to resolve before retrying, or call this function outside the BT RX thread to get blocking behavior. Queue size is controlled by @kconfig{CONFIG_BT_L2CAP_TX_BUF_COUNT}.

int **bt_gatt_resubscribe**(uint8_t id, const *bt_addr_le_t* *peer, struct *bt_gatt_subscribe_params* *params)

Resubscribe Attribute Value Notification subscription.

Resubscribe to Attribute Value Notification when already subscribed from a previous connection. The GATT server will remember subscription from previous connections when bonded, so resubscribing can be done without performing a new subscribe procedure after a power cycle.

Note: Notifications are asynchronous therefore the parameters need to remain valid while subscribed.

Parameters

- id Local identity (in most cases BT_ID_DEFAULT).
- **peer** Remote address.
- params Subscribe parameters.

Returns

0 in case of success or negative value in case of error.

int **bt_gatt_unsubscribe**(struct bt conn *conn, struct bt gatt subscribe params *params)

Unsubscribe Attribute Value Notification.

This procedure unsubscribe to value notification using the Client Characteristic Configuration handle. Notification callback with NULL data will be called if subscription was removed by this call, until then the parameters cannot be reused.

The Response comes in callback params->func. The callback is run from the BT RX thread.

This function will block while the ATT request queue is full, except when called from the BT RX thread, as this would cause a deadlock.

- conn Connection object.
- params Subscribe parameters.

Return values

- **0** Successfully queued request. Will call params->write on resolution.
- **-ENOMEM** ATT request queue is full and blocking would cause deadlock. Allow a pending request to resolve before retrying, or call this function outside the BT RX thread to get blocking behavior. Queue size is controlled by @kconfig{CONFIG_BT_L2CAP_TX_BUF_COUNT}.

void bt_gatt_cancel(struct bt conn *conn, void *params)

Try to cancel the first pending request identified by params.

This function does not release params for reuse. The usual callbacks for the request still apply. A successful cancel simulates a BT_ATT_ERR_UNLIKELY response from the server.

This function can cancel the following request functions:

- bt_gatt_exchange_mtu
- bt_gatt_discover
- bt_gatt_read
- bt_gatt_write
- bt_gatt_subscribe
- bt_gatt_unsubscribe

Parameters

- **conn** The connection the request was issued on.
- **params** The address **params** used in the request function call.

struct bt_gatt_exchange_params

#include <gatt.h> GATT Exchange MTU parameters.

Public Members

```
void (*func)(struct bt_conn *conn, uint8_t err, struct bt_gatt_exchange_params *params)
Response callback
```

struct bt_gatt_discover_params

#include <gatt.h> GATT Discover Attributes parameters.

Public Members

```
const struct bt_uuid *uuid

Discover UUID type

bt_gatt_discover_func_t func
```

Discover attribute callback

uint16_t end_handle

Discover end handle

uint8_t type

Discover type

struct bt_gatt_read_params

#include < gatt.h > GATT Read parameters.

Public Members

```
bt_gatt_read_func_t func
```

Read attribute callback.

size_t handle_count

If equals to 1 single.handle and single.offset are used. If greater than 1 multiple.handles are used. If equals to 0 by_uuid is used for Read Using Characteristic UUID.

struct bt_gatt_write_params

#include <gatt.h> GATT Write parameters.

Public Members

```
bt_gatt_write_func_t func
```

Response callback

uint16_t handle

Attribute handle

uint16 t offset

Attribute data offset

const void *data

Data to be written

uint16_t length

Length of the data

struct bt_gatt_subscribe_params

#include <gatt.h> GATT Subscribe parameters.

Public Functions

ATOMIC_DEFINE (flags, BT_GATT_SUBSCRIBE_NUM_FLAGS)

Subscription flags

Public Members

```
bt_gatt_notify_func_t notify
```

Notification value callback

bt_gatt_subscribe_func_t subscribe

Subscribe CCC write request response callback If given, called with the subscription parameters given when subscribing

```
bt_gatt_write_func_t write
```

Deprecated:

uint16_t value_handle

Subscribe value handle

uint16_t ccc_handle

Subscribe CCC handle

uint16_t value

Subscribe value

bt_security_t min_security

Minimum required security for received notification. Notifications and indications received over a connection with a lower security level are silently discarded.

```
union __unnamed61__
```

Public Members

struct $bt_gatt_discover_params$.[anonymous].[anonymous] $_$ included

uint16_t start_handle

Discover start handle

struct _included

Public Members

uint16_t attr_handle

Include service attribute declaration handle

uint16_t start_handle

Included service start handle

uint16_t end_handle

Included service end handle

union __unnamed64__

Public Members

```
struct bt_gatt_read_params.[anonymous].[anonymous] single
struct bt_gatt_read_params.[anonymous].[anonymous] multiple
struct bt_gatt_read_params.[anonymous].[anonymous] by_uuid
```

struct single

Public Members

uint16 t handle

Attribute handle.

uint16_t offset

Attribute data offset.

struct multiple

Public Members

uint16_t *handles

Attribute handles to read with Read Multiple Characteristic Values.

bool variable

If true use Read Multiple Variable Length Characteristic Values procedure. The values of the set of attributes may be of variable or unknown length. If false use Read Multiple Characteristic Values procedure. The values of the set of attributes must be of a known fixed length, with the exception of the last value that can have a variable length.

struct by_uuid

Public Members

```
uint16_t start_handle
```

First requested handle number.

uint16_t end_handle

Last requested handle number.

const struct bt_uuid *uuid

2 or 16 octet UUID.

1.7 Hands Free Profile (HFP)

1.7.1 API Reference

```
group bt_hfp
```

Hands Free AG Profile (HFP AG)

Hands Free Profile (HFP)

Defines

HFP_HF_DIGIT_ARRAY_SIZE

HFP_HF_MAX_OPERATOR_NAME_LEN

HFP_HF_CMD_OK

HFP_HF_CMD_ERROR

 ${\tt HFP_HF_CMD_CME_ERROR}$

HFP_HF_CMD_UNKNOWN_ERROR

Typedefs

```
typedef enum _hf_volume_type_t hf_volume_type_t
    bt hfp ag volume type
typedef enum _hfp_ag_call_status_t hfp_ag_call_status_t
     bt hf call status
typedef struct <u>_hfp_ag_get_config</u> hfp_ag_get_config
    bt ag configure setting
typedef struct _hfp_ag_cind_t hfp_ag_cind_t
     bt hf call status
typedef int (*bt_hfp_ag_discover_callback)(struct bt_conn *conn, uint8_t channel)
     hfp_ag discover callback function
         Param conn
             Pointer to bt_conn structure.
         Param channel
             the server channel of hfp ag
typedef enum _hf_volume_type_t hf_volume_type_t
     bt hfp ag volume type
typedef enum _hf_multiparty_call_option_t hf_multiparty_call_option_t
     bt hfp ag volume type
typedef struct _hf_waiting_call_state_t hf_waiting_call_state_t
Enums
enum _hf_volume_type_t
     bt hfp ag volume type
     Values:
     enumerator hf_volume_type_speaker
     enumerator hf_volume_type_mic
     enumerator hf_volume_type_speaker
     enumerator hf_volume_type_mic
```

```
enum _hfp_ag_call_status_t
    bt hf call status
    Values:
    enumerator hfp_ag_call_call_end
    enumerator hfp_ag_call_call_active
    enumerator hfp_ag_call_call_incoming
    enumerator hfp_ag_call_call_outgoing
enum hfp_ag_call_setup_status_t
    bt ag call setup status
    Values:
    enumerator HFP_AG_CALL_SETUP_STATUS_IDLE
    enumerator HFP_AG_CALL_SETUP_STATUS_INCOMING
    enumerator HFP_AG_CALL_SETUP_STATUS_OUTGOING_DIALING
    enumerator HFP_AG_CALL_SETUP_STATUS_OUTGOING_ALERTING
enum bt_hfp_hf_at_cmd
    Values:
    enumerator BT_HFP_HF_ATA
    enumerator BT_HFP_HF_AT_CHUP
enum _hf_volume_type_t
    bt hfp ag volume type
    Values:
    enumerator hf_volume_type_speaker
    enumerator hf_volume_type_mic
    enumerator hf_volume_type_speaker
    enumerator hf_volume_type_mic
```

```
enum _hf_multiparty_call_option_t
```

bt hfp ag volume type

Values:

enumerator hf_multiparty_call_option_one

enumerator hf_multiparty_call_option_two

enumerator **hf_multiparty_call_option_three**

enumerator hf_multiparty_call_option_four

enumerator hf_multiparty_call_option_five

Functions

int bt_hfp_ag_init(void)

BT HFP AG Initialize

This function called to initialize bt hfp ag

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_deinit(void)

BT HFP AG Deinitialize

This function called to initialize bt hfp ag

Returns

0 in case of success or otherwise in case of error.

```
int bt_hfp_ag_connect(struct bt_conn *conn, hfp_ag_get_config *config, struct bt_hfp_ag_cb *cb, struct bt_hfp_ag **phfp_ag)
```

hfp ag Connect.

This function is to be called after the conn parameter is obtained by performing a GAP procedure. The API is to be used to establish hfp ag connection between devices. This function only establish RFCOM connection. After connection success, the callback that is registered by bt_hfp_ag_register_connect_callback is called.

Parameters

- **conn** Pointer to bt_conn structure.
- **config** bt hfp ag congigure
- **cb** bt hfp ag congigure
- **phfp_ag** Pointer to pointer of bt hfp ag Connection object

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_disconnect(struct bt_hfp_ag *hfp_ag)

hfp ag DisConnect.

This function is to be called after the conn parameter is obtained by performing a GAP procedure. The API is to be used to establish hfp ag connection between devices. This function only establish RFCOM connection. After connection success, the callback that is registered by bt_hfp_ag_register_connect_callback is called.

Parameters

• phfp_ag – pointer to bt hfp ag connection object

Returns

0 in case of success or otherwise in case of error.

int **bt_hfp_ag_discover**(struct bt_conn *conn, bt_hfp_ag_discover_callback discoverCallback)

hfp ag discover

This function is to be called after the conn parameter is obtained by performing a GAP procedure. The API is to be used to establish hfp ag connection between devices.

Parameters

- phfp_ag pointer to bt hfp ag connection object
- discoverCallback pointer to discover callback function, defined in application

Returns

0 in case of success or otherwise in case of error.

void bt_hfp_ag_open_audio(struct bt_hfp_ag *hfp_ag, uint8_t codec)

hfp ag open audio for codec

This function is to open audio codec for hfp function

Parameters

• **phfp_ag** – pointer to bt hfp ag connection object

```
void bt_hfp_ag_close_audio(struct bt_hfp_ag *hfp_ag)
```

hfp ag close audio for codec

This function is to close audio codec for hfp function

Parameters

• **phfp_ag** – pointer to bt hfp ag connection object

int bt_hfp_ag_register_supp_features(struct bt_hfp_ag *hfp_ag, uint32_t supported_features)

configure hfp ag supported features.

if the function is not called, will use default supported features

hfp ag to configure hfp ag supported features

This function is to be configure hfp ag supported features. If the function is not called, will use default supported features

Parameters

- phfp_ag pointer to bt hfp ag connection object
- **supported_features** supported features of hfp ag

Returns

0 in case of success or otherwise in case of error.

uint32_t bt_hfp_ag_get_peer_supp_features(struct bt_hfp_ag *hfp_ag)

hfp ag to get peer hfp hp support feautes

This function is to be called to get hfp hp support feautes

Parameters

• phfp_ag – pointer to bt hfp ag connection object

Returns

the supported feature of hfp ag

int bt_hfp_ag_register_cind_features(struct bt_hfp_ag *hfp_ag, char *cind)

hfp ag to configure hfp ag supported features

This function is to be configure hfp ag cind setting supported features. If the function is not called, will use default supported features

Parameters

- **phfp_ag** pointer to bt hfp ag connection object
- cind pointer to hfp ag cwind

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_disable_voice_recognition(struct bt_hfp_ag *hfp_ag)

hfp ag to disable voice recognition

This function is o disabl voice recognition

Parameters

• **phfp_ag** – pointer to bt hfp ag connection object

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_enable_voice_recognition(struct bt_hfp_ag *hfp_ag)

hfp ag to enable voice recognition

This function is used to enable voice recognition

Parameters

• **phfp_ag** – pointer to bt hfp ag Connection object

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_disable_voice_ecnr(struct bt_hfp_ag *hfp_ag)

hfp ag to disable noise reduction and echo canceling

This function is o noise reduction and echo canceling

Parameters

• **phfp_ag** – pointer to bt hfp ag connection object

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_enable_voice_ecnr(struct bt_hfp_ag *hfp_ag)

hfp ag to enable noise reduction and echo canceling

This function is to enable noise reduction and echo canceling

Parameters

• phfp_ag – pointer to bt hfp ag connection object

Returns

0 in case of success or otherwise in case of error.

int **bt_hfp_ag_set_cops**(struct bt_hfp_ag *hfp_ag, char *name)

hfp ag to set the name of the currently selected Network operator by AG

This function is to set the name of the currently selected Network operator by AG

Parameters

- **phfp_ag** pointer to bt hfp ag connection object
- name the name of the currently selected Network operator by AG

Returns

0 in case of success or otherwise in case of error.

int **bt_hfp_ag_set_volume_control**(struct bt_hfp_ag *hfp_ag, hf_volume_type_t type, int value)

hfp ag to set volue of hfp hp

This function is to set volue of hfp hp

Parameters

- **phfp_ag** pointer to bt hfp ag connection object
- **type** the hfp hp volume type
- value the volue of volume

Returns

0 in case of success or otherwise in case of error.

int **bt_hfp_ag_set_inband_ring_tone**(struct bt_hfp_ag *hfp_ag, int value)

hfp ag to set inband ring tone support

This function is to set inband ring tone support

Parameters

- phfp_ag pointer to bt hfp ag connection object
- **value** the inband ring tone type

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_set_phnum_tag(struct bt_hfp_ag *hfp_ag, char *name)

hfp ag to set the attach a phone number to a voice Tag

This function is to set the attach a phone number to a voice Tag

- **phfp_ag** pointer to bt hfp ag connection object
- name the name of attach a phone number to a voice Tag

Returns

0 in case of success or otherwise in case of error.

void **bt_hfp_ag_call_status_pl**(struct bt_hfp_ag *hfp_ag, hfp_ag_call_status_t status)

hfp ag to set the call status

This function is to set the call status

Parameters

- phfp_ag pointer to bt hfp ag connection object
- **status** the ag call status

Returns

0 in case of success or otherwise in case of error.

int **bt_hfp_ag_handle_btrh**(struct bt_hfp_ag *hfp_ag, uint8_t option)

hfp ag to set the status of the "Response and Hold" state of the AG.

This function is to hfp ag to set the status of the "Response and Hold" state of the AG.

Parameters

- **phfp_ag** pointer to bt hfp ag connection object
- option the hfp ag "Response and Hold" state of the AG

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_handle_indicator_enable(struct bt_hfp_ag *hfp_ag, uint8_t index, uint8_t enable)

hfp ag to set the status of the "Response and Hold" state of the AG.

This function is to hfp ag to set the status of the "Response and Hold" state of the AG.

Parameters

- phfp_ag pointer to bt hfp ag connection object
- item 1 for Enhanced Safety, 2 for Battery Level
- enable 1 for enable

Returns

0 in case of success or otherwise in case of error.

void bt_hfp_ag_send_callring(struct bt hfp ag *hfp ag)

hfp ag to set ring command to hfp hp

This function is hfp ag to set ring command to hfp hp

Parameters

• phfp_ag – pointer to bt hfp ag connection object

int bt_hfp_ag_send_call_indicator(struct bt_hfp_ag *hfp_ag, uint8_t value)

hfp ag set call indicator to hfp hp

This function is hfp ag set call indicator to hfp hp

- **phfp_ag** pointer to bt hfp ag connection object
- value value of call indicator

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_callsetup_indicator(struct bt_hfp_ag *hfp_ag, uint8_t value)

hfp ag set call setup indicator to hfp hp

This function is hfp ag set call setup indicator to hfp hp

Parameters

- phfp_ag pointer to bt hfp ag connection object
- value value of call setup indicator

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_service_indicator(struct bt_hfp_ag *hfp_ag, uint8_t value)

hfp ag set service indicator to hfp hp

This function is hfp ag set service indicator to hfp hp

Parameters

- **phfp_ag** pointer to bt hfp ag connection object
- value value of service indicator

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_signal_indicator(struct bt_hfp_ag *hfp_ag, uint8_t value)

hfp ag set signal strength indicator to hfp hp

This function is hfp ag set signal strength indicator to hfp hp

Parameters

- phfp_ag pointer to bt hfp ag connection object
- **value** value of signal strength indicator

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_roaming_indicator(struct bt_hfp_ag *hfp_ag, uint8_t value)

hfp ag set roaming indicator to hfp hp

This function is hfp ag set roaming indicator to hfp hp

Parameters

- phfp_ag pointer to bt hfp ag connection object
- value value of roaming indicator

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_battery_indicator(struct bt_hfp_ag *hfp_ag, uint8_t value)

hfp ag set battery level indicator to hfp hp

This function is hfp ag set battery level indicator to hfp hp

Parameters

• phfp_ag – pointer to bt hfp ag connection object

• value – value of battery level indicator

Returns

0 in case of success or otherwise in case of error.

int bt_hfp_ag_send_ccwa_indicator(struct bt_hfp_ag *hfp_ag, char *number)

hfp ag set ccwa indicator to hfp hp

This function is hfp ag set ccwa indicator to hfp hp for mutiple call

Parameters

- **phfp_ag** pointer to bt hfp ag connection object
- value value of battery level indicator

Returns

0 in case of success or otherwise in case of error.

int **bt_hfp_ag_codec_selector**(struct bt_hfp_ag *hfp_ag, uint8_t value)

hfp ag set codec selector to hfp hp

This function is hfp ag set odec selector to hfp hp for codec negotiation

Parameters

- **phfp_ag** pointer to bt hfp ag connection object
- value value of codec selector

Returns

0 in case of success or otherwise in case of error.

```
int bt_hfp_ag_unknown_at_response(struct bt_hfp_ag *hfp_ag, uint8_t *unknow_at_rsp, uint16_t unknow_at_rsplen)
```

hfp ag set unknown at command response to hfp fp

This function is hfp ag set unknown at command response to hfp fp, the command is not supported on hfp ag profile, Need handle the unknown command on application

Parameters

- phfp_ag pointer to bt hfp ag connection object
- unknow_at_rsp string of unknown at command response
- unknow_at_rsplen string length of unknown at command response

Returns

0 in case of success or otherwise in case of error.

int **bt_hfp_hf_register**(struct *bt_hfp_hf_cb* *cb)

Register HFP HF profile.

Register Handsfree profile callbacks to monitor the state and get the required HFP details to display.

Parameters

• **cb** – callback structure.

Returns

0 in case of success or negative value in case of error.

int **bt_hfp_hf_send_cmd**(struct bt_conn *conn, enum bt_hfp_hf_at_cmd cmd)

Handsfree client Send AT.

Send specific AT commands to handsfree client profile.

Parameters

- conn Connection object.
- **cmd** AT command to be sent.

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_start_voice_recognition(struct bt_conn *conn)

Handsfree to enable voice recognition in the AG.

Parameters

• conn – Connection object.

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_stop_voice_recognition(struct bt_conn *conn)

Handsfree to Disable voice recognition in the AG.

Parameters

• **conn** – Connection object.

Returns

0 in case of success or negative value in case of error.

int **bt_hfp_hf_volume_update**(struct bt_conn *conn, hf_volume_type_t type, int volume)

Handsfree to update Volume with AG.

Parameters

- conn Connection object.
- **type** volume control target, speaker or microphone
- volume gain of the speaker of microphone, ranges 0 to 15

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_dial(struct bt_conn *conn, const char *number)

Place a call with a specified number, if number is NULL, last called number is called. As a precondition to use this API, Service Level Connection shall exist with AG.

Parameters

- conn Connection object.
- number number string of the call. If NULL, the last number is called(aka re-dial)

Returns

0 in case of success or negative value in case of error.

int **bt_hfp_hf_dial_memory**(struct bt_conn *conn, int location)

Place a call with number specified by location(speed dial). As a precondition, to use this API, Service Level Connection shall exist with AG.

- **conn** Connection object.
- **location** location of the number in the memory

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_last_dial(struct bt_conn *conn)

Place a call with number specified by location(speed dial). As a precondition, to use this API, Service Level connection shall exist with AG.

Parameters

• **conn** – Connection object.

Returns

0 in case of success or negative value in case of error.

int **bt_hfp_hf_multiparty_call_option**(struct bt_conn *conn, hf_multiparty_call_option_t option)

Place a call with number specified by location(speed dial). As a precondition, to use this API, Service Level Connection shall exist with AG.

Parameters

• **conn** – Connection object.

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_enable_clip_notification(struct bt_conn *conn)

Enable the CLIP notification.

Parameters

• **conn** – Connection object.

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_disable_clip_notification(struct bt_conn *conn)

Disable the CLIP notification.

Parameters

• **conn** – Connection object.

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_enable_call_waiting_notification(struct bt_conn *conn)

Enable the call waiting notification.

Parameters

• **conn** – Connection object.

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_disable_call_waiting_notification(struct bt_conn *conn)

Disable the call waiting notification.

Parameters

• **conn** – Connection object.

Returns

0 in case of success or negative value in case of error.

int bt_hfp_hf_get_last_voice_tag_number(struct bt_conn *conn)

Get the last voice tag nubmer, the mubmer will be fill callback event voicetag_phnum.

Parameters

• conn – Connection object.

Returns

0 in case of success or negative value in case of error.

struct _hfp_ag_get_config

#include <hfp_ag.h> bt ag configure setting

struct _hfp_ag_cind_t

#include <hfp_ag.h> bt hf call status

struct bt_hfp_ag_cb

#include <hfp_ag.h> HFP profile application callback.

Public Members

```
void (*connected)(struct bt_hfp_ag *hfp_ag)
```

AG connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param hfp_ag

bt hfp ag Connection object.

void (*disconnected)(struct bt_hfp_ag *hfp_ag)

AG disconnected callback to application

If this callback is provided it will be called whenever the connection gets disconnected, including when a connection gets rejected or cancelled or any error in SLC establisment.

Param hfp_ag

bt hfp ag Connection object.

void (*volume_control)(struct bt_hfp_ag *hfp_ag, hf_volume_type_t type, int value)

AG volume_control Callback

This callback provides volume_control indicator value to the application

Param hfp ag

bt hfp ag Connection object.

Param type

the hfp volue type, for speaker or mic.

Param value

service indicator value received from the AG.

void (*hfu_brsf)(struct bt_hfp_ag *hfp_ag, uint32_t value)

AG remote support feature Callback

This callback provides the remote hfp unit supported feature

Param hfp_ag

bt hfp ag Connection object.

Param value

call indicator he remote hfp unit supported feature received from the AG.

void (*ata_response)(struct bt_hfp_ag *hfp_ag)

AG remote call is answered Callback

This callback provides call indicator the call is answered to the application

Param hfp_ag

bt hfp ag Connection object.

void (*chup_response)(struct bt_hfp_ag *hfp_ag)

AG remote call is answered Callback

This callback provides call indicator the call is rejected to the application

Param hfp_ag

bt hfp ag Connection object.

void (*dial)(struct bt_hfp_ag *hfp_ag, char *number)

AG remote call is answered Callback

This callback provides call indicator the call is rejected to the application

Param hfp_ag

bt hfp ag Connection object.

Param value

call information.

void (*brva)(struct bt_hfp_ag *hfp_ag, uint32_t value)

AG remote voice recognition activation Callback

This callback provides call indicator voice recognition activation of peer HF to the application

Param hfp_ag

bt hfp ag Connection object.

Param value

voice recognition activation information.

void (*nrec)(struct bt_hfp_ag *hfp_ag, uint32_t value)

AG remote noise reduction and echo canceling Callback

This callback provides call indicator voice recognition activation of peer HF to the application

Param hfp ag

bt hfp ag Connection object.

Param value

Noise Reduction and Echo Canceling information.

void (*codec_negotiate)(struct bt_hfp_ag *hfp_ag, uint32_t value)

AG remote codec negotiate Callback

This callback provides codec negotiate information of peer HF to the application

Param hfp ag

bt hfp ag Connection object.

Param value

codec index of peer HF.

void (*chld)(struct bt_hfp_ag *hfp_ag, uint8_t option, uint8_t index)

AG multiparty call status indicator Callback

This callback provides multiparty call status indicator Callback of peer HF to the application

Param hfp_ag

bt hfp ag Connection object.

Param option

Multiparty call option.

Param index

Multiparty call index.

void (*unkown_at)(struct bt_hfp_ag *hfp_ag, char *value, uint32_t length)

AG unkown at Callback

This callback provides AG unkown at value to the application, the unkown at command could be handled by application

Param hfp_ag

bt hfp ag Connection object.

Param value

unknow AT string buffer

Param length

unknow AT string length.

struct bt_hfp_hf_cmd_complete

#include <hfp_hf.h> HFP HF Command completion field.

struct _hf_waiting_call_state_t

#include <hfp_hf.h>

struct bt_hfp_hf_cb

#include <hfp_hf.h> HFP profile application callback.

Public Members

void (*connected)(struct bt_conn *conn)

HF connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param conn

Connection object.

void (*disconnected)(struct bt_conn *conn)

HF disconnected callback to application

If this callback is provided it will be called whenever the connection gets disconnected, including when a connection gets rejected or cancelled or any error in SLC establishment.

Param conn

Connection object.

void (*service)(struct bt_conn *conn, uint32_t value)

HF indicator Callback

This callback provides service indicator value to the application

Param conn

Connection object.

Param value

service indicator value received from the AG.

void (*call)(struct bt_conn *conn, uint32_t value)

HF indicator Callback

This callback provides call indicator value to the application

Param conn

Connection object.

Param value

call indicator value received from the AG.

void (*call_setup)(struct bt_conn *conn, uint32_t value)

HF indicator Callback

This callback provides call setup indicator value to the application

Param conn

Connection object.

Param value

call setup indicator value received from the AG.

void (*call_held)(struct bt_conn *conn, uint32_t value)

HF indicator Callback

This callback provides call held indicator value to the application

Param conn

Connection object.

Param value

call held indicator value received from the AG.

void (*signal)(struct bt_conn *conn, uint32_t value)

HF indicator Callback

This callback provides signal indicator value to the application

Param conn

Connection object.

Param value

signal indicator value received from the AG.

void (*roam)(struct bt_conn *conn, uint32_t value)

HF indicator Callback

This callback provides roaming indicator value to the application

Param conn

Connection object.

Param value

roaming indicator value received from the AG.

void (*battery)(struct bt_conn *conn, uint32_t value)

HF indicator Callback

This callback battery service indicator value to the application

Param conn

Connection object.

Param value

battery indicator value received from the AG.

void (*voicetag_phnum)(struct bt_conn *conn, char *number)

HF voice tag phnum indicator Callback

This callback voice tag phnum indicator to the application

Param conn

Connection object.

Param voice

tag phnum value received from the AG.

void (*call_phnum)(struct bt_conn *conn, char *number)

HF calling phone number string indication callback to application

If this callback is provided it will be called whenever there is an incoming call and bt_hfp_hf_enable_clip_notification is called.

Param conn

Connection object.

Param char

to phone number string.

void (*waiting_call)(struct bt_conn *conn, hf_waiting_call_state_t *wcs)

HF waiting call indication callback to application

If this callback is provided it will be called in waiting call state

Param conn

Connection object.

Param pointer

to waiting call state information.

void (*ring_indication)(struct bt_conn *conn)

HF incoming call Ring indication callback to application

If this callback is provided it will be called whenever there is an incoming call.

Param conn

Connection object.

void (*cmd_complete_cb)(struct bt_conn *conn, struct bt_hfp_hf_cmd_complete *cmd)

HF notify command completed callback to application

The command sent from the application is notified about its status

Param conn

Connection object.

Param cmd

structure contains status of the command including cme.

1.8 Logical Link Control and Adaptation Protocol (L2CAP)

L2CAP layer enables connection-oriented channels which can be enable with the configuration option: CONFIG_BT_L2CAP_DYNAMIC_CHANNEL. This channels support segmentation and reassembly transparently, they also support credit based flow control making it suitable for data streams.

Channels instances are represented by the bt_12cap_chan struct which contains the callbacks in the $bt_12cap_chan_ops$ struct to inform when the channel has been connected, disconnected or when the encryption has changed. In addition to that it also contains the recv callback which is called whenever an incoming data has been received. Data received this way can be marked as processed by returning 0 or using $bt_12cap_chan_recv_complete()$ API if processing is asynchronous.

Note: The recv callback is called directly from RX Thread thus it is not recommended to block for long periods of time.

For sending data the bt_12cap_chan_send() API can be used noting that it may block if no credits are available, and resuming as soon as more credits are available.

Servers can be registered using $bt_12cap_server_register()$ API passing the bt_12cap_server struct which informs what psm it should listen to, the required security level sec_level , and the callback accept which is called to authorize incoming connection requests and allocate channel instances.

Client channels can be initiated with use of $bt_12cap_chan_connect()$ API and can be disconnected with the $bt_12cap_chan_disconnect()$ API. Note that the later can also disconnect channel instances created by servers.

1.8.1 API Reference

group bt_12cap

L2CAP.

Defines

BT_L2CAP_HDR_SIZE

L2CAP PDU header size, used for buffer size calculations

BT_L2CAP_TX_MTU

Maximum Transmission Unit (MTU) for an outgoing L2CAP PDU.

BT_L2CAP_RX_MTU

Maximum Transmission Unit (MTU) for an incoming L2CAP PDU.

BT_L2CAP_BUF_SIZE(mtu)

Helper to calculate needed buffer size for L2CAP PDUs. Useful for creating buffer pools.

• mtu – Needed L2CAP PDU MTU.

Returns

Needed buffer size to match the requested L2CAP PDU MTU.

BT_L2CAP_SDU_HDR_SIZE

L2CAP SDU header size, used for buffer size calculations

BT_L2CAP_SDU_TX_MTU

Maximum Transmission Unit for an unsegmented outgoing L2CAP SDU.

The Maximum Transmission Unit for an outgoing L2CAP SDU when sent without segmentation, i.e. a single L2CAP SDU will fit inside a single L2CAP PDU.

The MTU for outgoing L2CAP SDUs with segmentation is defined by the size of the application buffer pool.

BT_L2CAP_SDU_RX_MTU

Maximum Transmission Unit for an unsegmented incoming L2CAP SDU.

The Maximum Transmission Unit for an incoming L2CAP SDU when sent without segmentation, i.e. a single L2CAP SDU will fit inside a single L2CAP PDU.

The MTU for incoming L2CAP SDUs with segmentation is defined by the size of the application buffer pool. The application will have to define an alloc_buf callback for the channel in order to support receiving segmented L2CAP SDUs.

BT_L2CAP_SDU_BUF_SIZE(mtu)

Helper to calculate needed buffer size for L2CAP SDUs. Useful for creating buffer pools.

Parameters

• mtu – Required BT_L2CAP_*_SDU.

Returns

Needed buffer size to match the requested L2CAP SDU MTU.

BT_L2CAP_LE_CHAN(ch)

Helper macro getting container object of type $bt_l2cap_le_chan$ address having the same container chan member address as object in question.

Parameters

• **_ch** – Address of object of *bt_l2cap_chan* type

Returns

Address of in memory *bt_l2cap_le_chan* object type containing the address of in question object.

BT_L2CAP_CFG_OPT_MTU

configuration parameter options type

BT_L2CAP_CFG_OPT_FUSH_TIMEOUT

BT_L2CAP_CFG_OPT_QOS

BT_L2CAP_CFG_OPT_RETRANS_FC

BT_L2CAP_CFG_OPT_FCS

BT_L2CAP_CFG_OPT_EXT_FLOW_SPEC

BT_L2CAP_CFG_OPT_EXT_WIN_SIZE

BT_L2CAP_MODE_BASIC

L2CAP Operation Modes

BT_L2CAP_MODE_RTM

BT_L2CAP_MODE_FC

BT_L2CAP_MODE_ERTM

BT_L2CAP_MODE_SM

BT_L2CAP_FEATURE_FC

L2CAP Extended Feature Mask values

BT_L2CAP_FEATURE_RTM

BT_L2CAP_FEATURE_QOS

BT_L2CAP_FEATURE_ERTM

BT_L2CAP_FEATURE_SM

BT_L2CAP_FEATURE_FCS

BT_L2CAP_FEATURE_EFS_BR_EDR

BT_L2CAP_FEATURE_FIXED_CHANNELS

BT_L2CAP_FEATURE_EXTENDED_WINDOW_SIZE

BT_L2CAP_FEATURE_UCD

BT_L2CAP_CHAN_SEND_RESERVE

Headroom needed for outgoing L2CAP PDUs.

BT_L2CAP_SDU_CHAN_SEND_RESERVE

Headroom needed for outgoing L2CAP SDUs.

Typedefs

```
typedef\ void\ (\textbf{*bt\_12cap\_chan\_destroy\_t}) (struct\ \textit{bt\_12cap\_chan}\ \textbf{*chan})
```

Channel destroy callback.

Param chan

Channel object.

typedef enum bt_l2cap_chan_state_t

typedef enum bt_l2cap_chan_status bt_l2cap_chan_status_t

Enums

enum bt_12cap_chan_state

Life-span states of L2CAP CoC channel.

Used only by internal APIs dealing with setting channel to proper state depending on operational context.

Values:

enumerator BT_L2CAP_DISCONNECTED

Channel disconnected

enumerator BT_L2CAP_CONNECTING

Channel in connecting state

enumerator BT_L2CAP_CONFIG

Channel in config state, BR/EDR specific

enumerator BT_L2CAP_CONNECTED

Channel ready for upper layer traffic on it

enumerator BT_L2CAP_DISCONNECTING

Channel in disconnecting state

enum bt_12cap_chan_status

Status of L2CAP channel.

Values:

enumerator BT_L2CAP_STATUS_OUT

Channel output status

enumerator BT_L2CAP_STATUS_SHUTDOWN

Channel shutdown status.

Once **this** status is notified it means the channel will no longer be able to transmit or receive data.

enumerator BT_L2CAP_STATUS_ENCRYPT_PENDING

Channel encryption pending status.

enumerator BT_L2CAP_NUM_STATUS

Functions

int **bt_l2cap_server_register**(struct *bt_l2cap_server* *server)

Register L2CAP server.

Register L2CAP server for a PSM, each new connection is authorized using the accept() callback which in case of success shall allocate the channel structure to be used by the new connection.

