NXP Semiconductors
User's Guide

Document identifier: EFBTPALAPIRM
Rev. 2, 12 July 2022

EdgeFast BT PAL Documentation

CONTENTS:

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_bluetooth_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)

Convert connection interval to milliseconds.

Multiply by 1.25 to get milliseconds.

Note that this may be inaccurate, as something like 7.5 ms cannot be accurately presented with integers.

BT_CONN_INTERVAL_TO_US(interval)

Convert connection interval to microseconds.

Multiply by 1250 to get microseconds.

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

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 **bt_conn_type**

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_t**

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_OOB**

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 **bt_conn_le_path_loss_zone**

Path Loss zone that has been entered.

The path loss zone that has been entered in the most recent LE Path Loss Monitoring Threshold Change event as documented in Core Spec. Version 5.4 Vol.4, Part E, 7.7.65.32.

Note

BT_CONN_LE_PATH_LOSS_ZONE_UNAVAILABLE has been added to notify when path loss becomes unavailable.

Values:

enumerator **BT_CONN_LE_PATH_LOSS_ZONE_ENTERED_LOW**

Low path loss zone entered.

enumerator **BT_CONN_LE_PATH_LOSS_ZONE_ENTERED_MIDDLE**

Middle path loss zone entered.

enumerator **BT_CONN_LE_PATH_LOSS_ZONE_ENTERED_HIGH**

High path loss zone entered.

enumerator **BT_CONN_LE_PATH_LOSS_ZONE_UNAVAILABLE**

Path loss has become unavailable.

enum **bt_conn_auth_keypress**

Passkey Keypress Notification type.

The numeric values are the same as in the Core specification for Pairing Keypress Notification PDU.

Values:

enumerator **BT_CONN_AUTH_KEYPRESS_ENTRY_STARTED**

enumerator **BT_CONN_AUTH_KEYPRESS_DIGIT_ENTERED**

enumerator **BT_CONN_AUTH_KEYPRESS_DIGIT_ERASED**

enumerator **BT_CONN_AUTH_KEYPRESS_CLEARED**

enumerator **BT_CONN_AUTH_KEYPRESS_ENTRY_COMPLETED**

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 *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**(enum *bt_conn_type* type, void (*func)(struct bt_conn *conn, void *data), void *data)

Iterate through all bt_conn objects.

Iterates through all bt_conn objects that are alive in the Host allocator.

To find established connections, combine this with *bt_conn_get_info*. Check that *bt_conn_info::state* is *BT_CONN_STATE_CONNECTED*.

Thread safety: This API is thread safe, but it does not guarantee a sequentially-consistent view for objects allocated during the current invocation of this API. E.g. If preempted while allocations A then B then C happen then results may include A and C but miss B.

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 CONFIG_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) {CONFIG_BT_REMOTE_VERSION} must be enabled

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.

-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.

-ENOBUFS HCI command buffer is not available.

int **bt_conn_le_enhanced_get_tx_power_level**(struct bt_conn *conn, struct *bt_conn_le_tx_power* *tx_power)

Get local enhanced connection transmit power level.

Parameters

- **conn** – Connection object.
- **tx_power** – Transmit power level descriptor.

Return values

-ENOBUFS – HCI command buffer is not available.

Returns

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

int **bt_conn_le_get_remote_tx_power_level**(struct bt_conn *conn, enum *bt_conn_le_tx_power_phy* phy)

Get remote (peer) transmit power level.

Parameters

- **conn** – Connection object.
- **phy** – PHY information.

Return values

-ENOBUFS – HCI command buffer is not available.

Returns

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

int **bt_conn_le_set_tx_power_report_enable**(struct bt_conn *conn, bool local_enable, bool remote_enable)

Enable transmit power reporting.

Parameters

- **conn** – Connection object.
- **local_enable** – Enable/disable reporting for local.
- **remote_enable** – Enable/disable reporting for remote.

Return values

-ENOBUFS – HCI command buffer is not available.

Returns

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

int **bt_conn_le_set_path_loss_mon_param**(struct bt_conn *conn, const struct *bt_conn_le_path_loss_reporting_param* *param)

Set Path Loss Monitoring Parameters.

Change the configuration for path loss threshold change events for a given conn handle.

Note

To use this API {CONFIG_BT_PATH_LOSS_MONITORING} must be set.

Parameters

- **conn** – Connection object.
- **param** – Path Loss Monitoring parameters

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

int **bt_conn_le_set_path_loss_mon_enable**(struct bt_conn *conn, bool enable)

Enable or Disable Path Loss Monitoring.

Enable or disable Path Loss Monitoring, which will decide whether Path Loss Threshold events are sent from the controller to the host.

> **Note**
> To use this API {CONFIG_BT_PATH_LOSS_MONITORING} must be set.

Parameters

- **conn** – Connection Object.
- **enable** – Enable/disable path loss reporting.

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

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 {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 if {CONFIG_BT_SCAN_AND_INITIATE_IN_PARALLEL} is not enabled.

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_synced**(const struct bt_le_ext_adv *adv, const struct *bt_conn_le_create_synced_param* *synced_param, const struct *bt_le_conn_param* *conn_param, struct bt_conn **conn)

Create a connection to a synced device.

Initiate a connection to a synced device from a Periodic Advertising with Responses (PAwR) train.

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 Periodic Advertising Connection Procedure.

Parameters

- **adv** – **[in]** The adverting set the PAwR advertiser belongs to.
- **synced_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 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.

-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. Note that `sec` has no effect on the security level selected for the pairing process. The selection is instead controlled by the values of the registered *bt_conn_auth_cb*. 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 an error if the required level of security defined using `sec` 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 an error if the pairing procedure has already been initiated by the local device or the peer device.

Note

When {CONFIG_BT_SMP_SC_ONLY} is enabled then the security level will always be level 4.

When {CONFIG_BT_SMP_OOB_LEGACY_PAIR_ONLY} is enabled then the security level will always be level 3.

When *BT_SECURITY_FORCE_PAIR* within `sec` is enabled then the pairing procedure will always be initiated.

Parameters

- **conn** – Connection object.
- **sec** – Requested minimum 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.

int **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.

Return values

- **0** – Success.
- **-EEXIST** – if cb was already registered.

int **bt_conn_cb_unregister**(struct *bt_conn_cb* *cb)

Unregister connection callbacks.

Unregister the state of connections callbacks.

Parameters

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

Return values

- **0** – Success
- **-EINVAL** – If cb is NULL
- **-ENOENT** – if cb was not registered

static inline const char ***bt_security_err_to_str**(enum *bt_security_err* err)

Converts a security error to string.

Returns
The string representation of the security error code. If {CONFIG_BT_SECURITY_ERR_TO_STR} is not enabled, this just returns the empty string

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.

int **bt_conn_set_bondable**(struct bt_conn *conn, bool enable)

Set/clear the bonding flag for a given connection.

Set/clear the Bonding flag in the Authentication Requirements of SMP Pairing Request/Response data for a given connection.

The bonding flag for a given connection cannot be set/cleared if security procedures in the SMP module have already started. This function can be called only once per connection.

If the bonding flag is not set/cleared for a given connection, the value will depend on global configuration which is set using bt_set_bondable. The default value of the global configuration is defined using CONFIG_BT_BONDABLE Kconfig option.

Parameters

- **conn** – Connection object.
- **enable** – Value allowing/disallowing to be bondable.

void **bt_le_oob_set_sc_flag**(bool enable)

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

Set/clear the OOB data flag for LE SC SMP Pairing Request/Response data.

Parameters

- **enable** – Value allowing/disallowing remote LE SC OOB data.

void **bt_le_oob_set_legacy_flag**(bool enable)

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

Set/clear the OOB data flag for legacy SMP Pairing Request/Response data.

Parameters

- **enable** – Value allowing/disallowing remote legacy 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 {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 **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_keypress_notify**(struct bt_conn *conn, enum *bt_conn_auth_keypress* type)

Send Passkey Keypress Notification during pairing.

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

Requires {CONFIG_BT_PASSKEY_KEYPRESS}.

See also

bt_conn_auth_keypress.

Parameters

- **conn** – Destination for the notification.

- **type** – What keypress event type to send.

Return values

- **0** – Success
- **-EINVAL** – Improper use of the API.
- **-ENOMEM** – Failed to allocate.
- **-ENOBUFS** – Failed to allocate.

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_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

uint16_t **options**

Connection PHY options.

uint8_t **pref_tx_phy**

Bitmask of preferred transmit PHYs

uint8_t **pref_rx_phy**

Bitmask of preferred receive 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 **interval**

Connection interval

uint16_t **latency**

Connection peripheral latency

uint16_t **timeout**

Connection supervision timeout

struct **bt_conn_br_info**

#include <*conn.h*> BR/EDR Connection Info Structure

Public Members

const *bt_addr_t* ***dst**

Destination (Remote) BR/EDR address

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

enum *bt_conn_type* **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.

union **__unnamed__**

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.

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 {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

union **__unnamed__**

Public Members

struct *bt_conn_le_remote_info* **le**

LE connection remote info

struct *bt_conn_br_remote_info* **br**

BR/EDR connection remote info

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_tx_power_report**

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

Public Members

uint8_t **reason**

Reason for Transmit power reporting, as documented in Core Spec. Version 5.4 Vol. 4, Part E, 7.7.65.33.

enum *bt_conn_le_tx_power_phy* **phy**

Phy of Transmit power reporting.

int8_t **tx_power_level**

Transmit power level
- 0xXX - Transmit power level
  - Range: -127 to 20
  - Units: dBm
- 0x7E - Remote device is not managing power levels on this PHY.
- 0x7F - Transmit power level is not available

uint8_t **tx_power_level_flag**

Bit 0: Transmit power level is at minimum level. Bit 1: Transmit power level is at maximum level.

int8_t **delta**

Change in transmit power level
- 0xXX - Change in transmit power level (positive indicates increased power, negative indicates decreased power, zero indicates unchanged) Units: dB
- 0x7F - Change is not available or is out of range.

struct **bt_conn_le_path_loss_threshold_report**

#include <conn.h> LE Path Loss Monitoring Threshold Change Report Structure.

Public Members

enum *bt_conn_le_path_loss_zone* **zone**

Path Loss zone as documented in Core Spec. Version 5.4 Vol.4, Part E, 7.7.65.32.

uint8_t **path_loss**

Current path loss (dB).

struct **bt_conn_le_path_loss_reporting_param**

#include <conn.h> LE Path Loss Monitoring Parameters Structure as defined in Core Spec. Version 5.4 Vol.4, Part E, 7.8.119 LE Set Path Loss Reporting Parameters command.

Public Members

uint8_t **high_threshold**

High threshold for the path loss (dB).

uint8_t **high_hysteresis**

Hysteresis value for the high threshold (dB).

uint8_t **low_threshold**

Low threshold for the path loss (dB).

uint8_t **low_hysteresis**

Hysteresis value for the low threshold (dB).

uint16_t **min_time_spent**

Minimum time in number of connection events to be observed once the path loss crosses the threshold before an event is generated.

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 {CONFIG_BT_CREATE_CONN_TIMEOUT} timeout.

> **Note**
>
> Unused in *bt_conn_le_create_auto*

struct **bt_conn_le_create_synced_param**

#include <conn.h>

Public Members

const *bt_addr_le_t* ***peer**

Remote address.

The peer must be synchronized to the PAwR train.

uint8_t **subevent**

The subevent where the connection will be initiated.

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 {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 {CONFIG_BT_MAX_CONN}.

Param conn
Connection object.

Param reason
BT_HCI_ERR_* reason for the disconnection.

void (*recycled)(void)

A connection object has been returned to the pool.

This callback notifies the application that it might be able to allocate a connection object. No guarantee, first come, first serve.

Use this to e.g. re-start connectable advertising or scanning.

Treat this callback as an ISR, as it originates from *bt_conn_unref* which is used by the BT stack. Making Bluetooth API calls in this context is error-prone and strongly discouraged.

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

union **__unnamed__**

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_conn_oob_info*.[anonymous].[anonymous].[anonymous] **oob_config**

OOB data configuration

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.

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

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

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.

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.

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_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.

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.

1.5 Generic Access Profile (GAP)

1.5.1 API Reference

group bt_gap

Generic Access Profile (GAP)

Since

1.0

Version

1.0.0

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_SERIALIZED_SIZE(data_len)

Bluetooth data serialized size.

Get the size of a serialized *bt_data* given its data length.

Size of 'AD Structure'->'Length' field, equal to 1. Size of 'AD Structure'->'Data'->'AD Type' field, equal to 1. Size of 'AD Structure'->'Data'->'AD Data' field, equal to data_len.

See Core Specification Version 5.4 Vol. 3 Part C, 11, Figure 11.1.

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_ONE_TIME

This is the recommended default for connectable advertisers.

BT_LE_ADV_CONN_NAME

Deprecated:

This macro will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

BT_LE_ADV_CONN_NAME_AD

Deprecated:

This macro will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

BT_LE_ADV_CONN_DIR_LOW_DUTY(_peer)

BT_LE_ADV_NCONN

Non-connectable advertising with private address

BT_LE_ADV_NCONN_NAME

Deprecated:

This macro will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

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

Connectable extended advertising

BT_LE_EXT_ADV_CONN_NAME

Deprecated:

This macro will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

Connectable extended advertising with *BT_LE_ADV_OPT_USE_NAME*

BT_LE_EXT_ADV_SCAN

Scannable extended advertising

BT_LE_EXT_ADV_SCAN_NAME

Deprecated:

This macro will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

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

Deprecated:

This macro will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

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

Deprecated:

This macro will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

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_ACTIVE_CONTINUOUS

Helper macro to enable active scanning to discover new devices with window == interval.

Continuous scanning should be used to maximize the chances of receiving advertising packets.

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_PASSIVE_CONTINUOUS

Helper macro to enable passive scanning to discover new devices with window==interval.

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 {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.

Deprecated:

This option will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

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 *BT_LE_ADV_OPT_EXT_ADV* and not *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. *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.

```
BT_DATA(BT_DATA_NAME_COMPLETE, name, sizeof(name) - 1)
```

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.

This cannot be used with *bt_le_adv_start()*.

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.

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.

Deprecated:

This option will be removed in the near future, see https://github.com/zephyrproject-rtos/zephyr/issues/71686

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

> **Note**
>
> Requires *BT_LE_ADV_OPT_USE_NAME*

enumerator **BT_LE_ADV_OPT_USE_NRPA**

Advertise using a Non-Resolvable Private Address.

A new NRPA is set when updating the advertising parameters.

This is an advanced feature; most users will want to enable {CONFIG_BT_EXT_ADV} instead.

> **Note**
>
> Not implemented when {CONFIG_BT_PRIVACY}.
>
> Mutually exclusive with BT_LE_ADV_OPT_USE_IDENTITY.

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

enumerator **BT_LE_PER_ADV_SYNC_TRANSFER_OPT_REPORTING_INITIALLY_DISABLED**

Sync to received PAST packets but don't generate sync reports.

This option must not be set at the same time as *BT_LE_PER_ADV_SYNC_TRANSFER_OPT_FILTER_DUPLICATES*.

enumerator **BT_LE_PER_ADV_SYNC_TRANSFER_OPT_FILTER_DUPLICATES**

Sync to received PAST packets and generate sync reports with duplicate filtering.

This option must not be set at the same time as *BT_LE_PER_ADV_SYNC_TRANSFER_OPT_REPORTING_INITIALLY_*

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 {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.

int **bt_disable**(void)

Disable Bluetooth.

Disable Bluetooth. Can't be called before bt_enable has completed.

This API will clear all configured identities and keys that are not persistently stored with {CONFIG_BT_SETTINGS}. These can be restored with settings_load() before reenabling the stack.

This API does *not* clear previously registered callbacks like *bt_le_scan_cb_register* and *bt_conn_cb_register*. That is, the application shall not re-register them when the Bluetooth subsystem is re-enabled later.

Close and release HCI resources. Result is architecture dependent.

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*.

See also

{CONFIG_BT_DEVICE_NAME_MAX}.

Note

Requires {CONFIG_BT_DEVICE_NAME_DYNAMIC}.

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 {CONFIG_BT_SETTINGS} is enabled.

This symbol is linkable if {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()*. 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.

If supported by the HCI driver (indicated by setting {CONFIG_BT_HCI_SET_PUBLIC_ADDR}), the first call to this function can be used to set the controller's public identity address. This call must happen before calling *bt_enable()*. Subsequent calls always add/generate random static addresses.

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 {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 {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_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.

size_t **bt_data_get_len**(const struct *bt_data* data[], size_t data_count)

Get the total size (in bytes) of a given set of *bt_data* structures.

Parameters

- **data** – **[in]** Array of *bt_data* structures.
- **data_count** – **[in]** Number of *bt_data* structures in `data`.

Returns
Size of the concatenated data, built from the *bt_data* structure set.

size_t **bt_data_serialize**(const struct *bt_data* *input, uint8_t *output)

Serialize a *bt_data* struct into an advertising structure (a flat byte array).

The data are formatted according to the Bluetooth Core Specification v. 5.4, vol. 3, part C, 11.

Parameters

- **input** – **[in]** Single *bt_data* structure to read from.
- **output** – **[out]** Buffer large enough to store the advertising structure in `input`. The size of it must be at least the size of the `input->data_len + 2` (for the type and the length).

Returns
Number of bytes written in `output`.

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*.

This function cannot be used with *BT_LE_ADV_OPT_EXT_ADV* in the `param.options`. For extended advertising, the bt_le_ext_adv_* functions must be used.

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.

-ENOMEM No free connection objects available for connectable advertiser.

-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, const 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 sd_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.

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 CONFIG_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_set_subevent_data**(const struct bt_le_ext_adv *adv, uint8_t num_subevents, const struct *bt_le_per_adv_subevent_data_params* *params)

Set the periodic advertising with response subevent data.

Set the data for one or more subevents of a Periodic Advertising with Responses Advertiser in reply data request.

Parameters

- **adv** – The extended advertiser the PAwR train belongs to.
- **num_subevents** – The number of subevents to set data for.
- **params** – Subevent parameters.

Pre

There are `num_subevents` elements in `params`.

The controller has requested data for the subevents in `params`.

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 CONFIG_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

struct bt_le_per_adv_sync ***bt_le_per_adv_sync_lookup_index**(uint8_t index)

Get a periodic advertising sync object from the array index.

This function is to get the periodic advertising sync object from the array index. The array has CONFIG_BT_PER_ADV_SYNC_MAX elements.

Parameters

- **index** – The index of the periodic advertising sync object. The range of the index value is 0..CONFIG_BT_PER_ADV_SYNC_MAX-1

Returns
The periodic advertising sync object of the array index or NULL if invalid index.

int **bt_le_per_adv_sync_get_info**(struct bt_le_per_adv_sync *per_adv_sync, struct *bt_le_per_adv_sync_info* *info)

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.

struct bt_le_per_adv_sync ***bt_le_per_adv_sync_lookup_addr**(const *bt_addr_le_t* *adv_addr, uint8_t sid)

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.

int **bt_le_per_adv_sync_create**(const struct *bt_le_per_adv_sync_param* *param, struct bt_le_per_adv_sync **out_sync)

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.

This function does not timeout, and will continue to look for an advertiser until it either finds it or *bt_le_per_adv_sync_delete()* is called. It is thus suggested to implement a timeout when using this, if it is expected to find the advertiser within a reasonable timeframe.

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.

int **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.

Return values

- **0** – Success.
- **-EEXIST** – if `cb` was already registered.

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 {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 {CONFIG_BT_SCAN_WITH_IDENTITY} must be enabled.
>
> Setting the `param.timeout` parameter is not supported when {CONFIG_BT_PRIVACY} is enabled, when the param.type is *BT_LE_SCAN_TYPE_ACTIVE*. Supplying a non-zero timeout will result in an -EINVAL error code.

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.

int **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.

Return values

- **0** – Success.
- **-EEXIST** – if `cb` was already registered.

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.

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.

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.

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 {CONFIG_BT_RPA_TIMEOUT} is used as the RPA timeout.

This symbol is linkable if {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 {CONFIG_BT_PRIVACY} is enabled this will result in generating new Resolvable Private Address (RPA) that is valid for {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 {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 {CONFIG_BT_PRIVACY} is enabled this will result in generating new Resolvable Private Address (RPA) that is valid for {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.

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. May be NULL if callback registration through *bt_br_discovery_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_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.

void **bt_br_discovery_cb_register**(struct *bt_br_discovery_cb* *cb)

Register discovery packet callbacks.

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

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

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

void **bt_br_discovery_cb_unregister**(struct *bt_br_discovery_cb* *cb)

Unregister discovery packet callbacks.

Remove the callback structure from the list of discovery callbacks.

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

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.

int **bt_le_per_adv_sync_subevent**(struct bt_le_per_adv_sync *per_adv_sync, struct *bt_le_per_adv_sync_subevent_params* *params)

Synchronize with a subset of subevents.

Until this command is issued, the subevent(s) the controller is synchronized to is unspecified.

Parameters

- **per_adv_sync** – The periodic advertising sync object.
- **params** – Parameters.

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

int **bt_le_per_adv_set_response_data**(struct bt_le_per_adv_sync *per_adv_sync, const struct *bt_le_per_adv_response_params* *params, const struct net_buf_simple *data)

Set the data for a response slot in a specific subevent of the PAwR.

This function is called by the application to set the response data. The data for a response slot shall be transmitted only once.

Parameters

- **per_adv_sync** – The periodic advertising sync object.
- **params** – Parameters.
- **data** – The response data to send.

Returns
Zero on success or (negative) error code otherwise.

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_per_adv_data_request**

#include <bluetooth.h>

Public Members

uint8_t **start**

The first subevent data can be set for

uint8_t **count**

The number of subevents data can be set for

struct **bt_le_per_adv_response_info**

#include <bluetooth.h>

Public Members

uint8_t **subevent**

The subevent the response was received in

uint8_t **tx_status**

Status of the subevent indication.

0 if subevent indication was transmitted. 1 if subevent indication was not transmitted. All other values RFU.

int8_t **tx_power**

The TX power of the response in dBm

int8_t **rssi**

The RSSI of the response in dBm

uint8_t **cte_type**

The Constant Tone Extension (CTE) of the advertisement (bt_df_cte_type)

uint8_t **response_slot**

The slot the response was received in

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 {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 {CONFIG_BT_PRIVACY} is enabled then the timeout must be less than {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).

const *bt_addr_le_t* ***addr**

Current local advertising address used.

struct **bt_le_per_adv_subevent_data_params**

#include <bluetooth.h>

Public Members

uint8_t **subevent**

The subevent to set data for

uint8_t **response_slot_start**

The first response slot to listen to

uint8_t **response_slot_count**

The number of response slots to listen to

const struct net_buf_simple ***data**

The data to send

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. NULL if the controller failed to receive a subevent indication. Only happens if {CONFIG_BT_PER_ADV_SYNC_RSP} is enabled.

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 (N * 1.25 ms).

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_discovery_cb**

#include <*bluetooth.h*>

Public Members

void (*recv)(const struct *bt_br_discovery_result* *result)

Advertisement packet and scan response received callback.

Param result
Storage used for discovery results

void (*timeout)(const struct *bt_br_discovery_result* *results, size_t count)

The inquiry has stopped after discovery timeout.

Param results
Storage used for discovery results

Param count
Number of valid discovery results.

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.

struct **bt_le_per_adv_sync_subevent_params**

#include <bluetooth.h>

Public Members

uint16_t **properties**

Periodic Advertising Properties.

Bit 6 is include TxPower, all others RFU.

uint8_t **num_subevents**

Number of subevents to sync to

uint8_t *subevents

The subevent(s) to synchronize with.

The array must have *num_subevents* elements.

struct bt_le_per_adv_response_params

#include <bluetooth.h>

Public Members

uint16_t request_event

The periodic event counter of the request the response is sent to.

bt_le_per_adv_sync_recv_info

> **Note**
>
> The response can be sent up to one periodic interval after the request was received.

uint8_t request_subevent

The subevent counter of the request the response is sent to.

bt_le_per_adv_sync_recv_info

uint8_t response_subevent

The subevent the response shall be sent in

uint8_t response_slot

The response slot the response shall be sent in

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 bool **bt_addr_eq**(const *bt_addr_t* *a, const *bt_addr_t* *b)

Determine equality of two Bluetooth device addresses.

Return values

- **#true** – if the two addresses are equal
- **#false** – otherwise

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

Compare Bluetooth LE device addresses.

See also

bt_addr_le_eq

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 bool **bt_addr_le_eq**(const *bt_addr_le_t* *a, const *bt_addr_le_t* *b)

Determine equality of two Bluetooth LE device addresses.

The Bluetooth LE addresses are equal if and only if both the types and the 48-bit addresses are numerically equal.

Return values

- **#true** – if the two addresses are equal
- **#false** – otherwise

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.
- **len** – 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

Return values
0 – Success. The parsed address is stored in addr.

Returns
-EINVAL Invalid address string. str is not a well-formed Bluetooth address.

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.

Company Identifiers (see Bluetooth Assigned Numbers)

BT_COMP_ID_LF
The Linux Foundation

EIR/AD data type definitions

BT_DATA_FLAGS
AD flags

BT_DATA_UUID16_SOME
16-bit UUID, more available

BT_DATA_UUID16_ALL
16-bit UUID, all listed

BT_DATA_UUID32_SOME
32-bit UUID, more available

BT_DATA_UUID32_ALL
32-bit UUID, all listed

BT_DATA_UUID128_SOME
128-bit UUID, more available

BT_DATA_UUID128_ALL
128-bit UUID, all listed

BT_DATA_NAME_SHORTENED
Shortened name

BT_DATA_NAME_COMPLETE
Complete name

BT_DATA_TX_POWER
Tx Power

BT_DATA_SM_TK_VALUE
Security Manager TK Value

BT_DATA_SM_OOB_FLAGS
Security Manager OOB Flags

BT_DATA_PERIPHERAL_INT_RANGE
Peripheral Connection Interval Range

BT_DATA_SOLICIT16
Solicit UUIDs, 16-bit

BT_DATA_SOLICIT128
Solicit UUIDs, 128-bit

BT_DATA_SVC_DATA16
Service data, 16-bit UUID

BT_DATA_PUB_TARGET_ADDR
Public Target Address

BT_DATA_RAND_TARGET_ADDR
Random Target Address

BT_DATA_GAP_APPEARANCE
GAP appearance

BT_DATA_ADV_INT
Advertising Interval

BT_DATA_LE_BT_DEVICE_ADDRESS
LE Bluetooth Device Address

BT_DATA_LE_ROLE
LE Role

BT_DATA_SIMPLE_PAIRING_HASH
Simple Pairing Hash C256

BT_DATA_SIMPLE_PAIRING_RAND
Simple Pairing Randomizer R256

BT_DATA_SOLICIT32
Solicit UUIDs, 32-bit

BT_DATA_SVC_DATA32
Service data, 32-bit UUID

BT_DATA_SVC_DATA128
Service data, 128-bit UUID

BT_DATA_LE_SC_CONFIRM_VALUE
LE SC Confirmation Value

BT_DATA_LE_SC_RANDOM_VALUE

LE SC Random Value

BT_DATA_URI

URI

BT_DATA_INDOOR_POS

Indoor Positioning

BT_DATA_TRANS_DISCOVER_DATA

Transport Discovery Data

BT_DATA_LE_SUPPORTED_FEATURES

LE Supported Features

BT_DATA_CHANNEL_MAP_UPDATE_IND

Channel Map Update Indication

BT_DATA_MESH_PROV

Mesh Provisioning PDU

BT_DATA_MESH_MESSAGE

Mesh Networking PDU

BT_DATA_MESH_BEACON

Mesh Beacon

BT_DATA_BIG_INFO

BIGInfo

BT_DATA_BROADCAST_CODE

Broadcast Code

BT_DATA_CSIS_RSI

CSIS Random Set ID type

BT_DATA_ADV_INT_LONG

Advertising Interval long

BT_DATA_BROADCAST_NAME

Broadcast Name

BT_DATA_ENCRYPTED_AD_DATA

Encrypted Advertising Data

BT_DATA_3D_INFO

3D Information Data

BT_DATA_MANUFACTURER_DATA

Manufacturer Specific Data

BT_LE_AD_LIMITED

Limited Discoverable

BT_LE_AD_GENERAL

General Discoverable

BT_LE_AD_NO_BREDR

BR/EDR not supported

Appearance Values

Last Modified on 2023-01-05

BT_APPEARANCE_UNKNOWN

Generic Unknown

BT_APPEARANCE_GENERIC_PHONE

Generic Phone

BT_APPEARANCE_GENERIC_COMPUTER

Generic Computer

BT_APPEARANCE_COMPUTER_DESKTOP_WORKSTATION

Desktop Workstation

BT_APPEARANCE_COMPUTER_SERVER_CLASS

Server-class Computer

BT_APPEARANCE_COMPUTER_LAPTOP

Laptop

BT_APPEARANCE_COMPUTER_HANDHELD_PCPDA

Handheld PC/PDA (clamshell)

BT_APPEARANCE_COMPUTER_PALMSIZE_PCPDA

Palm-size PC/PDA

BT_APPEARANCE_COMPUTER_WEARABLE_COMPUTER

Wearable computer (watch size)