For fixed, SIG-assigned PSMs (in the range 0x0001-0x007f) the PSM should be assigned to server->psm before calling this API. For dynamic PSMs (in the range 0x0080-0x00ff) server->psm may be pre-set to a given value (this is however not recommended) or be left as 0, in which case upon return a newly allocated value will have been assigned to it. For dynamically allocated values the expectation is that it's exposed through a GATT service, and that's how L2CAP clients discover how to connect to the server.

Parameters

• **server** – Server structure.

Returns

0 in case of success or negative value in case of error.

int **bt_l2cap_br_server_register**(struct *bt_l2cap_server* *server)

Register L2CAP server on BR/EDR oriented connection.

Register L2CAP server for a PSM, each new connection is authorized using the accept() callback which in case of success shall allocate the channel structure to be used by the new connection.

Parameters

• **server** – Server structure.

Returns

0 in case of success or negative value in case of error.

int bt_l2cap_ecred_chan_connect(struct bt_conn *conn, struct bt_l2cap_chan **chans, uint16_t psm)

Connect Enhanced Credit Based L2CAP channels.

Connect up to 5 L2CAP channels by PSM, once the connection is completed each channel connected() callback will be called. If the connection is rejected disconnected() callback is called instead.

- conn Connection object.
- **chans** Array of channel objects.
- psm Channel PSM to connect to.

Returns

0 in case of success or negative value in case of error.

int bt_l2cap_ecred_chan_reconfigure(struct bt_l2cap_chan **chans, uint16_t mtu)

Reconfigure Enhanced Credit Based L2CAP channels.

Reconfigure up to 5 L2CAP channels. Channels must be from the same bt_conn. Once reconfiguration is completed each channel reconfigured() callback will be called. MTU cannot be decreased on any of provided channels.

Parameters

- **chans** Array of channel objects. Null-terminated. Elements after the first 5 are silently ignored.
- mtu Channel MTU to reconfigure to.

Returns

0 in case of success or negative value in case of error.

int **bt_12cap_chan_connect**(struct bt_conn *conn, struct bt_l2cap_chan *chan, uint16_t psm)

Connect L2CAP channel.

Connect L2CAP channel by PSM, once the connection is completed channel connected() callback will be called. If the connection is rejected disconnected() callback is called instead. Channel object passed (over an address of it) as second parameter shouldn't be instantiated in application as standalone. Instead of, application should create transport dedicated L2CAP objects, i.e. type of $bt_l2cap_le_chan$ for LE and/or type of $bt_l2cap_br_chan$ for BR/EDR. Then pass to this API the location (address) of bt_l2cap_chan type object which is a member of both transport dedicated objects.

Parameters

- conn Connection object.
- chan Channel object.
- psm Channel PSM to connect to.

Returns

0 in case of success or negative value in case of error.

int bt_l2cap_chan_disconnect(struct bt_l2cap_chan *chan)

Disconnect L2CAP channel.

Disconnect L2CAP channel, if the connection is pending it will be canceled and as a result the channel disconnected() callback is called. Regarding to input parameter, to get details see reference description to $bt_l2cap_chan_connect()$ API above.

Parameters

• **chan** – Channel object.

Returns

0 in case of success or negative value in case of error.

int **bt_12cap_chan_send**(struct *bt_12cap_chan* *chan, struct net_buf *buf)

Send data to L2CAP channel.

Send data from buffer to the channel. If credits are not available, buf will be queued and sent as and when credits are received from peer. Regarding to first input parameter, to get details see reference description to bt l2cap chan connect() API above.

When sending L2CAP data over an BR/EDR connection the application is sending L2CAP PDUs. The application is required to have reserved *BT_L2CAP_CHAN_SEND_RESERVE* bytes in the buffer before

sending. The application should use the *BT_L2CAP_BUF_SIZE()* helper to correctly size the buffers for the for the outgoing buffer pool.

When sending L2CAP data over an LE connection the application is sending L2CAP SDUs. The application can optionally reserve $BT_L2CAP_SDU_CHAN_SEND_RESERVE$ bytes in the buffer before sending. By reserving bytes in the buffer the stack can use this buffer as a segment directly, otherwise it will have to allocate a new segment for the first segment. If the application is reserving the bytes it should use the $BT_L2CAP_BUF_SIZE()$ helper to correctly size the buffers for the for the outgoing buffer pool. When segmenting an L2CAP SDU into L2CAP PDUs the stack will first attempt to allocate buffers from the original buffer pool of the L2CAP SDU before using the stacks own buffer pool.

Note: Buffer ownership is transferred to the stack in case of success, in case of an error the caller retains the ownership of the buffer.

Returns

Bytes sent in case of success or negative value in case of error.

int bt_l2cap_chan_recv_complete(struct bt_l2cap_chan *chan, struct net_buf *buf)

Complete receiving L2CAP channel data.

Complete the reception of incoming data. This shall only be called if the channel recv callback has returned -EINPROGRESS to process some incoming data. The buffer shall contain the original user_data as that is used for storing the credits/segments used by the packet.

Parameters

- **chan** Channel object.
- **buf** Buffer containing the data.

Returns

0 in case of success or negative value in case of error.

struct bt_12cap_chan

#include <l2cap.h> L2CAP Channel structure.

Public Members

struct bt_conn *conn

Channel connection reference

const struct bt_l2cap_chan_ops *ops

Channel operations reference

struct bt_l2cap_le_endpoint

#include <l2cap.h> LE L2CAP Endpoint structure.

```
uint16_t cid
```

Endpoint Channel Identifier (CID)

uint16_t mtu

Endpoint Maximum Transmission Unit

uint16 t mps

Endpoint Maximum PDU payload Size

uint16 tinit_credits

Endpoint initial credits

atomic_t credits

Endpoint credits

struct bt_12cap_le_chan

#include <l2cap.h> LE L2CAP Channel structure.

Public Members

struct bt_l2cap_chan chan

Common L2CAP channel reference object

struct bt_l2cap_le_endpoint **rx**

Channel Receiving Endpoint.

If the application has set an alloc_buf channel callback for the channel to support receiving segmented L2CAP SDUs the application should initialize the MTU of the Receiving Endpoint. Otherwise the MTU of the receiving endpoint will be initialized to $BT_L2CAP_SDU_RX_MTU$ by the stack.

uint16_t pending_rx_mtu

Pending RX MTU on ECFC reconfigure, used internally by stack

struct bt_l2cap_le_endpoint **tx**

Channel Transmission Endpoint

struct k_fifo tx_queue

Channel Transmission queue

struct net_buf *tx_buf

Channel Pending Transmission buffer

struct k_work tx_work

Channel Transmission work

```
struct net_buf *_sdu
        Segment SDU packet from upper layer
struct bt_12cap_br_endpoint
    #include <l2cap.h> BREDR L2CAP Endpoint structure.
    Public Members
    uint16_t cid
        Endpoint Channel Identifier (CID)
    uint16_t mtu
        Endpoint Maximum Transmission Unit
struct bt_12cap_br_chan
    #include <l2cap.h> BREDR L2CAP Channel structure.
    Public Members
    struct bt 12cap chan chan
        Common L2CAP channel reference object
    struct bt_l2cap_br_endpoint rx
        Channel Receiving Endpoint
    struct bt_l2cap_br_endpoint tx
        Channel Transmission Endpoint
    uint16 t psm
        Remote PSM to be connected
    uint8_t ident
        Helps match request context during CoC
struct bt_12cap_qos
    #include <l2cap.h> QUALITY OF SERVICE (QOS) OPTION
struct bt_12cap_retrans_fc
    #include <l2cap.h> RETRANSMISSION AND FLOW CONTROL OPTION
struct bt_12cap_ext_flow_spec
```

#include <l2cap.h> EXTENDED FLOW SPECIFICATION OPTION

struct bt_12cap_cfg_options

#include <l2cap.h> L2CAP configuration parameter options.

struct bt_12cap_chan_ops

#include <l2cap.h> L2CAP Channel operations structure.

Public Members

```
void (*connected)(struct bt_l2cap_chan *chan)
```

Channel connected callback.

If this callback is provided it will be called whenever the connection completes.

Param chan

The channel that has been connected

```
void (*disconnected)(struct bt_l2cap_chan *chan)
```

Channel disconnected callback.

If this callback is provided it will be called whenever the channel is disconnected, including when a connection gets rejected.

Param chan

The channel that has been Disconnected

```
void (*encrypt_change)(struct bt_l2cap_chan *chan, uint8_t hci_status)
```

Channel encrypt_change callback.

If this callback is provided it will be called whenever the security level changed (indirectly link encryption done) or authentication procedure fails. In both cases security initiator and responder got the final status (HCI status) passed by related to encryption and authentication events from local host's controller.

Param chan

The channel which has made encryption status changed.

Param status

HCI status of performed security procedure caused by channel security requirements. The value is populated by HCI layer and set to 0 when success and to non-zero (reference to HCI Error Codes) when security/authentication failed.

```
struct net_buf *(*alloc_seg)(struct bt_l2cap_chan *chan)
```

Channel alloc_seg callback.

If this callback is provided the channel will use it to allocate buffers to store segments. This avoids wasting big SDU buffers with potentially much smaller PDUs. If this callback is supplied, it must return a valid buffer.

Param chan

The channel requesting a buffer.

Return

Allocated buffer.

```
struct net_buf *(*alloc_buf)(struct bt_l2cap_chan *chan)
```

Channel alloc buf callback.

If this callback is provided the channel will use it to allocate buffers to store incoming data. Channels that requires segmentation must set this callback. If the application has not set a callback the L2CAP SDU MTU will be truncated to BT L2CAP SDU RX MTU.

Param chan

The channel requesting a buffer.

Return

Allocated buffer.

int (***recv**)(struct *bt_l2cap_chan* *chan, struct net_buf *buf)

Channel recv callback.

Param chan

The channel receiving data.

Param buf

Buffer containing incoming data.

Return

0 in case of success or negative value in case of error.

Return

-EINPROGRESS in case where user has to confirm once the data has been processed by calling *bt_l2cap_chan_recv_complete* passing back the buffer received with its original user_data which contains the number of segments/credits used by the packet.

```
void (*sent)(struct bt_l2cap_chan *chan)
```

Channel sent callback.

If this callback is provided it will be called whenever a SDU has been completely sent.

Param chan

The channel which has sent data.

```
void (*status)(struct bt_l2cap_chan *chan, atomic_t *status)
```

Channel status callback.

If this callback is provided it will be called whenever the channel status changes.

Param chan

The channel which status changed

Param status

The channel status

```
void (*reconfigured)(struct bt_l2cap_chan *chan)
```

Channel reconfigured callback.

If this callback is provided it will be called whenever peer or local device requested reconfiguration. Application may check updated MTU and MPS values by inspecting chan->le endpoints.

Param chan

The channel which was reconfigured

struct bt_12cap_server

#include <l2cap.h> L2CAP Server structure.

uint16_t psm

Server PSM.

Possible values: 0 A dynamic value will be auto-allocated when bt_l2cap_server_register() is called.

0x0001-0x007f Standard, Bluetooth SIG-assigned fixed values.

0x0080-0x00ff Dynamically allocated. May be pre-set by the application before server registration (not recommended however), or auto-allocated by the stack if the app gave 0 as the value.

bt_security_t sec_level

Required minimum security level

int (*accept)(struct bt_conn *conn, struct bt_l2cap_chan **chan)

Server accept callback.

This callback is called whenever a new incoming connection requires authorization.

Param conn

The connection that is requesting authorization

Param chan

Pointer to received the allocated channel

Return

0 in case of success or negative value in case of error.

Return

-ENOMEM if no available space for new channel.

Return

-EACCES if application did not authorize the connection.

Return

-EPERM if encryption key size is too short.

1.9 Serial Port Emulation (RFCOMM)

1.9.1 API Reference

group bt_rfcomm

RFCOMM.

Typedefs

typedef enum bt_rfcomm_role_t

Enums

```
enum [anonymous]

Values:

enumerator BT_RFCOMM_CHAN_HFP_HF

enumerator BT_RFCOMM_CHAN_HFP_AG

enumerator BT_RFCOMM_CHAN_HSP_AG

enumerator BT_RFCOMM_CHAN_HSP_HS

enumerator BT_RFCOMM_CHAN_SPP

enum bt_rfcomm_role

Role of RFCOMM session and dlc. Used only by internal APIs.

Values:

enumerator BT_RFCOMM_ROLE_ACCEPTOR

enumerator BT_RFCOMM_ROLE_INITIATOR
```

Functions

```
int bt_rfcomm_server_register(struct bt_rfcomm_server *server)
```

Register RFCOMM server.

```
(defined(CONFIG_BT_RFCOMM_ENABLE_CONTROL_CMD) && (CON-FIG_BT_RFCOMM_ENABLE_CONTROL_CMD > 0))
```

Register RFCOMM server for a channel, each new connection is authorized using the accept() callback which in case of success shall allocate the dlc structure to be used by the new connection.

Parameters

• **server** – Server structure.

Returns

0 in case of success or negative value in case of error.

```
int bt_rfcomm_dlc_connect(struct bt_conn *conn, struct bt_rfcomm_dlc *dlc, uint8_t channel)
```

Connect RFCOMM channel.

Connect RFCOMM dlc by channel, once the connection is completed dlc connected() callback will be called. If the connection is rejected disconnected() callback is called instead.

Parameters

- **conn** Connection object.
- dlc Dlc object.

• **channel** – Server channel to connect to.

Returns

0 in case of success or negative value in case of error.

```
int bt_rfcomm_dlc_send(struct bt_rfcomm_dlc *dlc, struct net_buf *buf)
```

Send data to RFCOMM.

Send data from buffer to the dlc. Length should be less than or equal to mtu.

Parameters

- dlc Dlc object.
- **buf** Data buffer.

Returns

Bytes sent in case of success or negative value in case of error.

```
int bt_rfcomm_dlc_disconnect(struct bt_rfcomm_dlc *dlc)
```

Disconnect RFCOMM dlc.

Disconnect RFCOMM dlc, if the connection is pending it will be canceled and as a result the dlc disconnected() callback is called.

Parameters

• dlc - Dlc object.

Returns

0 in case of success or negative value in case of error.

```
struct net_buf *bt_rfcomm_create_pdu(struct net_buf_pool *pool)
```

Allocate the buffer from pool after reserving head room for RFCOMM, L2CAP and ACL headers.

```
(defined(CONFIG_BT_RFCOMM_ENABLE_CONTROL_CMD) && (CONFIG_BT_RFCOMM_ENABLE_CONTROL_CMD > 0))
```

Parameters

• **pool** – Which pool to take the buffer from.

Returns

New buffer.

struct bt_rfcomm_dlc_ops

#include <rfcomm.h> RFCOMM DLC operations structure.

Public Members

```
void (*connected)(struct bt_rfcomm_dlc *dlc)
```

DLC connected callback

If this callback is provided it will be called whenever the connection completes.

Param dlc

The dlc that has been connected

```
void (*disconnected)(struct bt_rfcomm_dlc *dlc)
```

DLC disconnected callback

If this callback is provided it will be called whenever the dlc is disconnected, including when a connection gets rejected or cancelled (both incoming and outgoing)

Param dlc

The dlc that has been Disconnected

void (*recv)(struct bt_rfcomm_dlc *dlc, struct net_buf *buf)

DLC recv callback

Param dlc

The dlc receiving data.

Param buf

Buffer containing incoming data.

void (*sent)(struct bt_rfcomm_dlc *dlc, struct net_buf *buf)

DLC sent callback

Param dlc

The dlc receiving data.

Param buf

Buffer containing sending data.

struct bt_rfcomm_dlc

#include <rfcomm.h> RFCOMM DLC structure.

struct bt_rfcomm_server

#include <rfcomm.h>

Public Members

uint8_t channel

Server Channel

int (*accept)(struct bt_conn *conn, struct bt_rfcomm_dlc **dlc)

Server accept callback

This callback is called whenever a new incoming connection requires authorization.

Param conn

The connection that is requesting authorization

Param dlc

Pointer to received the allocated dlc

Return

0 in case of success or negative value in case of error.

1.10 Service Discovery Protocol (SDP)

1.10.1 API Reference

```
\mathit{group}~\mathtt{bt\_sdp}
```

Service Discovery Protocol (SDP)

Defines

BT_SDP_SDP_SERVER_SVCLASS

BT_SDP_BROWSE_GRP_DESC_SVCLASS

BT_SDP_PUBLIC_BROWSE_GROUP

BT_SDP_SERIAL_PORT_SVCLASS

BT_SDP_LAN_ACCESS_SVCLASS

BT_SDP_DIALUP_NET_SVCLASS

BT_SDP_IRMC_SYNC_SVCLASS

BT_SDP_OBEX_OBJPUSH_SVCLASS

BT_SDP_OBEX_FILETRANS_SVCLASS

BT_SDP_IRMC_SYNC_CMD_SVCLASS

BT_SDP_HEADSET_SVCLASS

BT_SDP_CORDLESS_TELEPHONY_SVCLASS

BT_SDP_AUDIO_SOURCE_SVCLASS

BT_SDP_AUDIO_SINK_SVCLASS

BT_SDP_AV_REMOTE_TARGET_SVCLASS

BT_SDP_ADVANCED_AUDIO_SVCLASS

BT_SDP_AV_REMOTE_SVCLASS

BT_SDP_AV_REMOTE_CONTROLLER_SVCLASS

BT_SDP_INTERCOM_SVCLASS

BT_SDP_FAX_SVCLASS

BT_SDP_HEADSET_AGW_SVCLASS

BT_SDP_WAP_SVCLASS

BT_SDP_WAP_CLIENT_SVCLASS

BT_SDP_PANU_SVCLASS

BT_SDP_NAP_SVCLASS

BT_SDP_GN_SVCLASS

BT_SDP_DIRECT_PRINTING_SVCLASS

BT_SDP_REFERENCE_PRINTING_SVCLASS

BT_SDP_IMAGING_SVCLASS

BT_SDP_IMAGING_RESPONDER_SVCLASS

BT_SDP_IMAGING_ARCHIVE_SVCLASS

BT_SDP_IMAGING_REFOBJS_SVCLASS

BT_SDP_HANDSFREE_SVCLASS

BT_SDP_HANDSFREE_AGW_SVCLASS

 ${\tt BT_SDP_DIRECT_PRT_REFOBJS_SVCLASS}$

BT_SDP_REFLECTED_UI_SVCLASS

BT_SDP_BASIC_PRINTING_SVCLASS

BT_SDP_PRINTING_STATUS_SVCLASS

BT_SDP_HID_SVCLASS

BT_SDP_HCR_SVCLASS

BT_SDP_HCR_PRINT_SVCLASS

BT_SDP_HCR_SCAN_SVCLASS

BT_SDP_CIP_SVCLASS

BT_SDP_VIDEO_CONF_GW_SVCLASS

BT_SDP_UDI_MT_SVCLASS

BT_SDP_UDI_TA_SVCLASS

BT_SDP_AV_SVCLASS

BT_SDP_SAP_SVCLASS

BT_SDP_PBAP_PCE_SVCLASS

BT_SDP_PBAP_PSE_SVCLASS

BT_SDP_PBAP_SVCLASS

BT_SDP_MAP_MSE_SVCLASS

BT_SDP_MAP_MCE_SVCLASS

BT_SDP_MAP_SVCLASS

BT_SDP_GNSS_SVCLASS

BT_SDP_GNSS_SERVER_SVCLASS

BT_SDP_MPS_SC_SVCLASS

BT_SDP_MPS_SVCLASS

BT_SDP_PNP_INFO_SVCLASS

BT_SDP_GENERIC_NETWORKING_SVCLASS

BT_SDP_GENERIC_FILETRANS_SVCLASS

BT_SDP_GENERIC_AUDIO_SVCLASS

BT_SDP_GENERIC_TELEPHONY_SVCLASS

BT_SDP_UPNP_SVCLASS

BT_SDP_UPNP_IP_SVCLASS

BT_SDP_UPNP_PAN_SVCLASS

BT_SDP_UPNP_LAP_SVCLASS

BT_SDP_UPNP_L2CAP_SVCLASS

BT_SDP_VIDEO_SOURCE_SVCLASS

BT_SDP_VIDEO_SINK_SVCLASS

BT_SDP_VIDEO_DISTRIBUTION_SVCLASS

BT_SDP_HDP_SVCLASS

BT_SDP_HDP_SOURCE_SVCLASS

BT_SDP_HDP_SINK_SVCLASS

BT_SDP_GENERIC_ACCESS_SVCLASS

BT_SDP_GENERIC_ATTRIB_SVCLASS

BT_SDP_APPLE_AGENT_SVCLASS

BT_SDP_SERVER_RECORD_HANDLE

BT_SDP_ATTR_RECORD_HANDLE

BT_SDP_ATTR_SVCLASS_ID_LIST

BT_SDP_ATTR_RECORD_STATE

BT_SDP_ATTR_SERVICE_ID

BT_SDP_ATTR_PROTO_DESC_LIST

BT_SDP_ATTR_BROWSE_GRP_LIST

BT_SDP_ATTR_LANG_BASE_ATTR_ID_LIST

BT_SDP_ATTR_SVCINFO_TTL

BT_SDP_ATTR_SERVICE_AVAILABILITY

BT_SDP_ATTR_PROFILE_DESC_LIST

BT_SDP_ATTR_DOC_URL

BT_SDP_ATTR_CLNT_EXEC_URL

BT_SDP_ATTR_ICON_URL

BT_SDP_ATTR_ADD_PROTO_DESC_LIST

BT_SDP_ATTR_GROUP_ID

BT_SDP_ATTR_IP_SUBNET

BT_SDP_ATTR_VERSION_NUM_LIST

BT_SDP_ATTR_SUPPORTED_FEATURES_LIST

BT_SDP_ATTR_GOEP_L2CAP_PSM

BT_SDP_ATTR_SVCDB_STATE

BT_SDP_ATTR_MPSD_SCENARIOS

BT_SDP_ATTR_MPMD_SCENARIOS

BT_SDP_ATTR_MPS_DEPENDENCIES

BT_SDP_ATTR_SERVICE_VERSION

BT_SDP_ATTR_EXTERNAL_NETWORK

BT_SDP_ATTR_SUPPORTED_DATA_STORES_LIST

BT_SDP_ATTR_DATA_EXCHANGE_SPEC

BT_SDP_ATTR_NETWORK

BT_SDP_ATTR_FAX_CLASS1_SUPPORT

BT_SDP_ATTR_REMOTE_AUDIO_VOLUME_CONTROL

BT_SDP_ATTR_MCAP_SUPPORTED_PROCEDURES

BT_SDP_ATTR_FAX_CLASS20_SUPPORT

BT_SDP_ATTR_SUPPORTED_FORMATS_LIST

BT_SDP_ATTR_FAX_CLASS2_SUPPORT

BT_SDP_ATTR_AUDIO_FEEDBACK_SUPPORT

BT_SDP_ATTR_NETWORK_ADDRESS

BT_SDP_ATTR_WAP_GATEWAY

BT_SDP_ATTR_HOMEPAGE_URL

BT_SDP_ATTR_WAP_STACK_TYPE

BT_SDP_ATTR_SECURITY_DESC

BT_SDP_ATTR_NET_ACCESS_TYPE

BT_SDP_ATTR_MAX_NET_ACCESSRATE

BT_SDP_ATTR_IP4_SUBNET

BT_SDP_ATTR_IP6_SUBNET

BT_SDP_ATTR_SUPPORTED_CAPABILITIES

BT_SDP_ATTR_SUPPORTED_FEATURES

BT_SDP_ATTR_SUPPORTED_FUNCTIONS

BT_SDP_ATTR_TOTAL_IMAGING_DATA_CAPACITY

BT_SDP_ATTR_SUPPORTED_REPOSITORIES

BT_SDP_ATTR_MAS_INSTANCE_ID

BT_SDP_ATTR_SUPPORTED_MESSAGE_TYPES

BT_SDP_ATTR_PBAP_SUPPORTED_FEATURES

BT_SDP_ATTR_MAP_SUPPORTED_FEATURES

BT_SDP_ATTR_SPECIFICATION_ID

BT_SDP_ATTR_VENDOR_ID

BT_SDP_ATTR_PRODUCT_ID

BT_SDP_ATTR_VERSION

BT_SDP_ATTR_PRIMARY_RECORD

BT_SDP_ATTR_VENDOR_ID_SOURCE

BT_SDP_ATTR_HID_DEVICE_RELEASE_NUMBER

BT_SDP_ATTR_HID_PARSER_VERSION

 ${\tt BT_SDP_ATTR_HID_DEVICE_SUBCLASS}$

BT_SDP_ATTR_HID_COUNTRY_CODE

BT_SDP_ATTR_HID_VIRTUAL_CABLE

BT_SDP_ATTR_HID_RECONNECT_INITIATE

BT_SDP_ATTR_HID_DESCRIPTOR_LIST

BT_SDP_ATTR_HID_LANG_ID_BASE_LIST

BT_SDP_ATTR_HID_SDP_DISABLE

BT_SDP_ATTR_HID_BATTERY_POWER

BT_SDP_ATTR_HID_REMOTE_WAKEUP

BT_SDP_ATTR_HID_PROFILE_VERSION

BT_SDP_ATTR_HID_SUPERVISION_TIMEOUT

BT_SDP_ATTR_HID_NORMALLY_CONNECTABLE

BT_SDP_ATTR_HID_BOOT_DEVICE

BT_SDP_PRIMARY_LANG_BASE

BT_SDP_ATTR_SVCNAME_PRIMARY

BT_SDP_ATTR_SVCDESC_PRIMARY

BT_SDP_ATTR_PROVNAME_PRIMARY

BT_SDP_DATA_NIL

BT_SDP_UINT8

BT_SDP_UINT16

BT_SDP_UINT32

BT_SDP_UINT64

BT_SDP_UINT128

BT_SDP_INT8

BT_SDP_INT16
BT_SDP_INT32
BT_SDP_INT64
BT_SDP_INT128
BT_SDP_UUID_UNSPEC
BT_SDP_UUID16
BT_SDP_UUID32
BT_SDP_UUID128
BT_SDP_TEXT_STR_UNSPEC
BT_SDP_TEXT_STR8
BT_SDP_TEXT_STR16
BT_SDP_TEXT_STR32
BT_SDP_BOOL
BT_SDP_SEQ_UNSPEC
BT_SDP_SEQ8
BT_SDP_SEQ16
BT_SDP_SEQ32
BT_SDP_ALT_UNSPEC
BT_SDP_ALT8
BT_SDP_ALT16
BT_SDP_ALT32

BT_SDP_URL_STR_UNSPEC

BT_SDP_URL_STR8

BT_SDP_URL_STR16

BT_SDP_URL_STR32

BT_SDP_TYPE_DESC_MASK

BT_SDP_SIZE_DESC_MASK

BT_SDP_SIZE_INDEX_OFFSET

BT_SDP_ARRAY_8(...)

Declare an array of 8-bit elements in an attribute.

BT_SDP_ARRAY_16(...)

Declare an array of 16-bit elements in an attribute.

BT_SDP_ARRAY_32(...)

Declare an array of 32-bit elements in an attribute.

BT_SDP_TYPE_SIZE(type)

Declare a fixed-size data element header.

Parameters

• **_type** – Data element header containing type and size descriptors.

BT_SDP_TYPE_SIZE_VAR(_type, _size)

Declare a variable-size data element header.

Parameters

- **_type** Data element header containing type and size descriptors.
- **_size** The actual size of the data.

BT_SDP_DATA_ELEM_LIST(...)

Declare a list of data elements.

BT_SDP_NEW_SERVICE

SDP New Service Record Declaration Macro.

Helper macro to declare a new service record. Default attributes: Record Handle, Record State, Language Base, Root Browse Group

BT_SDP_LIST(_att_id, _type_size, _data_elem_seq)

Generic SDP List Attribute Declaration Macro.

Helper macro to declare a list attribute.

Parameters

• _att_id – List Attribute ID.

- _data_elem_seq Data element sequence for the list.
- **_type_size** SDP type and size descriptor.

BT_SDP_SERVICE_ID(uuid)

SDP Service ID Attribute Declaration Macro.

Helper macro to declare a service ID attribute.

Parameters

• _uuid – Service ID 16bit UUID.

BT_SDP_SERVICE_NAME(name)

SDP Name Attribute Declaration Macro.

Helper macro to declare a service name attribute.

Parameters

• _name – Service name as a string (up to 256 chars).

BT_SDP_SUPPORTED_FEATURES(features)

SDP Supported Features Attribute Declaration Macro.

Helper macro to declare supported features of a profile/protocol.

Parameters

• **_features** – Feature mask as 16bit unsigned integer.

BT_SDP_RECORD(attrs)

SDP Service Declaration Macro.

Helper macro to declare a service.

Parameters

• _attrs – List of attributes for the service record.

Typedefs

typedef uint8_t (*bt_sdp_discover_func_t)(struct bt_conn *conn, struct bt_sdp_client_result *result)

Callback type reporting to user that there is a resolved result on remote for given UUID and the result record buffer can be used by user for further inspection.

A function of this type is given by the user to the *bt_sdp_discover_params* object. It'll be called on each valid record discovery completion for given UUID. When UUID resolution gives back no records then NULL is passed to the user. Otherwise user can get valid record(s) and then the internal hint 'next record' is set to false saying the UUID resolution is complete or the hint can be set by caller to true meaning that next record is available for given UUID. The returned function value allows the user to control retrieving follow-up resolved records if any. If the user doesn't want to read more resolved records for given UUID since current record data fulfills its requirements then should return BT_SDP_DISCOVER_UUID_STOP. Otherwise returned value means more subcall iterations are allowable.

Param conn

Connection object identifying connection to queried remote.

Param result

Object pointing to logical unparsed SDP record collected on base of response driven by given UUID.

Return

BT_SDP_DISCOVER_UUID_STOP in case of no more need to read next record data and continue discovery for given UUID. By returning BT_SDP_DISCOVER_UUID_CONTINUE user allows this discovery continuation.

Enums

enum [anonymous]

Helper enum to be used as return value of bt_sdp_discover_func_t. The value informs the caller to perform further pending actions or stop them.

Values:

enumerator BT_SDP_DISCOVER_UUID_STOP

enumerator BT_SDP_DISCOVER_UUID_CONTINUE

enum bt_sdp_proto

Protocols to be asked about specific parameters.

Values:

enumerator BT_SDP_PROTO_RFCOMM

enumerator BT_SDP_PROTO_L2CAP

Functions

int bt_sdp_register_service(struct bt_sdp_record *service)

Register a Service Record.

Register a Service Record. Applications can make use of macros such as BT_SDP_DECLARE_SERVICE, BT_SDP_LIST, BT_SDP_SERVICE_ID, BT_SDP_SERVICE_NAME, etc. A service declaration must start with BT_SDP_NEW_SERVICE.

Parameters

• service – Service record declared using BT_SDP_DECLARE_SERVICE.

Returns

0 in case of success or negative value in case of error.

int **bt_sdp_discover**(struct bt_conn *conn, const struct bt_sdp_discover_params *params)

Allows user to start SDP discovery session.

The function performs SDP service discovery on remote server driven by user delivered discovery parameters. Discovery session is made as soon as no SDP transaction is ongoing between peers and if any then this one is queued to be processed at discovery completion of previous one. On the service discovery completion the callback function will be called to get feedback to user about findings.

Parameters

• **conn** – Object identifying connection to remote.

• params – SDP discovery parameters.

Returns

0 in case of success or negative value in case of error.

int **bt_sdp_discover_cancel**(struct bt_conn *conn, const struct bt_sdp_discover_params *params)

Release waiting SDP discovery request.

It can cancel valid waiting SDP client request identified by SDP discovery parameters object.

Parameters

- conn Object identifying connection to remote.
- params SDP discovery parameters.

Returns

0 in case of success or negative value in case of error.

int **bt_sdp_get_proto_param**(const struct net_buf *buf, enum *bt_sdp_proto* proto, uint16_t *param)

Give to user parameter value related to given stacked protocol UUID.

API extracts specific parameter associated with given protocol UUID available in Protocol Descriptor List attribute.

Parameters

- **buf** Original buffered raw record data.
- **proto** Known protocol to be checked like RFCOMM or L2CAP.
- param On success populated by found parameter value.

Returns

0 on success when specific parameter associated with given protocol value is found, or negative if error occurred during processing.

int **bt_sdp_get_addl_proto_param**(const struct net_buf *buf, enum *bt_sdp_proto* proto, uint8_t param_index, uint16_t *param)

Get additional parameter value related to given stacked protocol UUID.

API extracts specific parameter associated with given protocol UUID available in Additional Protocol Descriptor List attribute.

Parameters

- **buf** Original buffered raw record data.
- proto Known protocol to be checked like RFCOMM or L2CAP.
- param_index There may be more than one parameter related to the given protocol UUID. This function returns the result that is indexed by this parameter. It's value is from 0, 0 means the first matched result, 1 means the second matched result.
- param [out] On success populated by found parameter value.

Returns

0 on success when a specific parameter associated with a given protocol value is found, or negative if error occurred during processing.

int **bt_sdp_get_profile_version**(const struct net_buf *buf, uint16_t profile, uint16_t *version)

Get profile version.

Helper API extracting remote profile version number. To get it proper generic profile parameter needs to be selected usually listed in SDP Interoperability Requirements section for given profile specification.

Parameters

- **buf** Original buffered raw record data.
- **profile** Profile family identifier the profile belongs.
- **version** On success populated by found version number.

Returns

0 on success, negative value if error occurred during processing.

int bt_sdp_get_features(const struct net_buf *buf, uint16_t *features)

Get SupportedFeatures attribute value.

Allows if exposed by remote retrieve SupportedFeature attribute.

Parameters

- **buf** Buffer holding original raw record data from remote.
- **features** On success object to be populated with SupportedFeature mask.

Returns

0 on success if feature found and valid, negative in case any error

struct bt_sdp_data_elem

#include <sdp.h> SDP Generic Data Element Value.

struct bt_sdp_attribute

#include <sdp.h> SDP Attribute Value.

struct bt_sdp_record

#include <sdp.h> SDP Service Record Value.

struct bt_sdp_client_result

#include <sdp.h> Generic SDP Client Query Result data holder.

struct bt_sdp_discover_params

#include <sdp.h> Main user structure used in SDP discovery of remote.

Public Members

const struct bt_uuid *uuid

UUID (service) to be discovered on remote SDP entity

bt_sdp_discover_func_t func

Discover callback to be called on resolved SDP record

struct net_buf_pool *pool

Memory buffer enabled by user for SDP query results

1.11 Advance Audio Distribution Profile (A2DP)

1.11.1 API Reference

```
group bt_a2dp
```

Advance Audio Distribution Profile (A2DP)

Defines

BT_A2DP_SBC_IE_LENGTH

SBC IE length

BT_A2DP_MPEG_1_2_IE_LENGTH

MPEG1,2 IE length

BT_A2DP_MPEG_2_4_IE_LENGTH

MPEG2,4 IE length

BT_A2DP_SOURCE_SBC_CODEC_BUFFER_SIZE

BT_A2DP_SOURCE_SBC_CODEC_BUFFER_NOCACHED_SIZE

BT_A2DP_SINK_SBC_CODEC_BUFFER_SIZE

BT_A2DP_SINK_SBC_CODEC_BUFFER_NOCACHED_SIZE

BT_A2DP_EP_CONTENT_PROTECTION_INIT

BT_A2DP_EP_RECOVERY_SERVICE_INIT

BT_A2DP_EP_REPORTING_SERVICE_INIT

BT_A2DP_EP_DELAY_REPORTING_INIT

BT_A2DP_EP_HEADER_COMPRESSION_INIT

BT_A2DP_EP_MULTIPLEXING_INIT

BT_A2DP_ENDPOINT_INIT(_role, _codec, _capability, _config, _codec_buffer, _codec_buffer_nocahced) define the audio endpoint

Parameters

- _role BT_A2DP_SOURCE or BT_A2DP_SINK.
- _codec value of enum bt_a2dp_codec_id.

- _capability the codec capability.
- **config** the default config to configure the peer same codec type endpoint.
- _codec_buffer the codec function used buffer.
- _codec_buffer_nocahced the codec function used nocached buffer.

BT_A2DP_SINK_ENDPOINT_INIT(codec, capability, codec buffer, codec buffer nocahced)

define the audio sink endpoint

Parameters

- _codec value of enum bt_a2dp_codec_id.
- _capability the codec capability.
- _codec_buffer the codec function used buffer.
- _codec_buffer_nocahced the codec function used nocached buffer.

BT_A2DP_SOURCE_ENDPOINT_INIT(_codec, _capability, _config, _codec_buffer, _codec_buffer_nocahced) define the audio source endpoint

Parameters

- _codec value of enum bt_a2dp_codec_id.
- _capability the codec capability.
- _config the default config to configure the peer same codec type endpoint.
- _codec_buffer the codec function used buffer.
- _codec_buffer_nocahced the codec function used nocached buffer.

BT_A2DP_SBC_SINK_ENDPOINT(_name)

define the default SBC sink endpoint that can be used as bt_a2dp_register_endpoint's parameter.

SBC is mandatory as a2dp specification, BT_A2DP_SBC_SINK_ENDPOINT is more convenient for user to register SBC endpoint.

Parameters

• _name – the endpoint variable name.

BT_A2DP_SBC_SOURCE_ENDPOINT(name, config freq)

define the default SBC source endpoint that can be used as bt_a2dp_register_endpoint's parameter.

SBC is mandatory as a2dp specification, BT_A2DP_SBC_SOURCE_ENDPOINT is more convenient for user to register SBC endpoint.

Parameters

- **_name** the endpoint variable name.
- **_config_freq** the frequency to configure the peer same codec type endpoint.

Typedefs

```
typedef uint8_t (*bt_a2dp_discover_peer_endpoint_cb_t)(struct bt_a2dp *a2dp, struct
bt_a2dp_endpoint *endpoint, int err)
    Get peer's endpoints callback.
Enums
enum bt_a2dp_codec_id
    Codec ID.
    Values:
    enumerator BT_A2DP_SBC
        Codec SBC
    enumerator BT_A2DP_MPEG1
        Codec MPEG-1
    enumerator BT_A2DP_MPEG2
        Codec MPEG-2
    enumerator BT_A2DP_ATRAC
        Codec ATRAC
    enumerator BT_A2DP_VENDOR
        Codec Non-A2DP
enum MEDIA_TYPE
    Stream End Point Media Type.
    Values:
    enumerator BT_A2DP_AUDIO
        Audio Media Type
    enumerator BT_A2DP_VIDEO
        Video Media Type
    enumerator BT_A2DP_MULTIMEDIA
        Multimedia Media Type
```

enum ROLE_TYPE

Values:

Stream End Point Role.

enumerator BT_A2DP_SOURCE

Source Role

enumerator BT_A2DP_SINK

Sink Role

enum [anonymous]

Helper enum to be used as return value of bt_a2dp_discover_peer_endpoint_cb_t. The value informs the caller to perform further pending actions or stop them.

Values:

enumerator BT_A2DP_DISCOVER_ENDPOINT_STOP

enumerator BT_A2DP_DISCOVER_ENDPOINT_CONTINUE

Functions

struct bt_a2dp *bt_a2dp_connect(struct bt_conn *conn)

A2DP Connect.

This function is to be called after the conn parameter is obtained by performing a GAP procedure. The API is to be used to establish A2DP connection between devices. This function only establish AVDTP L2CAP connection. After connection success, the callback that is registered by bt_a2dp_register_connect_callback is called.

Parameters

• **conn** – Pointer to bt_conn structure.

Returns

pointer to struct bt_a2dp in case of success or NULL in case of error.

int bt_a2dp_disconnect(struct bt_a2dp *a2dp)

disconnect 12cap a2dp

Parameters

• **a2dp** – The a2dp instance.

Returns

0 in case of success and error code in case of error.

int **bt_a2dp_register_endpoint**(struct *bt_a2dp_endpoint* *endpoint, uint8_t media_type, uint8_t role)

Endpoint Registration.

This function is used for registering the stream end points. The user has to take care of allocating the memory of the endpoint pointer and then pass the required arguments. Also, only one sep can be registered at a time. Multiple stream end points can be registered by calling multiple times. The endpoint registered first has a higher priority than the endpoint registered later. The priority is used in bt_a2dp_configure.

Parameters

- **endpoint** Pointer to *bt_a2dp_endpoint* structure.
- **media_type** Media type that the Endpoint is.

• **role** – Role of Endpoint.

Returns

0 in case of success and error code in case of error.

int bt_a2dp_register_connect_callback(struct bt a2dp connect cb *cb)

register connecting callback.

The cb is called when bt_a2dp_connect is called or it is connected by peer device.

Parameters

• **cb** – The callback function.

Returns

0 in case of success and error code in case of error.

```
int bt_a2dp_configure(struct bt_a2dp *a2dp, void (*result_cb)(int err))
```

configure control callback.

This function will get peer's all endpoints and select one endpoint based on the priority of registered endpoints, then configure the endpoint based on the "config" of endpoint. Note: (1) priority is described in bt_a2dp_register_endpoint; (2) "config" is the config field of struct bt_a2dp_endpoint that is registered by bt_a2dp_register_endpoint.

Parameters

- **a2dp** The a2dp instance.
- **result_cb** The result callback function.

Returns

0 in case of success and error code in case of error.

int **bt_a2dp_discover_peer_endpoints**(struct bt_a2dp *a2dp, bt_a2dp_discover_peer_endpoint_cb_t cb) get peer's endpoints.

bt_a2dp_configure can be called to configure a2dp. bt_a2dp_discover_peer_endpoints and bt_a2dp_configure_endpoint can be used too. In bt_a2dp_configure, the endpoint is selected automatically based on the prioriy. If bt_a2dp_configure fails, it means the default config of endpoint is not reasonal. bt_a2dp_discover_peer_endpoints and bt_a2dp_configure_endpoint can be used. bt_a2dp_discover_peer_endpoints is used to get peer endpoints. the peer endpoint is returned in the cb. then endpoint can be selected and configured by bt_a2dp_configure_endpoint. If user stops to discover more peer endpoints, return BT_A2DP_DISCOVER_ENDPOINT_STOP in the cb; if user wants to discover more peer endpoints, return BT_A2DP_DISCOVER_ENDPOINT_CONTINUE in the cb.

Parameters

- a2dp The a2dp instance.
- **cb** notify the result.

Returns

0 in case of success and error code in case of error.

```
int bt_a2dp_configure_endpoint(struct bt_a2dp *a2dp, struct bt_a2dp_endpoint *endpoint, struct bt_a2dp_endpoint *peer_endpoint, struct bt_a2dp_endpoint_config *config)
```

configure endpoint.

If the bt_a2dp_configure is failed or user want to change configured endpoint, user can call bt_a2dp_discover_peer_endpoints and this function to configure the selected endpoint.

Parameters

- **a2dp** The a2dp instance.
- **endpoint** The configured endpoint that is registered.
- **config** The config to configure the endpoint.

Returns

0 in case of success and error code in case of error.

int **bt_a2dp_deconfigure**(struct *bt_a2dp_endpoint* *endpoint)

revert the configuration, then it can be configured again.

Release the endpoint based on the endpoint's state. After this, the endpoint can be re-configured again.

Parameters

• **endpoint** – the registered endpoint.

Returns

0 in case of success and error code in case of error.

int bt_a2dp_start(struct bt_a2dp_endpoint *endpoint)

start a2dp streamer, it is source only.

Parameters

• **endpoint** – The endpoint.

Returns

0 in case of success and error code in case of error.

int **bt_a2dp_stop**(struct *bt_a2dp_endpoint* *endpoint)

stop a2dp streamer, it is source only.

Parameters

• **endpoint** – The registered endpoint.

Returns

0 in case of success and error code in case of error.

int **bt_a2dp_reconfigure**(struct *bt_a2dp_endpoint* *endpoint, struct *bt_a2dp_endpoint_config* *config) re-configure a2dp streamer

This function send the AVDTP_RECONFIGURE command

Parameters

- **a2dp** The a2dp instance.
- **endpoint** the endpoint.
- **config** The config to configure the endpoint.

Returns

0 in case of success and error code in case of error.

struct bt_a2dp_codec_ie

#include <a2dp.h> codec information elements for the endpoint

```
uint8_t len
```

Length of capabilities

uint8_t codec_ie[0]

codec information element

struct bt_a2dp_endpoint_config

#include <a2dp.h> The endpoint configuration.

Public Members

```
struct bt_a2dp_codec_ie *media_config
```

The media configuration content

struct bt_a2dp_endpoint_configure_result

#include <a2dp.h> The configuration result.

Public Members

```
int err
```

0 - success; other values - fail code

struct bt_a2dp *a2dp

which a2dp connection the endpoint is configured

struct bt_conn *conn

which conn the endpoint is configured

struct bt_a2dp_endpoint_config config

The configuration content

struct bt_a2dp_control_cb

#include <a2dp.h> The callback that is controlled by peer.

```
void (*configured)(struct bt_a2dp_endpoint_configure_result *config)
         a2dp is configured by peer.
             Param err
                a2dp configuration result.
     void (*deconfigured)(int err)
         a2dp is de-configured by peer.
              Param err
                a2dp configuration result.
     void (*start_play)(int err)
         The result of starting media streamer.
     void (*stop_play)(int err)
         the result of stopping media streaming.
     void (*sink_streamer_data)(uint8_t *data, uint32_t length)
         the media streaming data, only for sink.
              Param data
                the data buffer pointer.
              Param length
                the data length.
struct bt_a2dp_connect_cb
     #include \langle a2dp.h \rangle The connecting callback.
     Public Members
     void (*connected)(struct bt_a2dp *a2dp, int err)
         A a2dp connection has been established.
         This callback notifies the application of a a2dp connection. It means the AVDTP L2CAP connection.
         In case the err parameter is non-zero it means that the connection establishment failed.
              Param a2dp
                a2dp connection object.
             Param err
                error code.
     void (*disconnected)(struct bt a2dp *a2dp)
         A a2dp connection has been disconnected.
         This callback notifies the application that a a2dp connection has been disconnected.
              Param a2dp
                a2dp connection object.
struct bt_a2dp_endpoint
```

#include <a2dp.h> Stream End Point.

```
uint8_t codec_id

Code ID

struct bt_avdtp_seid_lsep info

Stream End Point Information

struct bt_a2dp_codec_ie *config

Pointer to codec default config

struct bt_a2dp_codec_ie *capabilities

Capabilities

struct bt_a2dp_control_cb control_cbs

endpoint control callbacks

uint8_t *codec_buffer

reserved codec related buffer (can be cacaheable ram)

uint8_t *codec_buffer_nocached

reserved codec related buffer (nocached)
```

1.12 Serial Port Profile (SPP)

1.12.1 API Reference

```
Serial Port Profile (SPP)

Typedefs

typedef enum bt_spp_role bt_spp_role_t
    SPP Role Value.

typedef struct _bt_spp_callback bt_spp_callback
    spp application callback function
    (defined(CONFIG_BT_SPP_ENABLE_CONTROL_CMD) && (CONFIG_BT_SPP_ENABLE_CONTROL_CMD > 0))

typedef int (*bt_spp_discover_callback)(struct bt_conn *conn, uint8_t count, uint16_t *channel)
    spp sdp discover callback function
```

Enums

```
enum bt_spp_role
```

SPP Role Value.

Values:

enumerator BT_SPP_ROLE_SERVER

enumerator BT_SPP_ROLE_CLIENT

Functions

```
int bt_spp_server_register(uint8_t channel, bt_spp_callback *cb)
```

Register a SPP server.

Register a SPP server channel, wait for spp connection from SPP client. Once it's connected by spp client, will notify application by calling cb->connected.

Parameters

- channel Registered server channel.
- **cb** Application callback.

Returns

0 in case of success or negative value in case of error.

int **bt_spp_discover**(struct bt conn *conn, *discover cb t* *cb)

Discover SPP server channel.

Discover peer SPP server channel after basic BR connection is created. Will notify application discover results by calling cb->cb.

Parameters

- conn BR connection handle.
- **cb** Discover callback.

Returns

0 in case of success or negative value in case of error.

Connect SPP server channel.

Create SPP connection with remote SPP server channel. Once connection is created successfully, will notify application by calling cb->connected.

Parameters

- conn Conn handle created with remote device.
- **channel** Remote server channel to be connected, if it's 0, will connect remote BT RFCOMM CHAN SPP channel.
- **cb** Application callback.
- spp SPP handle.

Returns

0 in case of success or negative value in case of error.

int **bt_spp_data_send**(struct bt_spp *spp, uint8_t *data, uint16_t len)

Send data to peer SPP device.

Send data to connected peer spp. Once data is sent, will notify application by calling cb->data_sent, which is provided by bt_spp_server_register or bt_spp_client_connect. If peer spp receives data, will notify application by calling cb->data_received.

Parameters

- spp SPP handle.
- data Data buffer.
- len Data length.

Returns

0 in case of success or negative value in case of error.

```
int bt_spp_disconnect(struct bt_spp *spp)
```

Disconnect SPP connection.

Disconnect SPP connection. Once connection is disconnected, will notify application by calling cb>disconnected, which is provided by bt_spp_server_register or bt_spp_client_connect.

Parameters

• spp – SPP handle.

Returns

0 in case of success or negative value in case of error.

int bt_spp_get_channel(struct bt_spp *spp, uint8_t *channel)

Get channel of SPP handle.

Parameters

- spp SPP handle.
- **channel** Pointer to channel of spp handle.

Returns

0 in case of success or negative value in case of error.

```
int bt_spp_get_role(struct bt_spp *spp, bt_spp_role_t *role)
```

Get role of SPP handle.

Parameters

- spp SPP handle.
- **role** Pointer to role of spp handle.

Returns

0 in case of success or negative value in case of error.

int **bt_spp_get_conn**(struct bt_spp *spp, struct bt_conn **conn)

Get conn handle of SPP handle.

Parameters

- spp SPP handle.
- conn Pointer to conn handle of spp handle.

Returns

0 in case of success or negative value in case of error.

struct _bt_spp_callback

#include <spp.h> spp application callback function

(defined(CONFIG_BT_SPP_ENABLE_CONTROL_CMD)
FIG_BT_SPP_ENABLE_CONTROL_CMD > 0))

&& (CON-

struct discover_cb_t

#include <spp.h> bt_spp_discover callback parameter

1.13 Universal Unique Identifiers (UUIDs)

1.13.1 API Reference

group bt_uuid

UUIDs.

Defines

BT_UUID_SIZE_16

Size in octets of a 16-bit UUID

BT_UUID_SIZE_32

Size in octets of a 32-bit UUID

BT_UUID_SIZE_128

Size in octets of a 128-bit UUID

BT_UUID_INIT_16(value)

Initialize a 16-bit UUID.

Parameters

• value – 16-bit UUID value in host endianness.

BT_UUID_INIT_32(value)

Initialize a 32-bit UUID.

Parameters

• value – 32-bit UUID value in host endianness.

BT_UUID_INIT_128(value...)

Initialize a 128-bit UUID.

Parameters

• **value** – 128-bit UUID array values in little-endian format. Can be combined with *BT_UUID_128_ENCODE* to initialize a UUID from the readable form of UUIDs.

BT_UUID_DECLARE_16(value)

Helper to declare a 16-bit UUID inline.

Parameters

• value – 16-bit UUID value in host endianness.

Returns

Pointer to a generic UUID.

BT_UUID_DECLARE_32 (value)

Helper to declare a 32-bit UUID inline.

Parameters

• value – 32-bit UUID value in host endianness.

Returns

Pointer to a generic UUID.

BT_UUID_DECLARE_128(value...)

Helper to declare a 128-bit UUID inline.

Parameters

• **value** – 128-bit UUID array values in little-endian format. Can be combined with *BT_UUID_128_ENCODE* to declare a UUID from the readable form of UUIDs.

Returns

Pointer to a generic UUID.

BT_UUID_16(__u)

Helper macro to access the 16-bit UUID from a generic UUID.

BT_UUID_32(u)

Helper macro to access the 32-bit UUID from a generic UUID.

```
BT_UUID_128( u)
```

Helper macro to access the 128-bit UUID from a generic UUID.

```
BT_UUID_128_ENCODE(w32, w1, w2, w3, w48)
```

Encode 128 bit UUID into array values in little-endian format.

Helper macro to initialize a 128-bit UUID array value from the readable form of UUIDs, or encode 128-bit UUID values into advertising data Can be combined with BT_UUID_DECLARE_128 to declare a 128-bit UUID.

Example of how to declare the UUID 6E400001-B5A3-F393-E0A9-E50E24DCCA9E

```
BT_UUID_DECLARE_128(
BT_UUID_128_ENCODE(0x6E400001, 0xB5A3, 0xF393, 0xE0A9, 0xE50E24DCCA9E))
```

Example of how to encode the UUID 6E400001-B5A3-F393-E0A9-E50E24DCCA9E into advertising data.

```
BT_DATA_BYTES(BT_DATA_UUID128_ALL,
BT_UUID_128_ENCODE(0x6E400001, 0xB5A3, 0xF393, 0xE0A9, 0xE50E24DCCA9E))
```

Just replace the hyphen by the comma and add 0x prefixes.

Parameters

• w32 – First part of the UUID (32 bits)

- w1 Second part of the UUID (16 bits)
- w2 Third part of the UUID (16 bits)
- w3 Fourth part of the UUID (16 bits)
- w48 Fifth part of the UUID (48 bits)

Returns

The comma separated values for UUID 128 initializer that may be used directly as an argument for BT_UUID_INIT_128 or BT_UUID_DECLARE_128

BT_UUID_16_ENCODE(w16)

Encode 16-bit UUID into array values in little-endian format.

Helper macro to encode 16-bit UUID values into advertising data.

Example of how to encode the UUID 0x180a into advertising data.

```
BT_DATA_BYTES(BT_DATA_UUID16_ALL, BT_UUID_16_ENCODE(0x180a))
```

Parameters

• **w16** – UUID value (16-bits)

Returns

The comma separated values for UUID 16 value that may be used directly as an argument for *BT_DATA_BYTES*.

BT_UUID_32_ENCODE(w32)

Encode 32-bit UUID into array values in little-endian format.

Helper macro to encode 32-bit UUID values into advertising data.

Example of how to encode the UUID 0x180a01af into advertising data.

```
BT_DATA_BYTES(BT_DATA_UUID32_ALL, BT_UUID_32_ENCODE(0x180a01af))
```

Parameters

• **w32** – UUID value (32-bits)

Returns

The comma separated values for UUID 32 value that may be used directly as an argument for *BT_DATA_BYTES*.

BT_UUID_STR_LEN

Recommended length of user string buffer for Bluetooth UUID.

The recommended length guarantee the output of UUID conversion will not lose valuable information about the UUID being processed. If the length of the UUID is known the string can be shorter.

BT_UUID_GAP_VAL

Generic Access UUID value.

BT_UUID_GAP

Generic Access.

BT_UUID_GATT_VAL

Generic attribute UUID value.

BT_UUID_GATT

Generic Attribute.

BT_UUID_IAS_VAL

Immediate Alert Service UUID value.

BT_UUID_IAS

Immediate Alert Service.

BT_UUID_LLS_VAL

Link Loss Service UUID value.

BT_UUID_LLS

Link Loss Service.

BT_UUID_TPS_VAL

Tx Power Service UUID value.

BT_UUID_TPS

Tx Power Service.

BT_UUID_CTS_VAL

Current Time Service UUID value.

BT_UUID_CTS_VAL

Current Time Service UUID value.

BT_UUID_CTS

Current Time Service.

BT_UUID_CTS

Current Time Service.

BT_UUID_HTS_VAL

Health Thermometer Service UUID value.

BT_UUID_HTS

Health Thermometer Service.

BT_UUID_DIS_VAL

Device Information Service UUID value.

BT_UUID_DIS

Device Information Service.

BT_UUID_HRS_VAL

Heart Rate Service UUID value.

BT_UUID_HRS

Heart Rate Service.

BT_UUID_BAS_VAL

Battery Service UUID value.

BT_UUID_BAS

Battery Service.

BT_UUID_HIDS_VAL

HID Service UUID value.

BT_UUID_HIDS

HID Service.

BT_UUID_RSCS_VAL

Running Speed and Cadence Service UUID value.

BT_UUID_RSCS

Running Speed and Cadence Service.

BT_UUID_CSC_VAL

Cycling Speed and Cadence Service UUID value.

BT_UUID_CSC

Cycling Speed and Cadence Service.

BT_UUID_ESS_VAL

Environmental Sensing Service UUID value.

BT_UUID_ESS

Environmental Sensing Service.

BT_UUID_BMS_VAL

Bond Management Service UUID value.

${\tt BT_UUID_BMS}$

Bond Management Service.

BT_UUID_CGMS_VAL

Continuous Glucose Monitoring Service UUID value.

BT_UUID_CGMS

Continuous Glucose Monitoring Service.

BT_UUID_IPSS_VAL

IP Support Service UUID value.

BT_UUID_IPSS

IP Support Service.

BT_UUID_HPS_VAL

HTTP Proxy Service UUID value.

BT_UUID_HPS

HTTP Proxy Service.

BT_UUID_OTS_VAL

Object Transfer Service UUID value.

BT_UUID_OTS

Object Transfer Service.

BT_UUID_MESH_PROV_VAL

Mesh Provisioning Service UUID value.

BT_UUID_MESH_PROV

Mesh Provisioning Service.

BT_UUID_MESH_PROXY_VAL

Mesh Proxy Service UUID value.

BT_UUID_MESH_PROXY

Mesh Proxy Service.

BT_UUID_AICS_VAL

Audio Input Control Service value.

BT_UUID_AICS

Audio Input Control Service.

BT_UUID_VCS_VAL

Volume Control Service value.

BT_UUID_VCS

Volume Control Service.

BT_UUID_VOCS_VAL

Volume Offset Control Service value.

BT_UUID_VOCS

Volume Offset Control Service.

BT_UUID_CSIS_VAL

Coordinated Set Identification Service value.

BT_UUID_CSIS

Coordinated Set Identification Service.

BT_UUID_MCS_VAL

Media Control Service value.

BT_UUID_MCS

Media Control Service.

BT_UUID_GMCS_VAL

Generic Media Control Service value.

BT_UUID_GMCS

Generic Media Control Service.

BT_UUID_TBS_VAL

Telephone Bearer Service value.

BT_UUID_TBS

Telephone Bearer Service.

BT_UUID_GTBS_VAL

Generic Telephone Bearer Service value.

BT_UUID_GTBS

Generic Telephone Bearer Service.

BT_UUID_MICS_VAL

Microphone Input Control Service value.

BT_UUID_MICS

Microphone Input Control Service.

BT_UUID_ASCS_VAL

Audio Stream Control Service value.

BT_UUID_ASCS

Audio Stream Control Service.

BT_UUID_BASS_VAL

Broadcast Audio Scan Service value.

BT_UUID_BASS

Broadcast Audio Scan Service.

BT_UUID_PACS_VAL

Published Audio Capabilities Service value.

BT_UUID_PACS

Published Audio Capabilities Service.

BT_UUID_BASIC_AUDIO_VAL

Basic Audio Announcement Service value.

BT_UUID_BASIC_AUDIO

Basic Audio Announcement Service.

BT_UUID_BROADCAST_AUDIO_VAL

Broadcast Audio Announcement Service value.

BT_UUID_BROADCAST_AUDIO

Broadcast Audio Announcement Service.

BT_UUID_CAS_VAL

Common Audio Service value.

BT_UUID_CAS

Common Audio Service.

BT_UUID_HAS_VAL

Hearing Access Service value.

BT_UUID_HAS

Hearing Access Service.

BT_UUID_GATT_PRIMARY_VAL

GATT Primary Service UUID value.

BT_UUID_GATT_PRIMARY

GATT Primary Service.

BT_UUID_GATT_SECONDARY_VAL

GATT Secondary Service UUID value.

BT_UUID_GATT_SECONDARY

GATT Secondary Service.

BT_UUID_GATT_INCLUDE_VAL

GATT Include Service UUID value.

BT_UUID_GATT_INCLUDE

GATT Include Service.

BT_UUID_GATT_CHRC_VAL

GATT Characteristic UUID value.

BT_UUID_GATT_CHRC

GATT Characteristic.

BT_UUID_GATT_CEP_VAL

GATT Characteristic Extended Properties UUID value.

BT_UUID_GATT_CEP

GATT Characteristic Extended Properties.

BT_UUID_GATT_CUD_VAL

GATT Characteristic User Description UUID value.

BT_UUID_GATT_CUD

GATT Characteristic User Description.

BT_UUID_GATT_CCC_VAL

GATT Client Characteristic Configuration UUID value.

BT_UUID_GATT_CCC

GATT Client Characteristic Configuration.

BT_UUID_GATT_SCC_VAL

GATT Server Characteristic Configuration UUID value.

BT_UUID_GATT_SCC

GATT Server Characteristic Configuration.

BT_UUID_GATT_CPF_VAL

GATT Characteristic Presentation Format UUID value.

BT_UUID_GATT_CPF

GATT Characteristic Presentation Format.

BT_UUID_GATT_CAF_VAL

GATT Characteristic Aggregated Format UUID value.

BT_UUID_GATT_CAF

GATT Characteristic Aggregated Format.

BT_UUID_VALID_RANGE_VAL

Valid Range Descriptor UUID value.

BT_UUID_VALID_RANGE

Valid Range Descriptor.

BT_UUID_HIDS_EXT_REPORT_VAL

HID External Report Descriptor UUID value.

BT_UUID_HIDS_EXT_REPORT

HID External Report Descriptor.

BT_UUID_HIDS_REPORT_REF_VAL

HID Report Reference Descriptor UUID value.

BT_UUID_HIDS_REPORT_REF

HID Report Reference Descriptor.

BT_UUID_ES_CONFIGURATION_VAL

Environmental Sensing Configuration Descriptor UUID value.

BT_UUID_ES_CONFIGURATION

Environmental Sensing Configuration Descriptor.

BT_UUID_ES_MEASUREMENT_VAL

Environmental Sensing Measurement Descriptor UUID value.

BT_UUID_ES_MEASUREMENT

Environmental Sensing Measurement Descriptor.

BT_UUID_ES_TRIGGER_SETTING_VAL

Environmental Sensing Trigger Setting Descriptor UUID value.

BT_UUID_ES_TRIGGER_SETTING

Environmental Sensing Trigger Setting Descriptor.

BT_UUID_GAP_DEVICE_NAME_VAL

GAP Characteristic Device Name UUID value.

BT_UUID_GAP_DEVICE_NAME

GAP Characteristic Device Name.

BT_UUID_GAP_APPEARANCE_VAL

GAP Characteristic Appearance UUID value.

BT_UUID_GAP_APPEARANCE

GAP Characteristic Appearance.

BT_UUID_GAP_PPCP_VAL

GAP Characteristic Peripheral Preferred Connection Parameters UUID value.

BT_UUID_GAP_PPCP

GAP Characteristic Peripheral Preferred Connection Parameters.

BT_UUID_GATT_SC_VAL

GATT Characteristic Service Changed UUID value.

BT_UUID_GATT_SC

GATT Characteristic Service Changed.

BT_UUID_ALERT_LEVEL_VAL

Alert Level UUID value.

BT_UUID_ALERT_LEVEL

Alert Level.

BT_UUID_TPS_TX_POWER_LEVEL_VAL

TPS Characteristic Tx Power Level UUID value.

BT_UUID_TPS_TX_POWER_LEVEL

TPS Characteristic Tx Power Level.

BT_UUID_BAS_BATTERY_LEVEL_VAL

BAS Characteristic Battery Level UUID value.

BT_UUID_BAS_BATTERY_LEVEL

BAS Characteristic Battery Level.

BT_UUID_HTS_MEASUREMENT_VAL

HTS Characteristic Measurement Value UUID value.

BT_UUID_HTS_MEASUREMENT

HTS Characteristic Measurement Value.

BT_UUID_HIDS_BOOT_KB_IN_REPORT_VAL

HID Characteristic Boot Keyboard Input Report UUID value.

BT_UUID_HIDS_BOOT_KB_IN_REPORT

HID Characteristic Boot Keyboard Input Report.

BT_UUID_DIS_SYSTEM_ID_VAL

DIS Characteristic System ID UUID value.

BT_UUID_DIS_SYSTEM_ID

DIS Characteristic System ID.

BT_UUID_DIS_MODEL_NUMBER_VAL

DIS Characteristic Model Number String UUID value.

BT_UUID_DIS_MODEL_NUMBER

DIS Characteristic Model Number String.

BT_UUID_DIS_SERIAL_NUMBER_VAL

DIS Characteristic Serial Number String UUID value.

BT_UUID_DIS_SERIAL_NUMBER

DIS Characteristic Serial Number String.

BT_UUID_DIS_FIRMWARE_REVISION_VAL

DIS Characteristic Firmware Revision String UUID value.

BT_UUID_DIS_FIRMWARE_REVISION

DIS Characteristic Firmware Revision String.

BT_UUID_DIS_HARDWARE_REVISION_VAL

DIS Characteristic Hardware Revision String UUID value.

BT_UUID_DIS_HARDWARE_REVISION

DIS Characteristic Hardware Revision String.

BT_UUID_DIS_SOFTWARE_REVISION_VAL

DIS Characteristic Software Revision String UUID value.

BT_UUID_DIS_SOFTWARE_REVISION

DIS Characteristic Software Revision String.

BT_UUID_DIS_MANUFACTURER_NAME_VAL

DIS Characteristic Manufacturer Name String UUID Value.

BT_UUID_DIS_MANUFACTURER_NAME

DIS Characteristic Manufacturer Name String.

BT_UUID_DIS_PNP_ID_VAL

DIS Characteristic PnP ID UUID value.

BT_UUID_DIS_PNP_ID

DIS Characteristic PnP ID.

BT_UUID_CTS_CURRENT_TIME_VAL

CTS Characteristic Current Time UUID value.

BT_UUID_CTS_CURRENT_TIME

CTS Characteristic Current Time.

BT_UUID_MAGN_DECLINATION_VAL

Magnetic Declination Characteristic UUID value.

BT_UUID_MAGN_DECLINATION

Magnetic Declination Characteristic.

BT_UUID_HIDS_BOOT_KB_OUT_REPORT_VAL

HID Boot Keyboard Output Report Characteristic UUID value.

BT_UUID_HIDS_BOOT_KB_OUT_REPORT

HID Boot Keyboard Output Report Characteristic.

BT_UUID_HIDS_BOOT_MOUSE_IN_REPORT_VAL

HID Boot Mouse Input Report Characteristic UUID value.

BT_UUID_HIDS_BOOT_MOUSE_IN_REPORT

HID Boot Mouse Input Report Characteristic.

BT_UUID_HRS_MEASUREMENT_VAL

HRS Characteristic Measurement Interval UUID value.

BT_UUID_HRS_MEASUREMENT

HRS Characteristic Measurement Interval.

BT_UUID_HRS_BODY_SENSOR_VAL

HRS Characteristic Body Sensor Location.

BT_UUID_HRS_BODY_SENSOR

HRS Characteristic Control Point.

BT_UUID_HRS_CONTROL_POINT_VAL

HRS Characteristic Control Point UUID value.

BT_UUID_HRS_CONTROL_POINT

HRS Characteristic Control Point.

BT_UUID_HIDS_INFO_VAL

HID Information Characteristic UUID value.

BT_UUID_HIDS_INFO

HID Information Characteristic.

BT_UUID_HIDS_REPORT_MAP_VAL

HID Report Map Characteristic UUID value.

BT_UUID_HIDS_REPORT_MAP

HID Report Map Characteristic.

BT_UUID_HIDS_CTRL_POINT_VAL

HID Control Point Characteristic UUID value.

BT_UUID_HIDS_CTRL_POINT

HID Control Point Characteristic.

BT_UUID_HIDS_REPORT_VAL

HID Report Characteristic UUID value.

BT_UUID_HIDS_REPORT

HID Report Characteristic.

BT_UUID_HIDS_PROTOCOL_MODE_VAL

HID Protocol Mode Characteristic UUID value.

BT_UUID_HIDS_PROTOCOL_MODE

HID Protocol Mode Characteristic.

BT_UUID_RECORD_ACCESS_CONTROL_POINT_VAL

Record Access Control Point Characteristic value.

BT_UUID_RECORD_ACCESS_CONTROL_POINT

Record Access Control Point.

BT_UUID_RSC_MEASUREMENT_VAL

RSC Measurement Characteristic UUID value.

BT_UUID_RSC_MEASUREMENT

RSC Measurement Characteristic.

BT_UUID_RSC_FEATURE_VAL

RSC Feature Characteristic UUID value.

BT_UUID_RSC_FEATURE

RSC Feature Characteristic.

BT_UUID_CSC_MEASUREMENT_VAL

CSC Measurement Characteristic UUID value.

BT_UUID_CSC_MEASUREMENT

CSC Measurement Characteristic.

BT_UUID_CSC_FEATURE_VAL

CSC Feature Characteristic UUID value.

BT_UUID_CSC_FEATURE

CSC Feature Characteristic.

BT_UUID_SENSOR_LOCATION_VAL

Sensor Location Characteristic UUID value.

BT_UUID_SENSOR_LOCATION

Sensor Location Characteristic.

BT_UUID_SC_CONTROL_POINT_VAL

SC Control Point Characteristic UUID value.

BT_UUID_SC_CONTROL_POINT

SC Control Point Characteristic.

BT_UUID_ELEVATION_VAL

Elevation Characteristic UUID value.

BT_UUID_ELEVATION

Elevation Characteristic.

BT_UUID_PRESSURE_VAL

Pressure Characteristic UUID value.