BT_APPEARANCE_COMPUTER_TABLET

Tablet

BT_APPEARANCE_COMPUTER_DOCKING_STATION

Docking Station

BT_APPEARANCE_COMPUTER_ALL_IN_ONE

All in One

BT_APPEARANCE_COMPUTER_BLADE_SERVER

Blade Server

BT_APPEARANCE_COMPUTER_CONVERTIBLE

Convertible

BT_APPEARANCE_COMPUTER_DETACHABLE

Detachable

BT_APPEARANCE_COMPUTER_IOT_GATEWAY

IoT Gateway

BT_APPEARANCE_COMPUTER_MINI_PC

Mini PC

BT_APPEARANCE_COMPUTER_STICK_PC

Stick PC

BT_APPEARANCE_GENERIC_WATCH

Generic Watch

BT_APPEARANCE_SPORTS_WATCH

Sports Watch

BT_APPEARANCE_SMARTWATCH

Smartwatch

BT_APPEARANCE_GENERIC_CLOCK

Generic Clock

BT_APPEARANCE_GENERIC_DISPLAY

Generic Display

BT_APPEARANCE_GENERIC_REMOTE

Generic Remote Control

BT_APPEARANCE_GENERIC_EYEGLASSES

Generic Eye-glasses

BT_APPEARANCE_GENERIC_TAG

Generic Tag

BT_APPEARANCE_GENERIC_KEYRING

Generic Keyring

BT_APPEARANCE_GENERIC_MEDIA_PLAYER

Generic Media Player

BT_APPEARANCE_GENERIC_BARCODE_SCANNER

Generic Barcode Scanner

BT_APPEARANCE_GENERIC_THERMOMETER

Generic Thermometer

BT_APPEARANCE_THERMOMETER_EAR

Ear Thermometer

BT_APPEARANCE_GENERIC_HEART_RATE

Generic Heart Rate Sensor

BT_APPEARANCE_HEART_RATE_BELT

Heart Rate Belt

BT_APPEARANCE_GENERIC_BLOOD_PRESSURE

Generic Blood Pressure

BT_APPEARANCE_BLOOD_PRESSURE_ARM

Arm Blood Pressure

BT_APPEARANCE_BLOOD_PRESSURE_WRIST

Wrist Blood Pressure

BT_APPEARANCE_GENERIC_HID

Generic Human Interface Device

BT_APPEARANCE_HID_KEYBOARD

Keyboard

BT_APPEARANCE_HID_MOUSE

Mouse

BT_APPEARANCE_HID_JOYSTICK

Joystick

BT_APPEARANCE_HID_GAMEPAD

Gamepad

BT_APPEARANCE_HID_DIGITIZER_TABLET

Digitizer Tablet

BT_APPEARANCE_HID_CARD_READER

Card Reader

BT_APPEARANCE_HID_DIGITAL_PEN

Digital Pen

BT_APPEARANCE_HID_BARCODE_SCANNER

Barcode Scanner

BT_APPEARANCE_HID_TOUCHPAD

Touchpad

BT_APPEARANCE_HID_PRESENTATION_REMOTE

Presentation Remote

BT_APPEARANCE_GENERIC_GLUCOSE

Generic Glucose Meter

BT_APPEARANCE_GENERIC_WALKING

Generic Running Walking Sensor

BT_APPEARANCE_WALKING_IN_SHOE

In-Shoe Running Walking Sensor

BT_APPEARANCE_WALKING_ON_SHOE

On-Shoe Running Walking Sensor

BT_APPEARANCE_WALKING_ON_HIP

On-Hip Running Walking Sensor

BT_APPEARANCE_GENERIC_CYCLING

Generic Cycling

BT_APPEARANCE_CYCLING_COMPUTER

Cycling Computer

BT_APPEARANCE_CYCLING_SPEED

Speed Sensor

BT_APPEARANCE_CYCLING_CADENCE

Cadence Sensor

BT_APPEARANCE_CYCLING_POWER

Power Sensor

BT_APPEARANCE_CYCLING_SPEED_CADENCE

Speed and Cadence Sensor

BT_APPEARANCE_GENERIC_CONTROL_DEVICE

Generic Control Device

BT_APPEARANCE_CONTROL_SWITCH

Switch

BT_APPEARANCE_CONTROL_MULTI_SWITCH

Multi-switch

BT_APPEARANCE_CONTROL_BUTTON

Button

BT_APPEARANCE_CONTROL_SLIDER

Slider

BT_APPEARANCE_CONTROL_ROTARY_SWITCH

Rotary Switch

BT_APPEARANCE_CONTROL_TOUCH_PANEL

Touch Panel

BT_APPEARANCE_CONTROL_SINGLE_SWITCH

Single Switch

BT_APPEARANCE_CONTROL_DOUBLE_SWITCH

Double Switch

BT_APPEARANCE_CONTROL_TRIPLE_SWITCH

Triple Switch

BT_APPEARANCE_CONTROL_BATTERY_SWITCH

Battery Switch

BT_APPEARANCE_CONTROL_ENERGY_HARVESTING_SWITCH

Energy Harvesting Switch

BT_APPEARANCE_CONTROL_PUSH_BUTTON

Push Button

BT_APPEARANCE_GENERIC_NETWORK_DEVICE

Generic Network Device

BT_APPEARANCE_NETWORK_ACCESS_POINT

Access Point

BT_APPEARANCE_NETWORK_MESH_DEVICE

Mesh Device

BT_APPEARANCE_NETWORK_MESH_PROXY

Mesh Network Proxy

BT_APPEARANCE_GENERIC_SENSOR

Generic Sensor

BT_APPEARANCE_SENSOR_MOTION

Motion Sensor

BT_APPEARANCE_SENSOR_AIR_QUALITY

Air quality Sensor

BT_APPEARANCE_SENSOR_TEMPERATURE

Temperature Sensor

BT_APPEARANCE_SENSOR_HUMIDITY

Humidity Sensor

BT_APPEARANCE_SENSOR_LEAK

Leak Sensor

BT_APPEARANCE_SENSOR_SMOKE

Smoke Sensor

BT_APPEARANCE_SENSOR_OCCUPANCY

Occupancy Sensor

BT_APPEARANCE_SENSOR_CONTACT

Contact Sensor

BT_APPEARANCE_SENSOR_CARBON_MONOXIDE

Carbon Monoxide Sensor

BT_APPEARANCE_SENSOR_CARBON_DIOXIDE

Carbon Dioxide Sensor

BT_APPEARANCE_SENSOR_AMBIENT_LIGHT

Ambient Light Sensor

BT_APPEARANCE_SENSOR_ENERGY

Energy Sensor

BT_APPEARANCE_SENSOR_COLOR_LIGHT

Color Light Sensor

BT_APPEARANCE_SENSOR_RAIN

Rain Sensor

BT_APPEARANCE_SENSOR_FIRE

Fire Sensor

BT_APPEARANCE_SENSOR_WIND

Wind Sensor

BT_APPEARANCE_SENSOR_PROXIMITY

Proximity Sensor

BT_APPEARANCE_SENSOR_MULTI

Multi-Sensor

BT_APPEARANCE_SENSOR_FLUSH_MOUNTED

Flush Mounted Sensor

BT_APPEARANCE_SENSOR_CEILING_MOUNTED

Ceiling Mounted Sensor

BT_APPEARANCE_SENSOR_WALL_MOUNTED

Wall Mounted Sensor

BT_APPEARANCE_MULTISENSOR

Multisensor

BT_APPEARANCE_SENSOR_ENERGY_METER

Energy Meter

BT_APPEARANCE_SENSOR_FLAME_DETECTOR

Flame Detector

BT_APPEARANCE_SENSOR_VEHICLE_TIRE_PRESSURE

Vehicle Tire Pressure Sensor

BT_APPEARANCE_GENERIC_LIGHT_FIXTURES

Generic Light Fixtures

BT_APPEARANCE_LIGHT_FIXTURES_WALL

Wall Light

BT_APPEARANCE_LIGHT_FIXTURES_CEILING

Ceiling Light

BT_APPEARANCE_LIGHT_FIXTURES_FLOOR

Floor Light

BT_APPEARANCE_LIGHT_FIXTURES_CABINET

Cabinet Light

BT_APPEARANCE_LIGHT_FIXTURES_DESK

Desk Light

BT_APPEARANCE_LIGHT_FIXTURES_TROFFER

Troffer Light

BT_APPEARANCE_LIGHT_FIXTURES_PENDANT

Pendant Light

BT_APPEARANCE_LIGHT_FIXTURES_IN_GROUND

In-ground Light

BT_APPEARANCE_LIGHT_FIXTURES_FLOOD

Flood Light

BT_APPEARANCE_LIGHT_FIXTURES_UNDERWATER

Underwater Light

BT_APPEARANCE_LIGHT_FIXTURES_BOLLARD_WITH

Bollard with Light

BT_APPEARANCE_LIGHT_FIXTURES_PATHWAY

Pathway Light

BT_APPEARANCE_LIGHT_FIXTURES_GARDEN

Garden Light

BT_APPEARANCE_LIGHT_FIXTURES_POLE_TOP

Pole-top Light

BT_APPEARANCE_SPOT_LIGHT

Spotlight

BT_APPEARANCE_LIGHT_FIXTURES_LINEAR

Linear Light

BT_APPEARANCE_LIGHT_FIXTURES_STREET

Street Light

BT_APPEARANCE_LIGHT_FIXTURES_SHELVES

Shelves Light

BT_APPEARANCE_LIGHT_FIXTURES_BAY

Bay Light

BT_APPEARANCE_LIGHT_FIXTURES_EMERGENCY_EXIT

Emergency Exit Light

BT_APPEARANCE_LIGHT_FIXTURES_CONTROLLER

Light Controller

BT_APPEARANCE_LIGHT_FIXTURES_DRIVER

Light Driver

BT_APPEARANCE_LIGHT_FIXTURES_BULB

Bulb

BT_APPEARANCE_LIGHT_FIXTURES_LOW_BAY

Low-bay Light

BT_APPEARANCE_LIGHT_FIXTURES_HIGH_BAY

High-bay Light

BT_APPEARANCE_GENERIC_FAN

Generic Fan

BT_APPEARANCE_FAN_CEILING

Ceiling Fan

BT_APPEARANCE_FAN_AXIAL

Axial Fan

BT_APPEARANCE_FAN_EXHAUST

Exhaust Fan

BT_APPEARANCE_FAN_PEDESTAL

Pedestal Fan

BT_APPEARANCE_FAN_DESK

Desk Fan

BT_APPEARANCE_FAN_WALL

Wall Fan

BT_APPEARANCE_GENERIC_HVAC

Generic HVAC

BT_APPEARANCE_HVAC_THERMOSTAT

Thermostat

BT_APPEARANCE_HVAC_HUMIDIFIER

Humidifier

BT_APPEARANCE_HVAC_DEHUMIDIFIER

De-humidifier

BT_APPEARANCE_HVAC_HEATER

Heater

BT_APPEARANCE_HVAC_RADIATOR

Radiator

BT_APPEARANCE_HVAC_BOILER

Boiler

BT_APPEARANCE_HVAC_HEAT_PUMP

Heat Pump

BT_APPEARANCE_HVAC_INFRARED_HEATER

Infrared Heater

BT_APPEARANCE_HVAC_RADIANT_PANEL_HEATER

Radiant Panel Heater

BT_APPEARANCE_HVAC_FAN_HEATER

Fan Heater

BT_APPEARANCE_HVAC_AIR_CURTAIN

Air Curtain

BT_APPEARANCE_GENERIC_AIR_CONDITIONING

Generic Air Conditioning

BT_APPEARANCE_GENERIC_HUMIDIFIER

Generic Humidifier

BT_APPEARANCE_GENERIC_HEATING

Generic Heating

BT_APPEARANCE_HEATING_RADIATOR

Radiator

BT_APPEARANCE_HEATING_BOILER

Boiler

BT_APPEARANCE_HEATING_HEAT_PUMP

Heat Pump

BT_APPEARANCE_HEATING_INFRARED_HEATER

Infrared Heater

BT_APPEARANCE_HEATING_RADIANT_PANEL_HEATER

Radiant Panel Heater

BT_APPEARANCE_HEATING_FAN_HEATER

Fan Heater

BT_APPEARANCE_HEATING_AIR_CURTAIN

Air Curtain

BT_APPEARANCE_GENERIC_ACCESS_CONTROL

Generic Access Control

BT_APPEARANCE_CONTROL_ACCESS_DOOR

Access Door

BT_APPEARANCE_CONTROL_GARAGE_DOOR

Garage Door

BT_APPEARANCE_CONTROL_EMERGENCY_EXIT_DOOR

Emergency Exit Door

BT_APPEARANCE_CONTROL_ACCESS_LOCK

Access Lock

BT_APPEARANCE_CONTROL_ELEVATOR

Elevator

BT_APPEARANCE_CONTROL_WINDOW

Window

BT_APPEARANCE_CONTROL_ENTRANCE_GATE

Entrance Gate

BT_APPEARANCE_CONTROL_DOOR_LOCK

Door Lock

BT_APPEARANCE_CONTROL_LOCKER

Locker

BT_APPEARANCE_GENERIC_MOTORIZED_DEVICE

Generic Motorized Device

BT_APPEARANCE_MOTORIZED_GATE

Motorized Gate

BT_APPEARANCE_MOTORIZED_AWNING

Awning

BT_APPEARANCE_MOTORIZED_BLINDS_OR_SHADES

Blinds or Shades

BT_APPEARANCE_MOTORIZED_CURTAINS

Curtains

BT_APPEARANCE_MOTORIZED_SCREEN

Screen

BT_APPEARANCE_GENERIC_POWER_DEVICE

Generic Power Device

BT_APPEARANCE_POWER_OUTLET

Power Outlet

BT_APPEARANCE_POWER_STRIP

Power Strip

BT_APPEARANCE_POWER_PLUG

Plug

BT_APPEARANCE_POWER_SUPPLY

Power Supply

BT_APPEARANCE_POWER_LED_DRIVER

LED Driver

BT_APPEARANCE_POWER_FLUORESCENT_LAMP_GEAR

Fluorescent Lamp Gear

BT_APPEARANCE_POWER_HID_LAMP_GEAR

HID Lamp Gear

BT_APPEARANCE_POWER_CHARGE_CASE

Charge Case

BT_APPEARANCE_POWER_POWER_BANK

Power Bank

BT_APPEARANCE_GENERIC_LIGHT_SOURCE

Generic Light Source

BT_APPEARANCE_LIGHT_SOURCE_INCANDESCENT_BULB

Incandescent Light Bulb

BT_APPEARANCE_LIGHT_SOURCE_LED_LAMP

LED Lamp

BT_APPEARANCE_LIGHT_SOURCE_HID_LAMP

HID Lamp

BT_APPEARANCE_LIGHT_SOURCE_FLUORESCENT_LAMP

Fluorescent Lamp

BT_APPEARANCE_LIGHT_SOURCE_LED_ARRAY

LED Array

BT_APPEARANCE_LIGHT_SOURCE_MULTICOLOR_LED_ARRAY

Multi-Color LED Array

BT_APPEARANCE_LIGHT_SOURCE_LOW_VOLTAGE_HALOGEN

Low voltage halogen

BT_APPEARANCE_LIGHT_SOURCE_OLED

Organic light emitting diode

BT_APPEARANCE_GENERIC_WINDOW_COVERING

Generic Window Covering

BT_APPEARANCE_WINDOW_SHADES

Window Shades

BT_APPEARANCE_WINDOW_BLINDS

Window Blinds

BT_APPEARANCE_WINDOW_AWNING

Window Awning

BT_APPEARANCE_WINDOW_CURTAIN

Window Curtain

BT_APPEARANCE_WINDOW_EXTERIOR_SHUTTER

Exterior Shutter

BT_APPEARANCE_WINDOW_EXTERIOR_SCREEN

Exterior Screen

BT_APPEARANCE_GENERIC_AUDIO_SINK

Generic Audio Sink

BT_APPEARANCE_AUDIO_SINK_STANDALONE_SPEAKER

Standalone Speaker

BT_APPEARANCE_AUDIO_SINK_SOUNDBAR

Soundbar

BT_APPEARANCE_AUDIO_SINK_BOOKSHELF_SPEAKER

Bookshelf Speaker

BT_APPEARANCE_AUDIO_SINK_STANDMOUNTED_SPEAKER

Standmounted Speaker

BT_APPEARANCE_AUDIO_SINK_SPEAKERPHONE

Speakerphone

BT_APPEARANCE_GENERIC_AUDIO_SOURCE

Generic Audio Source

BT_APPEARANCE_AUDIO_SOURCE_MICROPHONE

Microphone

BT_APPEARANCE_AUDIO_SOURCE_ALARM

Alarm

BT_APPEARANCE_AUDIO_SOURCE_BELL

Bell

BT_APPEARANCE_AUDIO_SOURCE_HORN

Horn

BT_APPEARANCE_AUDIO_SOURCE_BROADCASTING_DEVICE

Broadcasting Device

BT_APPEARANCE_AUDIO_SOURCE_SERVICE_DESK

Service Desk

BT_APPEARANCE_AUDIO_SOURCE_KIOSK

Kiosk

BT_APPEARANCE_AUDIO_SOURCE_BROADCASTING_ROOM

Broadcasting Room

BT_APPEARANCE_AUDIO_SOURCE_AUDITORIUM

Auditorium

BT_APPEARANCE_GENERIC_MOTORIZED_VEHICLE

Generic Motorized Vehicle

BT_APPEARANCE_VEHICLE_CAR

Car

BT_APPEARANCE_VEHICLE_LARGE_GOODS

Large Goods Vehicle

BT_APPEARANCE_VEHICLE_TWO_WHEELED

2-Wheeled Vehicle

BT_APPEARANCE_VEHICLE_MOTORBIKE

Motorbike

BT_APPEARANCE_VEHICLE_SCOOTER

Scooter

BT_APPEARANCE_VEHICLE_MOPED

Moped

BT_APPEARANCE_VEHICLE_THREE_WHEELED

3-Wheeled Vehicle

BT_APPEARANCE_VEHICLE_LIGHT

Light Vehicle

BT_APPEARANCE_VEHICLE_QUAD_BIKE

Quad Bike

BT_APPEARANCE_VEHICLE_MINIBUS

Minibus

BT_APPEARANCE_VEHICLE_BUS

Bus

BT_APPEARANCE_VEHICLE_TROLLEY

Trolley

BT_APPEARANCE_VEHICLE_AGRICULTURAL

Agricultural Vehicle

BT_APPEARANCE_VEHICLE_CAMPER_OR_CARAVAN

Camper/Caravan

BT_APPEARANCE_VEHICLE_RECREATIONAL

Recreational Vehicle/Motor Home

BT_APPEARANCE_GENERIC_DOMESTIC_APPLIANCE

Generic Domestic Appliance

BT_APPEARANCE_APPLIANCE_REFRIGERATOR

Refrigerator

BT_APPEARANCE_APPLIANCE_FREEZER

Freezer

BT_APPEARANCE_APPLIANCE_OVEN

Oven

BT_APPEARANCE_APPLIANCE_MICROWAVE

Microwave

BT_APPEARANCE_APPLIANCE_TOASTER

Toaster

BT_APPEARANCE_APPLIANCE_WASHING_MACHINE

Washing Machine

BT_APPEARANCE_APPLIANCE_DRYER

Dryer

BT_APPEARANCE_APPLIANCE_COFFEE_MAKER

Coffee maker

BT_APPEARANCE_APPLIANCE_CLOTHES_IRON

Clothes iron

BT_APPEARANCE_APPLIANCE_CURLING_IRON

Curling iron

BT_APPEARANCE_APPLIANCE_HAIR_DRYER

Hair dryer

BT_APPEARANCE_APPLIANCE_VACUUM_CLEANER

Vacuum cleaner

BT_APPEARANCE_APPLIANCE_ROBOTIC_VACUUM_CLEANER

Robotic vacuum cleaner

BT_APPEARANCE_APPLIANCE_RICE_COOKER

Rice cooker

BT_APPEARANCE_APPLIANCE_CLOTHES_STEAMER

Clothes steamer

BT_APPEARANCE_GENERIC_WEARABLE_AUDIO_DEVICE

Generic Wearable Audio Device

BT_APPEARANCE_WEARABLE_AUDIO_DEVICE_EARBUD

Earbud

BT_APPEARANCE_WEARABLE_AUDIO_DEVICE_HEADSET

Headset

BT_APPEARANCE_WEARABLE_AUDIO_DEVICE_HEADPHONES

Headphones

BT_APPEARANCE_WEARABLE_AUDIO_DEVICE_NECK_BAND

Neck Band

BT_APPEARANCE_GENERIC_AIRCRAFT

Generic Aircraft

BT_APPEARANCE_AIRCRAFT_LIGHT

Light Aircraft

BT_APPEARANCE_AIRCRAFT_MICROLIGHT

Microlight

BT_APPEARANCE_AIRCRAFT_PARAGLIDER

Paraglider

BT_APPEARANCE_AIRCRAFT_LARGE_PASSENGER

Large Passenger Aircraft

BT_APPEARANCE_GENERIC_AV_EQUIPMENT

Generic AV Equipment

BT_APPEARANCE_AV_EQUIPMENT_AMPLIFIER

Amplifier

BT_APPEARANCE_AV_EQUIPMENT_RECEIVER

Receiver

BT_APPEARANCE_AV_EQUIPMENT_RADIO

Radio

BT_APPEARANCE_AV_EQUIPMENT_TUNER

Tuner

BT_APPEARANCE_AV_EQUIPMENT_TURNTABLE

Turntable

BT_APPEARANCE_AV_EQUIPMENT_CD_PLAYER

CD Player

BT_APPEARANCE_AV_EQUIPMENT_DVD_PLAYER

DVD Player

BT_APPEARANCE_AV_EQUIPMENT_BLURAY_PLAYER

Bluray Player

BT_APPEARANCE_AV_EQUIPMENT_OPTICAL_DISC_PLAYER

Optical Disc Player

BT_APPEARANCE_AV_EQUIPMENT_SET_TOP_BOX

Set-Top Box

BT_APPEARANCE_GENERIC_DISPLAY_EQUIPMENT

Generic Display Equipment

BT_APPEARANCE_DISPLAY_EQUIPMENT_TELEVISION

Television

BT_APPEARANCE_DISPLAY_EQUIPMENT_MONITOR

Monitor

BT_APPEARANCE_DISPLAY_EQUIPMENT_PROJECTOR

Projector

BT_APPEARANCE_GENERIC_HEARING_AID

Generic Hearing aid

BT_APPEARANCE_HEARING_AID_IN_EAR

In-ear hearing aid

BT_APPEARANCE_HEARING_AID_BEHIND_EAR

Behind-ear hearing aid

BT_APPEARANCE_HEARING_AID_COCHLEAR_IMPLANT

Cochlear Implant

BT_APPEARANCE_GENERIC_GAMING

Generic Gaming

BT_APPEARANCE_HOME_VIDEO_GAME_CONSOLE

Home Video Game Console

BT_APPEARANCE_PORTABLE_HANDHELD_CONSOLE

Portable handheld console

BT_APPEARANCE_GENERIC_SIGNAGE

Generic Signage

BT_APPEARANCE_SIGNAGE_DIGITAL

Digital Signage

BT_APPEARANCE_SIGNAGE_ELECTRONIC_LABEL

Electronic Label

BT_APPEARANCE_GENERIC_PULSE_OXIMETER

Generic Pulse Oximeter

BT_APPEARANCE_PULSE_OXIMETER_FINGERTIP

Fingertip Pulse Oximeter

BT_APPEARANCE_PULSE_OXIMETER_WRIST

Wrist Worn Pulse Oximeter

BT_APPEARANCE_GENERIC_WEIGHT_SCALE

Generic Weight Scale

BT_APPEARANCE_GENERIC_PERSONAL_MOBILITY_DEVICE

Generic Personal Mobility Device

BT_APPEARANCE_MOBILITY_POWERED_WHEELCHAIR

Powered Wheelchair

BT_APPEARANCE_MOBILITY_SCOOTER

Mobility Scooter

BT_APPEARANCE_CONTINUOUS_GLUCOSE_MONITOR

Continuous Glucose Monitor

BT_APPEARANCE_GENERIC_INSULIN_PUMP

Generic Insulin Pump

BT_APPEARANCE_INSULIN_PUMP_DURABLE

Insulin Pump, durable pump

BT_APPEARANCE_INSULIN_PUMP_PATCH

Insulin Pump, patch pump

BT_APPEARANCE_INSULIN_PEN

Insulin Pen

BT_APPEARANCE_GENERIC_MEDICATION_DELIVERY

Generic Medication Delivery

BT_APPEARANCE_GENERIC_SPIROMETER

Generic Spirometer

BT_APPEARANCE_SPIROMETER_HANDHELD

Handheld Spirometer

BT_APPEARANCE_GENERIC_OUTDOOR_SPORTS

Generic Outdoor Sports Activity

BT_APPEARANCE_OUTDOOR_SPORTS_LOCATION

Location Display

BT_APPEARANCE_OUTDOOR_SPORTS_LOCATION_AND_NAV

Location and Navigation Display

BT_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD

Location Pod

BT_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD_AND_NAV

Location and Navigation Pod

Defined GAP timers

BT_GAP_SCAN_FAST_INTERVAL_MIN

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

Defines

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

Default data length

BT_GAP_DATA_LEN_MAX

Maximum data length

BT_GAP_DATA_TIME_DEFAULT

Default data time

BT_GAP_DATA_TIME_MAX

Maximum data time

BT_GAP_SID_MAX

Maximum advertising set number

BT_GAP_PER_ADV_MAX_SKIP

Maximum number of consecutive periodic advertisement events that can be skipped after a successful receive.

BT_GAP_PER_ADV_MIN_TIMEOUT

Minimum Periodic Advertising Timeout (N * 10 ms)

BT_GAP_PER_ADV_MAX_TIMEOUT

Maximum Periodic Advertising Timeout (N * 10 ms)

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.

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_
↪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_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_
↪ENCODE(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.

enum **[anonymous]**

Advertising PDU properties

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**

Unknown

enumerator **BT_GAP_SCA_251_500**

251 ppm to 500 ppm

enumerator **BT_GAP_SCA_151_250**

151 ppm to 250 ppm

enumerator **BT_GAP_SCA_101_150**

101 ppm to 150 ppm

enumerator **BT_GAP_SCA_76_100**

76 ppm to 100 ppm

enumerator **BT_GAP_SCA_51_75**

51 ppm to 75 ppm

enumerator **BT_GAP_SCA_31_50**

31 ppm to 50 ppm

enumerator **BT_GAP_SCA_21_30**

21 ppm to 30 ppm

enumerator **BT_GAP_SCA_0_20**

0 ppm to 20 ppm

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 *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_read_func_t **read**

Attribute read callback

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_authorization_cb**

#include <gatt.h> GATT authorization callback structure.

Public Members

bool (***read_authorize**)(struct bt_conn *conn, const struct *bt_gatt_attr* *attr)

Authorize the GATT read operation.

This callback allows the application to authorize the GATT read operation for the attribute that is being read.

Param conn
Connection object.

Param attr
The attribute that is being read.

Retval true
Authorize the operation and allow it to execute.

Retval false
Reject the operation and prevent it from executing.

bool (***write_authorize**)(struct bt_conn *conn, const struct *bt_gatt_attr* *attr)

Authorize the GATT write operation.

This callback allows the application to authorize the GATT write operation for the attribute that is being written.

Param conn
Connection object.

Param attr
The attribute that is being written.

Retval true
Authorize the operation and allow it to execute.

Retval false
Reject the operation and prevent it from executing.

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*).
- **_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.

`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

static inline const char *bt_gatt_err_to_str(int gatt_err)

Converts a GATT error to string.

The GATT errors are created with *BT_GATT_ERR*.

The error codes are described in the Bluetooth Core specification, Vol 3, Part F, Section 3.4.1.1.

The ATT and GATT documentation found in Vol 4, Part F and Part G describe when the different error codes are used.

See also the defined BT_ATT_ERR_* macros.

Returns
The string representation of the GATT error code. If {CONFIG_BT_ATT_ERR_TO_STR} is not enabled, this just returns the empty string.

void bt_gatt_cb_register(struct *bt_gatt_cb* *cb)

Register GATT callbacks.

Register callbacks to monitor the state of GATT. The callback struct must remain valid for the remainder of the program.

Parameters

- **cb** – Callback struct.

int **bt_gatt_authorization_cb_register**(const struct *bt_gatt_authorization_cb* *cb)

Register GATT authorization callbacks.

Register callbacks to perform application-specific authorization of GATT operations on all registered GATT attributes. The callback structure must remain valid throughout the entire duration of the Bluetooth subsys activity.

The {CONFIG_BT_GATT_AUTHORIZATION_CUSTOM} Kconfig must be enabled to make this API functional.

This API allows the user to register only one callback structure concurrently. Passing NULL unregisters the previous set of callbacks and makes it possible to register a new one.

Parameters

- **cb** – Callback struct.

Returns

Zero on success or negative error code otherwise

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 {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 {CONFIG_BT_GATT_CACHING} and {CONFIG_BT_SETTINGS} 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.

There are two situations where this function can be called: either before `bt_init()` has been called, or after `settings_load()` has been called. Registering a service in the middle is not supported and will return an error.

Parameters

- **svc** – Service containing the available attributes

Returns

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

-EAGAIN if `bt_init()` has been called but `settings_load()` hasn't yet.

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 {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 UUID.
- **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**

Private reference counter

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.

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.

tabularytabulary

params->type
BT_GATT_DISCOVER_PRIMARY
BT_GATT_DISCOVER_SECONDARY
BT_GATT_DISCOVER_INCLUDE
BT_GATT_DISCOVER_CHARACTERISTIC
BT_GATT_DISCOVER_STD_CHAR_DESC
BT_GATT_DISCOVER_STD_CHAR_DESC
BT_GATT_DISCOVER_STD_CHAR_DESC
BT_GATT_DISCOVER_STD_CHAR_DESC
BT_GATT_DISCOVER_DESCRIPTOR
BT_GATT_DISCOVER_ATTRIBUTE

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.

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.

When reading using by_uuid, `params->start_handle` is the attribute handle for this `data` item.

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.

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.

> **Note**
> The use of this type of discover is not recommended for discovering in ranges across multiple services/characteristics as it may incur in extra round trips.

enumerator **BT_GATT_DISCOVER_ATTRIBUTE**

Discover Attributes.

Discover Attributes of any type.

> **Note**
>
> The use of this type of discover is not recommended for discovering in ranges across multiple services/characteristics as it may incur in more round trips.

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.

> **Note**
>
> Internal use only.

enumerator **BT_GATT_SUBSCRIBE_FLAG_SENT**

Sent flag.

If set, indicates that a subscription request (CCC write) has already been sent in the active connection.

Used to avoid sending subscription requests multiple times when the {CONFIG_BT_GATT_AUTO_RESUBSCRIBE} quirk is enabled.

> **Note**
>
> Internal use only.

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 context specified by 'config BT_RECV_CONTEXT'. `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 {CONFIG_BT_ATT_TX_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 {CONFIG_BT_ATT_TX_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.

To perform a GATT Long Read procedure, start with a Characteristic Value Read (by setting `offset 0` and `handle_count 1`) and then return *BT_GATT_ITER_CONTINUE* from the callback. This is equivalent to calling *bt_gatt_read* again, but with the correct offset to continue the read. This may be repeated until the procedure is complete, which is signaled by the callback being called with `data` set to `NULL`.

Note that returning *BT_GATT_ITER_CONTINUE* is really starting a new ATT operation, so this can fail to allocate resources. However, all API errors are reported as if the server returned BT_ATT_ERR_UNLIKELY. There is no way to distinguish between this condition and a BT_ATT_ERR_UNLIKELY response from the server itself.

Note that the effect of returning *BT_GATT_ITER_CONTINUE* from the callback varies depending on the type of read operation.

The Response comes in callback `params->func`. The callback is run from the context specified by 'config BT_RECV_CONTEXT'. `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.

Parameters

- **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 {CONFIG_BT_ATT_TX_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 context specified by 'config BT_RECV_CONTEXT'. `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 {CONFIG_BT_ATT_TX_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 {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.

Parameters

- **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 {CONFIG_BT_ATT_TX_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 {CONFIG_BT_ATT_TX_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->subscribe`. The callback is run from the context specified by 'config BT_RECV_CONTEXT'. 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 `params` must remain valid while subscribed and cannot be reused for additional subscriptions whilst active.

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 {CONFIG_BT_ATT_TX_COUNT}.
- **-EALREADY** – if there already exist a subscription using the `params`.
- **-EBUSY** – if `params.ccc_handle` is 0 and {CONFIG_BT_GATT_AUTO_DISCOVER_CCC} is enabled and discovery for the `params` is already in progress.

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.

Parameters

- **conn** – Connection object.
- **params** – Subscribe parameters. The parameters shall be a *bt_gatt_subscribe_params* from a previous call to *bt_gatt_subscribe()*.

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 {CONFIG_BT_ATT_TX_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

union **__unnamed__**

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

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.

uint16_t **_att_mtu**

Internal

union **__unnamed__**

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.

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

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.

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

HFP_HF_CMD_CME_ERROR

HFP_HF_CMD_UNKNOWN_ERROR

Typedefs

typedef enum *_hf_ag_volume_type_t* **hf_ag_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 *_hfp_ag_get_config* **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 volume type

typedef enum *_hf_multiparty_call_option_t* **hf_multiparty_call_option_t**

bt hfp ag volume type

typedef struct *_hfp_hf_get_config* **hfp_hf_get_config**

bt hf configure setting

typedef struct *_hf_waiting_call_state_t* **hf_waiting_call_state_t**

typedef struct *_hf_indicator_status_t* **hf_indicator_status_t**

typedef int (***bt_hfp_hf_discover_callback**)(struct bt_conn *conn, uint8_t channel)

hfp_hf discover callback function

Param conn
Pointer to bt_conn structure.

Param channel
the server channel of hfp ag

Enums

enum **_hf_ag_volume_type_t**

bt hfp ag volume type

Values:

enumerator **hf_ag_volume_type_speaker**

enumerator **hf_ag_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 volume type

Values:

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_register_cb**(struct *bt_hfp_ag_cb* *cb)

hfp ag register callback

bt_hfp_ag_connect can register callback too.