BT_UUID_PRESSURE

Pressure Characteristic.

BT_UUID_TEMPERATURE_VAL

Temperature Characteristic UUID value.

BT_UUID_TEMPERATURE

Temperature Characteristic.

BT_UUID_HUMIDITY_VAL

Humidity Characteristic UUID value.

BT_UUID_HUMIDITY

Humidity Characteristic.

BT_UUID_TRUE_WIND_SPEED_VAL

True Wind Speed Characteristic UUID value.

BT_UUID_TRUE_WIND_SPEED

True Wind Speed Characteristic.

BT_UUID_TRUE_WIND_DIR_VAL

True Wind Direction Characteristic UUID value.

BT_UUID_TRUE_WIND_DIR

True Wind Direction Characteristic.

BT_UUID_APPARENT_WIND_SPEED_VAL

Apparent Wind Speed Characteristic UUID value.

BT_UUID_APPARENT_WIND_SPEED

Apparent Wind Speed Characteristic.

BT_UUID_APPARENT_WIND_DIR_VAL

Apparent Wind Direction Characteristic UUID value.

BT_UUID_APPARENT_WIND_DIR

Apparent Wind Direction Characteristic.

BT_UUID_GUST_FACTOR_VAL

Gust Factor Characteristic UUID value.

BT_UUID_GUST_FACTOR

Gust Factor Characteristic.

BT_UUID_POLLEN_CONCENTRATION_VAL

Pollen Concentration Characteristic UUID value.

BT_UUID_POLLEN_CONCENTRATION

Pollen Concentration Characteristic.

BT_UUID_UV_INDEX_VAL

UV Index Characteristic UUID value.

BT_UUID_UV_INDEX

UV Index Characteristic.

BT_UUID_IRRADIANCE_VAL

Irradiance Characteristic UUID value.

BT_UUID_IRRADIANCE

Irradiance Characteristic.

BT_UUID_RAINFALL_VAL

Rainfall Characteristic UUID value.

BT_UUID_RAINFALL

Rainfall Characteristic.

BT_UUID_WIND_CHILL_VAL

Wind Chill Characteristic UUID value.

BT_UUID_WIND_CHILL

Wind Chill Characteristic.

BT_UUID_HEAT_INDEX_VAL

Heat Index Characteristic UUID value.

BT_UUID_HEAT_INDEX

Heat Index Characteristic.

BT_UUID_DEW_POINT_VAL

Dew Point Characteristic UUID value.

BT_UUID_DEW_POINT

Dew Point Characteristic.

BT_UUID_DESC_VALUE_CHANGED_VAL

Descriptor Value Changed Characteristic UUID value.

BT_UUID_DESC_VALUE_CHANGED

Descriptor Value Changed Characteristic.

BT_UUID_MAGN_FLUX_DENSITY_2D_VAL

Magnetic Flux Density - 2D Characteristic UUID value.

BT_UUID_MAGN_FLUX_DENSITY_2D

Magnetic Flux Density - 2D Characteristic.

BT_UUID_MAGN_FLUX_DENSITY_3D_VAL

Magnetic Flux Density - 3D Characteristic UUID value.

BT_UUID_MAGN_FLUX_DENSITY_3D

Magnetic Flux Density - 3D Characteristic.

BT_UUID_BAR_PRESSURE_TREND_VAL

Barometric Pressure Trend Characteristic UUID value.

BT_UUID_BAR_PRESSURE_TREND

Barometric Pressure Trend Characteristic.

BT_UUID_BMS_CONTROL_POINT_VAL

Bond Management Control Point UUID value.

BT_UUID_BMS_CONTROL_POINT

Bond Management Control Point.

BT_UUID_BMS_FEATURE_VAL

Bond Management Feature UUID value.

BT_UUID_BMS_FEATURE

Bond Management Feature.

BT_UUID_CENTRAL_ADDR_RES_VAL

Central Address Resolution Characteristic UUID value.

BT_UUID_CENTRAL_ADDR_RES

Central Address Resolution Characteristic.

BT_UUID_CGM_MEASUREMENT_VAL

CGM Measurement Characteristic value.

BT_UUID_CGM_MEASUREMENT

CGM Measurement Characteristic.

BT_UUID_CGM_FEATURE_VAL

CGM Feature Characteristic value.

BT_UUID_CGM_FEATURE

CGM Feature Characteristic.

BT_UUID_CGM_STATUS_VAL

CGM Status Characteristic value.

BT_UUID_CGM_STATUS

CGM Status Characteristic.

BT_UUID_CGM_SESSION_START_TIME_VAL

CGM Session Start Time Characteristic value.

BT_UUID_CGM_SESSION_START_TIME

CGM Session Start Time.

BT_UUID_CGM_SESSION_RUN_TIME_VAL

CGM Session Run Time Characteristic value.

BT_UUID_CGM_SESSION_RUN_TIME

CGM Session Run Time.

BT_UUID_CGM_SPECIFIC_OPS_CONTROL_POINT_VAL

CGM Specific Ops Control Point Characteristic value.

BT_UUID_CGM_SPECIFIC_OPS_CONTROL_POINT

CGM Specific Ops Control Point.

BT_UUID_URI_VAL

URI UUID value.

BT_UUID_URI

URI.

BT_UUID_HTTP_HEADERS_VAL

HTTP Headers UUID value.

BT_UUID_HTTP_HEADERS

HTTP Headers.

BT_UUID_HTTP_STATUS_CODE_VAL

HTTP Status Code UUID value.

BT_UUID_HTTP_STATUS_CODE

HTTP Status Code.

BT_UUID_HTTP_ENTITY_BODY_VAL

HTTP Entity Body UUID value.

BT_UUID_HTTP_ENTITY_BODY

HTTP Entity Body.

BT_UUID_HTTP_CONTROL_POINT_VAL

HTTP Control Point UUID value.

BT_UUID_HTTP_CONTROL_POINT

HTTP Control Point.

BT_UUID_HTTPS_SECURITY_VAL

HTTPS Security UUID value.

BT_UUID_HTTPS_SECURITY

HTTPS Security.

BT_UUID_OTS_FEATURE_VAL

OTS Feature Characteristic UUID value.

BT_UUID_OTS_FEATURE

OTS Feature Characteristic.

BT_UUID_OTS_NAME_VAL

OTS Object Name Characteristic UUID value.

BT_UUID_OTS_NAME

OTS Object Name Characteristic.

BT_UUID_OTS_TYPE_VAL

OTS Object Type Characteristic UUID value.

BT_UUID_OTS_TYPE

OTS Object Type Characteristic.

BT_UUID_OTS_SIZE_VAL

OTS Object Size Characteristic UUID value.

BT_UUID_OTS_SIZE

OTS Object Size Characteristic.

BT_UUID_OTS_FIRST_CREATED_VAL

OTS Object First-Created Characteristic UUID value.

BT_UUID_OTS_FIRST_CREATED

OTS Object First-Created Characteristic.

BT_UUID_OTS_LAST_MODIFIED_VAL

OTS Object Last-Modified Characteristic UUI value.

BT_UUID_OTS_LAST_MODIFIED

OTS Object Last-Modified Characteristic.

BT_UUID_OTS_ID_VAL

OTS Object ID Characteristic UUID value.

BT_UUID_OTS_ID

OTS Object ID Characteristic.

BT_UUID_OTS_PROPERTIES_VAL

OTS Object Properties Characteristic UUID value.

BT_UUID_OTS_PROPERTIES

OTS Object Properties Characteristic.

BT_UUID_OTS_ACTION_CP_VAL

OTS Object Action Control Point Characteristic UUID value.

BT_UUID_OTS_ACTION_CP

OTS Object Action Control Point Characteristic.

BT_UUID_OTS_LIST_CP_VAL

OTS Object List Control Point Characteristic UUID value.

BT_UUID_OTS_LIST_CP

OTS Object List Control Point Characteristic.

BT_UUID_OTS_LIST_FILTER_VAL

OTS Object List Filter Characteristic UUID value.

BT_UUID_OTS_LIST_FILTER

OTS Object List Filter Characteristic.

BT_UUID_OTS_CHANGED_VAL

OTS Object Changed Characteristic UUID value.

BT_UUID_OTS_CHANGED

OTS Object Changed Characteristic.

BT_UUID_OTS_TYPE_UNSPECIFIED_VAL

OTS Unspecified Object Type UUID value.

BT_UUID_OTS_TYPE_UNSPECIFIED

OTS Unspecified Object Type.

BT_UUID_OTS_DIRECTORY_LISTING_VAL

OTS Directory Listing UUID value.

BT_UUID_OTS_DIRECTORY_LISTING

OTS Directory Listing.

BT_UUID_MESH_PROV_DATA_IN_VAL

Mesh Provisioning Data In UUID value.

BT_UUID_MESH_PROV_DATA_IN

Mesh Provisioning Data In.

BT_UUID_MESH_PROV_DATA_OUT_VAL

Mesh Provisioning Data Out UUID value.

BT_UUID_MESH_PROV_DATA_OUT

Mesh Provisioning Data Out.

BT_UUID_MESH_PROXY_DATA_IN_VAL

Mesh Proxy Data In UUID value.

BT_UUID_MESH_PROXY_DATA_IN

Mesh Proxy Data In.

BT_UUID_MESH_PROXY_DATA_OUT_VAL

Mesh Proxy Data Out UUID value.

BT_UUID_MESH_PROXY_DATA_OUT

Mesh Proxy Data Out.

BT_UUID_GATT_CLIENT_FEATURES_VAL

Client Supported Features UUID value.

BT_UUID_GATT_CLIENT_FEATURES

Client Supported Features.

BT_UUID_GATT_DB_HASH_VAL

Database Hash UUID value.

BT_UUID_GATT_DB_HASH

Database Hash.

BT_UUID_GATT_SERVER_FEATURES_VAL

Server Supported Features UUID value.

BT_UUID_GATT_SERVER_FEATURES

Server Supported Features.

BT_UUID_AICS_STATE_VAL

Audio Input Control Service State value.

BT_UUID_AICS_STATE

Audio Input Control Service State.

BT_UUID_AICS_GAIN_SETTINGS_VAL

Audio Input Control Service Gain Settings Properties value.

BT_UUID_AICS_GAIN_SETTINGS

Audio Input Control Service Gain Settings Properties.

BT_UUID_AICS_INPUT_TYPE_VAL

Audio Input Control Service Input Type value.

BT_UUID_AICS_INPUT_TYPE

Audio Input Control Service Input Type.

BT_UUID_AICS_INPUT_STATUS_VAL

Audio Input Control Service Input Status value.

BT_UUID_AICS_INPUT_STATUS

Audio Input Control Service Input Status.

BT_UUID_AICS_CONTROL_VAL

Audio Input Control Service Control Point value.

BT_UUID_AICS_CONTROL

Audio Input Control Service Control Point.

BT_UUID_AICS_DESCRIPTION_VAL

Audio Input Control Service Input Description value.

BT_UUID_AICS_DESCRIPTION

Audio Input Control Service Input Description.

BT_UUID_VCS_STATE_VAL

Volume Control Setting value.

BT_UUID_VCS_STATE

Volume Control Setting.

BT_UUID_VCS_CONTROL_VAL

Volume Control Control point value.

BT_UUID_VCS_CONTROL

Volume Control Control point.

BT_UUID_VCS_FLAGS_VAL

Volume Control Flags value.

BT_UUID_VCS_FLAGS

Volume Control Flags.

BT_UUID_VOCS_STATE_VAL

Volume Offset State value.

BT_UUID_VOCS_STATE

Volume Offset State.

BT_UUID_VOCS_LOCATION_VAL

Audio Location value.

BT_UUID_VOCS_LOCATION

Audio Location.

BT_UUID_VOCS_CONTROL_VAL

Volume Offset Control Point value.

BT_UUID_VOCS_CONTROL

Volume Offset Control Point.

BT_UUID_VOCS_DESCRIPTION_VAL

Volume Offset Audio Output Description value.

BT_UUID_VOCS_DESCRIPTION

Volume Offset Audio Output Description.

BT_UUID_CSIS_SET_SIRK_VAL

Set Identity Resolving Key value.

BT_UUID_CSIS_SET_SIRK

Set Identity Resolving Key.

BT_UUID_CSIS_SET_SIZE_VAL

Set size value.

BT_UUID_CSIS_SET_SIZE

Set size.

BT_UUID_CSIS_SET_LOCK_VAL

Set lock value.

BT_UUID_CSIS_SET_LOCK

Set lock.

BT_UUID_CSIS_RANK_VAL

Rank value.

BT_UUID_CSIS_RANK

Rank.

BT_UUID_MCS_PLAYER_NAME_VAL

Media player name value.

BT_UUID_MCS_PLAYER_NAME

Media player name.

BT_UUID_MCS_ICON_OBJ_ID_VAL

Media Icon Object ID value.

BT_UUID_MCS_ICON_OBJ_ID

Media Icon Object ID.

BT_UUID_MCS_ICON_URL_VAL

Media Icon URL value.

BT_UUID_MCS_ICON_URL

Media Icon URL.

BT_UUID_MCS_TRACK_CHANGED_VAL

Track Changed value.

BT_UUID_MCS_TRACK_CHANGED

Track Changed.

BT_UUID_MCS_TRACK_TITLE_VAL

Track Title value.

BT_UUID_MCS_TRACK_TITLE

Track Title.

BT_UUID_MCS_TRACK_DURATION_VAL

Track Duration value.

BT_UUID_MCS_TRACK_DURATION

Track Duration.

BT_UUID_MCS_TRACK_POSITION_VAL

Track Position value.

BT_UUID_MCS_TRACK_POSITION

Track Position.

BT_UUID_MCS_PLAYBACK_SPEED_VAL

Playback Speed value.

BT_UUID_MCS_PLAYBACK_SPEED

Playback Speed.

BT_UUID_MCS_SEEKING_SPEED_VAL

Seeking Speed value.

BT_UUID_MCS_SEEKING_SPEED

Seeking Speed.

BT_UUID_MCS_TRACK_SEGMENTS_OBJ_ID_VAL

Track Segments Object ID value.

BT_UUID_MCS_TRACK_SEGMENTS_OBJ_ID

Track Segments Object ID.

BT_UUID_MCS_CURRENT_TRACK_OBJ_ID_VAL

Current Track Object ID value.

BT_UUID_MCS_CURRENT_TRACK_OBJ_ID

Current Track Object ID.

BT_UUID_MCS_NEXT_TRACK_OBJ_ID_VAL

Next Track Object ID value.

BT_UUID_MCS_NEXT_TRACK_OBJ_ID

Next Track Object ID.

BT_UUID_MCS_PARENT_GROUP_OBJ_ID_VAL

Parent Group Object ID value.

BT_UUID_MCS_PARENT_GROUP_OBJ_ID

Parent Group Object ID.

BT_UUID_MCS_CURRENT_GROUP_OBJ_ID_VAL

Group Object ID value.

BT_UUID_MCS_CURRENT_GROUP_OBJ_ID

Group Object ID.

BT_UUID_MCS_PLAYING_ORDER_VAL

Playing Order value.

BT_UUID_MCS_PLAYING_ORDER

Playing Order.

BT_UUID_MCS_PLAYING_ORDERS_VAL

Playing Orders supported value.

BT_UUID_MCS_PLAYING_ORDERS

Playing Orders supported.

BT_UUID_MCS_MEDIA_STATE_VAL

Media State value.

BT_UUID_MCS_MEDIA_STATE

Media State.

BT_UUID_MCS_MEDIA_CONTROL_POINT_VAL

Media Control Point value.

BT_UUID_MCS_MEDIA_CONTROL_POINT

Media Control Point.

BT_UUID_MCS_MEDIA_CONTROL_OPCODES_VAL

Media control opcodes supported value.

BT_UUID_MCS_MEDIA_CONTROL_OPCODES

Media control opcodes supported.

BT_UUID_MCS_SEARCH_RESULTS_OBJ_ID_VAL

Search result object ID value.

BT_UUID_MCS_SEARCH_RESULTS_OBJ_ID

Search result object ID.

BT_UUID_MCS_SEARCH_CONTROL_POINT_VAL

Search control point value.

BT_UUID_MCS_SEARCH_CONTROL_POINT

Search control point.

BT_UUID_OTS_TYPE_MPL_ICON_VAL

Media Player Icon Object Type value.

BT_UUID_OTS_TYPE_MPL_ICON

Media Player Icon Object Type.

BT_UUID_OTS_TYPE_TRACK_SEGMENT_VAL

Track Segments Object Type value.

BT_UUID_OTS_TYPE_TRACK_SEGMENT

Track Segments Object Type.

BT_UUID_OTS_TYPE_TRACK_VAL

Track Object Type value.

BT_UUID_OTS_TYPE_TRACK

Track Object Type.

BT_UUID_OTS_TYPE_GROUP_VAL

Group Object Type value.

BT_UUID_OTS_TYPE_GROUP

Group Object Type.

BT_UUID_TBS_PROVIDER_NAME_VAL

Bearer Provider Name value.

BT_UUID_TBS_PROVIDER_NAME

Bearer Provider Name.

BT_UUID_TBS_UCI_VAL

Bearer UCI value.

BT_UUID_TBS_UCI

Bearer UCI.

BT_UUID_TBS_TECHNOLOGY_VAL

Bearer Technology value.

BT_UUID_TBS_TECHNOLOGY

Bearer Technology.

BT_UUID_TBS_URI_LIST_VAL

Bearer URI Prefixes Supported List value.

BT_UUID_TBS_URI_LIST

Bearer URI Prefixes Supported List.

BT_UUID_TBS_SIGNAL_STRENGTH_VAL

Bearer Signal Strength value.

BT_UUID_TBS_SIGNAL_STRENGTH

Bearer Signal Strength.

BT_UUID_TBS_SIGNAL_INTERVAL_VAL

Bearer Signal Strength Reporting Interval value.

BT_UUID_TBS_SIGNAL_INTERVAL

Bearer Signal Strength Reporting Interval.

BT_UUID_TBS_LIST_CURRENT_CALLS_VAL

Bearer List Current Calls value.

BT_UUID_TBS_LIST_CURRENT_CALLS

Bearer List Current Calls.

BT_UUID_CCID_VAL

Content Control ID value.

BT_UUID_CCID

Content Control ID.

BT_UUID_TBS_STATUS_FLAGS_VAL

Status flags value.

BT_UUID_TBS_STATUS_FLAGS

Status flags.

BT_UUID_TBS_INCOMING_URI_VAL

Incoming Call Target Caller ID value.

BT_UUID_TBS_INCOMING_URI

Incoming Call Target Caller ID.

BT_UUID_TBS_CALL_STATE_VAL

Call State value.

BT_UUID_TBS_CALL_STATE

Call State.

BT_UUID_TBS_CALL_CONTROL_POINT_VAL

Call Control Point value.

BT_UUID_TBS_CALL_CONTROL_POINT

Call Control Point.

BT_UUID_TBS_OPTIONAL_OPCODES_VAL

Optional Opcodes value.

BT_UUID_TBS_OPTIONAL_OPCODES

Optional Opcodes.

BT_UUID_TBS_TERMINATE_REASON_VAL

Terminate reason value.

BT_UUID_TBS_TERMINATE_REASON_VAL

BT_UUID_TBS_TERMINATE_REASON

Terminate reason.

BT_UUID_TBS_TERMINATE_REASON

BT_UUID_TBS_INCOMING_CALL_VAL

Incoming Call value.

BT_UUID_TBS_INCOMING_CALL

Incoming Call.

BT_UUID_TBS_FRIENDLY_NAME_VAL

Incoming Call Friendly name value.

BT_UUID_TBS_FRIENDLY_NAME

Incoming Call Friendly name.

BT_UUID_MICS_MUTE_VAL

Microphone Input Control Service Mute value.

BT_UUID_MICS_MUTE

Microphone Input Control Service Mute.

BT_UUID_ASCS_ASE_SNK_VAL

Audio Stream Endpoint Sink Characteristic value.

BT_UUID_ASCS_ASE_SNK

Audio Stream Endpoint Sink Characteristic.

BT_UUID_ASCS_ASE_SRC_VAL

Audio Stream Endpoint Source Characteristic value.

BT_UUID_ASCS_ASE_SRC

Audio Stream Endpoint Source Characteristic.

BT_UUID_ASCS_ASE_CP_VAL

Audio Stream Endpoint Control Point Characteristic value.

BT_UUID_ASCS_ASE_CP

Audio Stream Endpoint Control Point Characteristic.

BT_UUID_BASS_CONTROL_POINT_VAL

Broadcast Audio Scan Service Scan State value.

BT_UUID_BASS_CONTROL_POINT

Broadcast Audio Scan Service Scan State.

BT_UUID_BASS_RECV_STATE_VAL

Broadcast Audio Scan Service Receive State value.

BT_UUID_BASS_RECV_STATE

Broadcast Audio Scan Service Receive State.

BT_UUID_PACS_SNK_VAL

Sink PAC Characteristic value.

BT_UUID_PACS_SNK

Sink PAC Characteristic.

BT_UUID_PACS_SNK_LOC_VAL

Sink PAC Locations Characteristic value.

BT_UUID_PACS_SNK_LOC

Sink PAC Locations Characteristic.

BT_UUID_PACS_SRC_VAL

Source PAC Characteristic value.

BT_UUID_PACS_SRC

Source PAC Characteristic.

BT_UUID_PACS_SRC_LOC_VAL

Source PAC Locations Characteristic value.

BT_UUID_PACS_SRC_LOC

Source PAC Locations Characteristic.

BT_UUID_PACS_AVAILABLE_CONTEXT_VAL

Available Audio Contexts Characteristic value.

BT_UUID_PACS_AVAILABLE_CONTEXT

Available Audio Contexts Characteristic.

BT_UUID_PACS_SUPPORTED_CONTEXT_VAL

Supported Audio Context Characteristic value.

BT_UUID_PACS_SUPPORTED_CONTEXT

Supported Audio Context Characteristic.

BT_UUID_HAS_HEARING_AID_FEATURES_VAL

Hearing Aid Features Characteristic value.

BT_UUID_HAS_HEARING_AID_FEATURES

Hearing Aid Features Characteristic.

BT_UUID_HAS_PRESET_CONTROL_POINT_VAL

Hearing Aid Preset Control Point Characteristic value.

BT_UUID_HAS_PRESET_CONTROL_POINT

Hearing Aid Preset Control Point Characteristic.

BT_UUID_HAS_ACTIVE_PRESET_INDEX_VAL

Active Preset Index Characteristic value.

BT_UUID_HAS_ACTIVE_PRESET_INDEX

Active Preset Index Characteristic.

BT_UUID_RTUS_VAL

Reference Time Update UUID value.

BT_UUID_RTUS

Reference Time Update Service.

BT_UUID_RTUS_TIME_UPDATE_STATE_VAL

RTUS Characteristic Time Update State UUID value.

BT_UUID_RTUS_TIME_UPDATE_STATE

RTUS Characteristic Time Update State.

BT_UUID_RTUS_CONTROL_POINT_VAL

RTUS Characteristic Time Update COntrol Point UUID value.

BT_UUID_RTUS_CONTROL_POINT

RTUS Characteristic Time Update COntrol Point.

BT_UUID_CTS_LOCAL_TIME_INFORMATION_VAL

CTS Characteristic Local Time Information UUID value.

BT_UUID_CTS_LOCAL_TIME_INFORMATION

CTS Characteristic Local Time Information.

BT_UUID_CTS_REFERENCE_TIME_INFORMATION_VAL

CTS Characteristic Reference Time Information UUID value.

BT_UUID_CTS_REFERENCE_TIME_INFORMATION

CTS Characteristic Reference Time Information.

BT_UUID_NDTS_VAL

Next DST Change UUID value.

BT_UUID_NDTS

Next DST Change.

BT_UUID_NDTS_TIME_WITH_DTS_VAL

NDTS Time with DST UUID value.

BT_UUID_NDTS_TIME_WITH_DTS

Time with DST.	
BT_UUID_SDP_VAL	
BT_UUID_SDP	
BT_UUID_UDP_VAL	
BT_UUID_UDP	
BT_UUID_RFCOMM_VAL	
BT_UUID_RFCOMM	
BT_UUID_TCP_VAL	
BT_UUID_TCP	
BT_UUID_TCS_BIN_VAL	
BT_UUID_TCS_BIN	
BT_UUID_TCS_AT_VAL	
BT_UUID_TCS_AT	
BT_UUID_ATT_VAL	
BT_UUID_ATT	
BT_UUID_OBEX_VAL	
BT_UUID_OBEX	
BT_UUID_IP_VAL	
BT_UUID_IP	
BT_UUID_FTP_VAL	
BT_UUID_FTP	

BT_UUID_HTTP_VAL
BT_UUID_HTTP
BT_UUID_BNEP_VAL
BT_UUID_BNEP
BT_UUID_UPNP_VAL
BT_UUID_UPNP
BT_UUID_HIDP_VAL
BT_UUID_HIDP
BT_UUID_HCRP_CTRL_VAL
BT_UUID_HCRP_CTRL
BT_UUID_HCRP_DATA_VAL
BT_UUID_HCRP_DATA
BT_UUID_HCRP_NOTE_VAL
BT_UUID_HCRP_NOTE
BT_UUID_AVCTP_VAL
BT_UUID_AVCTP
BT_UUID_AVDTP_VAL
BT_UUID_AVDTP
BT_UUID_CMTP_VAL
BT_UUID_CMTP
BT_UUID_UDI_VAL

```
BT_UUID_UDI
BT_UUID_MCAP_CTRL_VAL
BT_UUID_MCAP_CTRL
BT_UUID_MCAP_DATA_VAL
BT_UUID_MCAP_DATA
BT_UUID_L2CAP_VAL
BT_UUID_L2CAP
Enums
enum [anonymous]
     Bluetooth UUID types.
     Values:
     enumerator BT_UUID_TYPE_16
         UUID type 16-bit.
     enumerator BT_UUID_TYPE_32
         UUID type 32-bit.
     enumerator BT_UUID_TYPE_128
         UUID type 128-bit.
Functions
int bt_uuid_cmp(const struct bt_uuid *u1, const struct bt_uuid *u2)
     Compare Bluetooth UUIDs.
     Compares 2 Bluetooth UUIDs, if the types are different both UUIDs are first converted to 128 bits format
     before comparing.
         Parameters
             • u1 – First Bluetooth UUID to compare
             • u2 – Second Bluetooth UUID to compare
             negative value if u1 < u2, 0 if u1 == u2, else positive
```

bool **bt_uuid_create**(struct *bt_uuid* *uuid, const uint8_t *data, uint8_t data_len)

Create a bt_uuid from a little-endian data buffer.

Create a *bt_uuid* from a little-endian data buffer. The data_len parameter is used to determine whether the UUID is in 16, 32 or 128 bit format (length 2, 4 or 16). Note: 32 bit format is not allowed over the air.

Parameters

- **uuid** Pointer to the *bt_uuid* variable
- data pointer to UUID stored in little-endian data buffer
- data_len length of the UUID in the data buffer

Returns

true if the data was valid and the UUID was successfully created.

```
void bt_uuid_to_str(const struct bt_uuid *uuid, char *str, size_t len)
```

Convert Bluetooth UUID to string.

Converts Bluetooth UUID to string. UUID can be in any format, 16-bit, 32-bit or 128-bit.

Parameters

- uuid Bluetooth UUID
- **str** pointer where to put converted string
- len length of str

struct bt_uuid

#include <uuid.h> This is a 'tentative' type and should be used as a pointer only.

struct bt_uuid_16

#include <uuid.h>

Public Members

struct bt_uuid uuid

UUID generic type.

uint 16_t val

UUID value, 16-bit in host endianness.

struct bt_uuid_32

#include <uuid.h>

Public Members

```
struct bt_uuid uuid

UUID generic type.

uint32_t val

UUID value, 32-bit in host endianness.

struct bt_uuid_128

#include <uuid.h>

Public Members

struct bt_uuid uuid

UUID generic type.

uint8_t val[16]

UUID value, 128-bit in little-endian format.
```

1.14 services

1.14.1 HTTP Proxy Service (HPS)

1.14.1.1 API Reference

```
group bt_hps
```

HTTP Proxy Service (HPS)

[Experimental] Users should note that the APIs can change as a part of ongoing development.

Defines

MAX_URI_LEN

MAX_HEADERS_LEN

MAX_BODY_LEN

Typedefs

```
typedef uint8_t hps_data_status_t
typedef uint8_t hps_http_command_t
typedef uint8_t hps_state_t
typedef uint8_t hps_flags_t
Enums
enum [anonymous]
    Values:
    enumerator HPS_HEADERS_RECEIVED
    enumerator HPS_HEADERS_TRUNCATED
    enumerator HPS_BODY_RECEIVED
    enumerator HPS_BODY_TRUNCATED
enum [anonymous]
    Values:
    enumerator HTTP_GET_REQ
    enumerator HTTP_HEAD_REQ
    enumerator HTTP_POST_REQ
    enumerator HTTP_PUT_REQ
    enumerator HTTP_DELETE_REQ
    enumerator HTTPS_GET_REQ
    enumerator HTTPS_HEAD_REQ
    enumerator HTTPS_POST_REQ
```

1.14. services 215

```
enumerator HTTPS_PUT_REQ
    enumerator HTTPS_DELETE_REQ
    enumerator HTTP_REQ_CANCEL
enum [anonymous]
    Values:
    enumerator IDLE_STATE
    enumerator BUSY_STATE
enum [anonymous]
    Values:
    enumerator URI_SET
    enumerator HEADERS_SET
    enumerator BODY_SET
enum [anonymous]
    Values:
    enumerator HPS_ERR_INVALID_REQUEST
    enumerator HPS_ERR_CCCD_IMPROPERLY_CONFIGURED
    enumerator HPS_ERR_PROC_ALREADY_IN_PROGRESS
enum [anonymous]
    Values:
    enumerator HTTPS_CERTIFICATE_INVALID
    enumerator HTTPS_CERTIFICATE_VALID
```

216

Functions

ssize_t write_http_headers (struct bt_conn *conn, const struct bt_gatt_attr *attr, const void *buf, uint16_t len, uint16_t offset, uint8_t flags)

HTTP Headers GATT write callback.

If called with conn == NULL, it is a local write.

Returns

Number of bytes written.

ssize_t write_http_entity_body(struct bt_conn *conn, const struct bt_gatt_attr *attr, const void *buf, uint16_t len, uint16_t offset, uint8_t flags)

HTTP Entity Body GATT write callback.

If called with conn == NULL, it is a local write.

Returns

Number of bytes written.

int bt_hps_init(osa_msgq_handle_t queue)

HTTP Proxy Server initialization.

Returns

Zero in case of success and error code in case of error.

void bt_hps_set_status_code(uint16 t http status)

Sets Status Code after HTTP request was fulfilled.

int bt_hps_notify(void)

Notify HTTP Status after Request was fulfilled.

This will send a GATT notification to the subscriber.

Returns

Zero in case of success and error code in case of error.

struct hps_status_t

#include <hps.h>

struct hps_config_t

#include <hps.h>

1.14.2 Health Thermometer Service (HTS)

1.14.2.1 API Reference

group bt_hts

Health Thermometer Service (HTS)

[Experimental] Users should note that the APIs can change as a part of ongoing development.

1.14. services 217

Defines

```
hts_unit_celsius_c
hts_unit_fahrenheit_c
hts_include_temp_type
Enums
enum [anonymous]
    Values:
    enumerator hts_no_temp_type
    enumerator hts_armpit
    enumerator hts_body
    enumerator hts_ear
    enumerator hts_finger
    enumerator hts_gastroInt
    enumerator hts_mouth
    enumerator hts_rectum
```

Functions

void bt_hts_indicate(void)

enumerator hts_toe

enumerator hts_tympanum

Notify indicate a temperature measurement.

This will send a GATT indication to all current subscribers. Awaits an indication response from peer.

Parameters

• none. -

Returns

Zero in case of success and error code in case of error.

struct temp_measurement

#include <hts.h>

1.14.3 Internet Protocol Support Profile (IPSP)

1.14.3.1 API Reference

group bt_ipsp

Internet Protocol Support Profile (IPSP)

Defines

USER_DATA_MIN

Typedefs

typedef int (*ipsp_rx_cb_t)(struct net_buf *buf)

Functions

```
int ipsp_init(ipsp_rx_cb_t pf_rx_cb)
```

Initialize the service.

This will setup the data receive callback.

Parameters

• **pf_rx_cb** – Pointer to the callback used for receiving data.

Returns

Zero in case of success and error code in case of error.

```
int ipsp_connect(struct bt_conn *conn)
```

Start a connection to an IPSP Node using this connection.

This will try to connect to the Node present.

Parameters

• conn – Pointer to the connection to be used.

Returns

Zero in case of success and error code in case of error.

int ipsp_send(struct net_buf *buf)

Send data to the peer IPSP Node/Router.

Parameters

• **conn** – Pointer to the buffer containing data.

Returns

Zero in case of success and error code in case of error.

1.14. services 219

```
int ipsp_listen(void)
          Setup an IPSP Server.
              Returns
                  Zero in case of success and error code in case of error.
1.14.4 Proximity Reporter (PXR)
1.14.4.1 API Reference
group bt_pxr
     Proximity Reporter (PXR)
     Typedefs
     typedef void (*alert_ui_cb)(uint8_t param)
     Enums
     enum [anonymous]
          Values:
          enumerator NO_ALERT
          enumerator MILD_ALERT
          enumerator HIGH_ALERT
```

Functions

Returns

Number of bytes read.

```
ssize_t write_ias_alert_level(struct bt_conn *conn, const struct bt_gatt_attr *attr, const void *buf, uint16_t len, uint16_t offset, uint8_t flags)

IAS Alert Level GATT write callback.

If called with conn == NULL, it is a local write.

Returns

Number of bytes written.

ssize_t read_lls_alert_level(struct bt_conn *conn, const struct bt_gatt_attr *attr, void *buf, uint16_t len, uint16_t offset)

IAS Alert Level GATT read callback.
```

```
ssize_t write_lls_alert_level(struct bt_conn *conn, const struct bt_gatt_attr *attr, const void *buf, uint16_t len, uint16_t offset, uint8_t flags)
```

LLS Alert Level GATT write callback.

If called with conn == NULL, it is a local write.

Returns

Number of bytes written.

ssize_t **read_tps_power_level**(struct bt_conn *conn, const struct *bt_gatt_attr* *attr, void *buf, uint16_t len, uint16_t offset)

TPS Power Level GATT read callback.

Returns

Number of bytes read.

ssize_t **read_tps_power_level_desc**(struct bt_conn *conn, const struct bt_gatt_attr *attr, void *buf, uint16 t len, uint16 t offset)

TPS Power Level Descriptor GATT read callback.

Returns

Number of bytes read.

uint8_t pxr_lls_get_alert_level(void)

Read LLS Alert Level locally.