Parameters

- **cb** – bt hfp ag callback

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 callback
- **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 funciton

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 funciton

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 featureshfp 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** – suppported 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_ag_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

Parameters

- **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

Parameters

- **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_set_clcc**(struct bt_hfp_ag *hfp_ag, char *call_list)

hfp ag set current call list 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

- **call_list** – point to current call list

Returns
0 in case of success or otherwise in case of error.

int **bt_hfp_ag_send_callheld_indicator**(struct bt_hfp_ag *hfp_ag, uint8_t value)

hfp ag set call hold status indicator to hfp hp

This function is hfp ag set call hold status indicator to hfp hp

Parameters

- **phfp_ag** – pointer to bt hfp ag connection object
- **value** – value of call hold status indicator

Returns
0 in case of success or otherwise in case of error.

int **bt_hfp_ag_get_cind_setting**(struct bt_hfp_ag *hfp_ag, *hfp_ag_cind_t* *cind_setting)

hfp ag get supported indicators for HFP AG

This function is user set supported indicators for HFP AG

Parameters

- **phfp_ag** – pointer to bt hfp ag connection object
- **cind_setting** – point to value of supported indicators

Returns
0 in case of success or otherwise in case of error.

int **bt_hfp_ag_set_cind_setting**(struct bt_hfp_ag *hfp_ag, *hfp_ag_cind_t* *cind_setting)

hfp ag set supported indicators for HFP AG

This function is set supported indicators for HFP AG

Parameters

- **phfp_ag** – pointer to bt hfp ag connection object
- **cind_setting** – point to value of supported indicators

Returns
0 in case of success or otherwise in case of error.

int **bt_hfp_ag_send_clip**(struct bt_hfp_ag *hfp_ag, uint8_t *clip_result)

hfp ag send the standard calling line identification notification unsolicited result code.

This function is send the standard calling line identification notification unsolicited result code to hf

Parameters

- **phfp_ag** – pointer to bt hfp ag connection object
- **clip_result** – point to standard calling line identification result

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 unkown at command response
- **unknow_at_rsplen** – string length of unkown 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_connect**(struct bt_conn *conn, uint8_t channel)

hfp hf 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 hf connection between devices. This function only establish RFCOM connection. After connection success, the callback that is registered by bt_hfp_hf_register is called.

Parameters

- **conn** – Pointer to bt_conn structure.
- **channel** – The refcom service channel.

Returns
0 in case of success or otherwise in case of error.

int **bt_hfp_hf_disconnect**(struct bt_conn *conn)

hfp hf 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 disconnect hfp ag connection between devices. This function only disconnect RFCOM connection. After connection success, the callback that is registered by bt_hfp_hf_register is called.

Parameters

- **conn** – pointer to connection object

Returns
0 in case of success or otherwise in case of error.

int **bt_hfp_hf_discover**(struct bt_conn *conn, *bt_hfp_hf_discover_callback* discoverCallback)

hfp hf 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 find the ag's rfcomm service channel.

Parameters

- **conn** – pointer to bt connection object

- **discoverCallback** – pointer to discover callback function,defined in application

Returns
0 in case of success or otherwise 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.

> **Note**
>
> Please note that send_cmd function and the following HFP APIs may return -EBUSY indicating a failure in sending with busy, requiring the application to retry sending. The possible reasons for this failure could be that an AT command is currently being sent by the application, or it could be caused by the HFP profile internally sending AT commands (such as AT+VGS, AT+BCS, AT+CLIP, AT+CCWA), resulting in the application's AT command being in a busy state.

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.

Parameters

- **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.

int **bt_hfp_hf_trigger_codec_connection**(struct bt_conn *conn)

Trigger codec connection.

Parameters

- **conn** – Connection object.

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

int **bt_hfp_hf_get_peer_indicator_status**(struct bt_conn *conn)

Get peer indicators' status.

Parameters

- **conn** – Connection object.

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

int **bt_hfp_hf_open_audio**(struct bt_conn *conn, uint8_t codec)

hfp hf open audio for codec

This function is to open audio codec for hfp funciton

Parameters

- **conn** – Connection object.
- **codec** – HFP Codec Id.

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

int **bt_hfp_hf_close_audio**(struct bt_conn *sco_conn)

hfp hf close audio for codec

This function is to close audio codec for hfp funciton

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 (***sco_connected**)(struct bt_hfp_ag *ag, struct bt_conn *sco_conn)

HF SCO/eSCO connected Callback

If this callback is provided it will be called whenever the SCO/eSCO connection completes.
Param ag
HFP AG object.
Param sco_conn
SCO/eSCO Connection object.

void (***sco_disconnected**)(struct bt_hfp_ag *ag)

HF SCO/eSCO disconnected Callback

If this callback is provided it will be called whenever the SCO/eSCO connection gets disconnected.
Param ag
HFP AG object.
Param sco_conn
SCO/eSCO Connection object.

void (*`get_config`)(struct bt_hfp_ag *hfp_ag, *hfp_ag_get_config* **config)

Get config after connection.

This callback is used to get config *hfp_ag_get_config* It is same as the second parameter of *bt_hfp_ag_connect*.

Param hfp_ag
bt hfp ag Connection object.

Param config
get the config from upper layer.

void (*`volume_control`)(struct bt_hfp_ag *hfp_ag, *hf_ag_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 (*clcc)(struct bt_hfp_ag *ag)

AG receive Standard list current calls command callback

If this callback is provided it will be called whenever the AT command AT+CLCC is received.

Param hfp_ag
bt hfp ag Connection object.

void (*memory_dial)(struct bt_hfp_ag *ag, uint32_t location)

AG memory dialing request Callback

If this callback is provided it will be called whenever a new call is requested with memory dialing from HFP unit

Param ag
HFP AG object.

Param location
AG memory call location

void (*last_dial)(struct bt_hfp_ag *ag)

AG last dialing request Callback

If this callback is provided it will be called whenever a new call is requested with last dialing from HFP unit side.

Param ag
HFP AG object.

void (*recv_dtmf_codes)(struct bt_hfp_ag *ag, char dtmf_code)

AG received a DTMF Code callback

If this callback is provided it will be called whenever the AT command 'AT+VTS=' is received from HFP unit side.

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 _hfp_hf_get_config

#include <*hfp_hf.h*> bt hf configure setting

struct _hf_waiting_call_state_t

#include <*hfp_hf.h*>

struct _hf_indicator_status_t

#include <*hfp_hf.h*>

struct bt_hfp_hf_cb

#include <*hfp_hf.h*> HFP profile application callback.

Note

Please note that blocking function calls should be not be called the callback function, otherwise, tThese blocking callbacks can potentially cause issues with the entire HFP AT Command state machine, preventing it from continuing to process subsequent AT commands.

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 (*`sco_connected`)(struct bt_conn *conn, struct bt_conn *sco_conn)

HF SCO/eSCO connected Callback

If this callback is provided it will be called whenever the SCO/eSCO connection completes.

Param conn
Connection object.
Param sco_conn
SCO/eSCO Connection object.

void (*`sco_disconnected`)(struct bt_conn *sco_conn, uint8_t reason)

HF SCO/eSCO disconnected Callback

If this callback is provided it will be called whenever the SCO/eSCO connection gets disconnected.

Param conn
Connection object.
Param reason
BT_HCI_ERR_* reason for the disconnection.

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 (*volume_update)(struct bt_conn *conn, *hf_volume_type_t* type, int volume)

HF volume update indication callback to application

If this callback is provided it will be called whenever there is an volume update event.

Param conn
Connection object.

Param type
bt hfp volume type.

Param volume
bt hfp volume level value.

void (*indicator_status)(struct bt_conn *conn, *hf_indicator_status_t* *status)

Indicator status.

This callback is used to notify the peer's indicators' status

Param conn
Connection object.

Param status
structure contains the result of indicator status.

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.

void (*ringtone_inband_set)(struct bt_conn *conn, uint8_t set)

HF notify inband ringtone set status callback to application

If this callback is provided it will be called whenever there is an volume update event.

Param conn
Connection object.

Param set
inband ringtone set status

void (*get_config)(*hfp_hf_get_config* **config)

Get config before connection.

This callback is used to get config *hfp_hf_get_config*

Param conn
Connection object.

Param config
get the config from upper layer.

1.8 Phone Book Access Profile (PBAP)

1.8.1 API Reference

group bt_pbap

Phone Book Access Profile (PBAP)

PBAP OBEX Headers

Constant Definitions for PBAP OBEX Headers

BT_PBAP_HDR_NAME

BT_PBAP_HDR_TYPE

BT_PBAP_HDR_TARGET

BT_PBAP_HDR_BODY

BT_PBAP_HDR_END_OF_BODY

BT_PBAP_HDR_WHO

BT_PBAP_HDR_CONNECTION_ID

BT_PBAP_HDR_APP_PARAM

BT_PBAP_HDR_AUTH_CHALLENGE

BT_PBAP_HDR_AUTH_RESPONSE

BT_PBAP_HDR_SRM

BT_PBAP_HDR_SRMP

PBAP Event Result Types

Constant Definitions for PBAP Event RESULT Types

BT_PBAP_CONTINUE_RSP

BT_PBAP_SUCCESS_RSP

BT_PBAP_BAD_REQ_RSP

BT_PBAP_UNAUTH_RSP

BT_PBAP_FORBIDDEN_RSP

BT_PBAP_NOT_FOUND_RSP

BT_PBAP_NOT_ACCEPTABLE_RSP

BT_PBAP_PRECOND_FAILED_RSP

BT_PBAP_NOT_IMPLEMENTED_RSP

BT_PBAP_NO_SERVICE_RSP

BT_PBAP_TYPE_HDR_PHONEBOOK

BT_PBAP_TYPE_HDR_VCARD_LIST

BT_PBAP_TYPE_HDR_VCARD

PBAP profile supported feature

Constant Definitions for PBAP profile supported featur

BT_PBAP_FEATURE_DOWNLOAD

BT_PBAP_FEATURE_BROWSING

BT_PBAP_FEATURE_DATABASE_IDENTIFIER

BT_PBAP_FEATURE_FOLDER_VERSION_COUNTERS

BT_PBAP_FEATURE_VCARD_SELECTING

BT_PBAP_FEATURE_ENHANCED_MISSED_CALLS

BT_PBAP_FEATURE_X_BT_UCL_VCARD_PROPERTY

BT_PBAP_FEATURE_X_BT_UID_VCARD_PROPERTY

BT_PBAP_FEATURE_CONTACT_REFERENCING

BT_PBAP_FEATURE_DEFAULT_CONTACT_IMAGE_FORMAT

PBAP Application Parameter Tag

Constant Definitions for PBAP Application Parameter Tag ID

BT_PBAP_TAG_ID_ORDER

BT_PBAP_TAG_ID_SEARCH_VALUE

BT_PBAP_TAG_ID_SEARCH_PROPERTY

BT_PBAP_TAG_ID_MAX_LIST_COUNT

BT_PBAP_TAG_ID_LIST_START_OFFSET

BT_PBAP_TAG_ID_PROPERTY_SELECTOR

BT_PBAP_TAG_ID_FORMAT

BT_PBAP_TAG_ID_PHONE_BOOK_SIZE

BT_PBAP_TAG_ID_NEW_MISSED_CALLS

BT_PBAP_TAG_ID_PRIMARY_FOLDER_VERSION

BT_PBAP_TAG_ID_SECONDARY_FOLDER_VERSION

BT_PBAP_TAG_ID_VCARD_SELECTOR

BT_PBAP_TAG_ID_DATABASE_IDENTIFIER

BT_PBAP_TAG_ID_VCARD_SELECTOR_OPERATOR

BT_PBAP_TAG_ID_RESET_NEW_MISSED_CALLS

BT_PBAP_TAG_ID_PBAP_SUPPORTED_FEATURES

BT_PBAP_ADD_PARAMS_ORDER(buf, value)

Add pbap application parameters into net buffer.

> **Note**
>
> Calling these functions should follow a call to net_buf_reserve.

Parameters

- **buf** – net buffer for TX.
- **val** – value to be written into buffer.
- **len** – the length of value.

BT_PBAP_ADD_PARAMS_SEARCH_VALUE(buf, value, len)

BT_PBAP_ADD_PARAMS_SEARCH_PROPERTY(buf, value)

BT_PBAP_ADD_PARAMS_MAX_LIST_COUNT(buf, value)

BT_PBAP_ADD_PARAMS_LIST_START_OFFSET(buf, value)

BT_PBAP_ADD_PARAMS_PROPERTY_SELECTOR(buf, value)

BT_PBAP_ADD_PARAMS_FORMAT(buf, value)

BT_PBAP_ADD_PARAMS_PHONE_BOOK_SIZE(buf, value)

BT_PBAP_ADD_PARAMS_NEW_MISSED_CALLS(buf, value)

BT_PBAP_ADD_PARAMS_PRIMARY_FOLDER_VERSION(buf, value)

BT_PBAP_ADD_PARAMS_SECONDARY_FOLDER_VERSION(buf, value)

BT_PBAP_ADD_PARAMS_VCARD_SELECTOR(buf, value)

BT_PBAP_ADD_PARAMS_DATABASE_IDENTIFIER(buf, value)

BT_PBAP_ADD_PARAMS_VCARD_SELECTOR_OPERATOR(buf, value)

BT_PBAP_ADD_PARAMS_RESET_NEW_MISSED_CALLS(buf, value)

BT_PBAP_ADD_PARAMS_PBAP_SUPPORTED_FEATURES(buf, value)

BT_PBAP_ADD_BODY(buf, val, len)

Add PBAP headers into net buffer.

Parameters

- **buf** – net buffer for TX.
- **val** – value to be written into buffer.
- **len** – the length of value.

BT_PBAP_ADD_END_OF_BODY(buf, val, len)

Defines

BT_PBAP_PCE_RSV_LEN_PULL_PHONEBOOK(pbap_pce, flags)

Reserve length for PullPhoneBook.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the pbap.

Parameters

- **pbap_pce** – PBAP PCE object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_PBAP_PCE_RSV_LEN_SET_PATH(pbap_pce)

Reserve length for SetPath.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the pbap.

Parameters

- **pbap_pce** – PBAP PCE object.

Returns
the reservation length.

BT_PBAP_PCE_RSV_LEN_PULL_VCARD_LISTING(pbap_pce, flags)

Reserve length for PullvCardListing.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the pbap.

Parameters

- **pbap_pce** – PBAP PCE object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_PBAP_PCE_RSV_LEN_PULL_VCARD_ENTRY(pbap_pce, flags)

Reserve length for PullvCardEntry.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the pbap.

Parameters

- **pbap_pce** – PBAP PCE object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

bt_pbap_pce_app_param_parse(buf, func, user_data)

Helper for parsing application parameters.

A helper for parsing the application parameters. The most common scenario is to call this helper on the application parameters received in the callback that was given to bt_pbap_register.

Parameters

- **buf** – net buffer returned in the callback registered by bt_pbap_register.
- **func** – Callback function which will be called for each tag that's found in the buffer. The callback should return true to continue parsing, or false to stop parsing.
- **user_data** – User data to be passed to the callback registered by bt_pbap_register.

bt_pbap_pce_get_body(buf, body, length)

Helper for getting body.

A helper for getting Body header. The most common scenario is to call this helper on the body received in the callback that was given to bt_pbap_register.

Parameters

- **buf** – net buffer returned in the callback registered by bt_pbap_register.
- **body** – pointer used for holding body data.
- **length** – the length of body data.

BT_PBAP_PSE_RSV_LEN_SEND_RESPONSE(pbap_pse)

Reserve length for PULL RESPONSE.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the pbap.

Parameters

- **pbap_pse** – PBAP PSE object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

bt_pbap_pse_app_param_parse(buf, func, user_data)

Helper for parsing application parameters.

A helper for parsing the application parameters. The most common scenario is to call this helper on the application parameters received in the callback that was given to bt_pbap_register.

Parameters

- **buf** – net buffer returned in the callback registered by bt_pbap_register.
- **func** – Callback function which will be called for each tag that's found in the buffer. The callback should return true to continue parsing, or false to stop parsing.
- **user_data** – User data to be passed to the callback registered by bt_pbap_register.

BT_PBAP_FLDR_VER_CNTR_SIZE

BT_PBAP_DATABASE_IDENTIFIER_SIZE

BT_PBAP_SUPPORTED_FEATURES_V11

Functions

int **bt_pbap_pce_register**(struct *bt_pbap_pce_cb* *cb)

PBAP PCE register

This function called to initialize pbap_pce and register callback.

Returns

0 in case of success or otherwise in case of error.

int **bt_pbap_pce_scn_connect**(struct bt_conn *conn, uint8_t channel, struct *bt_pbap_auth* *auth, uint32_t peer_feature, struct bt_pbap_pce **pbap_pce)

pbap pce Connect over rfcomm .

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 pbap connection based on RFCOM channel between devices. After connection success, the callback that is registered by bt_pbap_pce_register is called. If want to initiate authentication to pse, please input

Parameters

- **auth.** – If auth is be provided, its memory cannot be released during the entire connection hold process. If pce has not negotiated authentication information with pse, it is recommended not to initiate authentication. For example, pse is a phone.
- **conn** – Pointer to bt_conn structure.
- **channel** – RFCOM channel number, returned in SDP record
- **auth** – If want to authenticate pse, provide auth, refer to struct *bt_pbap_auth*.
- **peer_feature** – pbap_pse support feature.
- **pbap_pce** – Pointer to pointer of bt_pbap connection object

Returns

0 in case of success or otherwise in case of error.

int **bt_pbap_pce_psm_connect**(struct bt_conn *conn, uint16_t psm, struct *bt_pbap_auth* *auth, uint32_t peer_feature, struct bt_pbap_pce **pbap_pce)

pbap pce Connect over l2cap .

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 pbap pce connection between devices. This function can establish l2cap_psm connection. After connection success, the callback that is registered by bt_pbap_pce_register is called. if auth is be provided, its memory cannot be released during the entire connection hold process. If pce has not negotiated authentication information with pse, it is recommended not to initiate authentication. For example, pse is a phone.

Parameters

- **conn** – Pointer to bt_conn structure.
- **psm** – GoepL2capPsm, returned in SDP record
- **auth** – If want to authenticate pse, provide auth;
- **peer_feature** – pbap_pce support feature
- **pbap_pce** – PBAP PCE object

Returns

0 in case of success or otherwise in case of error.

int **bt_pbap_pce_disconnect**(struct bt_pbap_pce *pbap_pce)

pbap pce 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 disconnect pbap connection between devices. After disconnect, the callback that is registered by bt_pbap_pce_register is called.

Parameters

- **pbap_pce** – PBAP PCE object

Returns

0 in case of success or otherwise in case of error.

int **bt_pbap_pce_pull_phonebook**(struct bt_pbap_pce *pbap_pce, struct net_buf *buf, char *name, bool wait, enum bt_obex_req_flags flag)

pbap pce get phonebook request.

This function is to be called after the pabp conntion is established. This function can be called multiple times untill GET request is send completely. Application uses

Parameters

- **flag** – to send the chunked packet. It's recommended to intelligently allocate tX buffer size to put GET request in the single packet. This function can be called multiple times till the Phonebook object is retrieved completely. The tX net buf needs to use BT_PBAP_PCE_RSV_LEN_PULL_PHONEBOOK to reserve the length after it has been allocated and before it be used. Application can use BT_PBAP_ADD_xxx to add application parameters into tX net buffer. Setting
- **wait** – to true is to ask the pse to wait after sending its next response. After receive responese, the callback that is registered by bt_pbap_pce_register is called.
- **pbap_pce** – PBAP PCE object
- **buf** – tx net_buf
- **name** – Specific phone book path names

- **wait** – true - set SRMP is 1. false - exclude SRMP.
- **flag** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pce_set_phonebook_path**(struct bt_pbap_pce *pbap_pce, struct net_buf *buf, char *name)

pbap pce phone book path set.

This function is to be called after the pabp conn is established. This API is to be used to set sets the current folder in pse. When name is "/", go to parent directory. When name is ".." or "../", go up one level. When name is "child" or "./child", go to child For multilevel jumps, need to do it on a level-by-level basis After receive responese, the callback that is registered by bt_pbap_pce_register is called. The tx net_buf needs to use BT_PBAP_PCE_RSV_LEN_SET_PATH to reserve the length after it has been allocated and before it be used.

Parameters

- **pbap_pce** – PBAP PCE object
- **buf** – tx net_buf
- **name** – path name string.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pce_pull_vcard_listing**(struct bt_pbap_pce *pbap_pce, struct net_buf *buf, char *name, bool wait, enum bt_obex_req_flags flag)

pbap pce get vcardlisting request

This function is to be called after the pabp connction is established. This function can be called multiple times untill GET request is send completely. Application uses

Parameters

- **flag** – to send the chunked packet. It's recommended to intelligently allocate tX buffer size to put GET request in the single packet. This function can be called multiple times till the vcardlisting object is retrieved completely. The tX net buf needs to use BT_PBAP_PCE_RSV_LEN_PULL_VCARD_LISTING to reserve the length after it has been allocated and before it be used. Application can use BT_PBAP_ADD_xxx to add application parameters into tx net buffer. Setting
- **wait** – to true is to ask the pse to wait after sending its next response. After receive responese, the callback that is registered by bt_pbap_pce_register is called.
- **pbap_pce** – pointer to bt pbap pce connection object.
- **buf** – tx net_buf.
- **name** – Specific vcard listing name.
- **wait** – true - set SRMP is 1. false - exclude SRMP.
- **flag** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pce_pull_vcard_entry**(struct bt_pbap_pce *pbap_pce, struct net_buf *buf, char *name, bool wait, enum bt_obex_req_flags flag)

pbap pce get vcard entry request.

This function is to be called after the pabp connection is established. This function can be called multiple times till GET request is send completely. Application uses

Parameters

- **flag** – to send the chunked packet. It's recommended to intelligently allocate tX buffer size to put GET request in the single packet. This function can be called multiple times till the vcard entry object is retrieved completely. The tx net buf needs to use BT_PBAP_PCE_RSV_LEN_PULL_VCARD_ENTRY to reserve the length after it has been allocated and before it be used. Application can use BT_PBAP_ADD_xxx to add application parameters into tx net buffer. Setting
- **wait** – to true is to ask the pse to wait after sending its next response. After receive response, the callback that is registered by bt_pbap_pce_register is called.
- **pbap_pce** – pointer to bt pbap pce connection object
- **buf** – tx net_buf
- **name** – Specific vcard entry name
- **wait** – Reserved for future use. true - the remote device sets SRMP to 1. false - the remote device don't set SRMP.
- **flag** – refer to enum bt_obex_req_flags.

Returns

0 in case of success or otherwise in case of error.

int **bt_pbap_pce_abort**(struct bt_pbap_pce *pbap_pce)

Abort PBAP_PCE operation.

Abort PBAP_PCE GET operation. This cancels the current outstanding operation. The return value of -EINPROGRESS means abort is queued and pending. The current outstanding operation will be aborted when receiving next response from the pse. After abort, the callback that is registered by bt_pbap_pce_register is called.

Parameters

- **mce_mas** – PBAP_PCE object.

Returns

0 in case of success or -EINPROGRESS in case abort is queued or otherwise in case of error.

int **bt_pbap_pce_get_max_pkt_len**(struct bt_pbap_pce *pbap_pce, uint16_t *max_pkt_len)

Get maximum packet length.

This function is to get the maximum packet length in PBAP OBEX connection. This function returns immediately.

Parameters

- **pbap_pce** – pbap object.
- **max_pkt_len** – the return maximum packet length.

Returns

0 in case of success or otherwise in case of error.

int **bt_pbap_pse_register**(struct *bt_pbap_pse_cb* *cb)

PBAP PSE register

This function called to initialize pbap_pse and register callback.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pse_disconnect**(struct bt_pbap_pse *pbap_pse)

Disconnect PBAP connection.

Disconnect PBAP connection.

Parameters

- **pbap_pse** – PBAP PSE object.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pse_pull_phonebook_response**(struct bt_pbap_pse *pbap_pse, uint8_t result, struct net_buf *buf, bool wait)

PBAP PSE send response when receiving pull phonebook request from PCE.

This function called to send response when receiving pull phonebook request from PCE. This function is to be called after receiving pull phonebook request. This function can be called multiple times untill reponse is send completely. Application uses

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **flag** – to send the chunked packet. The tx net buf needs to use BT_PBAP_PCE_RSV_LEN_SEND_RESOPNSE to reserve the length after it has been allocated and before it be used. Application can use BT_PBAP_ADD_xxx to add application parameters and body into tX net buffer. Setting
- **wait** – to true is to ask the PCE to wait after sending its next response.
- **pbap_pse** – PBAP PSE object.
- **result** – response code: BT_PBAP_SUCCESS_RSP or BT_PBAP_CONTINUE_RSP in case of success or otherwise in case of error.
- **buf** – tX net buffer from pbap_pse
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pse_set_phonebook_path_response**(struct bt_pbap_pse *pbap_pse, uint8_t result)

PBAP PSE send response when receiving set phonebook path request from PCE.

This function is to send response when receiving set phonebook path request from PCE. When receiving request, the callback that is registered by bt_pbap_pse_register is called.

Parameters

- **pbap_pse** – PBAP PSE object.
- **result** – response code: BT_PBAP_SUCCESS_RSP or BT_PBAP_CONTINUE_RSP in case of success or otherwise in case of error.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pse_pull_vcard_listing_response**(struct bt_pbap_pse *pbap_pse, uint8_t result, struct net_buf *buf, bool wait)

PBAP PSE send response when receiving pull vcard listing request from PCE.

This function called to send response when receiving pull vcard listing request from PCE. This function is to be called after receiving pull vcard listing request. This function can be called multiple times untill reponse is send completely. Application uses

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **flag** – to send the chunked packet. The tx net buf needs to use BT_PBAP_PCE_RSV_LEN_SEND_RESOPNSE to reserve the length after it has been allocated and before it be used. Application can use BT_PBAP_ADD_xxx to add application parameters and body into tx net buffer. Setting
- **wait** – to true is to ask the PCE to wait after sending its next response.
- **pbap_pse** – PBAP PSE object.
- **result** – response code: BT_PBAP_SUCCESS_RSP or BT_PBAP_CONTINUE_RSP in case of success or otherwise in case of error.
- **buf** – tX net buffer from pbap_pse
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pse_pull_vcard_entry_response**(struct bt_pbap_pse *pbap_pse, uint8_t result, struct net_buf *buf, bool wait)

PBAP PSE send response when receiving pull vcard entry request from PCE.

This function called to send response when receiving pull vcard entry request from PCE. This function is to be called after pull vcard entry request. This function can be called multiple times untill reponse is send completely. Application uses

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **flag** – to send the chunked packet. The tX net buf needs to use BT_PBAP_PCE_RSV_LEN_PULL_RESOPNSE to reserve the length after it has been allocated and before it be used. Application can use BT_PBAP_ADD_xxx to add application parameters and body into tX net buffer. Setting

- **wait** – to true is to ask the PCE to wait after sending its next response.
- **pbap_pse** – PBAP PSE object.
- **result** – response code: BT_PBAP_SUCCESS_RSP or BT_PBAP_CONTINUE_RSP in case of success or otherwise in case of error.
- **buf** – tX net buffer from pbap_pse
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pse_get_max_pkt_len**(struct bt_pbap_pse *pbap_pse, uint16_t *max_pkt_len)

Get maximum packet length.

This function is to get the maximum packet length in PBAP OBEX connection. This function returns immediately.

Parameters

- **pbap_pce** – pbap object.
- **max_pkt_len** – the return maximum packet length.

Returns
0 in case of success or otherwise in case of error.

int **bt_pbap_pse_get_peer_supported_features**(struct bt_pbap_pse *pbap_pse, uint32_t *supported_features)

Get PBAP PCE supported features.

This function is to get PBAP PCE supported features. This function returns immediately.

Parameters

- **pbap_pce** – pbap object.
- **supported_features** – PBAP PCE supported feature.

Returns
0 in case of success or otherwise in case of error.

Variables

char ***user_id**

char ***pin_code**

uint8_t **opcode**

uint16_t **packet_length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/phonebook")]

struct *bt_pbap_pull_phonebook_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **packet_length**

uint8_t **Flags**

uint8_t **constant**

struct bt_obex_hdr_u32 **conn_id**

uint8_t **opcode**

uint16_t **packet_length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/vcard-listing")]

struct *bt_pbap_pull_vcard_listing_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **packet_length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/vcard")]

struct *bt_pbap_pull_vcard_entry_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srmp**

uint8_t **opcode**

uint16_t **packet_length**

uint8_t **obex_version**

uint8_t **flags**

uint16_t **maximum_packet_length**

uint8_t **target**[16]

uint8_t **opcode**

uint16_t **packet_length**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

struct **bt_pbap_pce_cb**

#include <pbap_pce.h> PBAP_PCE profile application callback.

Public Members

void (***connected**)(struct bt_pbap_pce *pbap_pce)

PBAP_PCE connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param pbap_pce
PBAP PCE object.

void (***disconnected**)(struct bt_pbap_pce *pbap_pce, uint8_t result)

PBAP_PCE disconnected callback to application

If this callback is provided it will be called whenever the disconnection completes.

Param pbap_pce
PBAP PCE object.
Param result
BT_PBAP_SUCCESS_RSP in case of success or reason causes of disconnection.

void (***get_auth_info**)(struct bt_pbap_pce *pbap_pce, struct *bt_pbap_auth* *pbap_atuh_info)

PBAP_PCE get authentication information callback to application

If this callback is provided it will be called whenever pse asks to authenticate pce, and pce does not provide authentication information when initiating the connection. The application can provide validation authenticate information in this callback.

Param pbap_pce
PBAP PCE object.
Param pbap_atuh_info
refer to struct *bt_pbap_auth*.

void (***abort**)(struct bt_pbap_pce *pbap_pce, uint8_t result)

PBAP_PCE abort callback to application

If this callback is provided it will be called whenever the abort completes.

Param pbap_pce
PBAP_PCE object.
Param result
BT_PBAP_SUCCESS_RSP in case of success or otherwise in case of error.

void (***pull_phonebook**)(struct bt_pbap_pce *pbap_pce, uint8_t result, struct net_buf *buf)

PBAP_PCE phonebook download callback to application

If this callback is provided it will be called whenever the phonebook download completes. Application can use bt_pbap_pce_app_param_parse to obtain application parameters and use bt_pbap_pce_get_body to obtain body from RX net buffer.

Param pbap_pce
PBAP PCE object.

Param result
BT_PBAP_SUCCESS_RSP or BT_PBAP_CONTINUE_RSP in case of success or otherwise in case of error.
Param buf
RX net buffer contains PBAP application parameters and body of phonebook object.

void (*set_phonebook_path)(struct bt_pbap_pce *pbap_pce, uint8_t result)

PBAP_PCE set phonebook path callback to application

If this callback is provided it will be called whenever the set phonebook path completes.
Param pbap_pce
PBAP PCE object.
Param result
BT_PBAP_SUCCESS_RSP or in case of success or otherwise in case of error.

void (*pull_vcard_listing)(struct bt_pbap_pce *pbap_pce, uint8_t result, struct net_buf *buf)

PBAP_PCE get phonebook vcardlist callback to application

If this callback is provided it will be called whenever the get phonebook vcardlist completes. Appplication can use bt_pbap_pce_app_param_parse to obtain application parameters and use bt_pbap_pce_get_body to obtain body from RX net buffer.
Param pbap_pce
PBAP PCE object.
Param result
BT_PBAP_SUCCESS_RSP or BT_PBAP_CONTINUE_RSP in case of success or otherwise in case of error.
Param buf
RX net buffer contains PBAP application parameters and body of vcard listing object.

void (*pull_vcard_entry)(struct bt_pbap_pce *pbap_pce, uint8_t result, struct net_buf *buf)

PBAP_PCE get phonebook vcard callback to application

If this callback is provided it will be called whenever the get phonebook vcard completes. Appplication can use bt_pbap_pce_app_param_parse to obtain application parameters and use bt_pbap_pce_get_body to obtain body from RX net buffer.
Param pbap_pce
PBAP PCE object.
Param result
BT_PBAP_SUCCESS_RSP or BT_PBAP_CONTINUE_RSP in case of success or otherwise in case of error.
Param buf
RX net buffer contains PBAP application parameters and body of vcard entry object.

struct bt_pbap_pse_cb

#include <pbap_pse.h> PBAP_PSE profile application callback.

Public Members

void (*connected)(struct bt_pbap_pse *pbap_pse)

PBAP_PSE connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param pbap_pse
PBAP PSE object.

Param result
BT_PBAP_SUCCESS_RSP in case of success or otherwise in case of error.

void (*disconnected)(struct bt_pbap_pse *pbap_pse, uint8_t result)

PBAP_PSE disconnected callback to application

If this callback is provided it will be called whenever the disconnection completes.

Param pbap_pse
PBAP PSE object.

Param result
BT_PBAP_SUCCESS_RSP in case of success or reason causes of disconnection.

void (*get_auth_info)(struct bt_pbap_pse *pbap_pse, struct *bt_pbap_auth* *pbap_auth_info, bool *active_auth)

PBAP_PSE get authentication information callback to application

If this callback is provided it will be called whenever pse asks to authenticate pse, and pse does not provide authentication information when initiating the connection. The application can provide validation authenticate information in this callback.

Param pbap_pse
PBAP PSE object.

Param pbap_auth_info
refer to struct *bt_pbap_auth*.

Param active_auth
if true that means pse actively authicate pce.

void (*abort)(struct bt_pbap_pse *pbap_pse, uint8_t result)

PBAP_PSE abort callback to application

If this callback is provided it will be called whenever the abort completes.

Param pbap_pse
PBAP_PSE object.

Param result
BT_PBAP_SUCCESS_RSP in case of success or otherwise in case of error.

void (*pull_phonebook)(struct bt_pbap_pse *pbap_pse, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

PBAP_PSE pull phonebook object callback to application

If this callback is provided it will be called whenever get pull phonebook object request from pce.

Param pbap_pse
PBAP PSE object.

Param buf
RX net buffer from pce.

Param name
Specific phone book path name getting from pce.

Param flag
refer to enum bt_obex_req_flags.

void (*set_phonebook_path)(struct bt_pbap_pse *pbap_pse, char *name)

PBAP_PSE set phonebook path callback to application

If this callback is provided it will be called whenever the set phonebook path request received. When name is "/", go to root directory. When name is "..", go up one level. When name is "child" or "./child", go to child

Param pbap_pse
PBAP PSE object.
Param name
path name

void (*pull_vcard_listing)(struct bt_pbap_pse *pbap_pse, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

PBAP_PSE pull vcardlisting object callback to application

If this callback is provided it will be called whenever get pull vcard listing object request from pce.

Param pbap_pse
PBAP PSE object.
Param buf
RX net buffer from pce
Param name
Specific vcard listing name getting from pce.
Param flag
refer to enum bt_obex_req_flags.

void (*pull_vcard_entry)(struct bt_pbap_pse *pbap_pse, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

PBAP_PSE pull vcard entry object callback to application

If this callback is provided it will be called whenever get pull vcard entry object request from pce.

Param pbap_pse
PBAP PSE object.
Param buf
RX net buffer from pce
Param name
Specific vcard entry name getting from pce.
Param flag
refer to enum bt_obex_req_flags

struct **bt_pbap_auth**

#include <pbap_types.h> PBAP Authentication structure. if auth is be provided, its memory cannot be released during the entire connection hold process Authentication of user_id is not supported now, support pin_code only now.

struct **bt_pbap_pull_phonebook_hdr**

#include <pbap_types.h>

struct **type**

struct **bt_pbap_set_path_hdr**

#include <pbap_types.h>

struct **bt_pbap_pull_vcard_listing_hdr**

#include <pbap_types.h>

struct **type**

struct **bt_pbap_pull_vcard_entry_hdr**

#include <pbap_types.h>

struct **type**

struct **bt_pbap_ops_get_cont_hdr**

#include <pbap_types.h>

struct **bt_pbap_connection_hdr**

#include <pbap_types.h>

struct **bt_pbap_push_response_hdr**

#include <pbap_types.h>

1.9 Message Access Profile (MAP)

1.9.1 API Reference

group **bt_map**

Message Access Profile (MAP)

MAP OBEX Headers

Constant Definitions for MAP OBEX Headers

BT_MAP_HDR_NAME

BT_MAP_HDR_TYPE

BT_MAP_HDR_TARGET

BT_MAP_HDR_BODY

BT_MAP_HDR_END_OF_BODY

BT_MAP_HDR_CONNECTION_ID

BT_MAP_HDR_APP_PARAM

MAP OBEX Response Codes

Constant Definitions for MAP OBEX Response Codes

BT_MAP_RSP_CONTINUE

BT_MAP_RSP_SUCCESS

BT_MAP_RSP_BAD_REQ

BT_MAP_RSP_NOT_IMPLEMENTED

BT_MAP_RSP_UNAUTH

BT_MAP_RSP_PRECOND_FAILED

BT_MAP_RSP_NOT_FOUND

BT_MAP_RSP_NOT_ACCEPTABLE

BT_MAP_RSP_SERVICE_UNAVBL

BT_MAP_RSP_FORBIDDEN

BT_MAP_RSP_INT_SERVER_ERR

MAP Message Type

Constant Definitions for MAP Message Type

BT_MAP_TYPE_SEND_EVENT

BT_MAP_TYPE_SET_NTF_REG

BT_MAP_TYPE_GET_FOLDER_LISTING

BT_MAP_TYPE_GET_MSG_LISTING

BT_MAP_TYPE_GET_MSG

BT_MAP_TYPE_SET_MSG_STATUS

BT_MAP_TYPE_PUSH_MSG

BT_MAP_TYPE_UPDATE_INBOX

BT_MAP_TYPE_GET_MAS_INST_INFO

BT_MAP_TYPE_SET_OWNER_STATUS

BT_MAP_TYPE_GET_OWNER_STATUS

BT_MAP_TYPE_GET_CONVO_LISTING

BT_MAP_TYPE_SET_NTF_FILTER

MAP Supported Features

Constant Definitions for MAP Supported Features

BT_MAP_NTF_REG_FEATURE

BT_MAP_NTF_FEATURE

BT_MAP_BROWSING_FEATURE

BT_MAP_UPLOADING_FEATURE

BT_MAP_DELETE_FEATURE

BT_MAP_INST_INFO_FEATURE

BT_MAP_EXT_EVENT_REPORT_1_1

BT_MAP_EXT_EVENT_VERSION_1_2

BT_MAP_MSG_FORMAT_VERSION_1_1

BT_MAP_MSG_LISTING_FORMAT_VERSION_1_1

BT_MAP_PERSISTENT_MSG_HANDLE

BT_MAP_DATABASE_ID

BT_MAP_FOLDER_VERSION_CNTR

BT_MAP_CONVO_VERSION_CNTR

BT_MAP_PARTICIPANT_PRESENCE_CHANGE_NTF

BT_MAP_PARTICIPANT_CHAT_STATE_CHANGE_NTF

BT_MAP_PBAP_CONTACT_CROSS_REF

BT_MAP_NTF_FILTERING

BT_MAP_UTC_OFFSET_TIMESTAMP_FORMAT

BT_MAP_SUPPORTED_FEATURES_CONNECT_REQ

BT_MAP_CONVO_LISTING

BT_MAP_OWNER_STATUS

BT_MAP_MSG_FORWARDING

MAP Application Parameter Tag

Constant Definitions for MAP Application Parameter Tag ID

BT_MAP_TAG_ID_MAX_LIST_COUNT

BT_MAP_TAG_ID_LIST_START_OFFSET

BT_MAP_TAG_ID_FILTER_MESSAGE_TYPE

BT_MAP_TAG_ID_FILTER_PERIOD_BEGIN

BT_MAP_TAG_ID_FILTER_PERIOD_END

BT_MAP_TAG_ID_FILTER_READ_STATUS

BT_MAP_TAG_ID_FILTER_RECIPIENT

BT_MAP_TAG_ID_FILTER_ORIGINATOR

BT_MAP_TAG_ID_FILTER_PRIORITY

BT_MAP_TAG_ID_ATTACHMENT

BT_MAP_TAG_ID_TRANSPARENT

BT_MAP_TAG_ID_RETRY

BT_MAP_TAG_ID_NEW_MESSAGE

BT_MAP_TAG_ID_NOTIFICATION_STATUS

BT_MAP_TAG_ID_MAS_INSTANCE_ID

BT_MAP_TAG_ID_PARAMETER_MASK

BT_MAP_TAG_ID_FOLDER_LISTING_SIZE

BT_MAP_TAG_ID_LISTING_SIZE

BT_MAP_TAG_ID_SUBJECT_LENGTH

BT_MAP_TAG_ID_CHARSET

BT_MAP_TAG_ID_FRACTION_REQUEST

BT_MAP_TAG_ID_FRACTION_DELIVER

BT_MAP_TAG_ID_STATUS_INDICATOR

BT_MAP_TAG_ID_STATUS_VALUE

BT_MAP_TAG_ID_MSE_TIME

BT_MAP_TAG_ID_DATABASE_IDENTIFIER

BT_MAP_TAG_ID_CONV_LIST_VER_CNTR

BT_MAP_TAG_ID_PRESENCE_AVAILABILITY

BT_MAP_TAG_ID_PRESENCE_TEXT

BT_MAP_TAG_ID_LAST_ACTIVITY

BT_MAP_TAG_ID_FILTER_LAST_ACTIVITY_BEGIN

BT_MAP_TAG_ID_FILTER_LAST_ACTIVITY_END

BT_MAP_TAG_ID_CHAT_STATE

BT_MAP_TAG_ID_CONVERSATION_ID

BT_MAP_TAG_ID_FOLDER_VER_CNTR

BT_MAP_TAG_ID_FILTER_MSG_HANDLE

BT_MAP_TAG_ID_NOTIFICATION_FILTER_MASK

BT_MAP_TAG_ID_CONV_PARAMETER_MASK

BT_MAP_TAG_ID_OWNER_UCI

BT_MAP_TAG_ID_EXTENDED_DATA

BT_MAP_TAG_ID_MAP_SUPPORTED_FEATURES

BT_MAP_TAG_ID_MESSAGE_HANDLE

BT_MAP_TAG_ID_MODIFY_TEXT

BT_MAP_ADD_MAX_LIST_COUNT(buf, val)

Add MAP application parameters into net buffer.

> **Note**
>
> Calling these functions should follow a call to net_buf_reserve.

Parameters

- **buf** – net buffer for TX.
- **val** – value to be written into buffer.
- **len** – the length of value.

BT_MAP_ADD_LIST_START_OFFSET(buf, val)

BT_MAP_ADD_FILTER_MESSAGE_TYPE(buf, val)

BT_MAP_ADD_FILTER_PERIOD_BEGIN(buf, val, len)

BT_MAP_ADD_FILTER_PERIOD_END(buf, val, len)

BT_MAP_ADD_FILTER_READ_STATUS(buf, val)

BT_MAP_ADD_FILTER_RECIPIENT(buf, val, len)

BT_MAP_ADD_FILTER_ORIGINATOR(buf, val, len)

BT_MAP_ADD_FILTER_PRIORITY(buf, val)

BT_MAP_ADD_ATTACHMENT(buf, val)

BT_MAP_ADD_TRANSPARENT(buf, val)

BT_MAP_ADD_RETRY(buf, val)

BT_MAP_ADD_NEW_MESSAGE(buf, val)

BT_MAP_ADD_NOTIFICATION_STATUS(buf, val)

BT_MAP_ADD_MAS_INSTANCE_ID(buf, val)

BT_MAP_ADD_PARAMETER_MASK(buf, val)

BT_MAP_ADD_FOLDER_LISTING_SIZE(buf, val)

BT_MAP_ADD_LISTING_SIZE(buf, val)

BT_MAP_ADD_SUBJECT_LENGTH(buf, val)

BT_MAP_ADD_CHARSET(buf, val)

BT_MAP_ADD_FRACTION_REQUEST(buf, val)

BT_MAP_ADD_FRACTION_DELIVER(buf, val)

BT_MAP_ADD_STATUS_INDICATOR(buf, val)

BT_MAP_ADD_STATUS_VALUE(buf, val)

BT_MAP_ADD_MSE_TIME(buf, val, len)

BT_MAP_ADD_DATABASE_IDENTIFIER(buf, val, len)

BT_MAP_ADD_CONV_LIST_VER_CNTR(buf, val, len)

BT_MAP_ADD_PRESENCE_AVAILABILITY(buf, val)

BT_MAP_ADD_PRESENCE_TEXT(buf, val, len)

BT_MAP_ADD_LAST_ACTIVITY(buf, val, len)

BT_MAP_ADD_FILTER_LAST_ACTIVITY_BEGIN(buf, val, len)

BT_MAP_ADD_FILTER_LAST_ACTIVITY_END(buf, val, len)

BT_MAP_ADD_CHAT_STATE(buf, val)

BT_MAP_ADD_CONVERSATION_ID(buf, val, len)

BT_MAP_ADD_FOLDER_VER_CNTR(buf, val, len)

BT_MAP_ADD_FILTER_MSG_HANDLE(buf, val, len)

BT_MAP_ADD_NOTIFICATION_FILTER_MASK(buf, val)

BT_MAP_ADD_CONV_PARAMETER_MASK(buf, val)

BT_MAP_ADD_OWNER_UCI(buf, val, len)

BT_MAP_ADD_EXTENDED_DATA(buf, val, len)

BT_MAP_ADD_MAP_SUPPORTED_FEATURES(buf, val)

BT_MAP_ADD_MESSAGE_HANDLE(buf, val, len)

BT_MAP_ADD_MODIFY_TEXT(buf, val)

BT_MAP_ADD_BODY(buf, val, len)

Add MAP headers into net buffer.

Parameters

- **buf** – net buffer for TX.
- **val** – value to be written into buffer.
- **len** – the length of value.

BT_MAP_ADD_END_OF_BODY(buf, val, len)

Defines

BT_MAP_MCE_RSV_LEN_GET_FOLDER_LISTING(mce_mas, flags)

Reserve length for GetFolderListing.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.
- **flags** – refer to enum bt_obex_req_flags.

Returns

the reservation length.

BT_MAP_MCE_RSV_LEN_SET_FOLDER(mce_mas)

Reserve length for SetFolder.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_GET_MSG_LISTING(mce_mas, name, flags)

Reserve length for GetMessageListing.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.
- **name** – the folder name string
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_GET_MSG(mce_mas, flags)

Reserve length for GetMessage.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_SET_MSG_STATUS(mce_mas, flags)

Reserve length for SetMessageStatus.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_PUSH_MSG(mce_mas, name, flags)

Reserve length for PushMessage.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.
- **name** – the folder name string
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_SET_NTF_REG(mce_mas)

Reserve length for SetNotificationRegistration.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_UPDATE_INBOX(mce_mas)

Reserve length for UpdateInbox.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_GET_MAS_INST_INFO(mce_mas, flags)

Reserve length for GetMASInstanceInformation.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_SET_OWNER_STATUS(mce_mas, flags)

Reserve length for SetOwnerStatus.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_GET_OWNER_STATUS(mce_mas, flags)

Reserve length for GetOwnerStatus.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.

- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_GET_CONVO_LISTING(mce_mas, flags)

Reserve length for GetConversationListing.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.
- **flags** – refer to enum bt_obex_req_flags.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_SET_NTF_FILTER(mce_mas)

Reserve length for SetNotificationFilter.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mas** – MCE MAS object.

Returns
the reservation length.

BT_MAP_MCE_RSV_LEN_SEND_EVENT_RESP(mce_mns)

Reserve length for SendEventResponse.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mce_mns** – MCE MNS object.

Returns
the reservation length.

bt_map_mce_app_param_parse(buf, func, user_data)

Helper for parsing application parameters.

A helper for parsing the application parameters. The most common scenario is to call this helper on the application parameters received in the callback that was given to bt_map_mce_mas_register or bt_map_mce_mns_register.

Parameters

- **buf** – net buffer received in the callback.
- **func** – Callback function which will be called for each tag that's found in the buffer. The callback should return true to continue parsing, or false to stop parsing.
- **user_data** – User data to be passed to the callback.

bt_map_mce_get_body(buf, body, length)

Helper for getting body data.

A helper for getting the body data. The most common scenario is to call this helper on the body received in the callback that was given to bt_map_mce_mas_register or bt_map_mce_mns_register.

Parameters

- **buf** – net buffer received in the callback.
- **body** – pointer used for holding body data.
- **length** – the length of the body data.

Returns

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

BT_MAP_MSE_RSV_LEN_SEND_EVENT(mse_mns, flags)

Reserve length for SendEvent.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mse_mns** – MSE MNS object.
- **flags** – refer to enum bt_obex_req_flags.

Returns

the reservation length.

BT_MAP_MSE_RSV_LEN_SEND_RESP(mse_mas)

Reserve length for SendResponse.

The reservation length will be as the 2nd parameter of net_buf_reserve, which is used for reserving space in front of net buffer. The reserved buffer is filled in the map.

Parameters

- **mse_mas** – MSE MAS object.

Returns

the reservation length.

bt_map_mse_app_param_parse(buf, func, user_data)

Helper for parsing application parameters.

A helper for parsing the application parameters. The most common scenario is to call this helper on the application parameters received in the callback that was given to bt_map_mse_mas_register or bt_map_mse_mns_register.

Parameters

- **buf** – net buffer received in the callback.
- **func** – Callback function which will be called for each tag that's found in the buffer. The callback should return true to continue parsing, or false to stop parsing.
- **user_data** – User data to be passed to the callback.

bt_map_mse_get_body(buf, body, length)

Helper for getting body data.

A helper for getting the body data. The most common scenario is to call this helper on the body received in the callback that was given to bt_map_mse_mas_register or bt_map_mse_mns_register.

Parameters

- **buf** – net buffer received in the callback.
- **body** – pointer used for holding body data.
- **length** – the length of the body data.

Returns

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

BT_MAP_VERSION_1_1

BT_MAP_VERSION_1_2

BT_MAP_VERSION_1_3

BT_MAP_VERSION_1_4

BT_MAP_MSG_HANDLE_SIZE

BT_MAP_CONVO_ID_SIZE

BT_MAP_DATABASE_ID_SIZE

BT_MAP_VER_CNT_SIZE

BT_MAP_MSE_MAS_SUPPORTED_FEATURES_V11

BT_MAP_MSE_MAS_SUPPORTED_FEATURES_V12

BT_MAP_MSE_MAS_SUPPORTED_FEATURES_V13

BT_MAP_MSE_MAS_SUPPORTED_FEATURES_V14

Functions

int **bt_map_mce_mas_register**(struct *bt_map_mce_mas_cb* *cb)

Register MCE MAS.

Register MAP profile MCE MAS callback to monitor the state change and event from MSE. This function just needs to be call one time for multiple MAS server instances. Application can use the parameter of mce_mas in the callback to determine the MAS server instance.

Parameters

- **cb** – callback structure.

Returns

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

int **bt_map_mce_mas_unregister**(void)

Unregister MCE MAS.

Unregister MAP profile MCE MAS callback. This function is to be called when there is no MAS OBEX connection.

Returns

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

int **bt_map_mce_psm_connect**(struct bt_conn *conn, uint16_t psm, uint32_t supported_features, struct bt_map_mce_mas **mce_mas)

Create MCE MAS connection based on L2CAP.

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 MAS OBEX connection between devices. This function establishes L2CAP connection. This function can be called multiple times to establish multiple MAS OBEX connections. After connection success, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **conn** – Pointer to bt_conn structure.
- **psm** – GoepL2capPsm, returned in SDP record.
- **supported_features** – partner device's supported features, returned in SDP record.
- **mce_mas** – MCE MAS object.

Returns

0 in case of success or otherwise in case of error.

int **bt_map_mce_scn_connect**(struct bt_conn *conn, uint8_t scn, uint32_t supported_features, struct bt_map_mce_mas **mce_mas)

Create MCE MAS connection based on RFCOM.

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 MAS OBEX connection between devices. This function establishes RFCOM connection. This function can be called multiple times to establish multiple MAS OBEX connections. After connection success, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **conn** – Pointer to bt_conn structure.
- **scn** – RFCOM server channel number, returned in SDP record.
- **supported_features** – partner device's supported features, returned in SDP record.

- **mce_mas** – MCE MAS object.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_disconnect**(struct bt_map_mce_mas *mce_mas)

Disconnect MCE MAS connection.

Disconnect MCE MAS OBEX connection. This function can be called multiple times to disconnect multiple MAS OBEX connections.

Parameters

- **mce_mas** – MCE MAS object.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_abort**(struct bt_map_mce_mas *mce_mas)

Abort MCE MAS operation.

Abort MCE MAS PUT or GET operation. This cancels the current outstanding operation. The return value of -EINPROGRESS means abort is queued and pending and the abort request will be sent after receiving next response from the MSE.

Parameters

- **mce_mas** – MCE MAS object.

Returns
0 in case of success or -EINPROGRESS in case abort is queued or otherwise in case of error.

int **bt_map_mce_get_folder_listing**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, bool wait, enum bt_obex_req_flags flags)

Send get folder listing request.

This function is to retrieve the Folder-Listing object from the current folder of the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. It's recommended to intelligently allocate TX buffer size to put GET request in the single packet. This function can be called multiple times till the Folder-Listing object is retrieved completely. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next message. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.
- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_set_folder**(struct bt_map_mce_mas *mce_mas, char *name)

Send set folder request.

This function is to navigate the folders of the MSE. When name is "/", go to root directory. When name is ".." or "../", go up one level. When name is "../child", go up one level and then go to child directory. When name is "child" or "./child", go to child directory. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **name** – the folder name string

Returns

0 in case of success or otherwise in case of error.

int **bt_map_mce_get_msg_listing**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, char *name, bool wait, enum bt_obex_req_flags flags)

Send get messages listing request.

This function is to retrieve the Messages-Listing objects from the current folder of the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. It's recommended to intelligently allocate TX buffer size to put GET request in the single packet. This function can be called multiple times till the Messages-Listing objects are retrieved completely. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. When name is NULL, get messages listing in the current folder. When name is not NULL, get messages listing in the child folder. Setting wait to true is to ask the partner to wait after sending its next message. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.
- **name** – the folder name string
- **wait** – true - set SRMP is 1. false - exclude SRMP.
- **flags** – refer to enum bt_obex_req_flags.

Returns

0 in case of success or otherwise in case of error.

int **bt_map_mce_get_msg**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, char *name, bool wait, enum bt_obex_req_flags flags)

Send get messages request.

This function is to retrieve a specific message from the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. It's recommended to intelligently allocate TX buffer size to put GET request in the single packet. This function can be called multiple times till the message is retrieved completely. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. When name is NULL, get messages listing in the current folder. When name is not NULL, get messages listing in the child folder. Setting wait to true is to ask the partner to wait after sending its next message. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.

- **buf** – TX net buffer allocated by application.
- **name** – the message handle string
- **wait** – true - set SRMP is 1. false - exclude SRMP.
- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_set_msg_status**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flags)

Send set message status request.

This function is to modify the status of a message on the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.
- **name** – the message handle string
- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_push_msg**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flags)

Send push message request.

This function is to push a message to a folder or conversation of the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. When name is NULL, push message into the current folder. When name is not NULL, push message into the child folder. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.
- **name** – the folder name string
- **actual** – actual number of bytes is sent.

- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_set_ntf_reg**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf)

Send set notification registration request.

This function is to register itself for being notified of the arrival of new messages. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_update_inbox**(struct bt_map_mce_mas *mce_mas)

Send update inbox request.

This function is to initiate an update of the MSE's inbox. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_get_mas_inst_info**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, bool wait, enum bt_obex_req_flags flags)

Send get MAS instance information request.

This function is to retrieve user-readable information about the MAS Instances provided by the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. It's recommended to intelligently allocate TX buffer size to put GET request in the single packet. This function can be called multiple times till the information is retrieved completely. Application can use BT_MAP_ADD_xxx to add application parameters into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next message. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.
- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_set_owner_status**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, enum bt_obex_req_flags flags)

Send set owner status request.

This function is to change the Presence, Chat State, or Last Activity of the owner on the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. Application can use BT_MAP_ADD_xxx to add application parameters into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.
- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_get_owner_status**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, bool wait, enum bt_obex_req_flags flags)

Send get owner status request.

This function is to retrieve the Presence, Chat State, or Last Activity of the owner on the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. It's recommended to intelligently allocate TX buffer size to put GET request in the single packet. This function can be called multiple times till the message is retrieved completely. Application can use BT_MAP_ADD_xxx to add application parameters into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next message. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exculde SMRP.
- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_get_convo_listing**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf, bool wait, enum bt_obex_req_flags flags)

Send get conversation listing request.

This function is retrieve Conversation-Listing objects from the MSE. This function can be called multiple times till the message is sent completely. Application uses the parameter of flags to send the chunked packet. It's recommended to intelligently allocate TX buffer size to put GET request in the single packet. This function can be called multiple times till the Conversation-Listing objects are retrieved completely. Application can use BT_MAP_ADD_xxx to add application parameters into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next message. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.
- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_set_ntf_filter**(struct bt_map_mce_mas *mce_mas, struct net_buf *buf)

Send set notification filter request.

This function is to specify which notifications to receive from the MSE. Application can use BT_MAP_ADD_xxx to add application parameters into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. When receiving response, the callback that is registered by bt_map_mce_mas_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **buf** – TX net buffer allocated by application.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_get_max_pkt_len**(struct bt_map_mce_mas *mce_mas, uint16_t *max_pkt_len)

Get MAS maximum packet length.

This function is to get the maximum packet length in MCE MAS OBEX connection. This function returns immediately.

Parameters

- **mce_mas** – MCE MAS object.
- **max_pkt_len** – the return maximum packet length.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_mns_register**(struct *bt_map_mce_mns_cb* *cb)

Register MCE MNS.

Register MAP profile MCE MNS callback to monitor the state change and event from MSE. This function just needs to be call one time.

Parameters

- **cb** – callback structure.

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

int **bt_map_mce_mns_unregister**(void)

Unregister MCE MNS.

Unregister MAP profile MCE MNS callback. This function is to be called when there is no MNS OBEX connection.

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

int **bt_map_mce_mns_disconnect**(struct bt_map_mce_mns *mce_mns)

Disconnect MCE MNS connection.

Disconnect MCE MNS OBEX connection.

Parameters

- **mce_mns** – MCE MNS object.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_send_event_response**(struct bt_map_mce_mns *mce_mns, uint8_t result, bool wait)

Send event response.

This function is to send response when receiving Event-Report object from MSE. Setting wait to true is to ask the partner to wait after sending its next message. When receiving Event-Report object, the callback that is registered by bt_map_mce_mns_register is called.

Parameters

- **mce_mas** – MCE MAS object.
- **result** – BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mce_mns_get_max_pkt_len**(struct bt_map_mce_mns *mce_mns, uint16_t *max_pkt_len)

Get MNS maximum packet length.

This function is to get the maximum packet length in MCE MNS OBEX connection. This function returns immediately.

Parameters

- **mce_mas** – MCE MAS object.
- **max_pkt_len** – the return maximum packet length.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_mas_register**(struct *bt_map_mse_mas_cb* *cb)

Register MSE MAS.

Register MAP profile MSE MAS callback to monitor the event from MCE. This function just needs to be call one time for multiple MAS server instances. Application can use the parameter of mse_mas in the callback to determine the MAS server instance.

Parameters

- **cb** – callback structure.

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

int **bt_map_mse_mas_unregister**(void)

Unregister MSE MAS.

Unregister MAP profile MSE MAS callback. This function is to be called when there is no MAS OBEX connection.

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

int **bt_map_mse_disconnect**(struct bt_map_mse_mas *mse_mas)

Disconnect MSE MAS connection.

Disconnect MSE MAS OBEX connection. This function can be called multiple times to disconnect multiple MAS OBEX connections.

Parameters

- **mse_mas** – MSE MAS object.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_get_folder_listing_response**(struct bt_map_mse_mas *mse_mas, uint8_t result, struct net_buf *buf, bool wait)

Send response when receiving get folder listing request from MCE.

This function is to send response when receiving get folder listing request from MCE. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next request. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_set_folder_response**(struct bt_map_mse_mas *mse_mas, uint8_t result)

Send response when receiving set folder request from MCE.

This function is to send response when receiving set folder request from MCE. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_get_msg_listing_response**(struct bt_map_mse_mas *mse_mas, uint8_t result, struct net_buf *buf, bool wait)

Send response when receiving get message listing request from MCE.

This function is to send response when receiving get message listing request from MCE. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next request. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns

0 in case of success or otherwise in case of error.

int **bt_map_mse_get_msg_response**(struct bt_map_mse_mas *mse_mas, uint8_t result, struct net_buf *buf, bool wait)

Send response when receiving get message request from MCE.

This function is to send response when receiving get message request from MCE. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next request. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_set_msg_status_response**(struct bt_map_mse_mas *mse_mas, uint8_t result)

Send response when receiving set message status request from MCE.

This function is to send response when receiving set message status request from MCE. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_push_msg_response**(struct bt_map_mse_mas *mse_mas, uint8_t result, char *name, bool wait)

Send response when receiving push message request from MCE.

This function is to send response when receiving push message request from MCE. Setting wait to true is to ask the partner to wait after sending its next request. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.
- **name** – pass a string if message handle is generated or NULL.
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_set_ntf_reg_response**(struct bt_map_mse_mas *mse_mas, uint8_t result)

Send response when receiving set notification registration request from MCE.

This function is to send response when receiving set notification registration from MCE. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_update_inbox_response**(struct bt_map_mse_mas *mse_mas, uint8_t result)

Send response when receiving update inbox request from MCE.

This function is to send response when receiving update inbox request from MCE. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_get_mas_inst_info_response**(struct bt_map_mse_mas *mse_mas, uint8_t result, struct net_buf *buf, bool wait)

Send response when receiving get MAS instance infomation request from MCE.

This function is to send response when receiving get MAS instance infomation request from MCE. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_OWNER_UCI and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next request. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_set_owner_status_response**(struct bt_map_mse_mas *mse_mas, uint8_t result)

Send response when receiving set owner status request from MCE.

This function is to send response when receiving set owner status request from MCE. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_get_owner_status_response**(struct bt_map_mse_mas *mse_mas, uint8_t result, struct net_buf *buf, bool wait)

Send response when receiving get owner status request from MCE.

This function is to send response when receiving get owner status from MCE. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_PRESENCE_AVAILABILITY and

BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next request. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns

0 in case of success or otherwise in case of error.

int **bt_map_mse_get_convo_listing_response**(struct bt_map_mse_mas *mse_mas, uint8_t result, struct net_buf *buf, bool wait)

Send response when receiving get conversation listing request from MCE.

This function is to send response when receiving get conversation listing from MCE. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAX_LIST_COUNT and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. Setting wait to true is to ask the partner to wait after sending its next request. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.
- **buf** – TX net buffer allocated by application.
- **wait** – true - set SRMP is 1. false - exclude SRMP.

Returns

0 in case of success or otherwise in case of error.

int **bt_map_mse_set_ntf_filter_response**(struct bt_map_mse_mas *mse_mas, uint8_t result)

Send response when receiving set notification filter request from MCE.

This function is to send response when receiving set notification filter request from MCE. When receiving request, the callback that is registered by bt_map_mse_mas_register is called.

Parameters

- **mse_mas** – MSE MAS object.
- **result** – response code - BT_MAP_RSP_SUCCESS, BT_MAP_RSP_CONTINUE or other error codes.

Returns

0 in case of success or otherwise in case of error.

int **bt_map_mse_get_max_pkt_len**(struct bt_map_mse_mas *mse_mas, uint16_t *max_pkt_len)

Get MAS maximum packet length.

This function is to get the maximum packet length in MSE MAS OBEX connection. This function returns immediately.

Parameters

- **mse_mas** – MSE MAS object.
- **max_pkt_len** – the return maximum packet length.

Returns

0 in case of success or otherwise in case of error.

int **bt_map_mse_mns_register**(struct *bt_map_mse_mns_cb* *cb)

Register MSE MNS.

Register MAP profile MSE MNS callback to monitor the event from MCE. This function just needs to be call one time.

Parameters

- **cb** – callback structure.

Returns

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

int **bt_map_mse_mns_unregister**(void)

Unregister MSE MNS.

Unregister MAP profile MSE MNS callback. This function is to be called when there is no MNS OBEX connection.

Returns

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

int **bt_map_mse_mns_psm_connect**(struct bt_conn *conn, uint16_t psm, uint32_t supported_features, struct bt_map_mse_mns **mse_mns)

Create MSE MNS connection based on L2CAP.