Returns

Number of bytes written.

uint8_t pxr_ias_get_alert_level(void)

Read IAS Alert Level locally.

Returns

Number of bytes written.

int8_t pxr_tps_get_power_level(void)

Read TPS Power Level locally.

Returns

Number of bytes written.

void pxr_tps_set_power_level(int8_t power_level)

Write TPS Power Level locally.

Returns

Number of bytes written.

int pxr_init(alert_ui_cb cb)

Initialize PXR Service.

Returns

Success or error.

int pxr_deinit(void)

Deinitialize PXR Service.

Returns

Success or error.

1.14. services 221

INDEX

Symbols	_hfp_ag_call_status_t.hfp_ag_call_call_outgoing
_BT_GATT_ATTRS_ARRAY_DEFINE (C macro), 92	(C enumerator), 122
_BT_GATT_SERVICE_ARRAY_ITEM (C macro), 92	_hfp_ag_cind_t (<i>C struct</i>), 132
_bt_gatt_ccc (<i>C struct</i>), 105	_hfp_ag_get_config(C struct), 132
_bt_gatt_ccc.cfg (C var), 105	[anonymous] (<i>C enum</i>), 5–7, 38, 41–43, 81, 82, 88, 95,
_bt_gatt_ccc.cfg_changed (C var), 105	109, 110, 149, 163, 169, 212, 215, 216, 218, 220
_bt_gatt_ccc.cfg_match (C var), 105	[anonymous].BODY_SET (C enumerator), 216
_bt_gatt_ccc.cfg_write(C var), 105	[anonymous].BT_A2DP_DISCOVER_ENDPOINT_CONTINUE
_bt_gatt_ccc.value (C var), 105	(C enumerator), 169
_bt_security (C enum), 6	[anonymous].BT_A2DP_DISCOVER_ENDPOINT_STOP
	(C enumerator), 169
_bt_security.BT_SECURITY_FORCE_PAIR (C enu-	[anonymous].BT_CONN_LE_OPT_CODED (C enumera-
merator), 6	tor), 7
_bt_security.BT_SECURITY_L0 (C enumerator), 6	[anonymous].BT_CONN_LE_OPT_NONE(C enumerator),
_bt_security.BT_SECURITY_L1 (C enumerator), 6	7
_bt_security.BT_SECURITY_L2 (C enumerator), 6	[anonymous].BT_CONN_LE_OPT_NO_1M (C enumera-
_bt_security.BT_SECURITY_L3 (C enumerator), 6	tor), 7
_bt_security.BT_SECURITY_L4 (C enumerator), 6	[anonymous].BT_CONN_LE_PHY_OPT_CODED_S2 (C
_bt_spp_callback (<i>C struct</i>), 177	enumerator), 5
_hf_multiparty_call_option_t(<i>C enum</i>), 122	flanonymous: ,RT_CONN_LE_PHY_OPT_CODED_S8 (C
_hf_multiparty_call_option_t.hf_multiparty_ca	111_OPTION_II ve _CONN_LE_FHI_OFI_CODED_36 (C
(C enumerator), 123	enumerator), 5
_hf_multiparty_call_option_t.hf_multiparty_ca	11anonymous BT_CONN_LE_PHY_OPT_NONE (C enu-
(Cenumerator) 123	meraior), 3
_hf_multiparty_call_option_t.hf_multiparty_ca	l[anonymous] BT_CONN_ROLE_CENTRAL (C enumera-
(C enumerator), 123	<i>tor</i>), 6
_hf_multiparty_call_option_t.hf_multiparty_ca	1hanonymous hree CONN_ROLE_PERIPHERAL (C enu-
$(C_{anymarator})$ 123	merator), 6
_hf_multiparty_call_option_t.hf_multiparty_ca	11 Lanonymous Looks TYPE_ALL (C enumerator), 5
(C enumerator), 123	[anonymous].bi_conn_life_bk (c enumerator), 3
_hf_volume_type_t (<i>C enum</i>), 121, 122	[anonymous].BT_CONN_TYPE_ISO(C enumerator), 5
_hf_volume_type_t.hf_volume_type_mic (C enu-	[anonymous].BT_CONN_TYPE_LE(C enumerator), 5
merator), 121, 122	[anonymous].BT_CONN_TYPE_SCO(<i>C enumerator</i>), 5
_hf_volume_type_t.hf_volume_type_speaker (C	$[anonymous]. BT_GAP_ADV_PROP_CONNECTABLE \qquad (C$
enumerator), 121, 122	enumerator), 81
_hf_waiting_call_state_t(C struct), 134	[anonymous].BT_GAP_ADV_PROP_DIRECTED ($C\ enu$ -
_hfp_ag_call_status_t (C enum), 121	merator), 82
_hfp_ag_call_status_t.hfp_ag_call_call_active	[anonymous].BT_GAP_ADV_PROP_EXT_ADV (C enu-
(C enumerator), 122	merator), 82
_hfp_ag_call_status_t.hfp_ag_call_call_end	$[anonymous]. \verb BT_GAP_ADV_PROP_SCANNABLE (C enu-$
(Conumerator) 122	merator), 82
hfn ag call status t.hfn ag call call incomi	n[anonymous].BT_GAP_ADV_PROP_SCAN_RESPONSE (C
(C enumerator), 122	enumerator), 82
(C chancianoi), 122	

- [anonymous].BT_GAP_ADV_TYPE_ADV_DIRECT_IND (*C enumerator*), 81
- [anonymous].BT_GAP_ADV_TYPE_ADV_IND (C enumerator), 81
- [anonymous].BT_GAP_ADV_TYPE_ADV_NONCONN_IND (C enumerator), 81
- [anonymous].BT_GAP_ADV_TYPE_ADV_SCAN_IND (C enumerator), 81
- [anonymous].BT_GAP_ADV_TYPE_EXT_ADV (C enumerator), 81
- [anonymous].BT_GAP_ADV_TYPE_SCAN_RSP (C enumerator), 81
- [anonymous].BT_GAP_CTE_AOA(C enumerator), 82
- [anonymous].BT_GAP_CTE_AOD_1US (C enumerator), 82
- [anonymous].BT_GAP_CTE_AOD_2US (*C enumerator*),
- [anonymous].BT_GAP_CTE_NONE(C enumerator), 82
- [anonymous].BT_GAP_LE_PHY_1M (C enumerator), 81
- [anonymous].BT_GAP_LE_PHY_2M (C enumerator), 81
- [anonymous] .BT_GAP_LE_PHY_CODED ($C\ enumerator$),
- [anonymous].BT_GAP_LE_PHY_NONE (*C enumerator*),
- [anonymous].BT_GAP_SCA_0_20 (C enumerator), 83
- [anonymous].BT_GAP_SCA_101_150 (*C enumerator*),
- [anonymous].BT_GAP_SCA_151_250 (*C enumerator*),
- $[{\tt anonymous}]. {\tt BT_GAP_SCA_21_30} \ ({\it Cenumerator}), 83$
- [anonymous].BT_GAP_SCA_251_500 (C enumerator), 82
- [anonymous].BT_GAP_SCA_31_50 (C enumerator), 83
- [anonymous].BT_GAP_SCA_51_75 (C enumerator), 82
- $\label{eq:cap_scale} \begin{tabular}{ll} [anonymous]. BT_GAP_SCA_76_100 & (C enumerator), \\ 82 & \\ \end{tabular}$
- [anonymous].BT_GAP_SCA_UNKNOWN (*C enumerator*),
- [anonymous].BT_GATT_DISCOVER_ATTRIBUTE (C enumerator), 110
- [anonymous].BT_GATT_DISCOVER_CHARACTERISTIC (C enumerator), 110
- [anonymous].BT_GATT_DISCOVER_DESCRIPTOR (C enumerator), 110
- [anonymous].BT_GATT_DISCOVER_INCLUDE (C enumerator), 110
- [anonymous].BT_GATT_DISCOVER_PRIMARY (C enumerator), 109
- [anonymous].BT_GATT_DISCOVER_SECONDARY (C enumerator), 109
- [anonymous].BT_GATT_DISCOVER_STD_CHAR_DESC (C enumerator), 110
- [anonymous].BT_GATT_ITER_CONTINUE (C enumerator), 95

- [anonymous].BT_GATT_ITER_STOP (C enumerator), 95
- [anonymous].BT_GATT_SUBSCRIBE_FLAG_NO_RESUB (C enumerator), 111
- [anonymous].BT_GATT_SUBSCRIBE_FLAG_VOLATILE (C enumerator), 110
- [anonymous].BT_GATT_SUBSCRIBE_FLAG_WRITE_PENDING (C enumerator), 111
- [anonymous].BT_GATT_SUBSCRIBE_NUM_FLAGS (C enumerator), 111
- [anonymous].BT_GATT_WRITE_FLAG_CMD ($C\ enumer-ator$), 88
- [anonymous].BT_GATT_WRITE_FLAG_EXECUTE (C enumerator), 88
- [anonymous].BT_GATT_WRITE_FLAG_PREPARE (C enumerator), 88
- [anonymous].BT_LE_ADV_OPT_ANONYMOUS (C enumerator), 40
- [anonymous].BT_LE_ADV_OPT_CODED(C enumerator),
- [anonymous].BT_LE_ADV_OPT_CONNECTABLE (C enumerator), 38
- [anonymous].BT_LE_ADV_OPT_DIR_ADDR_RPA (C enumerator).39
- [anonymous].BT_LE_ADV_OPT_DIR_MODE_LOW_DUTY (C enumerator), 39
- [anonymous].BT_LE_ADV_OPT_DISABLE_CHAN_37 (C enumerator), 41
- [anonymous].BT_LE_ADV_OPT_DISABLE_CHAN_38 (*C enumerator*), 41
- $[{\tt anonymous}]. {\tt BT_LE_ADV_OPT_DISABLE_CHAN_39}~(C \\ enumerator), 41$
- [anonymous].BT_LE_ADV_OPT_EXT_ADV (C enumerator), 40
- [anonymous].BT_LE_ADV_OPT_FILTER_CONN (C enumerator), 39
- [anonymous].BT_LE_ADV_OPT_FILTER_SCAN_REQ (C enumerator), 39
- [anonymous].BT_LE_ADV_OPT_FORCE_NAME_IN_AD (C enumerator), 41
- [anonymous].BT_LE_ADV_OPT_NONE (C enumerator), 38
- [anonymous].BT_LE_ADV_OPT_NOTIFY_SCAN_REQ (C enumerator), 39
- [anonymous].BT_LE_ADV_OPT_NO_2M(C enumerator),
- [anonymous].BT_LE_ADV_OPT_ONE_TIME (*C enumerator*), 38
- [anonymous].BT_LE_ADV_OPT_SCANNABLE (C enumerator), 39
- [anonymous].BT_LE_ADV_OPT_USE_IDENTITY $(Ce^{numerator})$, 38
- [anonymous].BT_LE_ADV_OPT_USE_NAME (*C enumerator*), 39

```
[anonymous].BT_LE_ADV_OPT_USE_TX_POWER
                                                                                                       [anonymous].BT_RFCOMM_CHAN_HSP_HS (C enumera-
                 enumerator), 40
                                                                                                                        tor), 149
                                                                                                       [anonymous].BT_RFCOMM_CHAN_SPP (C enumerator),
[anonymous].BT_LE_PER_ADV_OPT_INCLUDE_ADI (C
                enumerator), 41
[anonymous].BT_LE_PER_ADV_OPT_NONE (C enumer-
                                                                                                       [anonymous].BT_SDP_DISCOVER_UUID_CONTINUE (C
                ator), 41
                                                                                                                        enumerator), 163
[anonymous].BT_LE_PER_ADV_OPT_USE_TX_POWER
                                                                                                       [anonymous].BT_SDP_DISCOVER_UUID_STOP (C enu-
                 (C enumerator), 41
                                                                                                                        merator), 163
[anonymous].BT_LE_PER_ADV_SYNC_OPT_DONT_SYNC_ADAnonymous].BT_UUID_TYPE_128(Cenumerator), 212
                (C enumerator), 42
                                                                                                       [anonymous].BT_UUID_TYPE_16 (C enumerator), 212
[anonymous].BT_LE_PER_ADV_SYNC_OPT_DONT_SYNC_ADDOMD@bigsmous].BT_UUID_TYPE_32 (C enumerator), 212
                                                                                                       [anonymous].BUSY_STATE (C enumerator), 216
                (C enumerator), 42
[anonymous].BT_LE_PER_ADV_SYNC_OPT_DONT_SYNC_ADDO_200505mous].HEADERS_SET (C enumerator), 216
                                                                                                       [anonymous].HIGH_ALERT (C enumerator), 220
                (C enumerator), 42
[anonymous].BT_LE_PER_ADV_SYNC_OPT_FILTER_DUPL[AGDACHDED: HPS_BODY_RECEIVED (C enumerator),
                 (C enumerator), 42
[anonymous].BT_LE_PER_ADV_SYNC_OPT_NONE
                                                                                             (C [anonymous].HPS_BODY_TRUNCATED (C enumerator),
                                                                                                                        215
                enumerator), 41
[anonymous].BT_LE_PER_ADV_SYNC_OPT_REPORTING_INATIONALITY INSTRUMENTAL STATES AND STATES
                 (C enumerator), 42
                                                                                                                        (C enumerator), 216
[anonymous].BT_LE_PER_ADV_SYNC_OPT_SYNC_ONLY_CDANSON_STROOMES_EXTPS_ERR_INVALID_REQUEST (C enu-
                (C enumerator), 42
                                                                                                                        merator), 216
[anonymous].BT_LE_PER_ADV_SYNC_OPT_USE_PER_ADV[atriCsftymous].HPS_ERR_PROC_ALREADY_IN_PROGRESS
                 (C enumerator), 41
                                                                                                                        (C enumerator), 216
[anonymous].BT_LE_PER_ADV_SYNC_TRANSFER_OPT_NONE On on one of the companies of the companie
                (C enumerator), 42
                                                                                                                        tor), 215
[anonymous].BT_LE_PER_ADV_SYNC_TRANSFER_OPT_SYNandidothymbolas].HPS_HEADERS_TRUNCATED (C enumera-
                 (C enumerator), 42
                                                                                                                        tor), 215
[anonymous].BT_LE_PER_ADV_SYNC_TRANSFER_OPT_SYNandidymAddbs_11#STTPS_CERTIFICATE_INVALID (C enu-
                 (C enumerator), 42
                                                                                                                        merator), 216
[anonymous].BT_LE_PER_ADV_SYNC_TRANSFER_OPT_SYNandidynakolbs_PLESTTPS_CERTIFICATE_VALID (C enu-
                 (C enumerator), 42
                                                                                                                        merator), 216
[anonymous].BT_LE_PER_ADV_SYNC_TRANSFER_OPT_SYNandomymous]E.HTTPS_DELETE_REQ(C enumerator), 216
                (C enumerator), 42
                                                                                                       [anonymous].HTTPS_GET_REQ(C enumerator), 215
[anonymous].BT_LE_SCAN_OPT_CODED (C enumera-
                                                                                                       [anonymous].HTTPS_HEAD_REQ(C enumerator), 215
                tor), 43
                                                                                                       [anonymous].HTTPS_POST_REQ(C enumerator), 215
[anonymous].BT_LE_SCAN_OPT_FILTER_ACCEPT_LIST [anonymous].HTTPS_PUT_REQ(C enumerator), 215
                 (C enumerator), 43
                                                                                                       [anonymous].HTTP_DELETE_REQ(C enumerator), 215
[anonymous].BT_LE_SCAN_OPT_FILTER_DUPLICATE
                                                                                                       [anonymous].HTTP_GET_REQ(C enumerator), 215
                                                                                                       [anonymous].HTTP_HEAD_REQ(C enumerator), 215
                (C enumerator), 43
[anonymous].BT_LE_SCAN_OPT_NONE(C enumerator),
                                                                                                      [anonymous].HTTP_POST_REQ(C enumerator), 215
                                                                                                       [anonymous].HTTP_PUT_REQ(C enumerator), 215
[anonymous].BT_LE_SCAN_OPT_NO_1M (C enumera-
                                                                                                       [anonymous].HTTP_REQ_CANCEL (C enumerator), 216
                tor), 43
                                                                                                       [anonymous].IDLE_STATE (C enumerator), 216
[anonymous].BT_LE_SCAN_TYPE_ACTIVE (C enumer-
                                                                                                       [anonymous].MILD_ALERT (C enumerator), 220
                                                                                                       [anonymous].NO_ALERT (C enumerator), 220
                ator), 43
[anonymous].BT_LE_SCAN_TYPE_PASSIVE (C enu-
                                                                                                       [anonymous].URI_SET (C enumerator), 216
                merator), 43
                                                                                                       [anonymous].hts_armpit (C enumerator), 218
[anonymous].BT_RFCOMM_CHAN_HFP_AG (C enumera-
                                                                                                      [anonymous].hts_body (C enumerator), 218
                                                                                                       [anonymous].hts_ear(Cenumerator), 218
                tor), 149
[anonymous].BT_RFCOMM_CHAN_HFP_HF (C enumera-
                                                                                                       [anonymous].hts_finger(Cenumerator), 218
                tor), 149
                                                                                                       [anonymous].hts_gastroInt(Cenumerator), 218
[anonymous].BT_RFCOMM_CHAN_HSP_AG (C enumera-
                                                                                                       [anonymous].hts_mouth(Cenumerator), 218
                                                                                                       [anonymous].hts_no_temp_type(Cenumerator),218
                tor), 149
```

```
[anonymous].hts_rectum (C enumerator), 218
                                                 bt_a2dp_endpoint_configure_result.a2dp
                                                                                               (C
[anonymous].hts_toe(C enumerator), 218
                                                          var), 172
[anonymous].hts_tympanum(C enumerator), 218
                                                 bt_a2dp_endpoint_configure_result.config
                                                                                               (C
                                                          var), 172
                                                 bt_a2dp_endpoint_configure_result.conn
                                                                                               (C
                                                          var), 172
alert_ui_cb (C type), 220
                                                 bt_a2dp_endpoint_configure_result.err (C var),
В
                                                 BT_A2DP_ENDPOINT_INIT (C macro), 166
bt_a2dp_codec_id(Cenum), 168
                                                 BT_A2DP_EP_CONTENT_PROTECTION_INIT (C macro),
bt_a2dp_codec_id.BT_A2DP_ATRAC (C enumerator),
                                                 BT_A2DP_EP_DELAY_REPORTING_INIT (C macro), 166
bt_a2dp_codec_id.BT_A2DP_MPEG1 (C enumerator),
                                                 BT_A2DP_EP_HEADER_COMPRESSION_INIT (C macro),
                                                          166
bt_a2dp_codec_id.BT_A2DP_MPEG2 (C enumerator),
                                                 BT_A2DP_EP_MULTIPLEXING_INIT (C macro), 166
                                                 BT_A2DP_EP_RECOVERY_SERVICE_INIT (C macro), 166
bt_a2dp_codec_id.BT_A2DP_SBC(Cenumerator), 168
                                                 BT_A2DP_EP_REPORTING_SERVICE_INIT (C macro),
bt_a2dp_codec_id.BT_A2DP_VENDOR(C enumerator),
                                                          166
                                                 BT_A2DP_MPEG_1_2_IE_LENGTH (C macro), 166
bt_a2dp_codec_ie (C struct), 171
                                                 BT_A2DP_MPEG_2_4_IE_LENGTH (C macro), 166
bt_a2dp_codec_ie.codec_ie(C var), 172
                                                 bt_a2dp_reconfigure (C function), 171
bt_a2dp_codec_ie.len(C var), 172
                                                 bt_a2dp_register_connect_callback (C function),
bt_a2dp_configure (C function), 170
                                                          170
bt_a2dp_configure_endpoint(C function), 170
                                                 bt_a2dp_register_endpoint (C function), 169
bt_a2dp_connect (C function), 169
                                                 BT_A2DP_SBC_IE_LENGTH (C macro), 166
bt_a2dp_connect_cb (C struct), 173
                                                 BT_A2DP_SBC_SINK_ENDPOINT (C macro), 167
bt_a2dp_connect_cb.connected (C var), 173
                                                 BT_A2DP_SBC_SOURCE_ENDPOINT (C macro), 167
bt_a2dp_connect_cb.disconnected (C var), 173
                                                 BT_A2DP_SINK_ENDPOINT_INIT (C macro), 167
bt_a2dp_control_cb (C struct), 172
                                                 BT_A2DP_SINK_SBC_CODEC_BUFFER_NOCACHED_SIZE
bt_a2dp_control_cb.configured(C var), 173
                                                          (C macro), 166
bt_a2dp_control_cb.deconfigured (C var), 173
                                                 BT_A2DP_SINK_SBC_CODEC_BUFFER_SIZE (C macro),
bt_a2dp_control_cb.sink_streamer_data (C var),
                                                          166
                                                 BT_A2DP_SOURCE_ENDPOINT_INIT (C macro), 167
bt_a2dp_control_cb.start_play (C var), 173
                                                 BT_A2DP_SOURCE_SBC_CODEC_BUFFER_NOCACHED_SIZE
bt_a2dp_control_cb.stop_play (C var), 173
                                                          (C macro), 166
bt_a2dp_deconfigure (C function), 171
                                                 BT_A2DP_SOURCE_SBC_CODEC_BUFFER_SIZE
                                                                                               (C
bt_a2dp_disconnect (C function), 169
                                                          macro), 166
bt_a2dp_discover_peer_endpoint_cb_t (C type),
                                                 bt_a2dp_start (C function), 171
                                                 bt_a2dp_stop (C function), 171
bt_a2dp_discover_peer_endpoints (C function),
                                                 BT_ADDR_ANY (C macro), 71
        170
                                                 bt_addr_any (C var), 74
bt_a2dp_endpoint (C struct), 173
                                                 bt_addr_cmp (C function), 72
bt_a2dp_endpoint.capabilities (C var), 174
                                                 bt_addr_copy (C function), 73
bt_a2dp_endpoint.codec_buffer (C var), 174
                                                 bt_addr_from_str (C function), 74
bt_a2dp_endpoint.codec_buffer_nocached
                                                 BT_ADDR_IS_NRPA (C macro), 72
        var), 174
                                                 BT_ADDR_IS_RPA (C macro), 72
bt_a2dp_endpoint.codec_id(Cvar), 174
                                                 BT_ADDR_IS_STATIC (C macro), 72
bt_a2dp_endpoint.config(Cvar), 174
                                                 BT_ADDR_LE_ANONYMOUS (C macro), 71
bt_a2dp_endpoint.control_cbs (C var), 174
                                                 BT_ADDR_LE_ANY (C macro), 71
bt_a2dp_endpoint.info(C var), 174
                                                 bt_addr_le_any (C var), 74
bt_a2dp_endpoint_config (C struct), 172
                                                 bt_addr_le_cmp (C function), 72
bt_a2dp_endpoint_config.media_config (C var),
                                                 bt_addr_le_copy (C function), 73
                                                 bt_addr_le_create_nrpa (C function), 73
bt_a2dp_endpoint_configure_result (C struct),
                                                 bt_addr_le_create_static (C function), 73
        172
```

bt_addr_le_from_str(C function), 74	bt_buf_get_evt (C function), 33
bt_addr_le_is_identity (C function), 73	bt_buf_get_rx (C function), 32
bt_addr_le_is_rpa (C function), 73	bt_buf_get_tx (C function), 32
BT_ADDR_LE_NONE (C macro), 72	bt_buf_get_type (C function), 33
bt_addr_le_none (C var), 74	BT_BUF_ISO_RX_COUNT (C macro), 31
BT_ADDR_LE_PUBLIC (C macro), 71	BT_BUF_ISO_RX_SIZE (C macro), 31
BT_ADDR_LE_PUBLIC_ID (C macro), 71	BT_BUF_ISO_SIZE (C macro), 31
BT_ADDR_LE_RANDOM (C macro), 71	BT_BUF_RESERVE (C macro), 31
BT_ADDR_LE_RANDOM_ID (C macro), 71	BT_BUF_RX_COUNT (C macro), 31
BT_ADDR_LE_SIZE (C macro), 71	BT_BUF_RX_SIZE (C macro), 31
BT_ADDR_LE_STR_LEN (C macro), 72	<pre>bt_buf_set_type (C function), 33</pre>
bt_addr_le_t (C struct), 74	BT_BUF_SIZE (C macro), 31
bt_addr_le_to_str (C function), 74	bt_buf_type (<i>C enum</i>), 31
BT_ADDR_LE_UNRESOLVED (C macro), 71	<pre>bt_buf_type.BT_BUF_ACL_IN(C enumerator), 32</pre>
BT_ADDR_NONE (C macro), 71	<pre>bt_buf_type.BT_BUF_ACL_OUT (C enumerator), 32</pre>
bt_addr_none (C var), 74	<pre>bt_buf_type.BT_BUF_CMD (C enumerator), 31</pre>
BT_ADDR_SET_NRPA (C macro), 72	<pre>bt_buf_type.BT_BUF_EVT (C enumerator), 31</pre>
BT_ADDR_SET_RPA (C macro), 72	<pre>bt_buf_type.BT_BUF_H4 (C enumerator), 32</pre>
BT_ADDR_SET_STATIC (C macro), 72	<pre>bt_buf_type.BT_BUF_ISO_IN(C enumerator), 32</pre>
BT_ADDR_SIZE (C macro), 71	<pre>bt_buf_type.BT_BUF_ISO_OUT (C enumerator), 32</pre>
BT_ADDR_STR_LEN (C macro), 72	BT_COMP_ID_LF (C macro), 75
bt_addr_t (C struct), 74	<pre>bt_configure_data_path (C function), 59</pre>
<pre>bt_addr_to_str (C function), 73</pre>	<pre>bt_conn_auth_cancel (C function), 17</pre>
bt_bond_info(C struct), 71	bt_conn_auth_cb (C struct), 26
bt_bond_info.addr(Cvar),71	<pre>bt_conn_auth_cb.cancel (C var), 28</pre>
BT_BR_CONN_PARAM (C macro), 4	<pre>bt_conn_auth_cb.oob_data_request(C var), 28</pre>
bt_br_conn_param(C struct), 29	<pre>bt_conn_auth_cb.pairing_accept (C var), 26</pre>
BT_BR_CONN_PARAM_DEFAULT (C macro), 5	<pre>bt_conn_auth_cb.pairing_confirm(C var), 28</pre>
BT_BR_CONN_PARAM_INIT (C macro), 4	<pre>bt_conn_auth_cb.passkey_confirm(C var), 27</pre>
bt_br_discovery_cb_t (C type), 38	<pre>bt_conn_auth_cb.passkey_display(C var), 27</pre>
bt_br_discovery_param(C struct),70	<pre>bt_conn_auth_cb.passkey_entry(C var), 27</pre>
$bt_br_discovery_param.length(C var), 70$	<pre>bt_conn_auth_cb.pincode_entry(C var), 28</pre>
<pre>bt_br_discovery_param.limited(C var), 70</pre>	<pre>bt_conn_auth_cb_overlay (C function), 16</pre>
<pre>bt_br_discovery_result (C struct), 70</pre>	<pre>bt_conn_auth_cb_register(C function), 15</pre>
<pre>bt_br_discovery_resultpriv(C var), 70</pre>	bt_conn_auth_info_cb (C struct), 29
bt_br_discovery_result.addr(Cvar),70	<pre>bt_conn_auth_info_cb.bond_deleted(C var), 29</pre>
<pre>bt_br_discovery_result.cod(C var), 70</pre>	bt_conn_auth_info_cb.node(C var), 29
<pre>bt_br_discovery_result.eir(C var), 70</pre>	<pre>bt_conn_auth_info_cb.pairing_complete (C var),</pre>
<pre>bt_br_discovery_result.rssi(C var), 70</pre>	29
bt_br_discovery_start (C function), 57	<pre>bt_conn_auth_info_cb.pairing_failed(C var), 29</pre>
bt_br_discovery_stop(C function), 58	bt_conn_auth_info_cb_register (C function), 16
bt_br_oob (C struct), 70	bt_conn_auth_info_cb_unregister(C function), 16
bt_br_oob.addr (C var), 71	bt_conn_auth_pairing_confirm(C function), 17
bt_br_oob_get_local (C function), 58	bt_conn_auth_passkey_confirm(C function), 17
bt_br_set_connectable (C function), 58	bt_conn_auth_passkey_entry (C function), 16
bt_br_set_discoverable (C function), 58	bt_conn_auth_pincode_entry (C function), 17
BT_BUF_ACL_RX_SIZE (C macro), 31	bt_conn_br_info (C struct), 20
BT_BUF_ACL_SIZE (C macro), 31	bt_conn_br_remote_info(C struct), 21
BT_BUF_CMD_SIZE (C macro), 31	bt_conn_br_remote_info.features(C var), 21
BT_BUF_CMD_TX_SIZE (C macro), 31	bt_conn_br_remote_info.num_pages (C var), 21
bt_buf_data (C struct), 33	bt_conn_cb (C struct), 23
BT_BUF_EVT_RX_SIZE (C macro), 31	bt_conn_cb.connected (C var), 23
BT_BUF_EVT_SIZE (C macro), 31	bt_conn_cb.disconnected(C var), 23
bt_buf_get_cmd_complete(C function), 32	bt_conn_cb.identity_resolved(C var), 24
= -J (- J	=

```
bt_conn_cb.le_data_len_updated(C var), 25
                                                 bt_conn_le_data_len_param(C struct), 19
bt_conn_cb.le_param_req(Cvar), 23
                                                 bt_conn_le_data_len_param.tx_max_len (C var),
bt_conn_cb.le_param_updated(C var), 24
bt_conn_cb.le_phy_updated(C var), 25
                                                 bt_conn_le_data_len_param.tx_max_time (C var),
bt_conn_cb.remote_info_available (C var), 25
bt_conn_cb.security_changed(C var), 24
                                                 BT_CONN_LE_DATA_LEN_PARAM_INIT (C macro), 3
BT_CONN_CB_DEFINE (C macro), 4
                                                 bt_conn_le_data_len_update (C function), 11
                                                 bt_conn_le_get_tx_power_level (C function), 10
bt_conn_cb_register (C function), 14
bt_conn_create_auto_stop (C function), 12
                                                 bt_conn_le_info (C struct), 19
bt_conn_create_br (C function), 17
                                                 bt_conn_le_info.dst(C var), 19
bt_conn_create_sco (C function), 18
                                                 bt_conn_le_info.latency(C var), 19
bt_conn_disconnect (C function), 11
                                                 bt_conn_le_info.local(Cvar), 19
bt_conn_enc_key_size (C function), 14
                                                 bt_conn_le_info.phy(C var), 20
bt_conn_foreach (C function), 9
                                                 bt_conn_le_info.remote(C var), 19
                                                 bt_conn_le_info.src(C var), 19
bt_conn_get_dst (C function), 9
bt_conn_get_dst_br (C function), 9
                                                 bt_conn_le_info.timeout(C var), 20
bt_conn_get_info (C function), 10
                                                 bt_conn_le_param_update (C function), 11
bt_conn_get_remote_info (C function), 10
                                                 bt_conn_le_phy_info (C struct), 18
bt_conn_get_security (C function), 14
                                                 bt_conn_le_phy_info.rx_phy (C var), 18
                                                 BT_CONN_LE_PHY_PARAM (C macro), 3
bt_conn_index (C function), 9
bt_conn_info (C struct), 20
                                                 bt_conn_le_phy_param (C struct), 18
bt_conn_info.__unnamed39__(Cunion), 29
                                                 bt_conn_le_phy_param.pref_rx_phy(Cvar), 18
bt_conn_info.__unnamed39__.br(Cvar), 30
                                                 bt_conn_le_phy_param.pref_tx_phy(Cvar), 18
bt_conn_info.__unnamed39__.le(Cvar), 30
                                                 BT_CONN_LE_PHY_PARAM_1M (C macro), 3
bt_conn_info.id(Cvar), 20
                                                 BT_CONN_LE_PHY_PARAM_2M (C macro), 3
bt_conn_info.role(C var), 20
                                                 BT_CONN_LE_PHY_PARAM_ALL (C macro), 3
bt_conn_info.security(C var), 21
                                                 BT_CONN_LE_PHY_PARAM_CODED (C macro), 3
bt_conn_info.state(C var), 20
                                                 BT_CONN_LE_PHY_PARAM_INIT (C macro), 3
bt_conn_info.type(C var), 20
                                                 bt_conn_le_phy_update (C function), 11
bt_conn_info.[anonymous] (C var), 20
                                                 bt_conn_le_remote_info(C struct), 21
BT_CONN_INTERVAL_TO_MS (C macro), 4
                                                 bt_conn_le_remote_info.features(C var), 21
bt_conn_le_create (C function), 12
                                                 bt_conn_le_tx_power (C struct), 22
bt_conn_le_create_auto (C function), 12
                                                 bt_conn_le_tx_power.current_level (C var), 22
BT_CONN_LE_CREATE_CONN (C macro), 4
                                                 bt_conn_le_tx_power.max_level (C var), 22
BT_CONN_LE_CREATE_CONN_AUTO (C macro), 4
                                                 bt_conn_le_tx_power.phy (C var), 22
BT_CONN_LE_CREATE_PARAM (C macro), 4
                                                 bt_conn_le_tx_power_phy (C enum), 7
bt_conn_le_create_param (C struct), 22
                                                 bt_conn_le_tx_power_phy.BT_CONN_LE_TX_POWER_PHY_1M
bt_conn_le_create_param.interval(C var), 22
                                                          (C enumerator), 7
bt_conn_le_create_param.interval_coded
                                             (C bt_conn_le_tx_power_phy.BT_CONN_LE_TX_POWER_PHY_2M
        var), 22
                                                          (C enumerator), 7
bt_conn_le_create_param.options (C var), 22
                                                 bt_conn_le_tx_power_phy.BT_CONN_LE_TX_POWER_PHY_CODED_S2
bt_conn_le_create_param.timeout(C var), 22
                                                          (C enumerator), 7
bt_conn_le_create_param.window(C var), 22
                                                 bt_conn_le_tx_power_phy.BT_CONN_LE_TX_POWER_PHY_CODED_S8
bt_conn_le_create_param.window_coded (C var),
                                                          (C enumerator), 7
                                                 bt_conn_le_tx_power_phy.BT_CONN_LE_TX_POWER_PHY_NONE
BT_CONN_LE_CREATE_PARAM_INIT (C macro), 4
                                                          (C enumerator), 7
bt_conn_le_data_len_info (C struct), 18
                                                 bt_conn_lookup_addr_le (C function), 9
bt_conn_le_data_len_info.rx_max_len(C var), 19
                                                 bt_conn_oob_info (C struct), 25
bt_conn_le_data_len_info.rx_max_time (C var),
                                                 bt_conn_oob_info.__unnamed44__(Cunion), 30
                                                 bt_conn_oob_info.__unnamed44__.lesc (C struct),
bt_conn_le_data_len_info.tx_max_len(Cvar), 19
bt_conn_le_data_len_info.tx_max_time (C var),
                                                 bt_conn_oob_info.__unnamed44__.lesc(Cvar), 30
                                                 bt_conn_oob_info.__unnamed44__.lesc.oob_config
BT_CONN_LE_DATA_LEN_PARAM (C macro), 3
                                                          (C var), 30
```

```
bt_conn_oob_info.type(C var), 26
                                                  BT_DATA_MESH_MESSAGE (C macro), 76
bt_conn_oob_info.[anonymous] (C enum), 25
                                                 BT_DATA_MESH_PROV (C macro), 76
bt_conn_oob_info.[anonymous].BT_CONN_OOB_LE_LEGAQYATA_NAME_COMPLETE(C macro), 75
        (C enumerator), 25
                                                 BT_DATA_NAME_SHORTENED (C macro), 75
bt_conn_oob_info.[anonymous].BT_CONN_OOB_LE_SCbt_data_parse(C function), 56
        (C enumerator), 25
                                                 BT_DATA_SM_OOB_FLAGS (C macro), 75
bt_conn_pairing_feat (C struct), 26
                                                 BT_DATA_SM_TK_VALUE (C macro), 75
bt_conn_pairing_feat.auth_req(Cvar), 26
                                                 BT_DATA_SOLICIT128 (C macro), 75
bt_conn_pairing_feat.init_key_dist(C var), 26
                                                 BT_DATA_SOLICIT16 (C macro), 75
bt_conn_pairing_feat.io_capability(C var), 26
                                                 BT_DATA_SOLICIT32 (C macro), 76
bt_conn_pairing_feat.max_enc_key_size (C var),
                                                 BT_DATA_SVC_DATA128 (C macro), 76
                                                 BT_DATA_SVC_DATA16 (C macro), 75
bt_conn_pairing_feat.oob_data_flag(C var), 26
                                                 BT_DATA_SVC_DATA32 (C macro), 76
bt_conn_pairing_feat.resp_key_dist(C var), 26
                                                 BT_DATA_TX_POWER (C macro), 75
bt_conn_ref (C function), 8
                                                 BT_DATA_URI (C macro), 76
bt_conn_remote_info(C struct), 21
                                                 BT_DATA_UUID128_ALL (C macro), 75
bt_conn_remote_info.__unnamed41__(Cunion), 30
                                                 BT_DATA_UUID128_SOME (C macro), 75
bt_conn_remote_info.__unnamed41__.br (C var),
                                                 BT_DATA_UUID16_ALL (C macro), 75
                                                 BT_DATA_UUID16_SOME (C macro), 75
bt_conn_remote_info.__unnamed41__.le (C var),
                                                 BT_DATA_UUID32_ALL (C macro), 75
                                                 BT_DATA_UUID32_SOME (C macro), 75
bt_conn_remote_info.manufacturer(C var), 21
                                                 bt_enable (C function), 43
bt_conn_remote_info.subversion(C var), 21
                                                 bt_foreach_bond (C function), 59
bt_conn_remote_info.type (C var), 21
                                                 BT_GAP_ADV_FAST_INT_MAX_1 (C macro), 77
bt_conn_remote_info.version(C var), 21
                                                 BT_GAP_ADV_FAST_INT_MAX_2 (C macro), 77
BT_CONN_ROLE_MASTER (C macro), 4
                                                 BT_GAP_ADV_FAST_INT_MIN_1 (C macro), 77
{\tt BT\_CONN\_ROLE\_SLAVE}\;(C\;macro),\,4
                                                 BT_GAP_ADV_FAST_INT_MIN_2 (C macro), 77
bt_conn_set_security(C function), 13
                                                 BT_GAP_ADV_HIGH_DUTY_CYCLE_MAX_TIMEOUT
                                                                                               (C
bt_conn_state (C enum), 6
                                                          macro), 78
bt_conn_state.BT_CONN_STATE_CONNECTED (C enu-
                                                 BT_GAP_ADV_MAX_ADV_DATA_LEN (C macro), 77
        merator), 6
                                                 BT_GAP_ADV_MAX_EXT_ADV_DATA_LEN (C macro), 77
bt_conn_state.BT_CONN_STATE_CONNECTING
                                             (C BT_GAP_ADV_SLOW_INT_MAX (C macro), 77
                                                 BT_GAP_ADV_SLOW_INT_MIN (C macro), 77
        enumerator), 6
bt_conn_state.BT_CONN_STATE_DISCONNECTED
                                                 BT_GAP_DATA_LEN_DEFAULT (C macro), 78
        enumerator), 6
                                                 BT_GAP_DATA_LEN_MAX (C macro), 78
bt_conn_state.BT_CONN_STATE_DISCONNECTING (C
                                                 BT_GAP_DATA_TIME_DEFAULT (C macro), 78
        enumerator), 6
                                                 BT_GAP_DATA_TIME_MAX (C macro), 78
bt_conn_unref(C function), 9
                                                 BT_GAP_INIT_CONN_INT_MAX (C macro), 77
                                                 BT_GAP_INIT_CONN_INT_MIN (C macro), 77
BT_DATA (C macro), 34
                                                 BT_GAP_NO_TIMEOUT (C macro), 78
bt_data (C struct), 61
BT_DATA_BIG_INFO (C macro), 76
                                                 BT_GAP_PER_ADV_FAST_INT_MAX_1 (C macro), 77
                                                 BT_GAP_PER_ADV_FAST_INT_MAX_2 (C macro), 77
BT_DATA_BROADCAST_CODE (C macro), 76
BT_DATA_BYTES (C macro), 34
                                                 BT_GAP_PER_ADV_FAST_INT_MIN_1 (C macro), 77
BT_DATA_CHANNEL_MAP_UPDATE_IND (C macro), 76
                                                 BT_GAP_PER_ADV_FAST_INT_MIN_2 (C macro), 77
BT_DATA_CSIS_RSI (C macro), 76
                                                 BT_GAP_PER_ADV_INTERVAL_TO_MS (C macro), 78
                                                 BT_GAP_PER_ADV_MAX_INTERVAL (C macro), 78
BT_DATA_FLAGS (C macro), 75
BT_DATA_GAP_APPEARANCE (C macro), 75
                                                 BT_GAP_PER_ADV_MAX_SKIP (C macro), 78
BT_DATA_LE_BT_DEVICE_ADDRESS (C macro), 76
                                                 BT_GAP_PER_ADV_MAX_TIMEOUT (C macro), 78
BT_DATA_LE_ROLE (C macro), 76
                                                 BT_GAP_PER_ADV_MIN_INTERVAL (C macro), 78
BT_DATA_LE_SC_CONFIRM_VALUE (C macro), 76
                                                 BT_GAP_PER_ADV_MIN_TIMEOUT (C macro), 78
BT_DATA_LE_SC_RANDOM_VALUE (C macro), 76
                                                 BT_GAP_PER_ADV_SLOW_INT_MAX (C macro), 77
BT_DATA_LE_SUPPORTED_FEATURES (C macro), 76
                                                 BT_GAP_PER_ADV_SLOW_INT_MIN (C macro), 77
BT_DATA_MANUFACTURER_DATA (C macro), 76
                                                 BT_GAP_RSSI_INVALID (C macro), 78
BT_DATA_MESH_BEACON (C macro), 76
                                                 BT_GAP_SCAN_FAST_INTERVAL (C macro), 76
```