This function is to be called after MCE sets SetNotificationRegistration to ON. The API is to be used to establish MNS OBEX connection between devices. This function establishes L2CAP connection. This function can be called once as there is only one MNS connection for all MAS Instances. After connection success, the callback that is registered by bt_map_mse_mns_register is called.

Parameters

- **conn** – Pointer to bt_conn structure.
- **psm** – GoepL2capPsm, returned in SDP record.
- **supported_features** – partner device's supported features, returned in SDP record.
- **mse_mns** – MSE MNS object.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_mns_scn_connect**(struct bt_conn *conn, uint8_t scn, uint32_t supported_features, struct bt_map_mse_mns **mse_mns)

Create MSE MNS connection based on RFCOM.

This function is to be called after MCE sets SetNotificationRegistration to ON. The API is to be used to establish MNS OBEX connection between devices. This function establishes RFCOM connection. This function can be called once as there is only one MNS connection for all MAS Instances. After connection success, the callback that is registered by bt_map_mse_mns_register is called.

Parameters

- **conn** – Pointer to bt_conn structure.
- **scn** – RFCOM server channel number, returned in SDP record.
- **supported_features** – partner device's supported features, returned in SDP record.
- **mse_mns** – MSE MNS object.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_mns_disconnect**(struct bt_map_mse_mns *mse_mns)

Disconnect MSE MNS connection.

Disconnect MSE MNS OBEX connection.

Parameters

- **mse_mas** – MSE MNS object.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_send_event**(struct bt_map_mse_mns *mse_mns, struct net_buf *buf, enum bt_obex_req_flags flags)

Send event to MCE.

This function is to send event to MCE. This function can be called multiple times till the event object is sent completely. Application uses the parameter of flags to send the chunked packet. Application can use BT_MAP_ADD_xxx, such as BT_MAP_ADD_MAS_INSTANCE_ID and BT_MAP_ADD_BODY, to add application parameters and add body into TX net buffer. Calling BT_MAP_ADD_xxx to add application parameters should follow a call to net_buf_reserve. When receiving response, the callback that is registered by bt_map_mse_mns_register is called.

Note

the data size of the net_buf is managed by application. Application need to make sure the data size doesn't exceed the maximum packet length.

Parameters

- **mse_mas** – MSE MAS object.
- **buf** – TX net buffer allocated by application.
- **flags** – refer to enum bt_obex_req_flags.

Returns
0 in case of success or otherwise in case of error.

int **bt_map_mse_mns_get_max_pkt_len**(struct bt_map_mse_mns *mse_mns, uint16_t *max_pkt_len)

Get MNS maximum packet length.

This function is to get the maximum packet length in MSE MNS OBEX connection. This function returns immediately.

Parameters

- **mse_mas** – MSE MAS object.
- **max_pkt_len** – the return maximum packet length.

Returns
0 in case of success or otherwise in case of error.

Variables

void (***connected**)(struct bt_map_mce_mas *mce_mas)

MCE MAS connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param mce_mas
MAP MCE MAS object.

void (***disconnected**)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS 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 OBEX connection establisment.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (***abort**)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS abort Callback

This callback provides the result of abortion to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*get_folder_listing)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get folder listing Callback

This callback provides the folder listing object to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and folder listing object.

void (*set_folder)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set folder Callback

This callback provides the result of setting folder to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*get_msg_listing)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get message listing Callback

This callback provides the message listing object to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and message listing object.

void (*get_msg)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get message Callback

This callback provides the message object to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and message object.

void (*set_msg_status)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set message status Callback

This callback provides the result of setting message status to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*push_msg)(struct bt_map_mce_mas *mce_mas, uint8_t result, char *name)

MCE MAS push message Callback

This callback provides the result and message handle to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param name
the message handle returned by MSE

void (*set_ntf_reg)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set notification registration Callback

This callback provides the result of setting notification registration to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*update_inbox)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS update inbox Callback

This callback provides the result of updating inbox to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*get_mas_inst_info)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get MAS instance information Callback

This callback provides MAS instance information to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and MAS instance information.

void (*`set_owner_status`)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set owner status Callback

This callback provides the result of setting owner status to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

void (*`get_owner_status`)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get owner status Callback

This callback provides Presence, Chat State, or Last Activity of the owner on the MSE to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters(Presence, Chat State, or Last Activity).

void (*`get_convo_listing`)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get conversation listing Callback

This callback provides conversation listing object to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and conversation listing object.

Param wait
if true the remote device ask the local device to wait after sending the next message.

void (*set_ntf_filter)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set notification filter Callback

This callback provides the result of setting notification filter to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*connected)(struct bt_map_mce_mns *mce_mns)

MCE MNS connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param mce_mas
MAP MCE MNS object.

void (*disconnected)(struct bt_map_mce_mns *mce_mns, uint8_t result)

MCE MNS 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 OBEX connection establisment.

Param mce_mas
MAP MCE MNS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*send_event)(struct bt_map_mce_mns *mce_mns, struct net_buf *buf, enum bt_obex_req_flags flag)

MCE MNS send event Callback

This callback provides event report object to application.

Param mce_mas
MAP MCE MAS object.

Param buf
RX net buffer contains MAS instance ID and event report object.

Param flags
refer to enum bt_obex_req_flags.

void (*connected)(struct bt_map_mse_mas *mse_mas, uint16_t psm, uint8_t scn)

MSE MAS connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param mse_mas
MAP MSE MAS object.

Param psm
local L2CAP PSM.

Param scn
local RFCOM server channel.

void (*disconnected)(struct bt_map_mse_mas *mse_mas, uint8_t result)

MSE MAS 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 OBEX connection establisment.

Param mse_mas
MAP MSE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*abort)(struct bt_map_mse_mas *mse_mas)

MSE MAS abort Callback

This callback informs application to abort the current operation. The abort response is sent internally.

Param mse_mas
MAP MSE MAS object.

void (*get_folder_listing)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS get folder listing Callback

It is called when receiving get folder listing request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.

Param flags
refer to enum bt_obex_req_flags.

void (*set_folder)(struct bt_map_mse_mas *mse_mas, char *name)

MSE MAS set folder Callback

This callback provides the path to application. When name is "/", go to root directory. When name is "../", go up one level. When name is "../child", go up one level and then go to child directory. When name is "child", go to child directory.

Param mse_mas
MAP MSE MAS object.

Param name
the folder name string.

void (*get_msg_listing)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

MSE MAS get message listing Callback

It is called when receiving get message listing request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.

Param name
string - the child folder name, NULL - current directory.

Param flags
refer to enum bt_obex_req_flags.

void (***get_msg***)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

MSE MAS get message Callback

It is called when receiving get message request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.

Param name
string - the child folder name, NULL - current directory.

Param flags
refer to enum bt_obex_req_flags.

void (***set_msg_status***)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

MSE MAS set message status Callback

It is called when receiving set message status request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters and use bt_map_mse_get_body to obtain body from RX net buffer.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters and body.

Param name
string - the child folder name, NULL - current directory.

Param flags
refer to enum bt_obex_req_flags.

void (***push_msg***)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

MSE MAS push message Callback

It is called when receiving push message request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters and use bt_map_mse_get_body to obtain body from RX net buffer.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters and body.

Param name
string - the child folder name, NULL - current directory.

Param flags
refer to enum bt_obex_req_flags.

void (*set_ntf_reg)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS set notification registration Callback

It is called when receiving set notification registration request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters and use bt_map_mse_get_body to obtain body from RX net buffer.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains notification status and body.

Param flags
refer to enum bt_obex_req_flags.

void (*update_inbox)(struct bt_map_mse_mas *mse_mas)

MSE MAS update inbox Callback

It is called when receiving update inbox request.

Param mse_mas
MAP MSE MAS object.

void (*get_mas_inst_info)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS get MAS instance information Callback

It is called when receiving get MAS instance infomation request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.

Param flags
refer to enum bt_obex_req_flags.

void (*set_owner_status)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS set owner status Callback

It is called when receiving get set owner status request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.

Param flags
refer to enum bt_obex_req_flags.

void (*`get_owner_status`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS get owner status Callback

It is called when receiving get owner status request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.

Param flags
refer to enum bt_obex_req_flags.

void (*`get_convo_listing`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS get conversation listing Callback

It is called when receiving get conversation listing request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.

Param flags
refer to enum bt_obex_req_flags.

void (*`set_ntf_filter`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS set notification filter Callback

It is called when receiving get set notification filter request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters and use bt_map_mse_get_body to obtain body from RX net buffer.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.

Param flags
refer to enum bt_obex_req_flags.

void (*`connected`)(struct bt_map_mse_mns *mse_mns)

MSE MNS connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param mse_mas
MAP MSE MNS object.

void (*disconnected)(struct bt_map_mse_mns *mse_mns, uint8_t result)

MSE MNS 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 OBEX connection establisment.

Param mse_mas
MAP MSE MNS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*send_event)(struct bt_map_mse_mns *mse_mns, uint8_t result)

MSE MNS send event Callback

This callback provides send event result to application.

Param mse_mas
MAP MSE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

uint8_t opcode

uint16_t length

struct bt_obex_hdr_u32 conn_id

struct bt_obex_hdr_u8 srmp

uint8_t opcode

uint16_t length

struct bt_obex_hdr_u32 conn_id

uint8_t opcode

uint16_t length

uint8_t version

uint8_t flags

uint16_t max_pkt_len

uint8_t **uuid**[16]

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-obex/folder-listing")]

struct *bt_map_get_folder_listing_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

uint8_t **flags**

uint8_t **constant**

struct bt_obex_hdr_u32 **conn_id**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/MAP-msg-listing")]

struct *bt_map_get_msg_listing_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[(8U * 4U + 2U)]

struct *bt_map_get_msg_hdr*.[anonymous] **name**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/message")]

struct *bt_map_get_msg_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[(8U * 4U + 2U)]

struct *bt_map_set_msg_status_hdr*.[anonymous] **name**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/messageStatus")]

struct *bt_map_set_msg_status_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/message")]

struct *bt_map_push_msg_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/MAP-NotificationRegistration")]

struct *bt_map_set_ntf_reg_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/MAP-messageUpdate")]

struct *bt_map_update_inbox_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/MASInstanceInformation")]

struct *bt_map_get_mas_inst_info_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/ownerStatus")]

struct *bt_map_set_owner_status_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/ownerStatus")]

struct *bt_map_get_owner_status_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/MAP-convo-listing")]

struct *bt_map_get_convo_listing_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/MAP-notification-filter")]

struct *bt_map_set_ntf_filter_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u32 **conn_id**

struct bt_obex_hdr_u8 **srm**

uint8_t **hi**

uint16_t **length**

uint8_t **hv**[sizeof("x-bt/MAP-event-report")]

struct *bt_map_send_event_hdr*.[anonymous] **type**

uint8_t **opcode**

uint16_t **length**

struct bt_obex_hdr_u8 **srm**

struct bt_obex_hdr_u8 **srmp**

struct **bt_map_mce_mas_cb**

#include <map_mce.h> MAP MCE MAS application callback.

Public Members

void (***connected**)(struct bt_map_mce_mas *mce_mas)

MCE MAS connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param mce_mas
MAP MCE MAS object.

void (***disconnected**)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS 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 OBEX connection establisment.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (***abort**)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS abort Callback

This callback provides the result of abortion to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (***get_folder_listing**)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get folder listing Callback

This callback provides the folder listing object to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and folder listing object.

void (***set_folder**)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set folder Callback

This callback provides the result of setting folder to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*`get_msg_listing`)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get message listing Callback

This callback provides the message listing object to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and message listing object.

void (*`get_msg`)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get message Callback

This callback provides the message object to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and message object.

void (*`set_msg_status`)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set message status Callback

This callback provides the result of setting message status to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*`push_msg`)(struct bt_map_mce_mas *mce_mas, uint8_t result, char *name)

MCE MAS push message Callback

This callback provides the result and message handle to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param name
the message handle returned by MSE

void (*set_ntf_reg)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set notification registration Callback

This callback provides the result of setting notification registration to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*update_inbox)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS update inbox Callback

This callback provides the result of updating inbox to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*get_mas_inst_info)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get MAS instance information Callback

This callback provides MAS instance information to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters and MAS instance information.

void (*set_owner_status)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set owner status Callback

This callback provides the result of setting owner status to application.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

void (*get_owner_status)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get owner status Callback

This callback provides Presence, Chat State, or Last Activity of the owner on the MSE to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

Param buf
RX net buffer contains MAP application parameters(Presence, Chat State, or Last Activity).

void (*`get_convo_listing`)(struct bt_map_mce_mas *mce_mas, uint8_t result, struct net_buf *buf)

MCE MAS get conversation listing Callback

This callback provides conversation listing object to application. Appplication can use bt_map_mce_app_param_parse to obtain application parameters and use bt_map_mce_get_body to obtain body from RX net buffer.

Param mce_mas
MAP MCE MAS object.
Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.
Param buf
RX net buffer contains MAP application parameters and conversation listing object.
Param wait
if true the remote device ask the local device to wait after sending the next message.

void (*`set_ntf_filter`)(struct bt_map_mce_mas *mce_mas, uint8_t result)

MCE MAS set notification filter Callback

This callback provides the result of setting notification filter to application.

Param mce_mas
MAP MCE MAS object.
Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

struct `bt_map_mce_mns_cb`

#include <*map_mce.h*> MAP MCE MNS application callback.

Public Members

void (*`connected`)(struct bt_map_mce_mns *mce_mns)

MCE MNS connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param mce_mas
MAP MCE MNS object.

void (*`disconnected`)(struct bt_map_mce_mns *mce_mns, uint8_t result)

MCE MNS 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 OBEX connection establisment.

Param mce_mas
MAP MCE MNS object.
Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (*`send_event`)(struct bt_map_mce_mns *mce_mns, struct net_buf *buf, enum bt_obex_req_flags flag)

MCE MNS send event Callback

This callback provides event report object to application.

Param mce_mas
MAP MCE MAS object.

Param buf
RX net buffer contains MAS instance ID and event report object.

Param flags
refer to enum bt_obex_req_flags.

struct **bt_map_mse_mas_cb**

#include <map_mse.h> MAP MSE MAS application callback.

Public Members

void (***connected**)(struct bt_map_mse_mas *mse_mas, uint16_t psm, uint8_t scn)

MSE MAS connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param mse_mas
MAP MSE MAS object.

Param psm
local L2CAP PSM.

Param scn
local RFCOM server channel.

void (***disconnected**)(struct bt_map_mse_mas *mse_mas, uint8_t result)

MSE MAS 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 OBEX connection establisment.

Param mse_mas
MAP MSE MAS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (***abort**)(struct bt_map_mse_mas *mse_mas)

MSE MAS abort Callback

This callback informs application to abort the current operation. The abort response is sent internally.

Param mse_mas
MAP MSE MAS object.

void (***get_folder_listing**)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS get folder listing Callback

It is called when receiving get folder listing request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.

Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.
Param flags
refer to enum bt_obex_req_flags.

void (*`set_folder`)(struct bt_map_mse_mas *mse_mas, char *name)

MSE MAS set folder Callback

This callback provides the path to application. When name is "/", go to root directory. When name is "../", go up one level. When name is "../child", go up one level and then go to child directory. When name is "child", go to child directory.
Param mse_mas
MAP MSE MAS object.
Param name
the folder name string.

void (*`get_msg_listing`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

MSE MAS get message listing Callback

It is called when receiving get message listing request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.
Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains MAP application parameters.
Param name
string - the child folder name, NULL - current directory.
Param flags
refer to enum bt_obex_req_flags.

void (*`get_msg`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

MSE MAS get message Callback

It is called when receiving get message request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.
Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains MAP application parameters.
Param name
string - the child folder name, NULL - current directory.
Param flags
refer to enum bt_obex_req_flags.

void (*`set_msg_status`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

MSE MAS set message status Callback

It is called when receiving set message status request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters and use bt_map_mse_get_body to obtain body from RX net buffer.

Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains MAP application parameters and body.
Param name
string - the child folder name, NULL - current directory.
Param flags
refer to enum bt_obex_req_flags.

void (*push_msg)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, char *name, enum bt_obex_req_flags flag)

MSE MAS push message Callback

It is called when receiving push message request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters and use bt_map_mse_get_body to obtain body from RX net buffer.
Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains MAP application parameters and body.
Param name
string - the child folder name, NULL - current directory.
Param flags
refer to enum bt_obex_req_flags.

void (*set_ntf_reg)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS set notification registration Callback

It is called when receiving set notification registration request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters and use bt_map_mse_get_body to obtain body from RX net buffer.
Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains notification status and body.
Param flags
refer to enum bt_obex_req_flags.

void (*update_inbox)(struct bt_map_mse_mas *mse_mas)

MSE MAS update inbox Callback

It is called when receiving update inbox request.
Param mse_mas
MAP MSE MAS object.

void (*get_mas_inst_info)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS get MAS instance information Callback

It is called when receiving get MAS instance infomation request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.
Param mse_mas
MAP MSE MAS object.

Param buf
RX net buffer contains MAP application parameters.
Param flags
refer to enum bt_obex_req_flags.

void (*`set_owner_status`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS set owner status Callback

It is called when receiving get set owner status request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.
Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains MAP application parameters.
Param flags
refer to enum bt_obex_req_flags.

void (*`get_owner_status`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS get owner status Callback

It is called when receiving get owner status request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.
Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains MAP application parameters.
Param flags
refer to enum bt_obex_req_flags.

void (*`get_convo_listing`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS get conversation listing Callback

It is called when receiving get conversation listing request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters.
Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains MAP application parameters.
Param flags
refer to enum bt_obex_req_flags.

void (*`set_ntf_filter`)(struct bt_map_mse_mas *mse_mas, struct net_buf *buf, enum bt_obex_req_flags flag)

MSE MAS set notification filter Callback

It is called when receiving get set notification filter request. Appplication can use bt_map_mse_app_param_parse to obtain application parameters and use bt_map_mse_get_body to obtain body from RX net buffer.
Param mse_mas
MAP MSE MAS object.
Param buf
RX net buffer contains MAP application parameters.

Param flags
refer to enum bt_obex_req_flags.

struct **bt_map_mse_mns_cb**

#include <map_mse.h> MAP MSE MNS application callback.

Public Members

void (***connected**)(struct bt_map_mse_mns *mse_mns)

MSE MNS connected callback to application

If this callback is provided it will be called whenever the connection completes.

Param mse_mas
MAP MSE MNS object.

void (***disconnected**)(struct bt_map_mse_mns *mse_mns, uint8_t result)

MSE MNS 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 OBEX connection establisment.

Param mse_mas
MAP MSE MNS object.

Param result
BT_MAP_RSP_SUCCESS in case of success or otherwise in case of error.

void (***send_event**)(struct bt_map_mse_mns *mse_mns, uint8_t result)

MSE MNS send event Callback

This callback provides send event result to application.

Param mse_mas
MAP MSE MAS object.

Param result
BT_MAP_RSP_SUCCESS or BT_MAP_RSP_CONTINUE in case of success or otherwise in case of error.

struct **bt_map_ops_get_cont_hdr**

#include <map_types.h>

struct **bt_map_ops_put_cont_hdr**

#include <map_types.h>

struct **map_mce_connect_hdr**

#include <map_types.h>

struct **bt_map_get_folder_listing_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_set_folder_hdr**

#include <map_types.h>

struct **bt_map_get_msg_listing_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_get_msg_hdr**

#include <map_types.h>

struct **name**

struct **type**

struct **bt_map_set_msg_status_hdr**

#include <map_types.h>

struct **name**

struct **type**

struct **bt_map_push_msg_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_set_ntf_reg_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_update_inbox_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_get_mas_inst_info_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_set_owner_status_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_get_owner_status_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_get_convo_listing_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_set_ntf_filter_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_send_event_hdr**

#include <map_types.h>

struct **type**

struct **bt_map_resp_hdr**

#include <map_types.h>

1.10 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_l2cap_chan* struct which contains the callbacks in the *bt_l2cap_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_l2cap_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_l2cap_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_l2cap_server_register()* API passing the *bt_l2cap_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_l2cap_chan_connect()* API and can be disconnected with the *bt_l2cap_chan_disconnect()* API. Note that the later can also disconnect channel instances created by servers.

1.10.1 API Reference

group **bt_l2cap**

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.

Parameters

- **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 (***bt_l2cap_chan_destroy_t**)(struct *bt_l2cap_chan* *chan)

Channel destroy callback.

Param chan
Channel object.

typedef enum *bt_l2cap_chan_state* **bt_l2cap_chan_state_t**

Life-span states of L2CAP CoC channel.

Used only by internal APIs dealing with setting channel to proper state depending on operational context.

A channel enters the *BT_L2CAP_CONNECTING* state upon *bt_l2cap_chan_connect*, *bt_l2cap_ecred_chan_connect* or upon returning from *bt_l2cap_server::accept*.

When a channel leaves the *BT_L2CAP_CONNECTING* state, *bt_l2cap_chan_ops::connected* is called.

typedef enum *bt_l2cap_chan_status* **bt_l2cap_chan_status_t**

Status of L2CAP channel.

Enums

enum **bt_l2cap_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.

A channel enters the *BT_L2CAP_CONNECTING* state upon *bt_l2cap_chan_connect*, *bt_l2cap_ecred_chan_connect* or upon returning from *bt_l2cap_server::accept*.

When a channel leaves the *BT_L2CAP_CONNECTING* state, *bt_l2cap_chan_ops::connected* is called.

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_l2cap_chan_status**

Status of L2CAP channel.

Values:

enumerator **BT_L2CAP_STATUS_OUT**

Channel can send at least one PDU

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**

enum **bt_l2cap_br_psm**

Values:

enumerator **BT_BR_PSM_PBAP_PSE**

enumerator **BT_BR_PSM_MAP_MSE_1**

enumerator **BT_BR_PSM_MAP_MSE_2**

enumerator **BT_BR_PSM_MAP_MCE**

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.

Parameters

- **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, uint16_t mps)

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.
- **mps** – Channel MPS to reconfigure to

Returns

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

int **bt_l2cap_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_l2cap_chan_send**(struct *bt_l2cap_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.

Network buffer fragments (ie `buf->frags`) are not supported.

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 shall reserve *BT_L2CAP_SDU_CHAN_SEND_RESERVE* bytes in the buffer before sending.

The application can use the *BT_L2CAP_SDU_BUF_SIZE()* helper to correctly size the buffer to account for the reserved headroom.

When segmenting an L2CAP SDU into L2CAP PDUs the stack will first attempt to allocate buffers from the channel's `alloc_seg` callback and will fallback on the stack's global buffer pool (sized {CONFIG_BT_L2CAP_TX_BUF_COUNT}).

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.

Warning

The buffer's user_data *will* be overwritten by this function. Do not store anything in it. As soon as a call to this function has been made, consider ownership of user_data transferred into the stack.

Returns

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

-EINVAL if `buf` or `chan` is NULL.

-EINVAL if `chan` is not either BR/EDR or LE credit-based.

-EINVAL if buffer doesn't have enough bytes reserved to fit header.

-EINVAL if buffer's reference counter != 1

-EMSGSIZE if `buf` is larger than `chan`'s MTU.

-ENOTCONN if underlying conn is disconnected.

-ESHUTDOWN if L2CAP channel is disconnected.

-other (from lower layers) if chan is BR/EDR.

int **bt_l2cap_chan_give_credits**(struct *bt_l2cap_chan* *chan, uint16_t additional_credits)

Give credits to the remote.

Only available for channels using bt_l2cap_chan_ops.seg_recv. {CONFIG_BT_L2CAP_SEG_RECV} must be enabled to make this function available.

Each credit given allows the peer to send one segment.

This function depends on a valid `chan` object. Make sure to default-initialize or memset `chan` when allocating or reusing it for new connections.

Adding zero credits is not allowed.

Credits can be given before entering the *BT_L2CAP_CONNECTING* state. Doing so will adjust the 'initial credits' sent in the connection PDU.

Must not be called while the channel is in *BT_L2CAP_CONNECTING* state.

Returns
0 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_l2cap_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.

Public Members

uint16_t **cid**

Endpoint Channel Identifier (CID)

uint16_t **mtu**

Endpoint Maximum Transmission Unit

uint16_t **mps**

Endpoint Maximum PDU payload Size

atomic_t **credits**

Endpoint credits

struct **bt_l2cap_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.

This is the source of the MTU, MPS and credit values when sending L2CAP_LE_CREDIT_BASED_CONNECTION_REQ/RSP and L2CAP_CONFIGURATION_REQ.

uint16_t **pending_rx_mtu**

Pending RX MTU on ECFC reconfigure, used internally by stack

struct *bt_l2cap_le_endpoint* **tx**

Channel Transmission Endpoint.

This is an image of the remote's rx.

The MTU and MPS is controlled by the remote by L2CAP_LE_CREDIT_BASED_CONNECTION_REQ/RSP or L2CAP_CONFIGURATION_REQ.

struct k_fifo **tx_queue**

Channel Transmission queue (for SDUs)

struct **bt_l2cap_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

uint16_t **mps**

Endpoint Maximum PDU payload Size

uint16_t **init_credits**

Endpoint initial credits

atomic_t **credits**

Endpoint credits

struct **bt_l2cap_br_chan**

#include <l2cap.h> BREDR L2CAP Channel structure.

Public Members

struct *bt_l2cap_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 **pending_rx_mtu**

Pending RX MTU on ECFC reconfigure, used internally by stack

uint16_t **pending_rx_mps**

Pending RX MPS on ECFC reconfigure, used internally by stack

uint16_t **psm**

Remote PSM to be connected

uint8_t **ident**

Helps match request context during CoC

struct **bt_l2cap_qos**

#include <l2cap.h> QUALITY OF SERVICE (QOS) OPTION

struct **bt_l2cap_retrans_fc**

#include <l2cap.h> RETRANSMISSION AND FLOW CONTROL OPTION

struct **bt_l2cap_ext_flow_spec**

#include <l2cap.h> EXTENDED FLOW SPECIFICATION OPTION

struct **bt_l2cap_cfg_options**

#include <l2cap.h> L2CAP configuration parameter options.

struct **bt_l2cap_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.

-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.

This callback will be called once the controller marks the SDU as completed. When the controller does so is implementation dependent. It could be after the SDU is enqueued for transmission, or after it is sent on air.
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_l2cap_server**

#include <l2cap.h> L2CAP Server structure.

Public Members

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_server* *server, 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 server
Pointer to the server structure this callback relates to

Param chan
Pointer to received the allocated channel

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

-ENOMEM if no available space for new channel.

-EACCES if application did not authorize the connection.

-EPERM if encryption key size is too short.

1.11 Serial Port Emulation (RFCOMM)

1.11.1 API Reference

group **bt_rfcomm**

RFCOMM.

Typedefs

typedef enum *bt_rfcomm_role* **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**

enumerator **BT_RFCOMM_CHAN_PBAP_PSE**

enumerator **BT_RFCOMM_CHAN_MAP_MSE**

enumerator **BT_RFCOMM_CHAN_MAP_MCE**

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) && (CONFIG_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.

1.12 Service Discovery Protocol (SDP)

1.12.1 API Reference

group **bt_sdp**

Service class identifiers of standard services and service groups

BT_SDP_SDP_SERVER_SVCLASS
Service Discovery Server

BT_SDP_BROWSE_GRP_DESC_SVCLASS
Browse Group Descriptor

BT_SDP_PUBLIC_BROWSE_GROUP
Public Browse Group

BT_SDP_SERIAL_PORT_SVCLASS
Serial Port

BT_SDP_LAN_ACCESS_SVCLASS
LAN Access Using PPP

BT_SDP_DIALUP_NET_SVCLASS
Dialup Networking

BT_SDP_IRMC_SYNC_SVCLASS
IrMC Sync

BT_SDP_OBEX_OBJPUSH_SVCLASS
OBEX Object Push

BT_SDP_OBEX_FILETRANS_SVCLASS
OBEX File Transfer

BT_SDP_IRMC_SYNC_CMD_SVCLASS
IrMC Sync Command

BT_SDP_HEADSET_SVCLASS
Headset

BT_SDP_CORDLESS_TELEPHONY_SVCLASS

Cordless Telephony

BT_SDP_AUDIO_SOURCE_SVCLASS

Audio Source

BT_SDP_AUDIO_SINK_SVCLASS

Audio Sink

BT_SDP_AV_REMOTE_TARGET_SVCLASS

A/V Remote Control Target

BT_SDP_ADVANCED_AUDIO_SVCLASS

Advanced Audio Distribution

BT_SDP_AV_REMOTE_SVCLASS

A/V Remote Control

BT_SDP_AV_REMOTE_CONTROLLER_SVCLASS

A/V Remote Control Controller

BT_SDP_INTERCOM_SVCLASS

Intercom

BT_SDP_FAX_SVCLASS

Fax

BT_SDP_HEADSET_AGW_SVCLASS

Headset AG

BT_SDP_WAP_SVCLASS

WAP

BT_SDP_WAP_CLIENT_SVCLASS

WAP Client

BT_SDP_PANU_SVCLASS

Personal Area Networking User

BT_SDP_NAP_SVCLASS

Network Access Point

BT_SDP_GN_SVCLASS

Group Network

BT_SDP_DIRECT_PRINTING_SVCLASS
Direct Printing

BT_SDP_REFERENCE_PRINTING_SVCLASS
Reference Printing

BT_SDP_IMAGING_SVCLASS
Basic Imaging Profile

BT_SDP_IMAGING_RESPONDER_SVCLASS
Imaging Responder

BT_SDP_IMAGING_ARCHIVE_SVCLASS
Imaging Automatic Archive

BT_SDP_IMAGING_REFOBJS_SVCLASS
Imaging Referenced Objects

BT_SDP_HANDSFREE_SVCLASS
Handsfree

BT_SDP_HANDSFREE_AGW_SVCLASS
Handsfree Audio Gateway

BT_SDP_DIRECT_PRT_REFOBJS_SVCLASS
Direct Printing Reference Objects Service

BT_SDP_REFLECTED_UI_SVCLASS
Reflected UI

BT_SDP_BASIC_PRINTING_SVCLASS
Basic Printing

BT_SDP_PRINTING_STATUS_SVCLASS
Printing Status

BT_SDP_HID_SVCLASS
Human Interface Device Service

BT_SDP_HCR_SVCLASS
Hardcopy Cable Replacement

BT_SDP_HCR_PRINT_SVCLASS
HCR Print

BT_SDP_HCR_SCAN_SVCLASS
HCR Scan

BT_SDP_CIP_SVCLASS
Common ISDN Access

BT_SDP_VIDEO_CONF_GW_SVCLASS
Video Conferencing Gateway

BT_SDP_UDI_MT_SVCLASS
UDI MT

BT_SDP_UDI_TA_SVCLASS
UDI TA

BT_SDP_AV_SVCLASS
Audio/Video

BT_SDP_SAP_SVCLASS
SIM Access

BT_SDP_PBAP_PCE_SVCLASS
Phonebook Access Client

BT_SDP_PBAP_PSE_SVCLASS
Phonebook Access Server

BT_SDP_PBAP_SVCLASS
Phonebook Access

BT_SDP_MAP_MSE_SVCLASS
Message Access Server

BT_SDP_MAP_MCE_SVCLASS
Message Notification Server

BT_SDP_MAP_SVCLASS
Message Access Profile

BT_SDP_GNSS_SVCLASS
GNSS

BT_SDP_GNSS_SERVER_SVCLASS
GNSS Server

BT_SDP_MPS_SC_SVCLASS
MPS SC

BT_SDP_MPS_SVCLASS
MPS

BT_SDP_PNP_INFO_SVCLASS
PnP Information

BT_SDP_GENERIC_NETWORKING_SVCLASS
Generic Networking

BT_SDP_GENERIC_FILETRANS_SVCLASS
Generic File Transfer

BT_SDP_GENERIC_AUDIO_SVCLASS
Generic Audio

BT_SDP_GENERIC_TELEPHONY_SVCLASS
Generic Telephony

BT_SDP_UPNP_SVCLASS
UPnP Service

BT_SDP_UPNP_IP_SVCLASS
UPnP IP Service

BT_SDP_UPNP_PAN_SVCLASS
UPnP IP PAN

BT_SDP_UPNP_LAP_SVCLASS
UPnP IP LAP

BT_SDP_UPNP_L2CAP_SVCLASS
UPnP IP L2CAP

BT_SDP_VIDEO_SOURCE_SVCLASS
Video Source

BT_SDP_VIDEO_SINK_SVCLASS
Video Sink

BT_SDP_VIDEO_DISTRIBUTION_SVCLASS
Video Distribution

BT_SDP_HDP_SVCLASS

HDP

BT_SDP_HDP_SOURCE_SVCLASS

HDP Source

BT_SDP_HDP_SINK_SVCLASS

HDP Sink

BT_SDP_GENERIC_ACCESS_SVCLASS

Generic Access Profile

BT_SDP_GENERIC_ATTRIB_SVCLASS

Generic Attribute Profile

BT_SDP_APPLE_AGENT_SVCLASS

Apple Agent

Attribute identifier codes

Possible values for attribute-id are listed below. See SDP Spec, section "Service Attribute Definitions" for more details.