DE CAR CON TACE VENDOU (C) 35	1 1
BT_GAP_SCAN_FAST_WINDOW (C macro), 77	bt_gatt_chrc.uuid(C var), 90
BT_GAP_SCAN_SLOW_INTERVAL_1 (C macro), 77	bt_gatt_chrc.value_handle(C var), 90
BT_GAP_SCAN_SLOW_INTERVAL_2 (C macro), 77	BT_GATT_CHRC_AUTH (C macro), 85
BT_GAP_SCAN_SLOW_WINDOW_1 (C macro), 77	BT_GATT_CHRC_BROADCAST (C macro), 84
BT_GAP_SCAN_SLOW_WINDOW_2 (C macro), 77	BT_GATT_CHRC_EXT_PROP (C macro), 85
BT_GAP_SID_INVALID (C macro), 78	BT_GATT_CHRC_INDICATE (C macro), 85
BT_GAP_SID_MAX (C macro), 78	BT_GATT_CHRC_INIT (C macro), 93
BT_GAP_TX_POWER_INVALID (C macro), 78	BT_GATT_CHRC_NOTIFY (C macro), 85
bt_gatt_attr (<i>C struct</i>), 88	BT_GATT_CHRC_READ (C macro), 84
bt_gatt_attr.handle(<i>C var</i>), 89	BT_GATT_CHRC_WRITE (C macro), 85
<pre>bt_gatt_attr.perm(C var), 89</pre>	BT_GATT_CHRC_WRITE_WITHOUT_RESP (C macro), 85
bt_gatt_attr.user_data(C var),88	<pre>bt_gatt_complete_func_t (C type), 95</pre>
bt_gatt_attr.uuid(C var),88	BT_GATT_CPF (C macro), 94
<pre>bt_gatt_attr.write(C var), 88</pre>	bt_gatt_cpf(C struct), 91
bt_gatt_attr_func_t (C type), 95	<pre>bt_gatt_cpf.description(C var), 91</pre>
bt_gatt_attr_get_handle (C function), 97	bt_gatt_cpf.exponent(Cvar),91
bt_gatt_attr_next (C function), 97	bt_gatt_cpf.format(Cvar),91
bt_gatt_attr_read (C function), 98	bt_gatt_cpf.name_space(Cvar),91
bt_gatt_attr_read_ccc (C function), 99	bt_gatt_cpf.unit(C var),91
bt_gatt_attr_read_cep (C function), 100	BT_GATT_CUD (C macro), 94
bt_gatt_attr_read_chrc(C function), 99	BT_GATT_DESCRIPTOR (C macro), 94
bt_gatt_attr_read_cpf (C function), 101	bt_gatt_discover (C function), 112
bt_gatt_attr_read_cud (C function), 100	bt_gatt_discover_func_t (C type), 107
bt_gatt_attr_read_func_t (C type), 86	bt_gatt_discover_params (C struct), 116
bt_gatt_attr_read_included (<i>C function</i>), 99	bt_gatt_discover_paramsunnamed61 (C
bt_gatt_attr_read_service (C function), 98	union), 118
bt_gatt_attr_value_handle (C function), 98 bt_gatt_attr_value_handle (C function), 97	bt_gatt_discover_paramsunnamed61included
bt_gatt_attr_write_ccc (C function), 100	(C struct), 118
-	bt_gatt_discover_paramsunnamed61included
bt_gatt_attr_write_func_t (C type), 86	
BT_GATT_ATTRIBUTE (C macro), 94	(C var), 118
bt_gatt_cancel (C function), 116	bt_gatt_discover_paramsunnamed61included.attr_handl
bt_gatt_cb (C struct), 90	(C var), 119
bt_gatt_cb.att_mtu_updated(Cvar),90	bt_gatt_discover_paramsunnamed61included.end_handle
bt_gatt_cb_register(C function), 96	(C var), 119
BT_GATT_CCC (C macro), 93	bt_gatt_discover_paramsunnamed61included.start_hand
bt_gatt_ccc (C struct), 91	(C var), 119
bt_gatt_ccc.flags(Cvar),91	bt_gatt_discover_paramsunnamed61start_handle
bt_gatt_ccc_cfg (C struct), 104	(C var), 118
bt_gatt_ccc_cfg.id(C var), 105	bt_gatt_discover_params.end_handle(Cvar),116
<pre>bt_gatt_ccc_cfg.peer (C var), 105</pre>	<pre>bt_gatt_discover_params.func(C var), 116</pre>
bt_gatt_ccc_cfg.value(C var), 105	<pre>bt_gatt_discover_params.type (C var), 117</pre>
BT_GATT_CCC_INDICATE (C macro), 85	<pre>bt_gatt_discover_params.uuid(C var), 116</pre>
BT_GATT_CCC_INITIALIZER (C macro), 93	BT_GATT_ERR (C macro), 84
BT_GATT_CCC_MANAGED (C macro), 93	<pre>bt_gatt_exchange_mtu (C function), 111</pre>
BT_GATT_CCC_MAX (C macro), 93	bt_gatt_exchange_params (C struct), 116
BT_GATT_CCC_NOTIFY (C macro), 85	<pre>bt_gatt_exchange_params.func(C var), 116</pre>
BT_GATT_CEP (C macro), 94	<pre>bt_gatt_find_by_uuid (C function), 97</pre>
bt_gatt_cep (C struct), 90	bt_gatt_foreach_attr(C function), 97
bt_gatt_cep.properties (C var), 91	bt_gatt_foreach_attr_type (C function), 96
BT_GATT_CEP_RELIABLE_WRITE (C macro), 85	bt_gatt_get_mtu (<i>C function</i>), 104
BT_GATT_CEP_WRITABLE_AUX (C macro), 85	bt_gatt_include (C struct), 89
BT_GATT_CHARACTERISTIC (C macro), 93	bt_gatt_include.end_handle (C var), 90
bt_gatt_chrc (C struct), 90	bt_gatt_include.start_handle(C var), 90
bt_gatt_chrc.properties (C var), 90	bt_gatt_include.uuid (C var), 90
se_gate_cire.properties (& var), 30	be_yace_incrude. auta (o var), 10

```
BT_GATT_INCLUDE_SERVICE (C macro), 93
                                                 bt_gatt_read_params.__unnamed64__.by_uuid (C
                                                         var), 119
bt_gatt_indicate (C function), 103
bt_gatt_indicate_func_t (C type), 95
                                                 bt_gatt_read_params.__unnamed64__.by_uuid.end_handle
bt_gatt_indicate_params (C struct), 106
                                                         (C var), 120
                                                 bt_gatt_read_params.__unnamed64__.by_uuid.start_handle
bt_gatt_indicate_params._ref(C var), 107
bt_gatt_indicate_params.attr(Cvar), 106
                                                         (C var), 120
bt_gatt_indicate_params.data(C var), 107
                                                 bt_gatt_read_params.__unnamed64__.by_uuid.uuid
bt_gatt_indicate_params.destroy(C var), 107
                                                         (C \ var), 120
bt_gatt_indicate_params.func(Cvar), 106
                                                 bt_gatt_read_params.__unnamed64__.multiple
bt_gatt_indicate_params.len(C var), 107
                                                         (C struct), 119
bt_gatt_indicate_params.uuid(Cvar), 106
                                                 bt_gatt_read_params.__unnamed64__.multiple
bt_gatt_indicate_params_destroy_t (C type), 95
                                                         (C var), 119
bt_gatt_is_subscribed (C function), 104
                                                 bt_gatt_read_params.__unnamed64__.multiple.handles
bt_gatt_notify (C function), 102
                                                         (C var), 119
bt_gatt_notify_cb (C function), 101
                                                 bt_gatt_read_params.__unnamed64__.multiple.variable
bt_gatt_notify_func_t (C type), 109
                                                         (C \ var), 119
bt_gatt_notify_multiple (C function), 102
                                                 bt_gatt_read_params.__unnamed64__.single (C
bt_gatt_notify_params (C struct), 106
                                                         struct), 119
bt_gatt_notify_params.attr(C var), 106
                                                 bt_gatt_read_params.__unnamed64__.single (C
bt_gatt_notify_params.data(C var), 106
                                                         var), 119
bt_gatt_notify_params.func(Cvar), 106
                                                 bt_gatt_read_params.__unnamed64__.single.handle
bt_gatt_notify_params.len(Cvar), 106
                                                         (C var), 119
bt_gatt_notify_params.user_data(C var), 106
                                                 bt_gatt_read_params.__unnamed64__.single.offset
bt_gatt_notify_params.uuid(Cvar), 106
                                                         (C var), 119
bt_gatt_notify_uuid(C function), 103
                                                 bt_gatt_read_params.func(Cvar), 117
bt_gatt_perm (C enum), 87
                                                 bt_gatt_read_params.handle_count (C var), 117
bt_gatt_perm.BT_GATT_PERM_NONE (C enumerator),
                                                 bt_gatt_resubscribe (C function), 115
                                                 bt_gatt_scc (C struct), 91
bt_gatt_perm.BT_GATT_PERM_PREPARE_WRITE
                                                bt_gatt_scc.flags(C var), 91
                                             (C
                                                 BT_GATT_SCC_BROADCAST (C macro), 85
        enumerator), 87
                                                 BT_GATT_SECONDARY_SERVICE (C macro), 92
bt_gatt_perm.BT_GATT_PERM_READ (C enumerator),
                                                 BT_GATT_SERVICE (C macro), 92
                                                 bt_gatt_service (C struct), 89
bt_gatt_perm.BT_GATT_PERM_READ_AUTHEN (C enu-
                                                 bt_gatt_service.attr_count(Cvar),89
        merator), 87
bt_gatt_perm.BT_GATT_PERM_READ_ENCRYPT
                                             (C bt_gatt_service.attrs(C var), 89
        enumerator), 87
                                                 BT_GATT_SERVICE_DEFINE (C macro), 92
bt_gatt_perm.BT_GATT_PERM_READ_LESC (C enu-
                                                 BT_GATT_SERVICE_INSTANCE_DEFINE (C macro), 92
        merator), 87
                                                 bt_gatt_service_is_registered (C function), 96
bt_gatt_perm.BT_GATT_PERM_WRITE(C enumerator),
                                                 bt_gatt_service_register (C function), 96
                                                 bt_gatt_service_static (C struct), 89
bt_gatt_perm.BT_GATT_PERM_WRITE_AUTHEN
                                             (C bt_gatt_service_static.attr_count(C var), 89
        enumerator), 87
                                                 bt_gatt_service_static.attrs(C var), 89
bt_gatt_perm.BT_GATT_PERM_WRITE_ENCRYPT
                                             (C bt_gatt_service_unregister (C function), 96
                                                 bt_gatt_service_val(C struct), 89
        enumerator), 87
bt_gatt_perm.BT_GATT_PERM_WRITE_LESC (C enu-
                                                 bt_gatt_service_val.end_handle (C var), 89
                                                 bt_gatt_service_val.uuid(C var), 89
        merator), 88
BT_GATT_PRIMARY_SERVICE (C macro), 92
                                                 bt_gatt_subscribe (C function), 114
bt_gatt_read (C function), 112
                                                 bt_gatt_subscribe_func_t (C type), 109
                                                 bt_gatt_subscribe_params (C struct), 117
bt_gatt_read_func_t (C type), 108
bt_gatt_read_params (C struct), 117
                                                 bt_gatt_subscribe_params.ccc_handle (C var),
bt_gatt_read_params.__unnamed64__ (C union),
                                                 bt_gatt_subscribe_params.min_security (C var),
bt_gatt_read_params.__unnamed64__.by_uuid (C
                                                 bt_gatt_subscribe_params.notify(Cvar),118
        struct), 119
```

```
bt_gatt_subscribe_params.subscribe(C var), 118 bt_hfp_ag_send_callring(C function), 127
bt_gatt_subscribe_params.value(C var), 118
                                                 bt_hfp_ag_send_callsetup_indicator (C func-
bt_gatt_subscribe_params.value_handle (C var),
                                                 bt_hfp_ag_send_ccwa_indicator(C function), 129
        118
bt_gatt_subscribe_params.write(Cvar), 118
                                                 bt_hfp_ag_send_disable_voice_ecnr (C function),
bt_gatt_unsubscribe (C function), 115
bt_gatt_write(C function), 113
                                                 bt_hfp_ag_send_disable_voice_recognition (C
bt_gatt_write_func_t (C type), 108
                                                          function), 125
bt_gatt_write_params (C struct), 117
                                                 bt_hfp_ag_send_enable_voice_ecnr (C function),
bt_gatt_write_params.data(Cvar), 117
                                                          125
bt_gatt_write_params.func(Cvar), 117
                                                 bt_hfp_ag_send_enable_voice_recognition
                                                                                               (C
bt_gatt_write_params.handle(C var), 117
                                                          function), 125
bt_gatt_write_params.length(Cvar), 117
                                                 bt_hfp_ag_send_roaming_indicator (C function),
bt_gatt_write_params.offset(C var), 117
                                                          128
bt_gatt_write_without_response (C function), 114
                                                 bt_hfp_ag_send_service_indicator (C function),
bt_gatt_write_without_response_cb (C function),
        113
                                                 bt_hfp_ag_send_signal_indicator (C function),
bt_get_appearance (C function), 44
                                                          128
bt_get_name (C function), 44
                                                 bt_hfp_ag_set_cops (C function), 126
bt_hfp_ag_call_status_pl (C function), 127
                                                 bt_hfp_ag_set_inband_ring_tone (C function), 126
bt_hfp_ag_cb (C struct), 132
                                                 bt_hfp_ag_set_phnum_tag (C function), 126
bt_hfp_ag_cb.ata_response (C var), 133
                                                 bt_hfp_ag_set_volume_control (C function), 126
                                                 bt_hfp_ag_unknown_at_response (C function), 129
bt_hfp_ag_cb.brva (C var), 133
bt_hfp_ag_cb.chld(Cvar), 134
                                                 bt_hfp_hf_at_cmd (C enum), 122
bt_hfp_ag_cb.chup_response(C var), 133
                                                 bt_hfp_hf_at_cmd.BT_HFP_HF_AT_CHUP (C enumer-
bt_hfp_ag_cb.codec_negotiate (C var), 133
                                                          ator), 122
bt_hfp_ag_cb.connected(Cvar), 132
                                                 \verb|bt_hfp_hf_at_cmd.BT_HFP_HF_ATA| (C enumerator),
bt_hfp_ag_cb.dial(Cvar), 133
                                                          122
                                                 bt_hfp_hf_cb (C struct), 134
bt_hfp_ag_cb.disconnected(C var), 132
                                                 bt_hfp_hf_cb.battery (C var), 136
bt_hfp_ag_cb.hfu_brsf (C var), 132
bt_hfp_ag_cb.nrec (C var), 133
                                                 bt_hfp_hf_cb.call (C var), 135
bt_hfp_ag_cb.unkown_at (C var), 134
                                                 bt_hfp_hf_cb.call_held(Cvar), 135
bt_hfp_ag_cb.volume_control(C var), 132
                                                 bt_hfp_hf_cb.call_phnum(C var), 136
bt_hfp_ag_close_audio (C function), 124
                                                 bt_hfp_hf_cb.call_setup (C var), 135
bt_hfp_aq_codec_selector (C function), 129
                                                 bt_hfp_hf_cb.cmd_complete_cb(Cvar), 136
bt_hfp_ag_connect (C function), 123
                                                 bt_hfp_hf_cb.connected (C var), 134
bt_hfp_ag_deinit (C function), 123
                                                 bt_hfp_hf_cb.disconnected (C var), 134
bt_hfp_ag_disconnect (C function), 123
                                                 bt_hfp_hf_cb.ring_indication (C var), 136
bt_hfp_ag_discover (C function), 124
                                                 bt_hfp_hf_cb.roam(C var), 135
bt_hfp_ag_discover_callback (C type), 121
                                                 bt_hfp_hf_cb.service(Cvar), 135
bt_hfp_ag_get_peer_supp_features (C function),
                                                 bt_hfp_hf_cb.signal (C var), 135
        124
                                                 bt_hfp_hf_cb.voicetag_phnum (C var), 136
bt_hfp_ag_handle_btrh (C function), 127
                                                 bt_hfp_hf_cb.waiting_call (C var), 136
bt_hfp_ag_handle_indicator_enable (C function),
                                                 bt_hfp_hf_cmd_complete(C struct), 134
                                                 bt_hfp_hf_dial (C function), 130
bt_hfp_ag_init (C function), 123
                                                 bt_hfp_hf_dial_memory (C function), 130
                                                 bt_hfp_hf_disable_call_waiting_notification
bt_hfp_ag_open_audio (C function), 124
bt_hfp_ag_register_cind_features (C function),
                                                          (C function), 131
                                                 bt_hfp_hf_disable_clip_notification (C func-
bt_hfp_ag_register_supp_features (C function),
                                                          tion), 131
                                                 bt_hfp_hf_enable_call_waiting_notification
                                                          (C function), 131
bt_hfp_ag_send_battery_indicator (C function),
                                                 bt_hfp_hf_enable_clip_notification (C func-
bt_hfp_ag_send_call_indicator (C function), 127
                                                          tion), 131
```

```
bt_hfp_hf_get_last_voice_tag_number (C func- bt_l2cap_chan_ops.recv(C var), 147
        tion), 132
                                                  bt_12cap_chan_ops.sent (C var), 147
bt_hfp_hf_last_dial (C function), 131
                                                  bt_12cap_chan_ops.status (C var), 147
bt_hfp_hf_multiparty_call_option (C function),
                                                  bt_l2cap_chan_recv_complete (C function), 143
                                                  bt_12cap_chan_send (C function), 142
bt_hfp_hf_register (C function), 129
                                                  BT_L2CAP_CHAN_SEND_RESERVE (C macro), 139
bt_hfp_hf_send_cmd (C function), 129
                                                  bt_12cap_chan_state (C enum), 140
                                                  bt_l2cap_chan_state.BT_L2CAP_CONFIG (C enu-
bt_hfp_hf_start_voice_recognition (C function),
                                                          merator), 140
bt_hfp_hf_stop_voice_recognition (C function),
                                                  bt_12cap_chan_state.BT_L2CAP_CONNECTED
                                                                                               (C
                                                          enumerator), 140
bt_hfp_hf_volume_update (C function), 130
                                                  bt_12cap_chan_state.BT_L2CAP_CONNECTING
                                                                                               (C
bt_hps_init (C function), 217
                                                          enumerator), 140
                                                  bt_12cap_chan_state.BT_L2CAP_DISCONNECTED (C
bt_hps_notify (C function), 217
bt_hps_set_status_code (C function), 217
                                                          enumerator), 140
bt_hts_indicate (C function), 218
                                                  bt_12cap_chan_state.BT_L2CAP_DISCONNECTING
bt_id_create (C function), 45
                                                          (C enumerator), 140
                                                  bt_12cap_chan_state_t (C type), 140
BT_ID_DEFAULT (C macro), 34
bt_id_delete (C function), 46
                                                  bt_12cap_chan_status (C enum), 140
bt_id_get (C function), 44
                                                  bt_12cap_chan_status.BT_L2CAP_NUM_STATUS (C
bt_id_reset (C function), 45
                                                          enumerator), 141
bt_is_ready (C function), 44
                                                  bt_12cap_chan_status.BT_L2CAP_STATUS_ENCRYPT_PENDING
bt_12cap_br_chan (C struct), 145
                                                          (C enumerator), 141
bt_12cap_br_chan.chan(C var), 145
                                                  bt_l2cap_chan_status.BT_L2CAP_STATUS_OUT (C
bt_12cap_br_chan.ident (C var), 145
                                                          enumerator), 140
bt_12cap_br_chan.psm(Cvar), 145
                                                  bt_12cap_chan_status.BT_L2CAP_STATUS_SHUTDOWN
bt_12cap_br_chan.rx(Cvar), 145
                                                          (C enumerator), 140
bt_12cap_br_chan.tx(Cvar), 145
                                                  bt_12cap_chan_status_t (C type), 140
bt_12cap_br_endpoint (C struct), 145
                                                  bt_12cap_ecred_chan_connect (C function), 141
bt_l2cap_br_endpoint.cid(Cvar), 145
                                                  bt_12cap_ecred_chan_reconfigure (C function),
bt_12cap_br_endpoint.mtu(C var), 145
bt_12cap_br_server_register (C function), 141
                                                  bt_12cap_ext_flow_spec (C struct), 145
BT_L2CAP_BUF_SIZE (C macro), 137
                                                  BT_L2CAP_FEATURE_EFS_BR_EDR (C macro), 139
BT_L2CAP_CFG_OPT_EXT_FLOW_SPEC (C macro), 139
                                                  BT_L2CAP_FEATURE_ERTM (C macro), 139
BT_L2CAP_CFG_OPT_EXT_WIN_SIZE (C macro), 139
                                                                                                (C
                                                  BT_L2CAP_FEATURE_EXTENDED_WINDOW_SIZE
BT_L2CAP_CFG_OPT_FCS (C macro), 139
                                                          macro), 139
BT_L2CAP_CFG_OPT_FUSH_TIMEOUT (C macro), 138
                                                  BT_L2CAP_FEATURE_FC (C macro), 139
BT_L2CAP_CFG_OPT_MTU (C macro), 138
                                                  BT_L2CAP_FEATURE_FCS (C macro), 139
BT_L2CAP_CFG_OPT_QOS (C macro), 138
                                                  BT_L2CAP_FEATURE_FIXED_CHANNELS (C macro), 139
BT_L2CAP_CFG_OPT_RETRANS_FC (C macro), 138
                                                  BT_L2CAP_FEATURE_QOS (C macro), 139
bt_12cap_cfg_options (C struct), 145
                                                  BT_L2CAP_FEATURE_RTM (C macro), 139
                                                  BT_L2CAP_FEATURE_SM (C macro), 139
bt_12cap_chan (C struct), 143
bt_12cap_chan.conn (C var), 143
                                                  BT_L2CAP_FEATURE_UCD (C macro), 139
bt_12cap_chan.ops (C var), 143
                                                  BT_L2CAP_HDR_SIZE (C macro), 137
bt_12cap_chan_connect (C function), 142
                                                  BT_L2CAP_LE_CHAN (C macro), 138
                                                  bt_l2cap_le_chan (C struct), 144
bt_12cap_chan_destroy_t (C type), 140
bt_12cap_chan_disconnect (C function), 142
                                                  bt_12cap_le_chan._sdu (C var), 144
bt_12cap_chan_ops (C struct), 146
                                                  bt_l2cap_le_chan.chan(C var), 144
                                                  bt_l2cap_le_chan.pending_rx_mtu(Cvar), 144
bt_12cap_chan_ops.alloc_buf (C var), 146
bt_12cap_chan_ops.alloc_seg(Cvar), 146
                                                  bt_12cap_1e_chan.rx(Cvar), 144
bt_12cap_chan_ops.connected (C var), 146
                                                  bt_12cap_1e_chan.tx(Cvar), 144
bt_12cap_chan_ops.disconnected (C var), 146
                                                  bt_12cap_le_chan.tx_buf(Cvar), 144
bt_12cap_chan_ops.encrypt_change (C var), 146
                                                  bt_12cap_le_chan.tx_queue (C var), 144
bt_12cap_chan_ops.reconfigured (C var), 147
                                                  bt_12cap_le_chan.tx_work(Cvar), 144
```

bt_l2cap_le_endpoint (<i>C struct</i>), 143	BT_LE_DATA_LEN_PARAM_MAX (C macro), 3
<pre>bt_12cap_le_endpoint.cid(C var), 144</pre>	bt_le_ext_adv_cb (C struct), 60
bt_12cap_le_endpoint.credits(C var), 144	bt_le_ext_adv_cb.connected(C var), 60
<pre>bt_l2cap_le_endpoint.init_credits (C var), 144</pre>	bt_le_ext_adv_cb.scanned(C var), 60
bt_12cap_le_endpoint.mps(C var), 144	bt_le_ext_adv_cb.sent (C var), 60
bt_12cap_le_endpoint.mtu(C var), 144	BT_LE_EXT_ADV_CODED_NCONN (C macro), 35
BT_L2CAP_MODE_BASIC (C macro), 139	BT_LE_EXT_ADV_CODED_NCONN_IDENTITY (C macro),
BT_L2CAP_MODE_ERTM (C macro), 139	35
BT_L2CAP_MODE_FC (C macro), 139	BT_LE_EXT_ADV_CODED_NCONN_NAME (C macro), 35
BT_L2CAP_MODE_RTM (C macro), 139	BT_LE_EXT_ADV_CONN_NAME (C macro), 35
BT_L2CAP_MODE_SM (C macro), 139	<pre>bt_le_ext_adv_connected_info (C struct), 60</pre>
bt_12cap_qos (C struct), 145	<pre>bt_le_ext_adv_connected_info.conn (C var), 60</pre>
bt_12cap_retrans_fc(C struct), 145	<pre>bt_le_ext_adv_create (C function), 47</pre>
BT_L2CAP_RX_MTU (C macro), 137	<pre>bt_le_ext_adv_delete (C function), 49</pre>
BT_L2CAP_SDU_BUF_SIZE (C macro), 138	<pre>bt_le_ext_adv_get_index (C function), 49</pre>
BT_L2CAP_SDU_CHAN_SEND_RESERVE (C macro), 139	<pre>bt_le_ext_adv_get_info (C function), 49</pre>
BT_L2CAP_SDU_HDR_SIZE (C macro), 138	bt_le_ext_adv_info(C struct), 63
BT_L2CAP_SDU_RX_MTU (C macro), 138	<pre>bt_le_ext_adv_info.tx_power (C var), 63</pre>
BT_L2CAP_SDU_TX_MTU (C macro), 138	BT_LE_EXT_ADV_NCONN (C macro), 35
bt_12cap_server (<i>C struct</i>), 147	BT_LE_EXT_ADV_NCONN_IDENTITY (C macro), 35
bt_12cap_server.accept (C var), 148	BT_LE_EXT_ADV_NCONN_NAME (C macro), 35
bt_12cap_server.psm (<i>C var</i>), 148	bt_le_ext_adv_oob_get_local (C function), 57
bt_12cap_server.sec_level (C var), 148	BT_LE_EXT_ADV_SCAN_NAME (C macro), 35
bt_12cap_server_register (<i>C function</i>), 141	bt_le_ext_adv_scanned_info (C struct), 60
BT_L2CAP_TX_MTU (C macro), 137	bt_le_ext_adv_scanned_info.addr (<i>C var</i>), 60
BT_LE_AD_GENERAL (C macro), 76	bt_le_ext_adv_sent_info (C struct), 59
BT_LE_AD_LIMITED (C macro), 76	bt_le_ext_adv_sent_info.num_sent(C var), 60
BT_LE_AD_NO_BREDR (C macro), 76	bt_le_ext_adv_set_data (C function), 48
BT_LE_ADV_CONN (C macro), 35	bt_le_ext_adv_start (<i>C function</i>), 47
BT_LE_ADV_CONN_DIR (C macro), 35	BT_LE_EXT_ADV_START_DEFAULT (C macro), 36
BT_LE_ADV_CONN_DIR_LOW_DUTY (C macro), 35	BT_LE_EXT_ADV_START_PARAM (C macro), 36
BT_LE_ADV_CONN_NAME (C macro), 35	<pre>bt_le_ext_adv_start_param(C struct), 62</pre>
BT_LE_ADV_CONN_NAME_AD (C macro), 35	<pre>bt_le_ext_adv_start_param.num_events (C var),</pre>
BT_LE_ADV_NCONN (C macro), 35	63
BT_LE_ADV_NCONN_IDENTITY (C macro), 35	<pre>bt_le_ext_adv_start_param.timeout(C var), 63</pre>
BT_LE_ADV_NCONN_NAME (C macro), 35	BT_LE_EXT_ADV_START_PARAM_INIT (C macro), 36
BT_LE_ADV_PARAM (C macro), 34	<pre>bt_le_ext_adv_stop (C function), 48</pre>
bt_le_adv_param(C struct), 61	<pre>bt_le_ext_adv_update_param (C function), 49</pre>
bt_le_adv_param.id(C var),61	<pre>bt_le_filter_accept_list_add (C function), 55</pre>
bt_le_adv_param.interval_max(C var),62	<pre>bt_le_filter_accept_list_clear (C function), 55</pre>
bt_le_adv_param.interval_min(C var), 61	<pre>bt_le_filter_accept_list_remove (C function), 55</pre>
bt_le_adv_param.options (C var), 61	bt_le_oob (C struct), 69
bt_le_adv_param.peer(C var), 62	bt_le_oob.addr (<i>C var</i>), 70
bt_le_adv_param.secondary_max_skip(C var),61	bt_le_oob.le_sc_data (C var), 70
bt_le_adv_param.sid(C var), 61	bt_le_oob_get_local (C function), 56
BT_LE_ADV_PARAM_INIT (C macro), 34	bt_le_oob_get_sc_data (<i>C function</i>), 15
bt_le_adv_start (C function), 46	bt_le_oob_sc_data (<i>C struct</i>), 69
bt_le_adv_stop (C function), 47	bt_le_oob_sc_data.c (C var), 69
bt_le_adv_update_data (C function), 47	bt_le_oob_sc_data.r (<i>C var</i>), 69 bt_le_oob_sc_data.r (<i>C var</i>), 69
BT_LE_CONN_PARAM (C macro), 2	bt_le_oob_set_legacy_tk (C function), 14
bt_le_conn_param (C struct), 18	bt_le_oob_set_sc_data (C function), 14
BT_LE_CONN_PARAM_DEFAULT (C macro), 3	BT_LE_PER_ADV_DEFAULT (C macro), 36
BT_LE_CONN_PARAM_INIT (C macro), 2	bt_le_per_adv_list_add (C function), 53
BT_LE_DATA_LEN_PARAM_DEFAULT (C macro), 3	bt_le_per_adv_list_clear(C function), 54

```
bt_le_per_adv_list_remove (C function), 53
                                                 bt_le_per_adv_sync_synced_info.phy (C var), 63
BT_LE_PER_ADV_PARAM (C macro), 36
                                                 bt_le_per_adv_sync_synced_info.recv_enabled
                                                         (C var), 63
bt_le_per_adv_param (C struct), 62
bt_le_per_adv_param.interval_max(C var), 62
                                                 bt_le_per_adv_sync_synced_info.service_data
bt_le_per_adv_param.interval_min(C var), 62
                                                         (C var), 63
bt_le_per_adv_param.options (C var), 62
                                                 bt_le_per_adv_sync_synced_info.sid(C var), 63
BT_LE_PER_ADV_PARAM_INIT (C macro), 36
                                                 bt_le_per_adv_sync_term_info (C struct), 64
                                                 bt_le_per_adv_sync_term_info.addr(Cvar),64
bt_le_per_adv_set_data (C function), 50
bt_le_per_adv_set_info_transfer (C function), 52
                                                 bt_le_per_adv_sync_term_info.reason(C var), 64
bt_le_per_adv_set_param(Cfunction), 49
                                                 bt_le_per_adv_sync_term_info.sid(C var), 64
bt_le_per_adv_start (C function), 50
                                                 bt_le_per_adv_sync_transfer (C function), 52
bt_le_per_adv_stop (C function), 50
                                                 bt_le_per_adv_sync_transfer_param (C struct), 67
bt_le_per_adv_sync_cb (C struct), 65
                                                 bt_le_per_adv_sync_transfer_param.options (C
bt_le_per_adv_sync_cb.biginfo (C var), 65
                                                          var), 67
bt_le_per_adv_sync_cb.cte_report_cb(Cvar),66
                                                 bt_le_per_adv_sync_transfer_param.skip
                                                                                               (C
bt_le_per_adv_sync_cb.recv(C var), 65
                                                          var), 67
bt_le_per_adv_sync_cb.state_changed(C var), 65
                                                 bt_le_per_adv_sync_transfer_param.timeout (C
bt_le_per_adv_sync_cb.synced (C var), 65
                                                         var), 67
bt_le_per_adv_sync_cb.term(C var), 65
                                                 bt_le_per_adv_sync_transfer_subscribe (C func-
bt_le_per_adv_sync_cb_register (C function), 52
                                                         tion), 53
bt_le_per_adv_sync_create (C function), 51
                                                 bt_le_per_adv_sync_transfer_unsubscribe
                                                                                              (C
bt_le_per_adv_sync_delete (C function), 51
                                                         function), 53
                                                 BT_LE_SCAN_ACTIVE (C macro), 37
bt_le_per_adv_sync_get_index (C function), 50
bt_le_per_adv_sync_get_info (C function), 51
                                                 bt_le_scan_cb (C struct), 69
                                                 bt_le_scan_cb.recv(C var), 69
bt_le_per_adv_sync_info (C struct), 66
bt_le_per_adv_sync_info.addr (C var), 67
                                                 bt_le_scan_cb.timeout (C var), 69
bt_le_per_adv_sync_info.interval (C var), 67
                                                 bt_le_scan_cb_register(C function), 54
bt_le_per_adv_sync_info.phy (C var), 67
                                                 bt_le_scan_cb_t (C type), 37
bt_le_per_adv_sync_info.sid(C var), 67
                                                 bt_le_scan_cb_unregister (C function), 55
bt_le_per_adv_sync_lookup_addr (C function), 51
                                                 BT_LE_SCAN_CODED_ACTIVE (C macro), 37
bt_le_per_adv_sync_param (C struct), 66
                                                 BT_LE_SCAN_CODED_PASSIVE (C macro), 37
bt_le_per_adv_sync_param.addr(Cvar),66
                                                 BT_LE_SCAN_OPT_FILTER_WHITELIST (C macro), 36
bt_le_per_adv_sync_param.options(C var),66
                                                 BT_LE_SCAN_PARAM (C macro), 36
bt_le_per_adv_sync_param.sid(C var),66
                                                 bt_le_scan_param(C struct), 67
                                                 bt_le_scan_param.interval(C var), 67
bt_le_per_adv_sync_param.skip(C var), 66
bt_le_per_adv_sync_param.timeout (C var), 66
                                                 bt_le_scan_param.interval_coded(C var), 68
bt_le_per_adv_sync_recv_disable (C function), 52
                                                 bt_le_scan_param.options (C var), 67
bt_le_per_adv_sync_recv_enable (C function), 52
                                                 bt_le_scan_param.timeout(C var),68
bt_le_per_adv_sync_recv_info (C struct), 64
                                                 bt_le_scan_param.type(C var), 67
bt_le_per_adv_sync_recv_info.addr (C var), 64
                                                 bt_le_scan_param.window(C var), 67
bt_le_per_adv_sync_recv_info.cte_type (C var),
                                                 bt_le_scan_param.window_coded(C var), 68
                                                 BT_LE_SCAN_PARAM_INIT (C macro), 36
bt_le_per_adv_sync_recv_info.rssi(C var), 64
                                                 BT_LE_SCAN_PASSIVE (C macro), 37
                                                 bt_le_scan_recv_info (C struct), 68
bt_le_per_adv_sync_recv_info.sid(Cvar), 64
                                                 bt_le_scan_recv_info.addr(Cvar),68
bt_le_per_adv_sync_recv_info.tx_power (C var),
        64
                                                 bt_le_scan_recv_info.adv_props (C var), 68
bt_le_per_adv_sync_state_info (C struct), 64
                                                 bt_le_scan_recv_info.adv_type (C var), 68
bt_le_per_adv_sync_state_info.recv_enabled
                                                 bt_le_scan_recv_info.interval (C var), 68
        (C var), 65
                                                 bt_le_scan_recv_info.primary_phy (C var), 69
bt_le_per_adv_sync_synced_info (C struct), 63
                                                 bt_le_scan_recv_info.rssi(C var), 68
bt_le_per_adv_sync_synced_info.addr(Cvar),63
                                                 bt_le_scan_recv_info.secondary_phy(C var), 69
bt_le_per_adv_sync_synced_info.conn (C var), 64
                                                 bt_le_scan_recv_info.sid(C var), 68
bt_le_per_adv_sync_synced_info.interval
                                             (C bt_le_scan_recv_info.tx_power(C var), 68
        var), 63
                                                 bt_le_scan_start (C function), 54
```

bt_le_scan_stop (C function), 54	BT_SDP_ATTR_FAX_CLASS20_SUPPORT (C macro), 157
bt_le_set_auto_conn (C function), 13	BT_SDP_ATTR_FAX_CLASS2_SUPPORT (C macro), 157
<pre>bt_le_set_chan_map (C function), 56</pre>	BT_SDP_ATTR_GOEP_L2CAP_PSM (C macro), 156
<pre>bt_le_set_rpa_timeout (C function), 56</pre>	BT_SDP_ATTR_GROUP_ID (C macro), 156
BT_LE_SUPP_FEAT_16_ENCODE (C macro), 80	BT_SDP_ATTR_HID_BATTERY_POWER (C macro), 159
BT_LE_SUPP_FEAT_24_ENCODE (C macro), 79	BT_SDP_ATTR_HID_BOOT_DEVICE (C macro), 159
BT_LE_SUPP_FEAT_32_ENCODE (C macro), 79	BT_SDP_ATTR_HID_COUNTRY_CODE (C macro), 158
BT_LE_SUPP_FEAT_40_ENCODE (C macro), 78	BT_SDP_ATTR_HID_DESCRIPTOR_LIST (C macro), 159
BT_LE_SUPP_FEAT_8_ENCODE (C macro), 80	BT_SDP_ATTR_HID_DEVICE_RELEASE_NUMBER (C
BT_LE_SUPP_FEAT_VALIDATE (C macro), 80	macro), 158
bt_le_whitelist_add(Cfunction),55	BT_SDP_ATTR_HID_DEVICE_SUBCLASS (C macro), 158
bt_le_whitelist_clear (<i>C function</i>), 56	BT_SDP_ATTR_HID_LANG_ID_BASE_LIST (C macro),
bt_le_whitelist_rem(C function), 55	159
BT_PASSKEY_INVALID (C macro), 4	BT_SDP_ATTR_HID_NORMALLY_CONNECTABLE (C
bt_passkey_set (C function), 15	macro), 159
bt_ready_cb_t (C type), 37	BT_SDP_ATTR_HID_PARSER_VERSION (C macro), 158
bt_rfcomm_create_pdu (C function), 150	BT_SDP_ATTR_HID_PROFILE_VERSION (C macro), 159
bt_rfcomm_dlc(C struct), 151	BT_SDP_ATTR_HID_RECONNECT_INITIATE (C macro),
bt_rfcomm_dlc_connect(C function), 149	158
bt_rfcomm_dlc_disconnect(C function), 150	BT_SDP_ATTR_HID_REMOTE_WAKEUP (C macro), 159
bt_rfcomm_dlc_ops (C struct), 150	BT_SDP_ATTR_HID_SDP_DISABLE (C macro), 159
bt_rfcomm_dlc_ops.connected(C var), 150	BT_SDP_ATTR_HID_SUPERVISION_TIMEOUT (C macro),
bt_rfcomm_dlc_ops.disconnected(C var), 150	159
bt_rfcomm_dlc_ops.recv(C var), 151	BT_SDP_ATTR_HID_VIRTUAL_CABLE (C macro), 158
bt_rfcomm_dlc_ops.sent(C var), 151	BT_SDP_ATTR_HOMEPAGE_URL (C macro), 157
bt_rfcomm_dlc_send(C function), 150	BT_SDP_ATTR_ICON_URL (C macro), 156
bt_rfcomm_role(<i>Cenum</i>), 149	BT_SDP_ATTR_IP4_SUBNET (C macro), 157
bt_rfcomm_role.BT_RFCOMM_ROLE_ACCEPTOR (C	BT_SDP_ATTR_IP6_SUBNET (C macro), 157
enumerator), 149	BT_SDP_ATTR_IP_SUBNET (C macro), 156
bt_rfcomm_role.BT_RFCOMM_ROLE_INITIATOR (C	BT_SDP_ATTR_LANG_BASE_ATTR_ID_LIST (C macro),
enumerator), 149	156
bt_rfcomm_role_t(C type), 148	BT_SDP_ATTR_MAP_SUPPORTED_FEATURES (C macro),
bt_rfcomm_server(C struct), 151	158
bt_rfcomm_server.accept(Cvar),151	BT_SDP_ATTR_MAS_INSTANCE_ID (C macro), 158
bt_rfcomm_server.channel(Cvar),151	BT_SDP_ATTR_MAX_NET_ACCESSRATE (C macro), 157
bt_rfcomm_server_register(C function), 149	BT_SDP_ATTR_MCAP_SUPPORTED_PROCEDURES (C
BT_SDP_ADVANCED_AUDIO_SVCLASS (C macro), 152	<i>macro</i>), 157
BT_SDP_ALT16 (<i>C macro</i>), 160	BT_SDP_ATTR_MPMD_SCENARIOS (C macro), 156
BT_SDP_ALT32 (<i>C macro</i>), 160	BT_SDP_ATTR_MPS_DEPENDENCIES (C macro), 156
BT_SDP_ALT8 (<i>C macro</i>), 160	BT_SDP_ATTR_MPSD_SCENARIOS (C macro), 156
BT_SDP_ALT_UNSPEC (C macro), 160	BT_SDP_ATTR_NET_ACCESS_TYPE (C macro), 157
BT_SDP_APPLE_AGENT_SVCLASS (C macro), 155	BT_SDP_ATTR_NETWORK (C macro), 157
BT_SDP_ARRAY_16 (<i>C macro</i>), 161	BT_SDP_ATTR_NETWORK_ADDRESS (C macro), 157
BT_SDP_ARRAY_32 (<i>C macro</i>), 161	BT_SDP_ATTR_PBAP_SUPPORTED_FEATURES (C macro),
BT_SDP_ARRAY_8 (C macro), 161	158
BT_SDP_ATTR_ADD_PROTO_DESC_LIST (C macro), 156	BT_SDP_ATTR_PRIMARY_RECORD (C macro), 158
BT_SDP_ATTR_AUDIO_FEEDBACK_SUPPORT (C macro),	BT_SDP_ATTR_PRODUCT_ID (C macro), 158
157	BT_SDP_ATTR_PROFILE_DESC_LIST (C macro), 156
BT_SDP_ATTR_BROWSE_GRP_LIST (C macro), 156	BT_SDP_ATTR_PROTO_DESC_LIST (C macro), 156
BT_SDP_ATTR_CLNT_EXEC_URL (C macro), 156	BT_SDP_ATTR_PROVNAME_PRIMARY (C macro), 159
BT_SDP_ATTR_DATA_EXCHANGE_SPEC (C macro), 157	BT_SDP_ATTR_RECORD_HANDLE (C macro), 155
BT_SDP_ATTR_DOC_URL (C macro), 156	BT_SDP_ATTR_RECORD_STATE (C macro), 156
BT_SDP_ATTR_EXTERNAL_NETWORK (C macro), 157	${\tt BT_SDP_ATTR_REMOTE_AUDIO_VOLUME_CONTROL} \qquad (C$
BT_SDP_ATTR_FAX_CLASS1_SUPPORT (C macro), 157	<i>macro</i>), 157

/ 3	
BT_SDP_ATTR_SECURITY_DESC (C macro), 157	bt_sdp_discover (C function), 163
BT_SDP_ATTR_SERVICE_AVAILABILITY (C macro), 156	<pre>bt_sdp_discover_cancel (C function), 164</pre>
BT_SDP_ATTR_SERVICE_ID (C macro), 156	bt_sdp_discover_func_t (<i>C type</i>), 162
BT_SDP_ATTR_SERVICE_VERSION (C macro), 157	bt_sdp_discover_params (C struct), 165
BT_SDP_ATTR_SPECIFICATION_ID (C macro), 158	bt_sdp_discover_params.func(C var), 165
BT_SDP_ATTR_SUPPORTED_CAPABILITIES (C macro),	bt_sdp_discover_params.pool(Cvar), 165
158	bt_sdp_discover_params.uuid (C var), 165
BT_SDP_ATTR_SUPPORTED_DATA_STORES_LIST (C	BT_SDP_FAX_SVCLASS (C macro), 153
·	BT_SDP_GENERIC_ACCESS_SVCLASS (C macro), 155
macro), 157	· · · · · · · · · · · · · · · · · · ·
BT_SDP_ATTR_SUPPORTED_FEATURES (C macro), 158	BT_SDP_GENERIC_ATTRIB_SVCLASS (C macro), 155
${\tt BT_SDP_ATTR_SUPPORTED_FEATURES_LIST}~(C~macro),$	BT_SDP_GENERIC_AUDIO_SVCLASS (C macro), 155
156	BT_SDP_GENERIC_FILETRANS_SVCLASS (C macro), 155
$BT_SDP_ATTR_SUPPORTED_FORMATS_LIST$ (C macro),	$BT_SDP_GENERIC_NETWORKING_SVCLASS$ (C macro),
157	155
BT_SDP_ATTR_SUPPORTED_FUNCTIONS (C macro), 158	BT_SDP_GENERIC_TELEPHONY_SVCLASS (C macro), 155
BT_SDP_ATTR_SUPPORTED_MESSAGE_TYPES (C macro),	bt_sdp_get_addl_proto_param (C function), 164
158	bt_sdp_get_features (<i>C function</i>), 165
BT_SDP_ATTR_SUPPORTED_REPOSITORIES (C macro),	bt_sdp_get_profile_version (<i>C function</i>), 164
158	bt_sdp_get_proto_param (C function), 164
BT_SDP_ATTR_SVCDB_STATE (C macro), 156	BT_SDP_GN_SVCLASS (C macro), 153
BT_SDP_ATTR_SVCDESC_PRIMARY (C macro), 159	BT_SDP_GNSS_SERVER_SVCLASS (C macro), 154
BT_SDP_ATTR_SVCINFO_TTL (C macro), 156	BT_SDP_GNSS_SVCLASS (C macro), 154
BT_SDP_ATTR_SVCLASS_ID_LIST (C macro), 155	BT_SDP_HANDSFREE_AGW_SVCLASS (C macro), 153
BT_SDP_ATTR_SVCNAME_PRIMARY (C macro), 159	BT_SDP_HANDSFREE_SVCLASS (C macro), 153
${\tt BT_SDP_ATTR_TOTAL_IMAGING_DATA_CAPACITY} \qquad (C$	BT_SDP_HCR_PRINT_SVCLASS (C macro), 154
<i>macro</i>), 158	BT_SDP_HCR_SCAN_SVCLASS (C macro), 154
BT_SDP_ATTR_VENDOR_ID (C macro), 158	BT_SDP_HCR_SVCLASS (C macro), 154
BT_SDP_ATTR_VENDOR_ID_SOURCE (C macro), 158	BT_SDP_HDP_SINK_SVCLASS (C macro), 155
BT_SDP_ATTR_VERSION (C macro), 158	BT_SDP_HDP_SOURCE_SVCLASS (C macro), 155
BT_SDP_ATTR_VERSION_NUM_LIST (C macro), 156	BT_SDP_HDP_SVCLASS (C macro), 155
BT_SDP_ATTR_WAP_GATEWAY (C macro), 157	BT_SDP_HEADSET_AGW_SVCLASS (C macro), 153
BT_SDP_ATTR_WAP_STACK_TYPE (C macro), 157	BT_SDP_HEADSET_SVCLASS (C macro), 152
bt_sdp_attribute (C struct), 165	BT_SDP_HID_SVCLASS (C macro), 154
BT_SDP_AUDIO_SINK_SVCLASS (C macro), 152	BT_SDP_IMAGING_ARCHIVE_SVCLASS (C macro), 153
BT_SDP_AUDIO_SOURCE_SVCLASS (C macro), 152	BT_SDP_IMAGING_REFOBJS_SVCLASS (C macro), 153
BT_SDP_AV_REMOTE_CONTROLLER_SVCLASS (C macro),	BT_SDP_IMAGING_RESPONDER_SVCLASS (C macro), 153
153	BT_SDP_IMAGING_SVCLASS (C macro), 153
BT_SDP_AV_REMOTE_SVCLASS (C macro), 152	BT_SDP_INT128 (<i>C macro</i>), 160
BT_SDP_AV_REMOTE_TARGET_SVCLASS (C macro), 152	BT_SDP_INT16 (<i>C macro</i>), 159
BT_SDP_AV_SVCLASS (C macro), 154	BT_SDP_INT32 (C macro), 160
BT_SDP_BASIC_PRINTING_SVCLASS (C macro), 153	BT_SDP_INT64 (C macro), 160
BT_SDP_B00L (<i>C macro</i>), 160	BT_SDP_INT8 (C macro), 159
BT_SDP_BROWSE_GRP_DESC_SVCLASS (C macro), 152	BT_SDP_INTERCOM_SVCLASS (C macro), 153
BT_SDP_CIP_SVCLASS (C macro), 154	BT_SDP_IRMC_SYNC_CMD_SVCLASS (C macro), 152
<pre>bt_sdp_client_result (C struct), 165</pre>	BT_SDP_IRMC_SYNC_SVCLASS (C macro), 152
$BT_SDP_CORDLESS_TELEPHONY_SVCLASS$ (C macro),	BT_SDP_LAN_ACCESS_SVCLASS (C macro), 152
152	BT_SDP_LIST (C macro), 161
<pre>bt_sdp_data_elem (C struct), 165</pre>	BT_SDP_MAP_MCE_SVCLASS (C macro), 154
BT_SDP_DATA_ELEM_LIST (C macro), 161	BT_SDP_MAP_MSE_SVCLASS (C macro), 154
BT_SDP_DATA_NIL (C macro), 159	BT_SDP_MAP_SVCLASS (C macro), 154
BT_SDP_DIALUP_NET_SVCLASS (C macro), 152	BT_SDP_MPS_SC_SVCLASS (C macro), 154
BT_SDP_DIRECT_PRINTING_SVCLASS (C macro), 153	BT_SDP_MPS_SVCLASS (C macro), 154
BT_SDP_DIRECT_PRT_REFOBJS_SVCLASS (C macro),	BT_SDP_NAP_SVCLASS (C macro), 153
153	BT_SDP_NEW_SERVICE (C macro), 161
100	232 _M2m_32M 101 (0 mmcro), 101

```
BT_SDP_OBEX_FILETRANS_SVCLASS (C macro), 152
                                                  BT_SDP_URL_STR32 (C macro), 161
BT_SDP_OBEX_OBJPUSH_SVCLASS (C macro), 152
                                                  BT_SDP_URL_STR8 (C macro), 161
BT_SDP_PANU_SVCLASS (C macro), 153
                                                  BT_SDP_URL_STR_UNSPEC (C macro), 160
BT_SDP_PBAP_PCE_SVCLASS (C macro), 154
                                                  BT_SDP_UUID128 (C macro), 160
BT_SDP_PBAP_PSE_SVCLASS (C macro), 154
                                                  BT_SDP_UUID16 (C macro), 160
BT_SDP_PBAP_SVCLASS (C macro), 154
                                                  BT_SDP_UUID32 (C macro), 160
BT SDP PNP INFO SVCLASS (C macro), 154
                                                  BT SDP UUID UNSPEC (C macro), 160
BT_SDP_PRIMARY_LANG_BASE (C macro), 159
                                                  BT_SDP_VIDEO_CONF_GW_SVCLASS (C macro), 154
BT_SDP_PRINTING_STATUS_SVCLASS (C macro), 153
                                                  BT_SDP_VIDEO_DISTRIBUTION_SVCLASS (C macro),
bt_sdp_proto (C enum), 163
                                                          155
bt_sdp_proto.BT_SDP_PROTO_L2CAP(Cenumerator),
                                                  BT_SDP_VIDEO_SINK_SVCLASS (C macro), 155
                                                  BT_SDP_VIDEO_SOURCE_SVCLASS (C macro), 155
        163
bt_sdp_proto.BT_SDP_PROTO_RFCOMM (C enumera-
                                                  BT_SDP_WAP_CLIENT_SVCLASS (C macro), 153
                                                  BT_SDP_WAP_SVCLASS (C macro), 153
        tor), 163
BT_SDP_PUBLIC_BROWSE_GROUP (C macro), 152
                                                  bt_security_err(Cenum), 8
BT_SDP_RECORD (C macro), 162
                                                  bt_security_err.BT_SECURITY_ERR_AUTH_FAIL (C
bt_sdp_record (C struct), 165
                                                          enumerator), 8
BT_SDP_REFERENCE_PRINTING_SVCLASS (C macro),
                                                  bt_security_err.BT_SECURITY_ERR_AUTH_REQUIREMENT
                                                          (C enumerator), 8
BT_SDP_REFLECTED_UI_SVCLASS (C macro), 153
                                                  bt_security_err.BT_SECURITY_ERR_INVALID_PARAM
bt_sdp_register_service (C function), 163
                                                          (C enumerator), 8
BT_SDP_SAP_SVCLASS (C macro), 154
                                                  bt_security_err.BT_SECURITY_ERR_KEY_REJECTED
BT_SDP_SDP_SERVER_SVCLASS (C macro), 152
                                                          (C enumerator), 8
BT SDP SE016 (C macro), 160
                                                  bt_security_err.BT_SECURITY_ERR_OOB_NOT_AVAILABLE
BT_SDP_SEQ32 (C macro), 160
                                                          (C enumerator), 8
                                                  bt_security_err.BT_SECURITY_ERR_PAIR_NOT_ALLOWED
BT_SDP_SEQ8 (C macro), 160
BT_SDP_SEQ_UNSPEC (C macro), 160
                                                          (C enumerator), 8
BT_SDP_SERIAL_PORT_SVCLASS (C macro), 152
                                                  bt_security_err.BT_SECURITY_ERR_PAIR_NOT_SUPPORTED
BT_SDP_SERVER_RECORD_HANDLE (C macro), 155
                                                          (C enumerator), 8
BT_SDP_SERVICE_ID (C macro), 162
                                                  bt_security_err.BT_SECURITY_ERR_PIN_OR_KEY_MISSING
BT_SDP_SERVICE_NAME (C macro), 162
                                                          (C enumerator), 8
BT_SDP_SIZE_DESC_MASK (C macro), 161
                                                  bt_security_err.BT_SECURITY_ERR_SUCCESS
                                                                                                (C
BT_SDP_SIZE_INDEX_OFFSET (C macro), 161
                                                          enumerator), 8
BT_SDP_SUPPORTED_FEATURES (C macro), 162
                                                  bt_security_err.BT_SECURITY_ERR_UNSPECIFIED
BT_SDP_TEXT_STR16 (C macro), 160
                                                          (Cenumerator), 8
BT_SDP_TEXT_STR32 (C macro), 160
                                                  bt_security_flag(Cenum), 7
BT_SDP_TEXT_STR8 (C macro), 160
                                                  bt_security_flag.BT_SECURITY_FLAG_00B (C enu-
BT_SDP_TEXT_STR_UNSPEC (C macro), 160
                                                          merator), 7
                                                  \verb|bt_security_flag.BT_SECURITY_FLAG_SC| (C \textit{ enu-}
BT_SDP_TYPE_DESC_MASK (C macro), 161
BT_SDP_TYPE_SIZE (C macro), 161
                                                          merator), 7
BT_SDP_TYPE_SIZE_VAR (C macro), 161
                                                  bt_security_info(C struct), 20
BT_SDP_UDI_MT_SVCLASS (C macro), 154
                                                  bt_security_info.enc_key_size (C var), 20
BT_SDP_UDI_TA_SVCLASS (C macro), 154
                                                  bt_security_info.flags(C var), 20
BT_SDP_UINT128 (C macro), 159
                                                  bt_security_info.level(Cvar), 20
BT_SDP_UINT16 (C macro), 159
                                                  bt_security_t (C type), 5
BT_SDP_UINT32 (C macro), 159
                                                  bt_set_appearance (C function), 44
BT_SDP_UINT64 (C macro), 159
                                                  bt_set_bondable (C function), 14
BT_SDP_UINT8 (C macro), 159
                                                  bt_set_name (C function), 44
BT_SDP_UPNP_IP_SVCLASS (C macro), 155
                                                  bt_set_oob_data_flag (C function), 14
BT_SDP_UPNP_L2CAP_SVCLASS (C macro), 155
                                                  bt_spp_callback (C type), 174
BT_SDP_UPNP_LAP_SVCLASS (C macro), 155
                                                  bt_spp_client_connect (C function), 175
BT_SDP_UPNP_PAN_SVCLASS (C macro), 155
                                                  bt_spp_data_send (C function), 176
BT_SDP_UPNP_SVCLASS (C macro), 155
                                                  bt_spp_disconnect (C function), 176
BT_SDP_URL_STR16 (C macro), 161
                                                  bt_spp_discover (C function), 175
```