BT_SDP_ATTR_RECORD_HANDLE

Service Record Handle

BT_SDP_ATTR_SVCLASS_ID_LIST

Service Class ID List

BT_SDP_ATTR_RECORD_STATE

Service Record State

BT_SDP_ATTR_SERVICE_ID

Service ID

BT_SDP_ATTR_PROTO_DESC_LIST

Protocol Descriptor List

BT_SDP_ATTR_BROWSE_GRP_LIST

Browse Group List

BT_SDP_ATTR_LANG_BASE_ATTR_ID_LIST

Language Base Attribute ID List

BT_SDP_ATTR_SVCINFO_TTL
Service Info Time to Live

BT_SDP_ATTR_SERVICE_AVAILABILITY
Service Availability

BT_SDP_ATTR_PROFILE_DESC_LIST
Bluetooth Profile Descriptor List

BT_SDP_ATTR_DOC_URL
Documentation URL

BT_SDP_ATTR_CLNT_EXEC_URL
Client Executable URL

BT_SDP_ATTR_ICON_URL
Icon URL

BT_SDP_ATTR_ADD_PROTO_DESC_LIST
Additional Protocol Descriptor List

BT_SDP_ATTR_GROUP_ID
Group ID

BT_SDP_ATTR_IP_SUBNET
IP Subnet

BT_SDP_ATTR_VERSION_NUM_LIST
Version Number List

BT_SDP_ATTR_SUPPORTED_FEATURES_LIST
Supported Features List

BT_SDP_ATTR_GOEP_L2CAP_PSM
GOEP L2CAP PSM

BT_SDP_ATTR_SVCDB_STATE
Service Database State

BT_SDP_ATTR_MPSD_SCENARIOS
MPSD Scenarios

BT_SDP_ATTR_MPMD_SCENARIOS
MPMD Scenarios

BT_SDP_ATTR_MPS_DEPENDENCIES
Supported Profiles & Protocols

BT_SDP_ATTR_SERVICE_VERSION
Service Version

BT_SDP_ATTR_EXTERNAL_NETWORK
External Network

BT_SDP_ATTR_SUPPORTED_DATA_STORES_LIST
Supported Data Stores List

BT_SDP_ATTR_DATA_EXCHANGE_SPEC
Data Exchange Specification

BT_SDP_ATTR_NETWORK
Network

BT_SDP_ATTR_FAX_CLASS1_SUPPORT
Fax Class 1 Support

BT_SDP_ATTR_REMOTE_AUDIO_VOLUME_CONTROL
Remote Audio Volume Control

BT_SDP_ATTR_MCAP_SUPPORTED_PROCEDURES
MCAP Supported Procedures

BT_SDP_ATTR_FAX_CLASS20_SUPPORT
Fax Class 2.0 Support

BT_SDP_ATTR_SUPPORTED_FORMATS_LIST
Supported Formats List

BT_SDP_ATTR_FAX_CLASS2_SUPPORT
Fax Class 2 Support (vendor-specific)

BT_SDP_ATTR_AUDIO_FEEDBACK_SUPPORT
Audio Feedback Support

BT_SDP_ATTR_NETWORK_ADDRESS
Network Address

BT_SDP_ATTR_WAP_GATEWAY
WAP Gateway

BT_SDP_ATTR_HOMEPAGE_URL
Homepage URL

BT_SDP_ATTR_WAP_STACK_TYPE
WAP Stack Type

BT_SDP_ATTR_SECURITY_DESC
Security Description

BT_SDP_ATTR_NET_ACCESS_TYPE
Net Access Type

BT_SDP_ATTR_MAX_NET_ACCESSRATE
Max Net Access Rate

BT_SDP_ATTR_IP4_SUBNET
IPv4 Subnet

BT_SDP_ATTR_IP6_SUBNET
IPv6 Subnet

BT_SDP_ATTR_SUPPORTED_CAPABILITIES
BIP Supported Capabilities

BT_SDP_ATTR_SUPPORTED_FEATURES
BIP Supported Features

BT_SDP_ATTR_SUPPORTED_FUNCTIONS
BIP Supported Functions

BT_SDP_ATTR_TOTAL_IMAGING_DATA_CAPACITY
BIP Total Imaging Data Capacity

BT_SDP_ATTR_SUPPORTED_REPOSITORIES
Supported Repositories

BT_SDP_ATTR_MAS_INSTANCE_ID
MAS Instance ID

BT_SDP_ATTR_SUPPORTED_MESSAGE_TYPES
Supported Message Types

BT_SDP_ATTR_PBAP_SUPPORTED_FEATURES
PBAP Supported Features

BT_SDP_ATTR_MAP_SUPPORTED_FEATURES
MAP Supported Features

BT_SDP_ATTR_SPECIFICATION_ID
Specification ID

BT_SDP_ATTR_VENDOR_ID
Vendor ID

BT_SDP_ATTR_PRODUCT_ID
Product ID

BT_SDP_ATTR_VERSION
Version

BT_SDP_ATTR_PRIMARY_RECORD
Primary Record

BT_SDP_ATTR_VENDOR_ID_SOURCE
Vendor ID Source

BT_SDP_ATTR_HID_DEVICE_RELEASE_NUMBER
HID Device Release Number

BT_SDP_ATTR_HID_PARSER_VERSION
HID Parser Version

BT_SDP_ATTR_HID_DEVICE_SUBCLASS
HID Device Subclass

BT_SDP_ATTR_HID_COUNTRY_CODE
HID Country Code

BT_SDP_ATTR_HID_VIRTUAL_CABLE
HID Virtual Cable

BT_SDP_ATTR_HID_RECONNECT_INITIATE
HID Reconnect Initiate

BT_SDP_ATTR_HID_DESCRIPTOR_LIST
HID Descriptor List

BT_SDP_ATTR_HID_LANG_ID_BASE_LIST
HID Language ID Base List

BT_SDP_ATTR_HID_SDP_DISABLE

HID SDP Disable

BT_SDP_ATTR_HID_BATTERY_POWER

HID Battery Power

BT_SDP_ATTR_HID_REMOTE_WAKEUP

HID Remote Wakeup

BT_SDP_ATTR_HID_PROFILE_VERSION

HID Profile Version

BT_SDP_ATTR_HID_SUPERVISION_TIMEOUT

HID Supervision Timeout

BT_SDP_ATTR_HID_NORMALLY_CONNECTABLE

HID Normally Connectable

BT_SDP_ATTR_HID_BOOT_DEVICE

HID Boot Device

The Data representation in SDP PDUs (pps 339, 340 of BT SDP Spec)

These are the exact data type+size descriptor values that go into the PDU buffer.

The datatype (leading 5bits) + size descriptor (last 3 bits) is 8 bits. The size descriptor is critical to extract the right number of bytes for the data value from the PDU.

For most basic types, the datatype+size descriptor is straightforward. However for constructed types and strings, the size of the data is in the next "n" bytes following the 8 bits (datatype+size) descriptor. Exactly what the "n" is specified in the 3 bits of the data size descriptor.

TextString and URLString can be of size 2^{8, 16, 32} bytes DataSequence and DataSequenceAlternates can be of size 2^{8, 16, 32} The size are computed post-facto in the API and are not known apriori.

BT_SDP_DATA_NIL

Nil, the null type

BT_SDP_UINT8

Unsigned 8-bit integer

BT_SDP_UINT16

Unsigned 16-bit integer

BT_SDP_UINT32

Unsigned 32-bit integer

BT_SDP_UINT64
Unsigned 64-bit integer

BT_SDP_UINT128
Unsigned 128-bit integer

BT_SDP_INT8
Signed 8-bit integer

BT_SDP_INT16
Signed 16-bit integer

BT_SDP_INT32
Signed 32-bit integer

BT_SDP_INT64
Signed 64-bit integer

BT_SDP_INT128
Signed 128-bit integer

BT_SDP_UUID_UNSPEC
UUID, unspecified size

BT_SDP_UUID16
UUID, 16-bit

BT_SDP_UUID32
UUID, 32-bit

BT_SDP_UUID128
UUID, 128-bit

BT_SDP_TEXT_STR_UNSPEC
Text string, unspecified size

BT_SDP_TEXT_STR8
Text string, 8-bit length

BT_SDP_TEXT_STR16
Text string, 16-bit length

BT_SDP_TEXT_STR32
Text string, 32-bit length

BT_SDP_BOOL

Boolean

BT_SDP_SEQ_UNSPEC

Data element sequence, unspecified size

BT_SDP_SEQ8

Data element sequence, 8-bit length

BT_SDP_SEQ16

Data element sequence, 16-bit length

BT_SDP_SEQ32

Data element sequence, 32-bit length

BT_SDP_ALT_UNSPEC

Data element alternative, unspecified size

BT_SDP_ALT8

Data element alternative, 8-bit length

BT_SDP_ALT16

Data element alternative, 16-bit length

BT_SDP_ALT32

Data element alternative, 32-bit length

BT_SDP_URL_STR_UNSPEC

URL string, unspecified size

BT_SDP_URL_STR8

URL string, 8-bit length

BT_SDP_URL_STR16

URL string, 16-bit length

BT_SDP_URL_STR32

URL string, 32-bit length

Defines

BT_SDP_SERVER_RECORD_HANDLE

BT_SDP_PRIMARY_LANG_BASE

BT_SDP_ATTR_SVCNAME_PRIMARY

BT_SDP_ATTR_SVCDESC_PRIMARY

BT_SDP_ATTR_PROVNAME_PRIMARY

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**

enumerator **BT_SDP_PROTP_OBEX**

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

int **bt_sdp_get_goep_l2cap_psm**(const struct net_buf *buf, uint16_t *l2cap_psm)

Get GoepL2capPsm attribute value.

Helper API extracting remote GoepL2capPsm value.

Parameters

- **buf** – Buffer holding original raw record data from remote.
- **l2cap_psm** – On success object to be populated with GoepL2capPsm mask.

Returns
0 on success if feature found and valid, negative in case any error

int **bt_sdp_get_supported_repositories**(const struct net_buf *buf, uint8_t *supported_repositories)

Get PBAP Supported Repositories attribute value.

Allows if exposed by remote retrieve SupportedRepositories attribute.

Parameters

- **buf** – Buffer holding original raw record data from remote.
- **supported_repositories** – On success object to be populated with Supported Repositories mask.

Returns
0 on success if feature found and valid, negative in case any error

int **bt_sdp_get_pbap_map_ctn_features**(const struct net_buf *buf, uint32_t *features)

Get PBAP/MAP/CTN 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

int **bt_sdp_get_instance_id**(const struct net_buf *buf, uint8_t *id)

Get MASInstanceID/CASInstanceID attribute value.

Helper API extracting remote MASInstanceID/CASInstanceID value.

Parameters

- **buf** – Buffer holding original raw record data from remote.
- **id** – On success object to be populated with MASInstanceID/CASInstanceID mask.

Returns

0 on success if feature found and valid, negative in case any error

int **bt_sdp_get_supported_msg_type**(const struct net_buf *buf, uint8_t *supported_msg_type)

Get SupportedMessageTypes attribute value.

Helper API extracting remote SupportedMessageTypes value.

Parameters

- **buf** – Buffer holding original raw record data from remote.
- **supported_msg_type** – On success object to be populated with SupportedMessageTypes mask.

Returns

0 on success if feature found and valid, negative in case any error

int **bt_sdp_get_service_name**(const struct net_buf *buf, const char **name)

Get ServiceName attribute value.

Helper API extracting remote ServiceName value.

Parameters

- **buf** – Buffer holding original raw record data from remote.
- **name** – On success pointer to be populated with ServiceName 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.

Public Members

uint8_t **type**

Type of the data element

uint32_t **data_size**

Size of the data element

uint32_t **total_size**

Total size of the data element

struct **bt_sdp_attribute**

#include <*sdp.h*> SDP Attribute Value.

Public Members

uint16_t **id**

Attribute ID

struct *bt_sdp_data_elem* **val**

Attribute data

struct **bt_sdp_record**

#include <*sdp.h*> SDP Service Record Value.

Public Members

uint32_t **handle**

Redundant, for quick ref

struct *bt_sdp_attribute* ***attrs**

Base addr of attr array

size_t **attr_count**

Number of attributes

uint8_t **index**

Index of the record in LL

struct *bt_sdp_record* ***next**

Next service record

struct **bt_sdp_client_result**

#include <*sdp.h*> Generic SDP Client Query Result data holder.

Public Members

struct net_buf ***resp_buf**

buffer containing unparsed SDP record result for given UUID

bool **next_record_hint**

flag pointing that there are more result chunks for given UUID

const struct *bt_uuid* ***uuid**

Reference to UUID object on behalf one discovery was started

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

1.13 Advance Audio Distribution Profile (A2DP)

1.13.1 API Reference

group **bt_a2dp**

Advance Audio Distribution Profile (A2DP)

Defines

BT_A2DP_SBC_IE_LENGTH

SBC IE length

BT_A2DP_SBC_MEDIA_HDR_NUM_FRAMES_GET(hdr)

If the F bit is set to 0, this field indicates the number of frames contained in this packet. If the F bit is set to 1, this field indicates the number of remaining fragments, including the current fragment. Therefore, the last counter value shall be one.

BT_A2DP_SBC_MEDIA_HDR_L_GET(hdr)

Set to 1 for the last packet of a fragmented SBC frame, otherwise set to 0.

BT_A2DP_SBC_MEDIA_HDR_S_GET(hdr)

Set to 1 for the starting packet of a fragmented SBC frame, otherwise set to 0.

BT_A2DP_SBC_MEDIA_HDR_F_GET(hdr)

Set to 1 if the SBC frame is fragmented, otherwise set to 0.

BT_A2DP_SBC_MEDIA_HDR_NUM_FRAMES_SET(hdr, val)

If the F bit is set to 0, this field indicates the number of frames contained in this packet. If the F bit is set to 1, this field indicates the number of remaining fragments, including the current fragment. Therefore, the last counter value shall be one.

BT_A2DP_SBC_MEDIA_HDR_L_SET(hdr, val)

Set to 1 for the last packet of a fragmented SBC frame, otherwise set to 0.

BT_A2DP_SBC_MEDIA_HDR_S_SET(hdr, val)

Set to 1 for the starting packet of a fragmented SBC frame, otherwise set to 0.

BT_A2DP_SBC_MEDIA_HDR_F_SET(hdr, val)

Set to 1 if the SBC frame is fragmented, otherwise set to 0.

BT_A2DP_SBC_MEDIA_HDR_ENCODE(num_frames, l, s, f)

construct sbc header.

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

Stream End Point Role.

Values:

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 funciton 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 l2cap 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 configufed by bt_a2dp_configure_endpoint. If user stops to discover more peer endpoins, 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.

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.

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

Public Members

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.

Public Members

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. If CONFIG_A2DP_CODEC_EXTERNAL is set: if SBC, data's format is BT_AVDTP_MEDIA_HDR_SIZE + 1 byte SBC payload header + SBC media frames; if other codecs, data's format is BT_AVDTP_MEDIA_HDR_SIZE + codec media data. If CONFIG_A2DP_CODEC_EXTERNAL is not set: if SBC, data's format is PCM data; if other codecs, data's format is BT_AVDTP_MEDIA_HDR_SIZE + codec media data because stack doesn't support other codecs' encoder/decoder.

Param length
the data length.

struct **bt_a2dp_connect_cb**

#include <a2dp.h> 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.

Public Members

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)

1.14 Serial Port Profile (SPP)

1.14.1 API Reference

group **bt_spp**

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.

int **bt_spp_client_connect**(struct bt_conn *conn, uint8_t channel, *bt_spp_callback* *cb, struct bt_spp **spp)

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) && (CONFIG_BT_SPP_ENABLE_CONTROL_CMD > 0))

struct **discover_cb_t**
#include <*spp.h*> bt_spp_discover callback parameter

1.15 Audio/Video Remote Control Profile (AVRCP)

1.15.1 API Reference

group **bt_avrcp**
Audio/Video Remote Control Profile (AVRCP)

Defines

BT_AVRCP_COVER_ART_PSM
AVRCP cover art PSM

BT_AVRCP_SUBUNIT_TYPE_UNIT
AVRCP SubUnit Types

BT_AVRCP_SUBUNIT_TYPE_PANEL

BT_AVRCP_OPCODE_UNIT_INFO
AVRCP Command Opcode Types

BT_AVRCP_OPCODE_SUBUNIT_INFO

BT_AVRCP_OPCODE_PASS_THROUGH

BT_AVRCP_OPCODE_VENDOR_DEPENDENT

BT_AVRCP_OP_ID_SELECT
AVRCP operation ID PASS THROUGH

BT_AVRCP_OP_ID_UP

BT_AVRCP_OP_ID_DOWN

BT_AVRCP_OP_ID_LEFT

BT_AVRCP_OP_ID_RIGHT

BT_AVRCP_OP_ID_RIGHT_UP

BT_AVRCP_OP_ID_RIGHT_DOWN

BT_AVRCP_OP_ID_LEFT_UP

BT_AVRCP_OP_ID_LEFT_DOWN

BT_AVRCP_OP_ID_ROOT_MENU

BT_AVRCP_OP_ID_SETUP_MENU

BT_AVRCP_OP_ID_CONTENTS_MENU

BT_AVRCP_OP_ID_FAVORITE_MENU

BT_AVRCP_OP_ID_EXIT

BT_AVRCP_OP_ID_0

BT_AVRCP_OP_ID_1

BT_AVRCP_OP_ID_2

BT_AVRCP_OP_ID_3

BT_AVRCP_OP_ID_4

BT_AVRCP_OP_ID_5

BT_AVRCP_OP_ID_6

BT_AVRCP_OP_ID_7

BT_AVRCP_OP_ID_8

BT_AVRCP_OP_ID_9

BT_AVRCP_OP_ID_DOT

BT_AVRCP_OP_ID_ENTER

BT_AVRCP_OP_ID_CLEAR

BT_AVRCP_OP_ID_CHANNEL_UP

BT_AVRCP_OP_ID_CHANNEL_DOWN

BT_AVRCP_OP_ID_PREVIOUS_CHANNEL

BT_AVRCP_OP_ID_SOUND_SELECT

BT_AVRCP_OP_ID_INPUT_SELECT

BT_AVRCP_OP_ID_DISPLAY_INFORMATION

BT_AVRCP_OP_ID_HELP

BT_AVRCP_OP_ID_PAGE_UP

BT_AVRCP_OP_ID_PAGE_DOWN

BT_AVRCP_OP_ID_POWER

BT_AVRCP_OP_ID_VOLUME_UP

BT_AVRCP_OP_ID_VOLUME_DOWN

BT_AVRCP_OP_ID_MUTE

BT_AVRCP_OP_ID_PLAY

BT_AVRCP_OP_ID_STOP

BT_AVRCP_OP_ID_PAUSE

BT_AVRCP_OP_ID_RECORD

BT_AVRCP_OP_ID_REWIND

BT_AVRCP_OP_ID_FAST_FORWARD

BT_AVRCP_OP_ID_EJECT

BT_AVRCP_OP_ID_FORWARD

BT_AVRCP_OP_ID_BACKWARD

BT_AVRCP_OP_ID_ANGLE

BT_AVRCP_OP_ID_SUBPICTURE

BT_AVRCP_OP_ID_F1

BT_AVRCP_OP_ID_F2

BT_AVRCP_OP_ID_F3

BT_AVRCP_OP_ID_F4

BT_AVRCP_OP_ID_F5

BT_AVRCP_OP_ID_VENDOR_UNIQUE

BT_AVRCP_PDU_ID_GET_CAPABILITY

VENDOR DEPENDENT Transfer PDU ID

BT_AVRCP_PDU_ID_LIST_PLAYER_APP_SETTING_ATTR

BT_AVRCP_PDU_ID_LIST_PLAYER_APP_SETTING_VAL

BT_AVRCP_PDU_ID_GET_CUR_PLAYER_APP_SETTING_VAL

BT_AVRCP_PDU_ID_SET_PLAYER_APP_SETTING_VAL

BT_AVRCP_PDU_ID_GET_PLAYER_APP_SETTING_ATTR_TXT

BT_AVRCP_PDU_ID_GET_PLAYER_APP_SETTING_VAL_TXT

BT_AVRCP_PDU_ID_INFORM_DISPLAYABLE_CHAR_SET

BT_AVRCP_PDU_ID_INFORM_BATTERY_STATUS

BT_AVRCP_PDU_ID_GET_ELEMENT_ATTRIBUTE

BT_AVRCP_PDU_ID_GET_PLAY_STATUS

BT_AVRCP_PDU_ID_REGISTER_NOTIFICATION

BT_AVRCP_PDU_ID_REQUEST_CONTINUING_RESPONSE

BT_AVRCP_PDU_ID_ABORT_CONTINUING_RESPONSE

BT_AVRCP_PDU_ID_SET_ABSOLUTE_VOLUME

BT_AVRCP_PDU_ID_SET_ADDRESSED_PLAYER

BT_AVRCP_PDU_ID_SET_BROWSED_PLAYER

BT_AVRCP_PDU_ID_GET_FOLDER_ITEMS

BT_AVRCP_PDU_ID_CHANGE_PATH

BT_AVRCP_PDU_ID_GET_ITEM_ATTRIBUTES

BT_AVRCP_PDU_ID_PLAY_ITEMS

BT_AVRCP_PDU_ID_SEARCH

BT_AVRCP_PDU_ID_ADD_TO_NOW_PLAYING

BT_AVRCP_PDU_ID_GENERAL_REJECT

BT_AVRCP_PDU_ID_GET_TOTAL_NUM_ITEMS

BT_AVRCP_CAP_ID_COMPANY_ID
AVRCP Capability.

BT_AVRCP_CAP_ID_EVENTS_SUPPORTED

BT_AVRCP_RESPONSE_TYPE_NOT_IMPLEMENTED
AVRCP Message Response Types.

BT_AVRCP_RESPONSE_TYPE_ACCEPTED

BT_AVRCP_RESPONSE_TYPE_REJECTED

BT_AVRCP_RESPONSE_TYPE_INTERIM

BT_AVRCP_RESPONSE_TYPE_STABLE

BT_AVRCP_RESPONSE_TYPE_CHANGED

BT_AVRCP_PACKET_TYPE_SINGLE
AVRCP PDU Packet Type.

BT_AVRCP_PACKET_TYPE_START

BT_AVRCP_PACKET_TYPE_CONTINUE

BT_AVRCP_PACKET_TYPE_END

BT_AVRCP_EVENT_PLAYBACK_STATUS_CHANGED
AVRCP Events.

BT_AVRCP_EVENT_TRACK_CHANGED

BT_AVRCP_EVENT_TRACK_REACHED_END

BT_AVRCP_EVENT_TRACK_REACHED_START

BT_AVRCP_EVENT_PLAYBACK_POS_CHANGED

BT_AVRCP_EVENT_BATT_STATUS_CHANGED

BT_AVRCP_EVENT_SYSTEM_STATUS_CHANGED

BT_AVRCP_EVENT_PLAYER_APP_SETTING_CHANGED

BT_AVRCP_EVENT_NOW_PLAYING_CONTENT_CHANGED

BT_AVRCP_EVENT_AVAILABLE_PLAYER_CHANGED

BT_AVRCP_EVENT_ADDRESSED_PLAYER_CHANGED

BT_AVRCP_EVENT_UIDS_CHANGED

BT_AVRCP_EVENT_VOLUME_CHANGED

BT_AVRCP_PLAYER_APP_ATTR_ID_EQUALIZER
AVRCP Player Application Setting Attribute ID.

BT_AVRCP_PLAYER_APP_ATTR_ID_REPEAT

BT_AVRCP_PLAYER_APP_ATTR_ID_SHUFFLE

BT_AVRCP_PLAYER_APP_ATTR_ID_SCAN

BT_AVRCP_MEDIA_ATTR_ID_MEDIA_TITLE
AVRCP Metadata Attribute ID.

BT_AVRCP_MEDIA_ATTR_ID_ARTIST_NAME

BT_AVRCP_MEDIA_ATTR_ID_ALBUM_NAME

BT_AVRCP_MEDIA_ATTR_ID_MEDIA_NUMBER

BT_AVRCP_MEDIA_ATTR_ID_TOTAL_MEDIA_NUMBER

BT_AVRCP_MEDIA_ATTR_ID_GENRE

BT_AVRCP_MEDIA_ATTR_ID_PLAYING_TIME

BT_AVRCP_MEDIA_ATTR_ID_DEFAULT_COVER_ART

BT_AVRCP_SYSTEM_STATUS_POWER_ON
AVRCP System Status Code.

BT_AVRCP_SYSTEM_STATUS_POWER_OFF

BT_AVRCP_SYSTEM_STATUS_UNPLUGED

BT_AVRCP_BATTERY_STATUS_NORMAL
AVRCP Battery Status Code.

BT_AVRCP_BATTERY_STATUS_WARNING

BT_AVRCP_BATTERY_STATUS_CRITICAL

BT_AVRCP_BATTERY_STATUS_EXTERNAL

BT_AVRCP_BATTERY_STATUS_FULL

BT_AVRCP_METADATA_ERROR_INVALID_COMMAND
AVRCP Error Status Code.

BT_AVRCP_METADATA_ERROR_INVALID_PARAMETER

BT_AVRCP_METADATA_ERROR_PARAMETER_NOT_FOUND

BT_AVRCP_METADATA_ERROR_INTERNAL

BT_AVRCP_METADATA_ERROR_OPERATION_SUCCESSFUL

BT_AVRCP_METADATA_ERROR_UID_CHANGED

BT_AVRCP_COMMAND_TYPE_CONTROL
AVRCP Message Command Types.

BT_AVRCP_COMMAND_TYPE_STATUS

BT_AVRCP_COMMAND_TYPE_NOTIFY

BT_AVRCP_SCOPE_MEDIA_PLAYER_LIST
AVRCP Scopes.

BT_AVRCP_SCOPE_VIRTUAL_FILESYSTEM

BT_AVRCP_SCOPE_SEARCH

BT_AVRCP_SCOPE_NOW_PLAYING

BT_AVRCP_BOW_ERROR_INVALID_DIRECTION

AVRCP Browsing Error Status Code.

BT_AVRCP_BOW_ERROR_NOT_A_DIRECTORY

BT_AVRCP_BOW_ERROR_DOES_NOT_EXIST

BT_AVRCP_BOW_ERROR_INVALID_SCOPE

BT_AVRCP_BOW_ERROR_RANGE_OUT_OF_BOUNDS

BT_AVRCP_BOW_ERROR_DIRECTORY_NOT_HANDLED

BT_AVRCP_BOW_ERROR_MEDIA_IN_USE

BT_AVRCP_BOW_ERROR_NOW_PLAYING_LIST_FULL

BT_AVRCP_BOW_ERROR_SEARCH_NOT_SUPPORTED

BT_AVRCP_BOW_ERROR_SEARCH_IN_PROGRESS

BT_AVRCP_BOW_ERROR_INVALID_PLAYER_ID

BT_AVRCP_BOW_ERROR_PLAYER_NOT_BROWSABLE

BT_AVRCP_BOW_ERROR_PLAYER_NOT_ADDRESSED

BT_AVRCP_BOW_ERROR_NO_VALID_SEARCH_RESULTS

BT_AVRCP_BOW_ERROR_NO_AVAILABLE_PLAYERS

BT_AVRCP_BOW_ERROR_ADDR_PLAYER_CHANGED

BT_AVRCP_ITEM_TYPER_MEDIA_PLAYER

Browsable Item types

BT_AVRCP_ITEM_TYPER_FOLDER

BT_AVRCP_ITEM_TYPER_MEDIA

BT_AVRCP_CA_SUCCESS_RSP

AVRCP Cover Art OBEX Response Codes

BT_AVRCP_CA_CONTINUE_RSP

BT_AVRCP_CA_BAD_REQ_RSP

BT_AVRCP_CA_NOT_IMPLEMENTED_RSP

BT_AVRCP_CA_UNAUTH_RSP

BT_AVRCP_CA_PRECOND_FAILED_RSP

BT_AVRCP_CA_NOT_FOUND_RSP

BT_AVRCP_CA_NOT_ACCEPTABLE_RSP

BT_AVRCP_CA_NO_SERVICE_RSP

BT_AVRCP_CA_FORBIDDEN_RSP

BT_AVRCP_CA_SERVER_ERROR

Functions

int **bt_avrcp_register_callback**(struct *bt_avrcp_cb* *cb)

register callbacks to enable avrcp and monitor avrcp states.

Parameters

- **cb** – The callback structure.

Returns

0 in case of success and error code in case of error.

int **bt_avrcp_control_connect**(struct bt_conn *conn)

Create the AVRCP control l2cap connection.

The sender will be notified by the registered callback (control_connected).

Parameters

- **conn** – connection object.

Returns

0 in case of success and error code in case of error.

int **bt_avrcp_control_disconnect**(struct bt_conn *conn)

Release the AVRCP control l2cap connection.

The sender will be notified by the registered callback (control_disconnected).

Parameters

- **conn** – connection object.