h 1:	DT HILLD ACCC ACE CDC HAL (C) 207
bt_spp_discover_callback (C type), 174	BT_UUID_ASCS_ASE_SRC_VAL (C macro), 207
bt_spp_get_channel (C function), 176	BT_UUID_ASCS_VAL (C macro), 183
bt_spp_get_conn (<i>C function</i>), 176	BT_UUID_ATT (C macro), 210
bt_spp_get_role (C function), 176	BT_UUID_ATT_VAL (C macro), 210
bt_spp_role (C enum), 175	BT_UUID_AVCTP (C macro), 211
bt_spp_role.BT_SPP_ROLE_CLIENT (C enumerator),	BT_UUID_AVCTP_VAL (C macro), 211
175	BT_UUID_AVDTP (C macro), 211
bt_spp_role.BT_SPP_ROLE_SERVER (<i>C enumerator</i>),	BT_UUID_AVDTP_VAL (C macro), 211
175	BT_UUID_BAR_PRESSURE_TREND (C macro), 194
bt_spp_role_t (<i>C type</i>), 174	BT_UUID_BAR_PRESSURE_TREND_VAL (C macro), 194
bt_spp_server_register (C function), 175	BT_UUID_BAS (C macro), 181
bt_unpair (C function), 59	BT_UUID_BAS_BATTERY_LEVEL (C macro), 187
bt_uuid(C struct), 213	BT_UUID_BAS_BATTERY_LEVEL_VAL (C macro), 187
BT_UUID_128 (<i>C macro</i>), 178	BT_UUID_BAS_VAL (C macro), 181
bt_uuid_128 (<i>C struct</i>), 214	BT_UUID_BASIC_AUDIO (C macro), 184
bt_uuid_128.uuid(<i>C var</i>), 214	BT_UUID_BASIC_AUDIO_VAL (C macro), 184
bt_uuid_128.val (<i>C var</i>), 214	BT_UUID_BASS (C macro), 184
BT_UUID_128_ENCODE (C macro), 178	BT_UUID_BASS_CONTROL_POINT (C macro), 207
BT_UUID_16 (<i>C macro</i>), 178	BT_UUID_BASS_CONTROL_POINT_VAL (C macro), 207
bt_uuid_16 (<i>C struct</i>), 213	BT_UUID_BASS_RECV_STATE (C macro), 207
bt_uuid_16.uuid(<i>C var</i>), 213	BT_UUID_BASS_RECV_STATE_VAL (C macro), 207
bt_uuid_16.val (<i>C var</i>), 213	BT_UUID_BASS_VAL (C macro), 184
BT_UUID_16_ENCODE (C macro), 179	BT_UUID_BMS (C macro), 181
BT_UUID_32 (<i>C macro</i>), 178	BT_UUID_BMS_CONTROL_POINT (C macro), 194
bt_uuid_32 (<i>C struct</i>), 213	BT_UUID_BMS_CONTROL_POINT_VAL (C macro), 194
bt_uuid_32.uuid (<i>C var</i>), 214	BT_UUID_BMS_FEATURE (C macro), 194
bt_uuid_32.val (<i>C var</i>), 214	BT_UUID_BMS_FEATURE_VAL (C macro), 194
BT_UUID_32_ENCODE (C macro), 179	BT_UUID_BMS_VAL (C macro), 181
BT_UUID_AICS (C macro), 182	BT_UUID_BNEP (C macro), 211
BT_UUID_AICS_CONTROL (C macro), 199	BT_UUID_BNEP_VAL (C macro), 211
BT_UUID_AICS_CONTROL_VAL (C macro), 199	BT_UUID_BROADCAST_AUDIO (C macro), 184
BT_UUID_AICS_DESCRIPTION (C macro), 200	BT_UUID_BROADCAST_AUDIO_VAL (C macro), 184
BT_UUID_AICS_DESCRIPTION_VAL (C macro), 199	BT_UUID_CAS (C macro), 184
BT_UUID_AICS_GAIN_SETTINGS (C macro), 199	BT_UUID_CAS_VAL (C macro), 184
BT_UUID_AICS_GAIN_SETTINGS_VAL (C macro), 199	BT_UUID_CCID (C macro), 205
BT_UUID_AICS_INPUT_STATUS (C macro), 199	BT_UUID_CCID_VAL (C macro), 205
BT_UUID_AICS_INPUT_STATUS_VAL (C macro), 199	BT_UUID_CENTRAL_ADDR_RES (C macro), 194
BT_UUID_AICS_INPUT_TYPE (C macro), 199	BT_UUID_CENTRAL_ADDR_RES_VAL (C macro), 194
BT_UUID_AICS_INPUT_TYPE_VAL (C macro), 199	BT_UUID_CGM_FEATURE (C macro), 195
BT_UUID_AICS_STATE (C macro), 199	BT_UUID_CGM_FEATURE_VAL (C macro), 195
BT_UUID_AICS_STATE_VAL (C macro), 199	BT_UUID_CGM_MEASUREMENT (C macro), 194
	· · · · · · · · · · · · · · · · · · ·
BT_UUID_AICS_VAL (C macro), 182	BT_UUID_CGM_MEASUREMENT_VAL (C macro), 194
BT_UUID_ALERT_LEVEL (C macro), 187	BT_UUID_CGM_SESSION_RUN_TIME (C macro), 195
BT_UUID_ALERT_LEVEL_VAL (C macro), 187	BT_UUID_CGM_SESSION_RUN_TIME_VAL (C macro), 195
BT_UUID_APPARENT_WIND_DIR (C macro), 192	BT_UUID_CGM_SESSION_START_TIME (C macro), 195
BT_UUID_APPARENT_WIND_DIR_VAL (C macro), 192	BT_UUID_CGM_SESSION_START_TIME_VAL (C macro),
BT_UUID_APPARENT_WIND_SPEED (C macro), 192	195
BT_UUID_APPARENT_WIND_SPEED_VAL (C macro), 192	BT_UUID_CGM_SPECIFIC_OPS_CONTROL_POINT (C
BT_UUID_ASCS (C macro), 184	<i>macro</i>), 195
BT_UUID_ASCS_ASE_CP (C macro), 207	BT_UUID_CGM_SPECIFIC_OPS_CONTROL_POINT_VAL
BT_UUID_ASCS_ASE_CP_VAL (C macro), 207	(C macro), 195
BT_UUID_ASCS_ASE_SNK (C macro), 207	BT_UUID_CGM_STATUS (C macro), 195
BT_UUID_ASCS_ASE_SNK_VAL (C macro), 207	BT_UUID_CGM_STATUS_VAL (C macro), 195
BT_UUID_ASCS_ASE_SRC (C macro), 207	BT_UUID_CGMS (C macro), 182

```
BT_UUID_CGMS_VAL (C macro), 181
                                                  BT_UUID_DIS_SERIAL_NUMBER (C macro), 188
bt_uuid_cmp (C function), 212
                                                  BT_UUID_DIS_SERIAL_NUMBER_VAL (C macro), 188
BT_UUID_CMTP (C macro), 211
                                                  BT_UUID_DIS_SOFTWARE_REVISION (C macro), 188
BT_UUID_CMTP_VAL (C macro), 211
                                                  BT_UUID_DIS_SOFTWARE_REVISION_VAL (C macro),
bt_uuid_create (C function), 212
BT_UUID_CSC (C macro), 181
                                                  BT_UUID_DIS_SYSTEM_ID (C macro), 188
BT UUID CSC FEATURE (C macro), 191
                                                  BT UUID DIS SYSTEM ID VAL (C macro), 188
BT_UUID_CSC_FEATURE_VAL (C macro), 191
                                                  BT_UUID_DIS_VAL (C macro), 180
BT_UUID_CSC_MEASUREMENT (C macro), 191
                                                  BT UUID ELEVATION (C macro), 191
                                                  BT_UUID_ELEVATION_VAL (C macro), 191
BT_UUID_CSC_MEASUREMENT_VAL (C macro), 191
BT_UUID_CSC_VAL (C macro), 181
                                                  BT_UUID_ES_CONFIGURATION (C macro), 186
BT_UUID_CSIS (C macro), 183
                                                  BT_UUID_ES_CONFIGURATION_VAL (C macro), 186
BT_UUID_CSIS_RANK (C macro), 201
                                                  BT_UUID_ES_MEASUREMENT (C macro), 186
BT_UUID_CSIS_RANK_VAL (C macro), 201
                                                  BT_UUID_ES_MEASUREMENT_VAL (C macro), 186
BT_UUID_CSIS_SET_LOCK (C macro), 201
                                                  BT_UUID_ES_TRIGGER_SETTING (C macro), 186
BT_UUID_CSIS_SET_LOCK_VAL (C macro), 201
                                                  BT_UUID_ES_TRIGGER_SETTING_VAL (C macro), 186
BT_UUID_CSIS_SET_SIRK (C macro), 201
                                                  BT_UUID_ESS (C macro), 181
BT_UUID_CSIS_SET_SIRK_VAL (C macro), 201
                                                  BT_UUID_ESS_VAL (C macro), 181
BT_UUID_CSIS_SET_SIZE (C macro), 201
                                                  BT_UUID_FTP (C macro), 210
BT_UUID_CSIS_SET_SIZE_VAL (C macro), 201
                                                  BT_UUID_FTP_VAL (C macro), 210
BT_UUID_CSIS_VAL (C macro), 183
                                                  BT_UUID_GAP (C macro), 179
BT_UUID_CTS (C macro), 180
                                                  BT_UUID_GAP_APPEARANCE (C macro), 187
BT_UUID_CTS_CURRENT_TIME (C macro), 189
                                                  BT_UUID_GAP_APPEARANCE_VAL (C macro), 187
BT UUID CTS CURRENT TIME VAL (C macro), 189
                                                  BT UUID GAP DEVICE NAME (C macro), 187
BT_UUID_CTS_LOCAL_TIME_INFORMATION (C macro),
                                                  BT_UUID_GAP_DEVICE_NAME_VAL (C macro), 187
                                                  BT_UUID_GAP_PPCP (C macro), 187
BT_UUID_CTS_LOCAL_TIME_INFORMATION_VAL
                                                 BT_UUID_GAP_PPCP_VAL (C macro), 187
        macro), 209
                                                  BT_UUID_GAP_VAL (C macro), 179
BT_UUID_CTS_REFERENCE_TIME_INFORMATION
                                                 BT_UUID_GATT (C macro), 180
                                             (C
                                                  BT_UUID_GATT_CAF (C macro), 186
        macro), 209
                                                  BT_UUID_GATT_CAF_VAL (C macro), 186
BT_UUID_CTS_REFERENCE_TIME_INFORMATION_VAL
        (C macro), 209
                                                  BT_UUID_GATT_CCC (C macro), 185
BT_UUID_CTS_VAL (C macro), 180
                                                  BT_UUID_GATT_CCC_VAL (C macro), 185
BT_UUID_DECLARE_128 (C macro), 178
                                                  BT_UUID_GATT_CEP (C macro), 185
BT_UUID_DECLARE_16 (C macro), 177
                                                  BT_UUID_GATT_CEP_VAL (C macro), 185
                                                  BT_UUID_GATT_CHRC (C macro), 185
BT_UUID_DECLARE_32 (C macro), 178
BT_UUID_DESC_VALUE_CHANGED (C macro), 194
                                                  BT_UUID_GATT_CHRC_VAL (C macro), 185
BT_UUID_DESC_VALUE_CHANGED_VAL (C macro), 193
                                                  BT_UUID_GATT_CLIENT_FEATURES (C macro), 198
BT_UUID_DEW_POINT (C macro), 193
                                                  BT_UUID_GATT_CLIENT_FEATURES_VAL (C macro), 198
BT_UUID_DEW_POINT_VAL (C macro), 193
                                                  BT_UUID_GATT_CPF (C macro), 186
BT_UUID_DIS (C macro), 180
                                                  BT_UUID_GATT_CPF_VAL (C macro), 185
BT_UUID_DIS_FIRMWARE_REVISION (C macro), 188
                                                  BT_UUID_GATT_CUD (C macro), 185
BT_UUID_DIS_FIRMWARE_REVISION_VAL (C macro),
                                                  BT_UUID_GATT_CUD_VAL (C macro), 185
        188
                                                  BT_UUID_GATT_DB_HASH (C macro), 199
BT_UUID_DIS_HARDWARE_REVISION (C macro), 188
                                                  BT_UUID_GATT_DB_HASH_VAL (C macro), 199
BT_UUID_DIS_HARDWARE_REVISION_VAL (C macro),
                                                  BT_UUID_GATT_INCLUDE (C macro), 185
                                                  BT_UUID_GATT_INCLUDE_VAL (C macro), 185
BT_UUID_DIS_MANUFACTURER_NAME (C macro), 189
                                                  BT_UUID_GATT_PRIMARY (C macro), 184
BT_UUID_DIS_MANUFACTURER_NAME_VAL (C macro),
                                                  BT_UUID_GATT_PRIMARY_VAL (C macro), 184
                                                  BT_UUID_GATT_SC (C macro), 187
        189
BT_UUID_DIS_MODEL_NUMBER (C macro), 188
                                                  BT_UUID_GATT_SC_VAL (C macro), 187
BT_UUID_DIS_MODEL_NUMBER_VAL (C macro), 188
                                                  BT_UUID_GATT_SCC (C macro), 185
BT_UUID_DIS_PNP_ID (C macro), 189
                                                  BT_UUID_GATT_SCC_VAL (C macro), 185
BT_UUID_DIS_PNP_ID_VAL (C macro), 189
                                                  BT_UUID_GATT_SECONDARY (C macro), 185
```

```
BT_UUID_GATT_SECONDARY_VAL (C macro), 185
                                                  BT_UUID_HIDS_REPORT_REF_VAL (C macro), 186
BT_UUID_GATT_SERVER_FEATURES (C macro), 199
                                                  BT_UUID_HIDS_REPORT_VAL (C macro), 190
BT_UUID_GATT_SERVER_FEATURES_VAL (C macro), 199
                                                  BT_UUID_HIDS_VAL (C macro), 181
BT_UUID_GATT_VAL (C macro), 179
                                                  BT_UUID_HPS (C macro), 182
                                                  BT_UUID_HPS_VAL (C macro), 182
BT_UUID_GMCS (C macro), 183
BT_UUID_GMCS_VAL (C macro), 183
                                                  BT_UUID_HRS (C macro), 181
BT UUID GTBS (C macro), 183
                                                  BT UUID HRS BODY SENSOR (C macro), 190
BT_UUID_GTBS_VAL (C macro), 183
                                                  BT_UUID_HRS_BODY_SENSOR_VAL (C macro), 189
BT_UUID_GUST_FACTOR (C macro), 192
                                                  BT_UUID_HRS_CONTROL_POINT (C macro), 190
BT_UUID_GUST_FACTOR_VAL (C macro), 192
                                                  BT_UUID_HRS_CONTROL_POINT_VAL (C macro), 190
BT_UUID_HAS (C macro), 184
                                                  BT_UUID_HRS_MEASUREMENT (C macro), 189
BT_UUID_HAS_ACTIVE_PRESET_INDEX (C macro), 209
                                                  BT_UUID_HRS_MEASUREMENT_VAL (C macro), 189
BT_UUID_HAS_ACTIVE_PRESET_INDEX_VAL (C macro),
                                                  BT_UUID_HRS_VAL (C macro), 181
                                                  BT_UUID_HTS (C macro), 180
BT_UUID_HAS_HEARING_AID_FEATURES (C macro), 208
                                                  BT_UUID_HTS_MEASUREMENT (C macro), 188
BT_UUID_HAS_HEARING_AID_FEATURES_VAL
                                                 BT_UUID_HTS_MEASUREMENT_VAL (C macro), 187
        macro), 208
                                                  BT_UUID_HTS_VAL (C macro), 180
BT_UUID_HAS_PRESET_CONTROL_POINT (C macro), 208
                                                  BT_UUID_HTTP (C macro), 211
BT_UUID_HAS_PRESET_CONTROL_POINT_VAL
                                                  BT_UUID_HTTP_CONTROL_POINT (C macro), 196
                                                  BT_UUID_HTTP_CONTROL_POINT_VAL (C macro), 196
        macro), 208
BT_UUID_HAS_VAL (C macro), 184
                                                  BT_UUID_HTTP_ENTITY_BODY (C macro), 196
BT_UUID_HCRP_CTRL (C macro), 211
                                                  BT_UUID_HTTP_ENTITY_BODY_VAL (C macro), 196
BT_UUID_HCRP_CTRL_VAL (C macro), 211
                                                  BT_UUID_HTTP_HEADERS (C macro), 195
                                                  BT UUID HTTP HEADERS VAL (C macro), 195
BT UUID HCRP DATA (C macro), 211
BT_UUID_HCRP_DATA_VAL (C macro), 211
                                                  BT_UUID_HTTP_STATUS_CODE (C macro), 196
BT UUID HCRP NOTE (C macro), 211
                                                  BT_UUID_HTTP_STATUS_CODE_VAL (C macro), 195
BT_UUID_HCRP_NOTE_VAL (C macro), 211
                                                  BT_UUID_HTTP_VAL (C macro), 210
BT_UUID_HEAT_INDEX (C macro), 193
                                                  BT_UUID_HTTPS_SECURITY (C macro), 196
BT_UUID_HEAT_INDEX_VAL (C macro), 193
                                                  BT_UUID_HTTPS_SECURITY_VAL (C macro), 196
BT_UUID_HIDP (C macro), 211
                                                  BT_UUID_HUMIDITY (C macro), 192
BT_UUID_HIDP_VAL (C macro), 211
                                                  BT_UUID_HUMIDITY_VAL (C macro), 192
BT_UUID_HIDS (C macro), 181
                                                  BT_UUID_IAS (C macro), 180
BT_UUID_HIDS_BOOT_KB_IN_REPORT (C macro), 188
                                                  BT_UUID_IAS_VAL (C macro), 180
BT_UUID_HIDS_BOOT_KB_IN_REPORT_VAL (C macro),
                                                  BT_UUID_INIT_128 (C macro), 177
                                                  BT_UUID_INIT_16 (C macro), 177
        188
BT UUID HIDS BOOT KB OUT REPORT (C macro), 189
                                                  BT_UUID_INIT_32 (C macro), 177
BT_UUID_HIDS_BOOT_KB_OUT_REPORT_VAL (C macro),
                                                  BT_UUID_IP (C macro), 210
                                                  BT_UUID_IP_VAL (C macro), 210
BT_UUID_HIDS_BOOT_MOUSE_IN_REPORT (C macro),
                                                  BT_UUID_IPSS (C macro), 182
                                                  BT_UUID_IPSS_VAL (C macro), 182
        189
BT_UUID_HIDS_BOOT_MOUSE_IN_REPORT_VAL
                                             (C BT_UUID_IRRADIANCE (C macro), 193
                                                  BT_UUID_IRRADIANCE_VAL (C macro), 193
        macro), 189
BT_UUID_HIDS_CTRL_POINT (C macro), 190
                                                  BT UUID L2CAP (C macro), 212
BT_UUID_HIDS_CTRL_POINT_VAL (C macro), 190
                                                  BT_UUID_L2CAP_VAL (C macro), 212
BT_UUID_HIDS_EXT_REPORT (C macro), 186
                                                  BT_UUID_LLS (C macro), 180
                                                  BT_UUID_LLS_VAL (C macro), 180
BT_UUID_HIDS_EXT_REPORT_VAL (C macro), 186
BT_UUID_HIDS_INFO (C macro), 190
                                                  BT_UUID_MAGN_DECLINATION (C macro), 189
BT_UUID_HIDS_INFO_VAL (C macro), 190
                                                  BT_UUID_MAGN_DECLINATION_VAL (C macro), 189
BT_UUID_HIDS_PROTOCOL_MODE (C macro), 190
                                                  BT_UUID_MAGN_FLUX_DENSITY_2D (C macro), 194
BT_UUID_HIDS_PROTOCOL_MODE_VAL (C macro), 190
                                                  BT_UUID_MAGN_FLUX_DENSITY_2D_VAL (C macro), 194
                                                  BT_UUID_MAGN_FLUX_DENSITY_3D (C macro), 194
BT_UUID_HIDS_REPORT (C macro), 190
BT_UUID_HIDS_REPORT_MAP (C macro), 190
                                                  BT_UUID_MAGN_FLUX_DENSITY_3D_VAL (C macro), 194
BT_UUID_HIDS_REPORT_MAP_VAL (C macro), 190
                                                  BT_UUID_MCAP_CTRL (C macro), 212
BT_UUID_HIDS_REPORT_REF (C macro), 186
                                                  BT_UUID_MCAP_CTRL_VAL (C macro), 212
```

BT_UUID_MCAP_DATA (C macro), 212	BT_UUID_MCS_TRACK_SEGMENTS_OBJ_ID_VAL (C
BT_UUID_MCAP_DATA_VAL (C macro), 212	macro), 202
BT_UUID_MCS (C macro), 183	BT_UUID_MCS_TRACK_TITLE (C macro), 202
BT_UUID_MCS_CURRENT_GROUP_OBJ_ID (C macro), 203	BT_UUID_MCS_TRACK_TITLE_VAL (C macro), 202
BT_UUID_MCS_CURRENT_GROUP_OBJ_ID_VAL (C	BT_UUID_MCS_VAL (C macro), 183
macro), 203	BT_UUID_MESH_PROV (C macro), 182
BT_UUID_MCS_CURRENT_TRACK_OBJ_ID (C macro), 202	BT_UUID_MESH_PROV_DATA_IN (C macro), 198
BT_UUID_MCS_CURRENT_TRACK_OBJ_ID_VAL (C	BT_UUID_MESH_PROV_DATA_IN_VAL (C macro), 198
macro), 202	BT_UUID_MESH_PROV_DATA_OUT (C macro), 198
BT_UUID_MCS_ICON_OBJ_ID (C macro), 201	BT_UUID_MESH_PROV_DATA_OUT_VAL (C macro), 198
BT_UUID_MCS_ICON_OBJ_ID_VAL (C macro), 201	BT_UUID_MESH_PROV_VAL (C macro), 182
BT_UUID_MCS_ICON_URL (C macro), 201	BT_UUID_MESH_PROXY (C macro), 182
BT_UUID_MCS_ICON_URL_VAL (C macro), 201	BT_UUID_MESH_PROXY_DATA_IN (C macro), 198
BT_UUID_MCS_MEDIA_CONTROL_OPCODES (C macro),	BT_UUID_MESH_PROXY_DATA_IN_VAL (C macro), 198
204	BT_UUID_MESH_PROXY_DATA_OUT (C macro), 198
BT_UUID_MCS_MEDIA_CONTROL_OPCODES_VAL (C	BT_UUID_MESH_PROXY_DATA_OUT_VAL (C macro), 198
macro), 203	BT_UUID_MESH_PROXY_VAL (C macro), 182
BT_UUID_MCS_MEDIA_CONTROL_POINT (C macro), 203	BT_UUID_MICS (C macro), 183
BT_UUID_MCS_MEDIA_CONTROL_POINT_VAL (C macro),	BT_UUID_MICS_MUTE (C macro), 207
203	BT_UUID_MICS_MUTE_VAL (C macro), 207
BT_UUID_MCS_MEDIA_STATE (C macro), 203	BT_UUID_MICS_VAL (C macro), 183
BT_UUID_MCS_MEDIA_STATE_VAL (C macro), 203	BT_UUID_NDTS (C macro), 209
BT_UUID_MCS_NEXT_TRACK_OBJ_ID (C macro), 203	BT_UUID_NDTS_TIME_WITH_DTS (C macro), 209
BT_UUID_MCS_NEXT_TRACK_OBJ_ID_VAL (C macro),	BT_UUID_NDTS_TIME_WITH_DTS_VAL (C macro), 209
203	BT_UUID_NDTS_VAL (C macro), 209
BT_UUID_MCS_PARENT_GROUP_OBJ_ID (C macro), 203	BT_UUID_OBEX (C macro), 210
BT_UUID_MCS_PARENT_GROUP_OBJ_ID_VAL (C macro),	BT_UUID_OBEX_VAL (C macro), 210
203	BT_UUID_OTS (C macro), 182
BT_UUID_MCS_PLAYBACK_SPEED (C macro), 202	BT_UUID_OTS_ACTION_CP (C macro), 197
BT_UUID_MCS_PLAYBACK_SPEED_VAL (C macro), 202	BT_UUID_OTS_ACTION_CP_VAL (C macro), 197
BT_UUID_MCS_PLAYER_NAME (C macro), 201	BT_UUID_OTS_CHANGED (C macro), 198
BT_UUID_MCS_PLAYER_NAME_VAL (C macro), 201	BT_UUID_OTS_CHANGED_VAL (C macro), 197
BT_UUID_MCS_PLAYING_ORDER (C macro), 203	BT_UUID_OTS_DIRECTORY_LISTING (C macro), 198
BT_UUID_MCS_PLAYING_ORDER_VAL (C macro), 203	BT_UUID_OTS_DIRECTORY_LISTING_VAL (C macro),
BT_UUID_MCS_PLAYING_ORDERS (C macro), 203	198
BT_UUID_MCS_PLAYING_ORDERS_VAL (C macro), 203	BT_UUID_OTS_FEATURE (C macro), 196
BT_UUID_MCS_SEARCH_CONTROL_POINT (C macro), 204	BT_UUID_OTS_FEATURE_VAL (C macro), 196
BT_UUID_MCS_SEARCH_CONTROL_POINT_VAL (C	BT_UUID_OTS_FIRST_CREATED (C macro), 197
<i>macro</i>), 204	BT_UUID_OTS_FIRST_CREATED_VAL (C macro), 197
$BT_UUID_MCS_SEARCH_RESULTS_OBJ_ID$ (C macro),	BT_UUID_OTS_ID (C macro), 197
204	BT_UUID_OTS_ID_VAL (C macro), 197
BT_UUID_MCS_SEARCH_RESULTS_OBJ_ID_VAL (C	BT_UUID_OTS_LAST_MODIFIED (C macro), 197
macro), 204	BT_UUID_OTS_LAST_MODIFIED_VAL (C macro), 197
BT_UUID_MCS_SEEKING_SPEED (C macro), 202	BT_UUID_OTS_LIST_CP (C macro), 197
BT_UUID_MCS_SEEKING_SPEED_VAL (C macro), 202	BT_UUID_OTS_LIST_CP_VAL (C macro), 197
BT_UUID_MCS_TRACK_CHANGED (C macro), 202	BT_UUID_OTS_LIST_FILTER (C macro), 197
BT_UUID_MCS_TRACK_CHANGED_VAL (C macro), 201	BT_UUID_OTS_LIST_FILTER_VAL (C macro), 197
BT_UUID_MCS_TRACK_DURATION (C macro), 202	BT_UUID_OTS_NAME (C macro), 196
BT_UUID_MCS_TRACK_DURATION_VAL (C macro), 202	BT_UUID_OTS_NAME_VAL (C macro), 196
BT_UUID_MCS_TRACK_POSITION (C macro), 202	BT_UUID_OTS_PROPERTIES (C macro), 197
BT_UUID_MCS_TRACK_POSITION_VAL (C macro), 202	BT_UUID_OTS_PROPERTIES_VAL (C macro), 197
BT_UUID_MCS_TRACK_SEGMENTS_OBJ_ID (C macro),	BT_UUID_OTS_SIZE (C macro), 196
202	BT_UUID_OTS_SIZE_VAL (C macro), 196
	BT IIIID OTS TYPE (C macro) 196

```
BT_UUID_OTS_TYPE_GROUP (C macro), 204
                                                 BT_UUID_SC_CONTROL_POINT (C macro), 191
BT_UUID_OTS_TYPE_GROUP_VAL (C macro), 204
                                                 BT_UUID_SC_CONTROL_POINT_VAL (C macro), 191
BT_UUID_OTS_TYPE_MPL_ICON (C macro), 204
                                                 BT_UUID_SDP (C macro), 210
BT_UUID_OTS_TYPE_MPL_ICON_VAL (C macro), 204
                                                 BT_UUID_SDP_VAL (C macro), 210
BT_UUID_OTS_TYPE_TRACK (C macro), 204
                                                 BT_UUID_SENSOR_LOCATION (C macro), 191
BT_UUID_OTS_TYPE_TRACK_SEGMENT (C macro), 204
                                                 BT_UUID_SENSOR_LOCATION_VAL (C macro), 191
BT UUID OTS TYPE TRACK SEGMENT VAL (C macro).
                                                 BT UUID SIZE 128 (C macro), 177
                                                 BT_UUID_SIZE_16 (C macro), 177
BT_UUID_OTS_TYPE_TRACK_VAL (C macro), 204
                                                 BT_UUID_SIZE_32 (C macro), 177
BT_UUID_OTS_TYPE_UNSPECIFIED (C macro), 198
                                                 BT_UUID_STR_LEN (C macro), 179
BT_UUID_OTS_TYPE_UNSPECIFIED_VAL(C macro), 198
                                                 BT_UUID_TBS (C macro), 183
BT_UUID_OTS_TYPE_VAL (C macro), 196
                                                 BT_UUID_TBS_CALL_CONTROL_POINT (C macro), 206
BT_UUID_OTS_VAL (C macro), 182
                                                 BT_UUID_TBS_CALL_CONTROL_POINT_VAL (C macro),
BT_UUID_PACS (C macro), 184
                                                          206
BT_UUID_PACS_AVAILABLE_CONTEXT (C macro), 208
                                                 BT_UUID_TBS_CALL_STATE (C macro), 206
BT_UUID_PACS_AVAILABLE_CONTEXT_VAL (C macro),
                                                 BT_UUID_TBS_CALL_STATE_VAL (C macro), 206
        208
                                                 BT_UUID_TBS_FRIENDLY_NAME (C macro), 207
BT_UUID_PACS_SNK (C macro), 207
                                                 BT_UUID_TBS_FRIENDLY_NAME_VAL (C macro), 206
BT_UUID_PACS_SNK_LOC (C macro), 208
                                                 BT_UUID_TBS_INCOMING_CALL (C macro), 206
                                                 BT_UUID_TBS_INCOMING_CALL_VAL (C macro), 206
BT_UUID_PACS_SNK_LOC_VAL (C macro), 208
BT_UUID_PACS_SNK_VAL (C macro), 207
                                                 BT_UUID_TBS_INCOMING_URI (C macro), 206
BT_UUID_PACS_SRC (C macro), 208
                                                 BT_UUID_TBS_INCOMING_URI_VAL (C macro), 206
BT_UUID_PACS_SRC_LOC (C macro), 208
                                                 BT_UUID_TBS_LIST_CURRENT_CALLS (C macro), 205
BT_UUID_PACS_SRC_LOC_VAL (C macro), 208
                                                 BT UUID TBS LIST CURRENT CALLS VAL (C macro).
BT_UUID_PACS_SRC_VAL (C macro), 208
                                                          205
BT_UUID_PACS_SUPPORTED_CONTEXT (C macro), 208
                                                 BT_UUID_TBS_OPTIONAL_OPCODES (C macro), 206
BT_UUID_PACS_SUPPORTED_CONTEXT_VAL (C macro),
                                                 BT_UUID_TBS_OPTIONAL_OPCODES_VAL (C macro), 206
                                                 BT_UUID_TBS_PROVIDER_NAME (C macro), 204
        208
BT_UUID_PACS_VAL (C macro), 184
                                                 BT_UUID_TBS_PROVIDER_NAME_VAL (C macro), 204
BT_UUID_POLLEN_CONCENTRATION (C macro), 193
                                                 BT_UUID_TBS_SIGNAL_INTERVAL (C macro), 205
                                                 BT_UUID_TBS_SIGNAL_INTERVAL_VAL (C macro), 205
BT_UUID_POLLEN_CONCENTRATION_VAL (C macro), 193
BT_UUID_PRESSURE (C macro), 192
                                                 BT_UUID_TBS_SIGNAL_STRENGTH (C macro), 205
BT_UUID_PRESSURE_VAL (C macro), 191
                                                 BT_UUID_TBS_SIGNAL_STRENGTH_VAL (C macro), 205
BT_UUID_RAINFALL (C macro), 193
                                                 BT_UUID_TBS_STATUS_FLAGS (C macro), 206
BT_UUID_RAINFALL_VAL (C macro), 193
                                                 BT_UUID_TBS_STATUS_FLAGS_VAL (C macro), 205
BT_UUID_RECORD_ACCESS_CONTROL_POINT (C macro),
                                                 BT_UUID_TBS_TECHNOLOGY (C macro), 205
                                                 BT_UUID_TBS_TECHNOLOGY_VAL (C macro), 205
BT_UUID_RECORD_ACCESS_CONTROL_POINT_VAL
                                             (C BT_UUID_TBS_TERMINATE_REASON (C macro), 206
                                                 BT_UUID_TBS_TERMINATE_REASON_VAL (C macro), 206
        macro), 190
BT_UUID_RFCOMM (C macro), 210
                                                 BT_UUID_TBS_UCI (C macro), 205
BT_UUID_RFCOMM_VAL (C macro), 210
                                                 BT_UUID_TBS_UCI_VAL (C macro), 205
                                                 BT_UUID_TBS_URI_LIST (C macro), 205
BT_UUID_RSC_FEATURE (C macro), 191
BT_UUID_RSC_FEATURE_VAL (C macro), 191
                                                 BT_UUID_TBS_URI_LIST_VAL (C macro), 205
BT_UUID_RSC_MEASUREMENT (C macro), 191
                                                 BT_UUID_TBS_VAL (C macro), 183
BT_UUID_RSC_MEASUREMENT_VAL (C macro), 191
                                                 BT_UUID_TCP (C macro), 210
BT_UUID_RSCS (C macro), 181
                                                 BT_UUID_TCP_VAL (C macro), 210
BT_UUID_RSCS_VAL (C macro), 181
                                                 BT_UUID_TCS_AT (C macro), 210
BT_UUID_RTUS (C macro), 209
                                                 BT_UUID_TCS_AT_VAL (C macro), 210
BT_UUID_RTUS_CONTROL_POINT (C macro), 209
                                                 BT_UUID_TCS_BIN (C macro), 210
BT_UUID_RTUS_CONTROL_POINT_VAL (C macro), 209
                                                 BT_UUID_TCS_BIN_VAL (C macro), 210
BT_UUID_RTUS_TIME_UPDATE_STATE (C macro), 209
                                                 BT_UUID_TEMPERATURE (C macro), 192
BT_UUID_RTUS_TIME_UPDATE_STATE_VAL (C macro),
                                                 BT_UUID_TEMPERATURE_VAL (C macro), 192
        209
                                                 bt_uuid_to_str (C function), 213
BT_UUID_RTUS_VAL (C macro), 209
                                                 BT_UUID_TPS (C macro), 180
```

```
BT_UUID_TPS_TX_POWER_LEVEL (C macro), 187
                                                   hfp_aq_call_setup_status_t.HFP_AG_CALL_SETUP_STATUS_OUTGO
BT_UUID_TPS_TX_POWER_LEVEL_VAL (C macro), 187
                                                            (C enumerator), 122
BT_UUID_TPS_VAL (C macro), 180
                                                   hfp_ag_call_status_t (C type), 121
BT_UUID_TRUE_WIND_DIR (C macro), 192
                                                   hfp_ag_cind_t (C type), 121
BT_UUID_TRUE_WIND_DIR_VAL (C macro), 192
                                                   hfp_ag_get_config(C type), 121
BT_UUID_TRUE_WIND_SPEED (C macro), 192
                                                   HFP_HF_CMD_CME_ERROR (C macro), 120
BT_UUID_TRUE_WIND_SPEED_VAL (C macro), 192
                                                   HFP_HF_CMD_ERROR (C macro), 120
BT_UUID_UDI (C macro), 211
                                                   HFP_HF_CMD_OK (C macro), 120
BT_UUID_UDI_VAL (C macro), 211
                                                   HFP_HF_CMD_UNKNOWN_ERROR (C macro), 120
BT_UUID_UDP (C macro), 210
                                                   HFP_HF_DIGIT_ARRAY_SIZE (C macro), 120
BT_UUID_UDP_VAL (C macro), 210
                                                   HFP_HF_MAX_OPERATOR_NAME_LEN (C macro), 120
BT_UUID_UPNP (C macro), 211
                                                   hps_config_t (C struct), 217
BT_UUID_UPNP_VAL (C macro), 211
                                                   hps_data_status_t (C type), 215
BT_UUID_URI (C macro), 195
                                                   hps_flags_t (C type), 215
BT_UUID_URI_VAL (C macro), 195
                                                   hps_http_command_t (C type), 215
BT_UUID_UV_INDEX (C macro), 193
                                                   hps_state_t (C type), 215
BT_UUID_UV_INDEX_VAL (C macro), 193
                                                   hps_status_t (C struct), 217
BT_UUID_VALID_RANGE (C macro), 186
                                                   hts_include_temp_type (C macro), 218
BT_UUID_VALID_RANGE_VAL (C macro), 186
                                                   hts_unit_celsius_c (C macro), 218
BT_UUID_VCS (C macro), 182
                                                   hts_unit_fahrenheit_c (C macro), 218
BT_UUID_VCS_CONTROL (C macro), 200
BT_UUID_VCS_CONTROL_VAL (C macro), 200
BT_UUID_VCS_FLAGS (C macro), 200
                                                   ipsp_connect (C function), 219
BT_UUID_VCS_FLAGS_VAL (C macro), 200
                                                   ipsp_init (C function), 219
BT_UUID_VCS_STATE (C macro), 200
                                                   ipsp_listen (C function), 219
BT_UUID_VCS_STATE_VAL (C macro), 200
                                                   ipsp_rx_cb_t (C type), 219
BT_UUID_VCS_VAL (C macro), 182
                                                   ipsp_send (C function), 219
BT_UUID_VOCS (C macro), 183
BT_UUID_VOCS_CONTROL (C macro), 200
                                                    M
BT_UUID_VOCS_CONTROL_VAL (C macro), 200
                                                   MAX_BODY_LEN (C macro), 214
BT_UUID_VOCS_DESCRIPTION (C macro), 200
                                                   MAX_HEADERS_LEN (C macro), 214
BT_UUID_VOCS_DESCRIPTION_VAL (C macro), 200
                                                   MAX_URI_LEN (C macro), 214
BT_UUID_VOCS_LOCATION (C macro), 200
                                                   MEDIA_TYPE (C enum), 168
BT_UUID_VOCS_LOCATION_VAL (C macro), 200
                                                   MEDIA_TYPE.BT_A2DP_AUDIO (C enumerator), 168
BT_UUID_VOCS_STATE (C macro), 200
                                                   \texttt{MEDIA\_TYPE.BT\_A2DP\_MULTIMEDIA} (C enumerator),
BT_UUID_VOCS_STATE_VAL (C macro), 200
BT_UUID_VOCS_VAL (C macro), 183
                                                   MEDIA_TYPE.BT_A2DP_VIDEO (C enumerator), 168
BT_UUID_WIND_CHILL (C macro), 193
BT_UUID_WIND_CHILL_VAL (C macro), 193
                                                   Р
                                                   pxr_deinit (C function), 221
D
                                                   pxr_ias_get_alert_level (C function), 221
discover_cb_t (C struct), 177
                                                   pxr_init (C function), 221
                                                   pxr_lls_get_alert_level (C function), 221
Н
                                                   pxr_tps_get_power_level (C function), 221
hf_multiparty_call_option_t (C type), 121
                                                   pxr_tps_set_power_level (C function), 221
hf_volume_type_t (C type), 121
                                                    R
hf_waiting_call_state_t(C type), 121
hfp_ag_call_setup_status_t(Cenum), 122
                                                         lls_alert_level (C function), 220
S_IDLE
hfp_ag_call_setup_status_t.HFP_AG_CALL_SETUP
                                                             _power_level (C function), 221
        (C enumerator), 122
                                                             _power_level_desc (C function), 221
hfp_ag_call_setup_status_t.HFP_AG_CALL_SETUP
                                                             E (C'enum), 168
        (C enumerator), 122
                                                                  A2DP_SINK (C enumerator), 169
NG_ALFRTING
A2DP_SOURCE (C enumerator), 168
hfp_ag_call_setup_status_t.HFP_AG_CALL_SETUP_
        (C enumerator), 122
```

Т

temp_measurement(*C struct*), 218

U

USER_DATA_MIN (C macro), 219

W

write_http_entity_body (C function), 217
write_http_headers (C function), 217
write_ias_alert_level (C function), 220
write_lls_alert_level (C function), 220

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorlQ, QorlQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, elQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.



© NXP B.V. 2021.

All rights reserved.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 12 July 2022
Document identifier: EFBTPALAPIRM