Returns

0 in case of success and error code in case of error.

struct **bt_avrcp_header**

#include <avrcp.h> avrcp message header

Public Members

uint8_t **ctype_response**

profile identifier (it is omitted) command type or response code

uint8_t **subunit_id**

subunit id

uint8_t **subunit_type**

subunit type

uint8_t **op_code**

operation code

union **__unnamed__**

Public Members

struct *bt_avrcp_header*.[anonymous].[anonymous] **[anonymous]**

uint8_t **tl_pt_cr_ipid**

struct **__unnamed__**

Public Members

uint8_t **ipid**
IPID

uint8_t **cr**
command/response

uint8_t **packet_type**
packet type

uint8_t **tl**
transaction label

struct **bt_avrcp_unit_info**
#include <*avrcp.h*> unit info

struct **bt_avrcp_subunit_info**
#include <*avrcp.h*> subunit info

struct **bt_avrcp_pass_through**
#include <*avrcp.h*> pass through message

Public Members

uint8_t **op_id**
operation id

uint8_t **state_flag**
state flag

uint8_t **op_data_len**
operation data length

uint8_t **op_data**[0]
operation data

struct **bt_avrcp_register_ntfy**
#include <*avrcp.h*> register notification

Public Members

uint8_t **event_id**

event id

uint32_t **playback_interval**

Playback interval

struct **bt_avrcp_player_app_setting_attr_ids**

#include <avrcp.h> player application setting attribute IDs

Public Members

uint8_t **num_of_attr**

number of attributes

uint8_t **attr_ids**[0]

attribute IDs

struct **bt_avrcp_player_attr_value**

#include <avrcp.h> attribute value

struct **bt_avrcp_player_app_attr_values**

#include <avrcp.h> player application values

Public Members

uint8_t **num_of_attr**

number of attributes

struct *bt_avrcp_player_attr_value* **attr_vals**[0]

attribute value

struct **bt_avrcp_get_player_app_setting_value_text**

#include <avrcp.h> get player application setting values text

Public Members

uint8_t **num_of_value**
number of attributes

uint8_t **value_ids**[0]
attribute IDs

struct **bt_avrcp_inform_displayable_char_set**
#include <*avrcp.h*> inform displayable character set

struct **bt_avrcp_get_element_attrs**
#include <*avrcp.h*> get element attributes

Public Members

uint8_t **identifier**[8]
PLAYING (0x0)

struct **bt_avrcp_add_to_now_playing**
#include <*avrcp.h*> AddToNowPlaying.

struct **bt_avrcp_play_item**
#include <*avrcp.h*> PlayItem.

struct **bt_avrcp_company_id**
#include <*avrcp.h*> company id

Public Members

uint8_t **company_id0**
company id

struct **bt_avrcp_capability_company_id**
#include <*avrcp.h*> capability id

Public Members

uint8_t **capability_id**
capability id

uint8_t **capability_count**
capability count

struct *bt_avrcp_company_id* **company_ids**[0]
capability IDs

struct **bt_avrcp_capability_events_supported**
#include <avrcp.h> capability event supports

Public Members

uint8_t **capability_id**
capability id

uint8_t **capability_count**
capability count

uint8_t **event_ids**[0]
events supported

struct **bt_avrcp_player_app_setting_values**
#include <avrcp.h> player application setting attribute IDs

Public Members

uint8_t **num_of_value**
number of attributes

uint8_t **value_ids**[0]
attribute IDs

struct **bt_avrcp_player_attr_value_text**
#include <avrcp.h> attribute/value text

struct **bt_avrcp_player_get_txt_rsp**
#include <avrcp.h> get text response

struct **bt_avrcp_element_attr**
#include <avrcp.h> element attribute

struct **bt_avrcp_player_get_element_attr_rsp**

#include <*avrcp.h*> get element attribute response

struct **bt_avrcp_play_status_rsp**

#include <*avrcp.h*> play status response

struct **bt_avrcp_event_rsp**

#include <*avrcp.h*> event response data format

union **__unnamed__**

Public Members

uint8_t **play_status**

EVENT_PLAYBACK_STATUS_CHANGED

uint8_t **identifier**[8]

EVENT_TRACK_CHANGED

uint32_t **playback_pos**

EVENT_PLAYBACK_POS_CHANGED

uint8_t **battery_status**

EVENT_BATT_STATUS_CHANGED

uint8_t **system_status**

EVENT_SYSTEM_STATUS_CHANGED

struct *bt_avrcp_event_rsp*.[anonymous].[anonymous] **setting_changed**

EVENT_PLAYER_APPLICATION_SETTING_CHANGED

struct *bt_avrcp_event_rsp*.[anonymous].[anonymous] **addressed_player_changed**

EVENT_ADDRESSED_PLAYER_CHANGED

uint16_t **uid_counter**

EVENT_UIDS_CHANGED

uint8_t **absolute_volume**

EVENT_VOLUME_CHANGED

struct **setting_changed**

EVENT_PLAYER_APPLICATION_SETTING_CHANGED

struct **addressed_player_changed**

EVENT_ADDRESSED_PLAYER_CHANGED

struct **bt_avrcp_vendor_header**

#include <*avrcp.h*> vendor dependent message header

Public Members

uint32_t **company_id**

company id

uint8_t **pdu_id**

pdu id

uint8_t **packet_type**

packet type. It's value can be BT_AVRCP_PACKET_TYPE_SINGLE BT_AVRCP_PACKET_TYPE_START BT_AVRCP_PACKET_TYPE_CONTINUE BT_AVRCP_PACKET_TYPE_END

uint16_t **parameter_len**

parameter length

struct **bt_avrcp_vendor**

#include <*avrcp.h*> vendor dependent message

Public Members

uint8_t **company_id0**

company id

uint8_t **pdu_id**

pdu id

uint8_t **packet_type**

packet type

uint8_t **reserved**

reserved

uint16_t **parameter_len**

parameter length

union **__unnamed__**

Public Members

uint8_t **parameter**

parameters

struct *bt_avrcp_player_app_setting_attr_ids* **player_attr_ids**

struct *bt_avrcp_player_app_attr_values* **player_attr_values**

struct *bt_avrcp_get_player_app_setting_value_text* **player_value_text**

struct *bt_avrcp_inform_displayable_char_set* **inform_char_set**

struct *bt_avrcp_get_element_attrs* **element_attrs**

struct *bt_avrcp_play_item* **play_item**

struct *bt_avrcp_add_to_now_playing* **add_now_play**

struct *bt_avrcp_register_ntfy* **register_notify**

struct *bt_avrcp_capability_company_id* **comapny_id_rsp**

struct *bt_avrcp_capability_events_supported* **events_suported_rsp**

struct *bt_avrcp_player_app_setting_attr_ids* **attr_ids_rsp**

struct *bt_avrcp_player_app_setting_values* **values_rsp**

struct *bt_avrcp_player_app_attr_values* **attr_values_rsp**

struct *bt_avrcp_player_get_txt_rsp* **get_txt_rsp**

struct *bt_avrcp_player_get_element_attr_rsp* **element_attr_rsp**

struct *bt_avrcp_play_status_rsp* **play_status_rsp**

struct *bt_avrcp_event_rsp* **event_rsp**

struct **bt_avrcp_control_msg**

#include <avrcp.h> avrcp control message

union **__unnamed__**

Public Members

struct *bt_avrcp_unit_info* **unit_info**

struct *bt_avrcp_subunit_info* **subunit_info**

struct *bt_avrcp_pass_through* **pass_th**

struct *bt_avrcp_vendor* **vendor**

struct **bt_avrcp_browsing_header**

#include <avrcp.h> browsing message header

Public Members

uint8_t **pdu_id**

profile identifier (it is omitted) pdu id

uint16_t **parameter_len**

parameter length

union **__unnamed__**

Public Members

struct *bt_avrcp_browsing_header*.[anonymous].[anonymous] **[anonymous]**

uint8_t **tl_pt_cr_ipid**

struct **__unnamed__**

Public Members

uint8_t **ipid**

IPID

uint8_t **cr**

command/response

uint8_t **packet_type**

packet type

uint8_t **tl**

transaction label

struct **bt_avrcp_get_folder_items_cmd**

#include <*avrcp.h*> get folder items player list

Public Members

uint32_t **attr_list**[0]

Appendix E in AVRCP spec every attribute id is 4 bytes.

struct **bt_avrcp_set_browsed_player_cmd**

#include <*avrcp.h*> set browsed player

struct **bt_avrcp_change_path_cmd**

#include <*avrcp.h*> change path

struct **bt_avrcp_get_item_attrs_cmd**

#include <*avrcp.h*> get item attributes

Public Members

uint32_t **attr_list**[0]

Appendix E in AVRCP spec every attribute id is 4 bytes.

struct **bt_avrcp_search_cmd**

#include <*avrcp.h*> search

struct **bt_avrcp_get_total_num_of_items_cmd**

#include <*avrcp.h*> get total number of items

struct **bt_avrcp_browsing_cmd**

#include <*avrcp.h*> avrcp browsing command

union **__unnamed__**

Public Members

struct *bt_avrcp_get_folder_items_cmd* **folder_items**

struct *bt_avrcp_set_browsed_player_cmd* **set_browsed_player**

struct *bt_avrcp_change_path_cmd* **change_path**

struct *bt_avrcp_get_item_attrs_cmd* **get_item_attrs**

struct *bt_avrcp_search_cmd* **search**

struct *bt_avrcp_get_total_num_of_items_cmd* **get_total_num_of_items**

struct **bt_avrcp_player_item**

#include <*avrcp.h*> player item

struct **bt_avrcp_folder_item**

#include <*avrcp.h*> folder item

struct **bt_avrcp_attr_val_entry**

#include <*avrcp.h*> media item attribute

struct **bt_avrcp_media_item**

#include <*avrcp.h*> media item

struct **bt_avrcp_item**

#include <*avrcp.h*> avrcp item

union **__unnamed__**

Public Members

struct *bt_avrcp_player_item* **player_item**

struct *bt_avrcp_folder_item* **folder_item**

struct *bt_avrcp_media_item* **media_item**

struct **bt_avrcp_get_folder_items_rsp**

#include <*avrcp.h*> get folder items (response)

struct **bt_avrcp_folder_name**

#include <avrcp.h> get folder name (response)

struct **bt_avrcp_set_browsed_player_rsp**

#include <avrcp.h> set browsed player (response)

struct **bt_avrcp_change_path_rsp**

#include <avrcp.h> change path (response)

struct **bt_avrcp_get_item_attrs_rsp**

#include <avrcp.h> get item attributes (response)

Public Members

struct *bt_avrcp_attr_val_entry* **attrs**[0]

Appendix E in AVRCP spec every attribute id is 4 bytes.

struct **bt_avrcp_search_rsp**

#include <avrcp.h> search (response)

struct **bt_avrcp_get_total_num_of_items_rsp**

#include <avrcp.h> get total number of items (response)

struct **bt_avrcp_browsing_rsp**

#include <avrcp.h> browsing message (response)

union **__unnamed__**

Public Members

struct *bt_avrcp_get_folder_items_rsp* **folder_items**

struct *bt_avrcp_set_browsed_player_rsp* **set_browsed_player**

struct *bt_avrcp_change_path_rsp* **change_path**

struct *bt_avrcp_get_item_attrs_rsp* **get_item_attrs**

struct *bt_avrcp_search_rsp* **search**

struct *bt_avrcp_get_total_num_of_items_rsp* **get_total_num_of_items**

struct **bt_avrcp_cb**

#include <avrcp.h> The connecting callback.

Public Members

void (*`control_connected`)(struct bt_conn *conn, int err)

avrcp control connection has been established.

This callback notifies the application of avrcp control connection.

Param conn
connection object.

Param err
error code.

void (*`control_disconnected`)(struct bt_conn *conn, int err)

avrcp control connection has been disconnected.

This callback notifies the application of avrcp control disconnection.

Param conn
connection object.

void (*`browsing_connected`)(struct bt_conn *conn, int err)

avrcp browsing connection has been established.

This callback notifies the application of avrcp browsing connection.

Param conn
connection object.

Param err
error code.

void (*`browsing_disconnected`)(struct bt_conn *conn, int err)

avrcp browsing connection has been disconnected.

This callback notifies the application of avrcp browsing disconnection.

Param conn
connection object.

Param err
error code.

void (*`send_result`)(struct bt_conn *conn, int err)

avrcp send result callback.

This callback notifies the application that a avrcp command send result.

Param conn
connection object.

Param err
error code.

1.16 Universal Unique Identifiers (UUIDs)

1.16.1 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

Current Time Service.

BT_UUID_RTUS_VAL

Reference Time Update Service UUID value.

BT_UUID_RTUS

Reference Time Update Service.

BT_UUID_NDSTS_VAL

Next DST Change Service UUID value.

BT_UUID_NDSTS

Next DST Change Service.

BT_UUID_GS_VAL

Glucose Service UUID value.

BT_UUID_GS

Glucose 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_NAS_VAL

Network Availability Service UUID value.

BT_UUID_NAS

Network Availability Service.

BT_UUID_WDS_VAL

Watchdog Service UUID value.

BT_UUID_WDS

Watchdog Service.

BT_UUID_HRS_VAL

Heart Rate Service UUID value.

BT_UUID_HRS

Heart Rate Service.

BT_UUID_PAS_VAL

Phone Alert Service UUID value.

BT_UUID_PAS

Phone Alert Service.

BT_UUID_BAS_VAL

Battery Service UUID value.

BT_UUID_BAS

Battery Service.

BT_UUID_BPS_VAL

Blood Pressure Service UUID value.

BT_UUID_BPS

Blood Pressure Service.

BT_UUID_ANS_VAL

Alert Notification Service UUID value.

BT_UUID_ANS

Alert Notification Service.

BT_UUID_HIDS_VAL

HID Service UUID value.

BT_UUID_HIDS

HID Service.

BT_UUID_SPS_VAL

Scan Parameters Service UUID value.

BT_UUID_SPS

Scan Parameters Service.

BT_UUID_RSCS_VAL

Running Speed and Cadence Service UUID value.

BT_UUID_RSCS

Running Speed and Cadence Service.

BT_UUID_AIOS_VAL

Automation IO Service UUID value.

BT_UUID_AIOS

Automation IO Service.

BT_UUID_CSC_VAL

Cycling Speed and Cadence Service UUID value.

BT_UUID_CSC

Cycling Speed and Cadence Service.

BT_UUID_CPS_VAL

Cycling Power Service UUID value.

BT_UUID_CPS

Cycling Power Service.

BT_UUID_LNS_VAL

Location and Navigation Service UUID value.

BT_UUID_LNS

Location and Navigation Service.

BT_UUID_ESS_VAL

Environmental Sensing Service UUID value.

BT_UUID_ESS

Environmental Sensing Service.

BT_UUID_BCS_VAL

Body Composition Service UUID value.

BT_UUID_BCS

Body Composition Service.

BT_UUID_UDS_VAL

User Data Service UUID value.

BT_UUID_UDS

User Data Service.

BT_UUID_WSS_VAL

Weight Scale Service UUID value.

BT_UUID_WSS

Weight Scale Service.

BT_UUID_BMS_VAL

Bond Management Service UUID value.

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_IPS_VAL

Indoor Positioning Service UUID value.

BT_UUID_IPS

Indoor Positioning Service.

BT_UUID_POS_VAL

Pulse Oximeter Service UUID value.

BT_UUID_POS

Pulse Oximeter Service.

BT_UUID_HPS_VAL

HTTP Proxy Service UUID value.

BT_UUID_HPS

HTTP Proxy Service.

BT_UUID_TDS_VAL

Transport Discovery Service UUID value.

BT_UUID_TDS

Transport Discovery Service.

BT_UUID_OTS_VAL

Object Transfer Service UUID value.

BT_UUID_OTS

Object Transfer Service.

BT_UUID_FMS_VAL

Fitness Machine Service UUID value.

BT_UUID_FMS

Fitness Machine 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_MESH_PROXY_SOLICITATION_VAL

Proxy Solicitation UUID value.

BT_UUID_RCSRV_VAL

Reconnection Configuration Service UUID value.

BT_UUID_RCSRV

Reconnection Configuration Service.

BT_UUID_IDS_VAL

Insulin Delivery Service UUID value.

BT_UUID_IDS

Insulin Delivery Service.

BT_UUID_BSS_VAL

Binary Sensor Service UUID value.

BT_UUID_BSS

Binary Sensor Service.

BT_UUID_ECS_VAL

Emergency Configuration Service UUID value.

BT_UUID_ECS

Emergency Configuration Service.

BT_UUID_ACLS_VAL

Authorization Control Service UUID value.

BT_UUID_ACLS

Authorization Control Service.

BT_UUID_PAMS_VAL

Physical Activity Monitor Service UUID value.

BT_UUID_PAMS

Physical Activity Monitor Service.

BT_UUID_AICS_VAL

Audio Input Control Service UUID value.

BT_UUID_AICS

Audio Input Control Service.

BT_UUID_VCS_VAL

Volume Control Service UUID value.

BT_UUID_VCS

Volume Control Service.

BT_UUID_VOCS_VAL

Volume Offset Control Service UUID value.

BT_UUID_VOCS

Volume Offset Control Service.

BT_UUID_CSIS_VAL

Coordinated Set Identification Service UUID value.

BT_UUID_CSIS

Coordinated Set Identification Service.

BT_UUID_DTS_VAL

Device Time Service UUID value.

BT_UUID_DTS

Device Time Service.

BT_UUID_MCS_VAL

Media Control Service UUID value.

BT_UUID_MCS

Media Control Service.

BT_UUID_GMCS_VAL

Generic Media Control Service UUID value.

BT_UUID_GMCS

Generic Media Control Service.

BT_UUID_CTES_VAL

Constant Tone Extension Service UUID value.

BT_UUID_CTES

Constant Tone Extension Service.

BT_UUID_TBS_VAL

Telephone Bearer Service UUID value.

BT_UUID_TBS

Telephone Bearer Service.

BT_UUID_GTBS_VAL

Generic Telephone Bearer Service UUID value.

BT_UUID_GTBS

Generic Telephone Bearer Service.

BT_UUID_MICS_VAL

Microphone Control Service UUID value.

BT_UUID_MICS

Microphone Control Service.

BT_UUID_ASCS_VAL

Audio Stream Control Service UUID value.

BT_UUID_ASCS

Audio Stream Control Service.

BT_UUID_BASS_VAL

Broadcast Audio Scan Service UUID value.

BT_UUID_BASS

Broadcast Audio Scan Service.

BT_UUID_PACS_VAL

Published Audio Capabilities Service UUID value.

BT_UUID_PACS

Published Audio Capabilities Service.

BT_UUID_BASIC_AUDIO_VAL

Basic Audio Announcement Service UUID value.

BT_UUID_BASIC_AUDIO

Basic Audio Announcement Service.

BT_UUID_BROADCAST_AUDIO_VAL

Broadcast Audio Announcement Service UUID value.

BT_UUID_BROADCAST_AUDIO

Broadcast Audio Announcement Service.

BT_UUID_CAS_VAL

Common Audio Service UUID value.

BT_UUID_CAS

Common Audio Service.

BT_UUID_HAS_VAL

Hearing Access Service UUID value.

BT_UUID_HAS

Hearing Access Service.

BT_UUID_TMAS_VAL

Telephony and Media Audio Service UUID value.

BT_UUID_TMAS

Telephony and Media Audio Service.

BT_UUID_PBA_VAL

Public Broadcast Announcement Service UUID value.

BT_UUID_PBA

Public Broadcast Announcement 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_VAL_TRIGGER_SETTING_VAL

Value Trigger Setting Descriptor UUID value.

BT_UUID_VAL_TRIGGER_SETTING

Value Trigger Setting 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_TM_TRIGGER_SETTING_VAL

Time Trigger Setting Descriptor UUID value.

BT_UUID_TM_TRIGGER_SETTING

Time 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_PPF_VAL

GAP Characteristic Peripheral Privacy Flag UUID value.

BT_UUID_GAP_PPF

GAP Characteristic Peripheral Privacy Flag.

BT_UUID_GAP_RA_VAL

GAP Characteristic Reconnection Address UUID value.

BT_UUID_GAP_RA

GAP Characteristic Reconnection Address.

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

GATT Characteristic Alert Level UUID value.

BT_UUID_ALERT_LEVEL

GATT Characteristic 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_GATT_DT_VAL

GATT Characteristic Date Time UUID value.

BT_UUID_GATT_DT

GATT Characteristic Date Time.

BT_UUID_GATT_DW_VAL

GATT Characteristic Day of Week UUID value.

BT_UUID_GATT_DW

GATT Characteristic Day of Week.

BT_UUID_GATT_DDT_VAL

GATT Characteristic Day Date Time UUID value.

BT_UUID_GATT_DDT

GATT Characteristic Day Date Time.

BT_UUID_GATT_ET256_VAL

GATT Characteristic Exact Time 256 UUID value.

BT_UUID_GATT_ET256

GATT Characteristic Exact Time 256.

BT_UUID_GATT_DST_VAL

GATT Characteristic DST Offset UUID value.

BT_UUID_GATT_DST

GATT Characteristic DST Offset.

BT_UUID_GATT_TZ_VAL

GATT Characteristic Time Zone UUID value.

BT_UUID_GATT_TZ

GATT Characteristic Time Zone.

BT_UUID_GATT_LTI_VAL

GATT Characteristic Local Time Information UUID value.

BT_UUID_GATT_LTI

GATT Characteristic Local Time Information.

BT_UUID_GATT_TDST_VAL

GATT Characteristic Time with DST UUID value.

BT_UUID_GATT_TDST

GATT Characteristic Time with DST.

BT_UUID_GATT_TA_VAL

GATT Characteristic Time Accuracy UUID value.

BT_UUID_GATT_TA

GATT Characteristic Time Accuracy.

BT_UUID_GATT_TS_VAL

GATT Characteristic Time Source UUID value.

BT_UUID_GATT_TS

GATT Characteristic Time Source.

BT_UUID_GATT_RTI_VAL

GATT Characteristic Reference Time Information UUID value.

BT_UUID_GATT_RTI

GATT Characteristic Reference Time Information.

BT_UUID_GATT_TUCP_VAL

GATT Characteristic Time Update Control Point UUID value.

BT_UUID_GATT_TUCP

GATT Characteristic Time Update Control Point.

BT_UUID_GATT_TUS_VAL

GATT Characteristic Time Update State UUID value.

BT_UUID_GATT_TUS

GATT Characteristic Time Update State.

BT_UUID_GATT_GM_VAL

GATT Characteristic Glucose Measurement UUID value.

BT_UUID_GATT_GM

GATT Characteristic Glucose Measurement.

BT_UUID_BAS_BATTERY_LEVEL_VAL

BAS Characteristic Battery Level UUID value.

BT_UUID_BAS_BATTERY_LEVEL

BAS Characteristic Battery Level.

BT_UUID_BAS_BATTERY_POWER_STATE_VAL

BAS Characteristic Battery Power State UUID value.

BT_UUID_BAS_BATTERY_POWER_STATE

BAS Characteristic Battery Power State.

BT_UUID_BAS_BATTERY_LEVEL_STATE_VAL

BAS Characteristic Battery Level StateUUID value.

BT_UUID_BAS_BATTERY_LEVEL_STATE

BAS Characteristic Battery Level State.

BT_UUID_HTS_MEASUREMENT_VAL

HTS Characteristic Temperature Measurement UUID value.

BT_UUID_HTS_MEASUREMENT

HTS Characteristic Temperature Measurement Value.

BT_UUID_HTS_TEMP_TYP_VAL

HTS Characteristic Temperature Type UUID value.

BT_UUID_HTS_TEMP_TYP

HTS Characteristic Temperature Type.

BT_UUID_HTS_TEMP_INT_VAL

HTS Characteristic Intermediate Temperature UUID value.

BT_UUID_HTS_TEMP_INT

HTS Characteristic Intermediate Temperature.

BT_UUID_HTS_TEMP_C_VAL

HTS Characteristic Temperature Celsius UUID value.

BT_UUID_HTS_TEMP_C

HTS Characteristic Temperature Celsius.

BT_UUID_HTS_TEMP_F_VAL

HTS Characteristic Temperature Fahrenheit UUID value.

BT_UUID_HTS_TEMP_F

HTS Characteristic Temperature Fahrenheit.

BT_UUID_HTS_INTERVAL_VAL

HTS Characteristic Measurement Interval UUID value.

BT_UUID_HTS_INTERVAL

HTS Characteristic Measurement Interval.

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_GATT_IEEE_RCDL_VAL

GATT Characteristic IEEE Regulatory Certification Data List UUID Value.

BT_UUID_GATT_IEEE_RCDL

GATT Characteristic IEEE Regulatory Certification Data List.

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_GATT_LLAT_VAL

GATT Characteristic Legacy Latitude UUID Value.

BT_UUID_GATT_LLAT

GATT Characteristic Legacy Latitude.

BT_UUID_GATT_LLON_VAL

GATT Characteristic Legacy Longitude UUID Value.

BT_UUID_GATT_LLON

GATT Characteristic Legacy Longitude.

BT_UUID_GATT_POS_2D_VAL

GATT Characteristic Position 2D UUID Value.

BT_UUID_GATT_POS_2D

GATT Characteristic Position 2D.

BT_UUID_GATT_POS_3D_VAL

GATT Characteristic Position 3D UUID Value.

BT_UUID_GATT_POS_3D

GATT Characteristic Position 3D.

BT_UUID_GATT_SR_VAL

GATT Characteristic Scan Refresh UUID Value.

BT_UUID_GATT_SR

GATT Characteristic Scan Refresh.

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_GATT_GMC_VAL

GATT Characteristic Glucose Measurement Context UUID Value.

BT_UUID_GATT_GMC

GATT Characteristic Glucose Measurement Context.

BT_UUID_GATT_BPM_VAL

GATT Characteristic Blood Pressure Measurement UUID Value.

BT_UUID_GATT_BPM

GATT Characteristic Blood Pressure Measurement.

BT_UUID_GATT_ICP_VAL

GATT Characteristic Intermediate Cuff Pressure UUID Value.

BT_UUID_GATT_ICP

GATT Characteristic Intermediate Cuff Pressure.

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_GATT_REM_VAL

GATT Characteristic Removable UUID Value.

BT_UUID_GATT_REM

GATT Characteristic Removable.

BT_UUID_GATT_SRVREQ_VAL

GATT Characteristic Service Required UUID Value.

BT_UUID_GATT_SRVREQ

GATT Characteristic Service Required.

BT_UUID_GATT_SC_TEMP_C_VAL

GATT Characteristic Scientific Temperature in Celsius UUID Value.

BT_UUID_GATT_SC_TEMP_C

GATT Characteristic Scientific Temperature in Celsius.

BT_UUID_GATT_STRING_VAL

GATT Characteristic String UUID Value.

BT_UUID_GATT_STRING

GATT Characteristic String.

BT_UUID_GATT_NETA_VAL

GATT Characteristic Network Availability UUID Value.

BT_UUID_GATT_NETA

GATT Characteristic Network Availability.

BT_UUID_GATT_ALRTS_VAL

GATT Characteristic Alert Status UUID Value.

BT_UUID_GATT_ALRTS

GATT Characteristic Alert Status.

BT_UUID_GATT_RCP_VAL

GATT Characteristic Ringer Control Point UUID Value.

BT_UUID_GATT_RCP

GATT Characteristic Ringer Control Point.

BT_UUID_GATT_RS_VAL

GATT Characteristic Ringer Setting UUID Value.

BT_UUID_GATT_RS

GATT Characteristic Ringer Setting.

BT_UUID_GATT_ALRTCID_MASK_VAL

GATT Characteristic Alert Category ID Bit Mask UUID Value.

BT_UUID_GATT_ALRTCID_MASK

GATT Characteristic Alert Category ID Bit Mask.

BT_UUID_GATT_ALRTCID_VAL

GATT Characteristic Alert Category ID UUID Value.

BT_UUID_GATT_ALRTCID

GATT Characteristic Alert Category ID.

BT_UUID_GATT_ALRTNCP_VAL

GATT Characteristic Alert Notification Control Point Value.

BT_UUID_GATT_ALRTNCP

GATT Characteristic Alert Notification Control Point.

BT_UUID_GATT_UALRTS_VAL

GATT Characteristic Unread Alert Status UUID Value.

BT_UUID_GATT_UALRTS

GATT Characteristic Unread Alert Status.

BT_UUID_GATT_NALRT_VAL

GATT Characteristic New Alert UUID Value.

BT_UUID_GATT_NALRT

GATT Characteristic New Alert.

BT_UUID_GATT_SNALRTC_VAL

GATT Characteristic Supported New Alert Category UUID Value.

BT_UUID_GATT_SNALRTC

GATT Characteristic Supported New Alert Category.

BT_UUID_GATT_SUALRTC_VAL

GATT Characteristic Supported Unread Alert Category UUID Value.

BT_UUID_GATT_SUALRTC

GATT Characteristic Supported Unread Alert Category.

BT_UUID_GATT_BPF_VAL

GATT Characteristic Blood Pressure Feature UUID Value.

BT_UUID_GATT_BPF

GATT Characteristic Blood Pressure Feature.

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_GATT_SIW_VAL

GATT Characteristic Scan Interval Windows UUID Value.

BT_UUID_GATT_SIW

GATT Characteristic Scan Interval Windows.

BT_UUID_DIS_PNP_ID_VAL

DIS Characteristic PnP ID UUID value.

BT_UUID_DIS_PNP_ID

DIS Characteristic PnP ID.

BT_UUID_GATT_GF_VAL

GATT Characteristic Glucose Feature UUID Value.

BT_UUID_GATT_GF

GATT Characteristic Glucose Feature.

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_SC_CONTROL_POINT_VAL

SC Control Point Characteristic UUID value.

BT_UUID_SC_CONTROL_POINT

SC Control Point Characteristic.

BT_UUID_GATT_DI_VAL

GATT Characteristic Digital Input UUID Value.

BT_UUID_GATT_DI

GATT Characteristic Digital Input.

BT_UUID_GATT_DO_VAL

GATT Characteristic Digital Output UUID Value.

BT_UUID_GATT_DO

GATT Characteristic Digital Output.

BT_UUID_GATT_AI_VAL

GATT Characteristic Analog Input UUID Value.

BT_UUID_GATT_AI

GATT Characteristic Analog Input.

BT_UUID_GATT_AO_VAL

GATT Characteristic Analog Output UUID Value.

BT_UUID_GATT_AO

GATT Characteristic Analog Output.

BT_UUID_GATT_AGGR_VAL

GATT Characteristic Aggregate UUID Value.

BT_UUID_GATT_AGGR

GATT Characteristic Aggregate.

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_GATT_PLX_SCM_VAL

GATT Characteristic PLX Spot-Check Measurement UUID Value.

BT_UUID_GATT_PLX_SCM

GATT Characteristic PLX Spot-Check Measurement.

BT_UUID_GATT_PLX_CM_VAL

GATT Characteristic PLX Continuous Measurement UUID Value.

BT_UUID_GATT_PLX_CM

GATT Characteristic PLX Continuous Measurement.

BT_UUID_GATT_PLX_F_VAL

GATT Characteristic PLX Features UUID Value.

BT_UUID_GATT_PLX_F

GATT Characteristic PLX Features.

BT_UUID_GATT_POPE_VAL

GATT Characteristic Pulse Oximetry Pulastile Event UUID Value.

BT_UUID_GATT_POPE

GATT Characteristic Pulse Oximetry Pulsatile Event.

BT_UUID_GATT_POCP_VAL

GATT Characteristic Pulse Oximetry Control Point UUID Value.

BT_UUID_GATT_POCP

GATT Characteristic Pulse Oximetry Control Point.

BT_UUID_GATT_CPS_CPM_VAL

GATT Characteristic Cycling Power Measurement UUID Value.

BT_UUID_GATT_CPS_CPM

GATT Characteristic Cycling Power Measurement.

BT_UUID_GATT_CPS_CPV_VAL

GATT Characteristic Cycling Power Vector UUID Value.

BT_UUID_GATT_CPS_CPV

GATT Characteristic Cycling Power Vector.

BT_UUID_GATT_CPS_CPF_VAL

GATT Characteristic Cycling Power Feature UUID Value.

BT_UUID_GATT_CPS_CPF

GATT Characteristic Cycling Power Feature.

BT_UUID_GATT_CPS_CPCP_VAL

GATT Characteristic Cycling Power Control Point UUID Value.

BT_UUID_GATT_CPS_CPCP

GATT Characteristic Cycling Power Control Point.

BT_UUID_GATT_LOC_SPD_VAL

GATT Characteristic Location and Speed UUID Value.

BT_UUID_GATT_LOC_SPD

GATT Characteristic Location and Speed.

BT_UUID_GATT_NAV_VAL

GATT Characteristic Navigation UUID Value.

BT_UUID_GATT_NAV

GATT Characteristic Navigation.

BT_UUID_GATT_PQ_VAL

GATT Characteristic Position Quality UUID Value.

BT_UUID_GATT_PQ

GATT Characteristic Position Quality.

BT_UUID_GATT_LNF_VAL

GATT Characteristic LN Feature UUID Value.

BT_UUID_GATT_LNF

GATT Characteristic LN Feature.

BT_UUID_GATT_LNCP_VAL

GATT Characteristic LN Control Point UUID Value.

BT_UUID_GATT_LNCP

GATT Characteristic LN Control Point.

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_GATT_TREND_VAL

GATT Characteristic Trend UUID Value.

BT_UUID_GATT_TREND

GATT Characteristic Trend.

BT_UUID_DESC_VALUE_CHANGED_VAL

Descriptor Value Changed Characteristic UUID value.

BT_UUID_DESC_VALUE_CHANGED

Descriptor Value Changed Characteristic.

BT_UUID_GATT_AEHRLL_VAL

GATT Characteristic Aerobic Heart Rate Low Limit UUID Value.

BT_UUID_GATT_AEHRLL

GATT Characteristic Aerobic Heart Rate Lower Limit.

BT_UUID_GATT_AETHR_VAL

GATT Characteristic Aerobic Threshold UUID Value.

BT_UUID_GATT_AETHR

GATT Characteristic Aerobic Threshold.

BT_UUID_GATT_AGE_VAL

GATT Characteristic Age UUID Value.

BT_UUID_GATT_AGE

GATT Characteristic Age.

BT_UUID_GATT_ANHRLL_VAL

GATT Characteristic Anaerobic Heart Rate Lower Limit UUID Value.

BT_UUID_GATT_ANHRLL

GATT Characteristic Anaerobic Heart Rate Lower Limit.

BT_UUID_GATT_ANHRUL_VAL

GATT Characteristic Anaerobic Heart Rate Upper Limit UUID Value.

BT_UUID_GATT_ANHRUL

GATT Characteristic Anaerobic Heart Rate Upper Limit.

BT_UUID_GATT_ANTHR_VAL

GATT Characteristic Anaerobic Threshold UUID Value.

BT_UUID_GATT_ANTHR

GATT Characteristic Anaerobic Threshold.

BT_UUID_GATT_AEHRUL_VAL

GATT Characteristic Aerobic Heart Rate Upper Limit UUID Value.

BT_UUID_GATT_AEHRUL

GATT Characteristic Aerobic Heart Rate Upper Limit.

BT_UUID_GATT_DATE_BIRTH_VAL

GATT Characteristic Date of Birth UUID Value.

BT_UUID_GATT_DATE_BIRTH

GATT Characteristic Date of Birth.

BT_UUID_GATT_DATE_THRASS_VAL

GATT Characteristic Date of Threshold Assessment UUID Value.

BT_UUID_GATT_DATE_THRASS

GATT Characteristic Date of Threshold Assessment.

BT_UUID_GATT_EMAIL_VAL

GATT Characteristic Email Address UUID Value.

BT_UUID_GATT_EMAIL

GATT Characteristic Email Address.

BT_UUID_GATT_FBHRLL_VAL

GATT Characteristic Fat Burn Heart Rate Lower Limit UUID Value.

BT_UUID_GATT_FBHRLL

GATT Characteristic Fat Burn Heart Rate Lower Limit.

BT_UUID_GATT_FBHRUL_VAL

GATT Characteristic Fat Burn Heart Rate Upper Limit UUID Value.

BT_UUID_GATT_FBHRUL

GATT Characteristic Fat Burn Heart Rate Upper Limit.

BT_UUID_GATT_FIRST_NAME_VAL

GATT Characteristic First Name UUID Value.

BT_UUID_GATT_FIRST_NAME

GATT Characteristic First Name.

BT_UUID_GATT_5ZHRL_VAL

GATT Characteristic Five Zone Heart Rate Limits UUID Value.

BT_UUID_GATT_5ZHRL

GATT Characteristic Five Zone Heart Rate Limits.

BT_UUID_GATT_GENDER_VAL

GATT Characteristic Gender UUID Value.

BT_UUID_GATT_GENDER

GATT Characteristic Gender.

BT_UUID_GATT_HR_MAX_VAL

GATT Characteristic Heart Rate Max UUID Value.

BT_UUID_GATT_HR_MAX

GATT Characteristic Heart Rate Max.

BT_UUID_GATT_HEIGHT_VAL

GATT Characteristic Height UUID Value.

BT_UUID_GATT_HEIGHT

GATT Characteristic Height.

BT_UUID_GATT_HC_VAL

GATT Characteristic Hip Circumference UUID Value.

BT_UUID_GATT_HC

GATT Characteristic Hip Circumference.

BT_UUID_GATT_LAST_NAME_VAL

GATT Characteristic Last Name UUID Value.

BT_UUID_GATT_LAST_NAME

GATT Characteristic Last Name.

BT_UUID_GATT_MRHR_VAL

GATT Characteristic Maximum Recommended Heart Rate> UUID Value.

BT_UUID_GATT_MRHR

GATT Characteristic Maximum Recommended Heart Rate.

BT_UUID_GATT_RHR_VAL

GATT Characteristic Resting Heart Rate UUID Value.

BT_UUID_GATT_RHR

GATT Characteristic Resting Heart Rate.

BT_UUID_GATT_AEANTHR_VAL

GATT Characteristic Sport Type for Aerobic and Anaerobic Thresholds UUID Value.

BT_UUID_GATT_AEANTHR

GATT Characteristic Sport Type for Aerobic and Anaerobic Threshold.

BT_UUID_GATT_3ZHRL_VAL

GATT Characteristic Three Zone Heart Rate Limits UUID Value.

BT_UUID_GATT_3ZHRL

GATT Characteristic Three Zone Heart Rate Limits.

BT_UUID_GATT_2ZHRL_VAL

GATT Characteristic Two Zone Heart Rate Limits UUID Value.

BT_UUID_GATT_2ZHRL

GATT Characteristic Two Zone Heart Rate Limits.

BT_UUID_GATT_VO2_MAX_VAL

GATT Characteristic VO2 Max UUID Value.

BT_UUID_GATT_VO2_MAX

GATT Characteristic VO2 Max.

BT_UUID_GATT_WC_VAL

GATT Characteristic Waist Circumference UUID Value.

BT_UUID_GATT_WC

GATT Characteristic Waist Circumference.

BT_UUID_GATT_WEIGHT_VAL

GATT Characteristic Weight UUID Value.

BT_UUID_GATT_WEIGHT

GATT Characteristic Weight.

BT_UUID_GATT_DBCHINC_VAL

GATT Characteristic Database Change Increment UUID Value.

BT_UUID_GATT_DBCHINC

GATT Characteristic Database Change Increment.

BT_UUID_GATT_USRIDX_VAL

GATT Characteristic User Index UUID Value.

BT_UUID_GATT_USRIDX

GATT Characteristic User Index.

BT_UUID_GATT_BCF_VAL

GATT Characteristic Body Composition Feature UUID Value.

BT_UUID_GATT_BCF

GATT Characteristic Body Composition Feature.

BT_UUID_GATT_BCM_VAL

GATT Characteristic Body Composition Measurement UUID Value.

BT_UUID_GATT_BCM

GATT Characteristic Body Composition Measurement.

BT_UUID_GATT_WM_VAL

GATT Characteristic Weight Measurement UUID Value.

BT_UUID_GATT_WM

GATT Characteristic Weight Measurement.

BT_UUID_GATT_WSF_VAL

GATT Characteristic Weight Scale Feature UUID Value.

BT_UUID_GATT_WSF

GATT Characteristic Weight Scale Feature.

BT_UUID_GATT_USRCP_VAL

GATT Characteristic User Control Point UUID Value.

BT_UUID_GATT_USRCP

GATT Characteristic User Control Point.

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_GATT_LANG_VAL

GATT Characteristic Language UUID Value.

BT_UUID_GATT_LANG

GATT Characteristic Language.

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_GATT_IPC_VAL

GATT Characteristic Indoor Positioning Configuration UUID Value.

BT_UUID_GATT_IPC

GATT Characteristic Indoor Positioning Configuration.

BT_UUID_GATT_LAT_VAL

GATT Characteristic Latitude UUID Value.

BT_UUID_GATT_LAT

GATT Characteristic Latitude.

BT_UUID_GATT_LON_VAL

GATT Characteristic Longitude UUID Value.

BT_UUID_GATT_LON

GATT Characteristic Longitude.

BT_UUID_GATT_LNCOORD_VAL

GATT Characteristic Local North Coordinate UUID Value.

BT_UUID_GATT_LNCOORD

GATT Characteristic Local North Coordinate.

BT_UUID_GATT_LECOORD_VAL

GATT Characteristic Local East Coordinate UUID Value.

BT_UUID_GATT_LECOORD

GATT Characteristic Local East Coordinate.

BT_UUID_GATT_FN_VAL

GATT Characteristic Floor Number UUID Value.

BT_UUID_GATT_FN

GATT Characteristic Floor Number.

BT_UUID_GATT_ALT_VAL

GATT Characteristic Altitude UUID Value.

BT_UUID_GATT_ALT

GATT Characteristic Altitude.

BT_UUID_GATT_UNCERTAINTY_VAL

GATT Characteristic Uncertainty UUID Value.

BT_UUID_GATT_UNCERTAINTY

GATT Characteristic Uncertainty.

BT_UUID_GATT_LOC_NAME_VAL

GATT Characteristic Location Name UUID Value.

BT_UUID_GATT_LOC_NAME

GATT Characteristic Location Name.

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_GATT_TDS_CP_VAL

GATT Characteristic TDS Control Point UUID Value.

BT_UUID_GATT_TDS_CP

GATT Characteristic TDS Control Point.

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_GATT_RPAO_VAL

GATT Characteristic Resolvable Private Address Only UUID Value.

BT_UUID_GATT_RPAO

GATT Characteristic Resolvable Private Address Only.

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_GATT_FMF_VAL

GATT Characteristic Fitness Machine Feature UUID Value.

BT_UUID_GATT_FMF

GATT Characteristic Fitness Machine Feature.

BT_UUID_GATT_TD_VAL

GATT Characteristic Treadmill Data UUID Value.

BT_UUID_GATT_TD

GATT Characteristic Treadmill Data.

BT_UUID_GATT_CTD_VAL

GATT Characteristic Cross Trainer Data UUID Value.

BT_UUID_GATT_CTD

GATT Characteristic Cross Trainer Data.

BT_UUID_GATT_STPCD_VAL

GATT Characteristic Step Climber Data UUID Value.

BT_UUID_GATT_STPCD

GATT Characteristic Step Climber Data.

BT_UUID_GATT_STRCD_VAL

GATT Characteristic Stair Climber Data UUID Value.

BT_UUID_GATT_STRCD

GATT Characteristic Stair Climber Data.

BT_UUID_GATT_RD_VAL

GATT Characteristic Rower Data UUID Value.

BT_UUID_GATT_RD

GATT Characteristic Rower Data.

BT_UUID_GATT_IBD_VAL

GATT Characteristic Indoor Bike Data UUID Value.

BT_UUID_GATT_IBD

GATT Characteristic Indoor Bike Data.

BT_UUID_GATT_TRSTAT_VAL

GATT Characteristic Training Status UUID Value.

BT_UUID_GATT_TRSTAT

GATT Characteristic Training Status.

BT_UUID_GATT_SSR_VAL

GATT Characteristic Supported Speed Range UUID Value.

BT_UUID_GATT_SSR

GATT Characteristic Supported Speed Range.

BT_UUID_GATT_SIR_VAL

GATT Characteristic Supported Inclination Range UUID Value.

BT_UUID_GATT_SIR

GATT Characteristic Supported Inclination Range.

BT_UUID_GATT_SRLR_VAL

GATT Characteristic Supported Resistance Level Range UUID Value.

BT_UUID_GATT_SRLR

GATT Characteristic Supported Resistance Level Range.

BT_UUID_GATT_SHRR_VAL

GATT Characteristic Supported Heart Rate Range UUID Value.

BT_UUID_GATT_SHRR

GATT Characteristic Supported Heart Rate Range.

BT_UUID_GATT_SPR_VAL

GATT Characteristic Supported Power Range UUID Value.

BT_UUID_GATT_SPR

GATT Characteristic Supported Power Range.

BT_UUID_GATT_FMCP_VAL

GATT Characteristic Fitness Machine Control Point UUID Value.

BT_UUID_GATT_FMCP

GATT Characteristic Fitness Machine Control Point.

BT_UUID_GATT_FMS_VAL

GATT Characteristic Fitness Machine Status UUID Value.

BT_UUID_GATT_FMS

GATT Characteristic Fitness Machine Status.

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_NNN_VAL

GATT Characteristic New Number Needed UUID Value.

BT_UUID_GATT_NNN

GATT Characteristic New Number Needed.

BT_UUID_GATT_AC_VAL

GATT Characteristic Average Current UUID Value.

BT_UUID_GATT_AC

GATT Characteristic Average Current.

BT_UUID_GATT_AV_VAL

GATT Characteristic Average Voltage UUID Value.

BT_UUID_GATT_AV

GATT Characteristic Average Voltage.

BT_UUID_GATT_BOOLEAN_VAL

GATT Characteristic Boolean UUID Value.

BT_UUID_GATT_BOOLEAN

GATT Characteristic Boolean.

BT_UUID_GATT_CRDFP_VAL

GATT Characteristic Chromatic Distance From Planckian UUID Value.

BT_UUID_GATT_CRDFP

GATT Characteristic Chromatic Distance From Planckian.

BT_UUID_GATT_CRCOORDS_VAL

GATT Characteristic Chromaticity Coordinates UUID Value.

BT_UUID_GATT_CRCOORDS

GATT Characteristic Chromaticity Coordinates.

BT_UUID_GATT_CRCCT_VAL

GATT Characteristic Chromaticity In CCT And Duv Values UUID Value.

BT_UUID_GATT_CRCCT

GATT Characteristic Chromaticity In CCT And Duv Values.

BT_UUID_GATT_CRT_VAL

GATT Characteristic Chromaticity Tolerance UUID Value.

BT_UUID_GATT_CRT

GATT Characteristic Chromaticity Tolerance.

BT_UUID_GATT_CIEIDX_VAL

GATT Characteristic CIE 13.3-1995 Color Rendering Index UUID Value.

BT_UUID_GATT_CIEIDX

GATT Characteristic CIE 13.3-1995 Color Rendering Index.

BT_UUID_GATT_COEFFICIENT_VAL

GATT Characteristic Coefficient UUID Value.

BT_UUID_GATT_COEFFICIENT

GATT Characteristic Coefficient.

BT_UUID_GATT_CCTEMP_VAL

GATT Characteristic Correlated Color Temperature UUID Value.

BT_UUID_GATT_CCTEMP

GATT Characteristic Correlated Color Temperature.

BT_UUID_GATT_COUNT16_VAL

GATT Characteristic Count 16 UUID Value.

BT_UUID_GATT_COUNT16

GATT Characteristic Count 16.

BT_UUID_GATT_COUNT24_VAL

GATT Characteristic Count 24 UUID Value.

BT_UUID_GATT_COUNT24

GATT Characteristic Count 24.

BT_UUID_GATT_CNTRCODE_VAL

GATT Characteristic Country Code UUID Value.

BT_UUID_GATT_CNTRCODE

GATT Characteristic Country Code.

BT_UUID_GATT_DATEUTC_VAL

GATT Characteristic Date UTC UUID Value.

BT_UUID_GATT_DATEUTC

GATT Characteristic Date UTC.

BT_UUID_GATT_EC_VAL

GATT Characteristic Electric Current UUID Value.

BT_UUID_GATT_EC

GATT Characteristic Electric Current.

BT_UUID_GATT_ECR_VAL

GATT Characteristic Electric Current Range UUID Value.

BT_UUID_GATT_ECR

GATT Characteristic Electric Current Range.

BT_UUID_GATT_ECSPEC_VAL

GATT Characteristic Electric Current Specification UUID Value.

BT_UUID_GATT_ECSPEC

GATT Characteristic Electric Current Specification.

BT_UUID_GATT_ECSTAT_VAL

GATT Characteristic Electric Current Statistics UUID Value.

BT_UUID_GATT_ECSTAT

GATT Characteristic Electric Current Statistics.

BT_UUID_GATT_ENERGY_VAL

GATT Characteristic Energy UUID Value.

BT_UUID_GATT_ENERGY

GATT Characteristic Energy.

BT_UUID_GATT_EPOD_VAL

GATT Characteristic Energy In A Period Of Day UUID Value.

BT_UUID_GATT_EPOD

GATT Characteristic Energy In A Period Of Day.

BT_UUID_GATT_EVTSTAT_VAL

GATT Characteristic Event Statistics UUID Value.

BT_UUID_GATT_EVTSTAT

GATT Characteristic Event Statistics.

BT_UUID_GATT_FSTR16_VAL

GATT Characteristic Fixed String 16 UUID Value.

BT_UUID_GATT_FSTR16

GATT Characteristic Fixed String 16.

BT_UUID_GATT_FSTR24_VAL

GATT Characteristic Fixed String 24 UUID Value.

BT_UUID_GATT_FSTR24

GATT Characteristic Fixed String 24.

BT_UUID_GATT_FSTR36_VAL

GATT Characteristic Fixed String 36 UUID Value.

BT_UUID_GATT_FSTR36

GATT Characteristic Fixed String 36.

BT_UUID_GATT_FSTR8_VAL

GATT Characteristic Fixed String 8 UUID Value.

BT_UUID_GATT_FSTR8

GATT Characteristic Fixed String 8.

BT_UUID_GATT_GENLVL_VAL

GATT Characteristic Generic Level UUID Value.

BT_UUID_GATT_GENLVL

GATT Characteristic Generic Level.

BT_UUID_GATT_GTIN_VAL

GATT Characteristic Global Trade Item Number UUID Value.

BT_UUID_GATT_GTIN

GATT Characteristic Global Trade Item Number.

BT_UUID_GATT_ILLUM_VAL

GATT Characteristic Illuminance UUID Value.

BT_UUID_GATT_ILLUM

GATT Characteristic Illuminance.

BT_UUID_GATT_LUMEFF_VAL

GATT Characteristic Luminous Efficacy UUID Value.

BT_UUID_GATT_LUMEFF

GATT Characteristic Luminous Efficacy.

BT_UUID_GATT_LUMNRG_VAL

GATT Characteristic Luminous Energy UUID Value.

BT_UUID_GATT_LUMNRG

GATT Characteristic Luminous Energy.

BT_UUID_GATT_LUMEXP_VAL

GATT Characteristic Luminous Exposure UUID Value.

BT_UUID_GATT_LUMEXP

GATT Characteristic Luminous Exposure.

BT_UUID_GATT_LUMFLX_VAL

GATT Characteristic Luminous Flux UUID Value.

BT_UUID_GATT_LUMFLX

GATT Characteristic Luminous Flux.

BT_UUID_GATT_LUMFLXR_VAL

GATT Characteristic Luminous Flux Range UUID Value.

BT_UUID_GATT_LUMFLXR

GATT Characteristic Luminous Flux Range.

BT_UUID_GATT_LUMINT_VAL

GATT Characteristic Luminous Intensity UUID Value.

BT_UUID_GATT_LUMINT

GATT Characteristic Luminous Intensity.

BT_UUID_GATT_MASSFLOW_VAL

GATT Characteristic Mass Flow UUID Value.

BT_UUID_GATT_MASSFLOW

GATT Characteristic Mass Flow.

BT_UUID_GATT_PERLGHT_VAL

GATT Characteristic Perceived Lightness UUID Value.

BT_UUID_GATT_PERLGHT

GATT Characteristic Perceived Lightness.

BT_UUID_GATT_PER8_VAL

GATT Characteristic Percentage 8 UUID Value.

BT_UUID_GATT_PER8

GATT Characteristic Percentage 8.

BT_UUID_GATT_PWR_VAL

GATT Characteristic Power UUID Value.

BT_UUID_GATT_PWR

GATT Characteristic Power.

BT_UUID_GATT_PWRSPEC_VAL

GATT Characteristic Power Specification UUID Value.

BT_UUID_GATT_PWRSPEC

GATT Characteristic Power Specification.

BT_UUID_GATT_RRICR_VAL

GATT Characteristic Relative Runtime In A Current Range UUID Value.

BT_UUID_GATT_RRICR

GATT Characteristic Relative Runtime In A Current Range.

BT_UUID_GATT_RRIGLR_VAL

GATT Characteristic Relative Runtime In A Generic Level Range UUID Value.

BT_UUID_GATT_RRIGLR

GATT Characteristic Relative Runtime In A Generic Level Range.

BT_UUID_GATT_RVIVR_VAL

GATT Characteristic Relative Value In A Voltage Range UUID Value.

BT_UUID_GATT_RVIVR

GATT Characteristic Relative Value In A Voltage Range.

BT_UUID_GATT_RVIIR_VAL

GATT Characteristic Relative Value In A Illuminance Range UUID Value.

BT_UUID_GATT_RVIIR

GATT Characteristic Relative Value In A Illuminance Range.

BT_UUID_GATT_RVIPOD_VAL

GATT Characteristic Relative Value In A Period Of Day UUID Value.

BT_UUID_GATT_RVIPOD

GATT Characteristic Relative Value In A Period Of Day.

BT_UUID_GATT_RVITR_VAL

GATT Characteristic Relative Value In A Temperature Range UUID Value.

BT_UUID_GATT_RVITR

GATT Characteristic Relative Value In A Temperature Range.

BT_UUID_GATT_TEMP8_VAL

GATT Characteristic Temperature 8 UUID Value.

BT_UUID_GATT_TEMP8

GATT Characteristic Temperature 8.

BT_UUID_GATT_TEMP8_IPOD_VAL

GATT Characteristic Temperature 8 In A Period Of Day UUID Value.

BT_UUID_GATT_TEMP8_IPOD

GATT Characteristic Temperature 8 In A Period Of Day.

BT_UUID_GATT_TEMP8_STAT_VAL

GATT Characteristic Temperature 8 Statistics UUID Value.

BT_UUID_GATT_TEMP8_STAT

GATT Characteristic Temperature 8 Statistics.

BT_UUID_GATT_TEMP_RNG_VAL

GATT Characteristic Temperature Range UUID Value.

BT_UUID_GATT_TEMP_RNG

GATT Characteristic Temperature Range.

BT_UUID_GATT_TEMP_STAT_VAL

GATT Characteristic Temperature Statistics UUID Value.

BT_UUID_GATT_TEMP_STAT

GATT Characteristic Temperature Statistics.

BT_UUID_GATT_TIM_DC8_VAL

GATT Characteristic Time Decihour 8 UUID Value.

BT_UUID_GATT_TIM_DC8

GATT Characteristic Time Decihour 8.

BT_UUID_GATT_TIM_EXP8_VAL

GATT Characteristic Time Exponential 8 UUID Value.

BT_UUID_GATT_TIM_EXP8

GATT Characteristic Time Exponential 8.

BT_UUID_GATT_TIM_H24_VAL

GATT Characteristic Time Hour 24 UUID Value.

BT_UUID_GATT_TIM_H24

GATT Characteristic Time Hour 24.

BT_UUID_GATT_TIM_MS24_VAL

GATT Characteristic Time Millisecond 24 UUID Value.

BT_UUID_GATT_TIM_MS24

GATT Characteristic Time Millisecond 24.

BT_UUID_GATT_TIM_S16_VAL

GATT Characteristic Time Second 16 UUID Value.

BT_UUID_GATT_TIM_S16

GATT Characteristic Time Second 16.

BT_UUID_GATT_TIM_S8_VAL

GATT Characteristic Time Second 8 UUID Value.

BT_UUID_GATT_TIM_S8

GATT Characteristic Time Second 8.

BT_UUID_GATT_V_VAL

GATT Characteristic Voltage UUID Value.

BT_UUID_GATT_V

GATT Characteristic Voltage.

BT_UUID_GATT_V_SPEC_VAL

GATT Characteristic Voltage Specification UUID Value.

BT_UUID_GATT_V_SPEC

GATT Characteristic Voltage Specification.

BT_UUID_GATT_V_STAT_VAL

GATT Characteristic Voltage Statistics UUID Value.

BT_UUID_GATT_V_STAT

GATT Characteristic Voltage Statistics.

BT_UUID_GATT_VOLF_VAL

GATT Characteristic Volume Flow UUID Value.

BT_UUID_GATT_VOLF

GATT Characteristic Volume Flow.

BT_UUID_GATT_CRCOORD_VAL

GATT Characteristic Chromaticity Coordinate (not Coordinates) UUID Value.

BT_UUID_GATT_CRCOORD

GATT Characteristic Chromaticity Coordinate (not Coordinates)

BT_UUID_GATT_RCF_VAL

GATT Characteristic RC Feature UUID Value.

BT_UUID_GATT_RCF

GATT Characteristic RC Feature.

BT_UUID_GATT_RCSET_VAL

GATT Characteristic RC Settings UUID Value.

BT_UUID_GATT_RCSET

GATT Characteristic RC Settings.

BT_UUID_GATT_RCCP_VAL

GATT Characteristic Reconnection Configuration Control Point UUID Value.

BT_UUID_GATT_RCCP

GATT Characteristic Reconnection Configuration Control Point.

BT_UUID_GATT_IDD_SC_VAL

GATT Characteristic IDD Status Changed UUID Value.

BT_UUID_GATT_IDD_SC

GATT Characteristic IDD Status Changed.

BT_UUID_GATT_IDD_S_VAL

GATT Characteristic IDD Status UUID Value.

BT_UUID_GATT_IDD_S

GATT Characteristic IDD Status.

BT_UUID_GATT_IDD_AS_VAL

GATT Characteristic IDD Annunciation Status UUID Value.

BT_UUID_GATT_IDD_AS

GATT Characteristic IDD Annunciation Status.

BT_UUID_GATT_IDD_F_VAL

GATT Characteristic IDD Features UUID Value.

BT_UUID_GATT_IDD_F

GATT Characteristic IDD Features.

BT_UUID_GATT_IDD_SRCP_VAL

GATT Characteristic IDD Status Reader Control Point UUID Value.

BT_UUID_GATT_IDD_SRCP

GATT Characteristic IDD Status Reader Control Point.

BT_UUID_GATT_IDD_CCP_VAL

GATT Characteristic IDD Command Control Point UUID Value.

BT_UUID_GATT_IDD_CCP

GATT Characteristic IDD Command Control Point.

BT_UUID_GATT_IDD_CD_VAL

GATT Characteristic IDD Command Data UUID Value.

BT_UUID_GATT_IDD_CD

GATT Characteristic IDD Command Data.

BT_UUID_GATT_IDD_RACP_VAL

GATT Characteristic IDD Record Access Control Point UUID Value.

BT_UUID_GATT_IDD_RACP

GATT Characteristic IDD Record Access Control Point.

BT_UUID_GATT_IDD_HD_VAL

GATT Characteristic IDD History Data UUID Value.

BT_UUID_GATT_IDD_HD

GATT Characteristic IDD History Data.

BT_UUID_GATT_CLIENT_FEATURES_VAL

GATT Characteristic Client Supported Features UUID value.

BT_UUID_GATT_CLIENT_FEATURES

GATT Characteristic Client Supported Features.

BT_UUID_GATT_DB_HASH_VAL

GATT Characteristic Database Hash UUID value.

BT_UUID_GATT_DB_HASH

GATT Characteristic Database Hash.

BT_UUID_GATT_BSS_CP_VAL

GATT Characteristic BSS Control Point UUID Value.

BT_UUID_GATT_BSS_CP

GATT Characteristic BSS Control Point.

BT_UUID_GATT_BSS_R_VAL

GATT Characteristic BSS Response UUID Value.

BT_UUID_GATT_BSS_R

GATT Characteristic BSS Response.

BT_UUID_GATT_EMG_ID_VAL

GATT Characteristic Emergency ID UUID Value.

BT_UUID_GATT_EMG_ID

GATT Characteristic Emergency ID.

BT_UUID_GATT_EMG_TXT_VAL

GATT Characteristic Emergency Text UUID Value.

BT_UUID_GATT_EMG_TXT

GATT Characteristic Emergency Text.

BT_UUID_GATT_ACS_S_VAL

GATT Characteristic ACS Status UUID Value.

BT_UUID_GATT_ACS_S

GATT Characteristic ACS Status.

BT_UUID_GATT_ACS_DI_VAL

GATT Characteristic ACS Data In UUID Value.

BT_UUID_GATT_ACS_DI

GATT Characteristic ACS Data In.

BT_UUID_GATT_ACS_DON_VAL

GATT Characteristic ACS Data Out Notify UUID Value.

BT_UUID_GATT_ACS_DON

GATT Characteristic ACS Data Out Notify.

BT_UUID_GATT_ACS_DOI_VAL

GATT Characteristic ACS Data Out Indicate UUID Value.

BT_UUID_GATT_ACS_DOI

GATT Characteristic ACS Data Out Indicate.

BT_UUID_GATT_ACS_CP_VAL

GATT Characteristic ACS Control Point UUID Value.

BT_UUID_GATT_ACS_CP

GATT Characteristic ACS Control Point.

BT_UUID_GATT_EBPM_VAL

GATT Characteristic Enhanced Blood Pressure Measurement UUID Value.

BT_UUID_GATT_EBPM

GATT Characteristic Enhanced Blood Pressure Measurement.

BT_UUID_GATT_EICP_VAL

GATT Characteristic Enhanced Intermediate Cuff Pressure UUID Value.

BT_UUID_GATT_EICP

GATT Characteristic Enhanced Intermediate Cuff Pressure.

BT_UUID_GATT_BPR_VAL

GATT Characteristic Blood Pressure Record UUID Value.

BT_UUID_GATT_BPR

GATT Characteristic Blood Pressure Record.

BT_UUID_GATT_RU_VAL

GATT Characteristic Registered User UUID Value.

BT_UUID_GATT_RU

GATT Characteristic Registered User.

BT_UUID_GATT_BR_EDR_HD_VAL

GATT Characteristic BR-EDR Handover Data UUID Value.

BT_UUID_GATT_BR_EDR_HD

GATT Characteristic BR-EDR Handover Data.

BT_UUID_GATT_BT_SIG_D_VAL

GATT Characteristic Bluetooth SIG Data UUID Value.

BT_UUID_GATT_BT_SIG_D

GATT Characteristic Bluetooth SIG Data.

BT_UUID_GATT_SERVER_FEATURES_VAL

GATT Characteristic Server Supported Features UUID value.

BT_UUID_GATT_SERVER_FEATURES

GATT Characteristic Server Supported Features.

BT_UUID_GATT_PHY_AMF_VAL

GATT Characteristic Physical Activity Monitor Features UUID Value.

BT_UUID_GATT_PHY_AMF

GATT Characteristic Physical Activity Monitor Features.

BT_UUID_GATT_GEN_AID_VAL

GATT Characteristic General Activity Instantaneous Data UUID Value.

BT_UUID_GATT_GEN_AID

GATT Characteristic General Activity Instantaneous Data.

BT_UUID_GATT_GEN_ASD_VAL

GATT Characteristic General Activity Summary Data UUID Value.

BT_UUID_GATT_GEN_ASD

GATT Characteristic General Activity Summary Data.

BT_UUID_GATT_CR_AID_VAL

GATT Characteristic CardioRespiratory Activity Instantaneous Data UUID Value.

BT_UUID_GATT_CR_AID

GATT Characteristic CardioRespiratory Activity Instantaneous Data.

BT_UUID_GATT_CR_ASD_VAL

GATT Characteristic CardioRespiratory Activity Summary Data UUID Value.

BT_UUID_GATT_CR_ASD

GATT Characteristic CardioRespiratory Activity Summary Data.

BT_UUID_GATT_SC_ASD_VAL

GATT Characteristic Step Counter Activity Summary Data UUID Value.

BT_UUID_GATT_SC_ASD

GATT Characteristic Step Counter Activity Summary Data.

BT_UUID_GATT_SLP_AID_VAL

GATT Characteristic Sleep Activity Instantaneous Data UUID Value.

BT_UUID_GATT_SLP_AID

GATT Characteristic Sleep Activity Instantaneous Data.

BT_UUID_GATT_SLP_ASD_VAL

GATT Characteristic Sleep Activity Summary Data UUID Value.

BT_UUID_GATT_SLP_ASD

GATT Characteristic Sleep Activity Summary Data.

BT_UUID_GATT_PHY_AMCP_VAL

GATT Characteristic Physical Activity Monitor Control Point UUID Value.

BT_UUID_GATT_PHY_AMCP

GATT Characteristic Physical Activity Monitor Control Point.

BT_UUID_GATT_ACS_VAL

GATT Characteristic Activity Current Session UUID Value.

BT_UUID_GATT_ACS

GATT Characteristic Activity Current Session.

BT_UUID_GATT_PHY_ASDESC_VAL

GATT Characteristic Physical Activity Session Descriptor UUID Value.

BT_UUID_GATT_PHY_ASDESC

GATT Characteristic Physical Activity Session Descriptor.

BT_UUID_GATT_PREF_U_VAL

GATT Characteristic Preferred Units UUID Value.

BT_UUID_GATT_PREF_U

GATT Characteristic Preferred Units.

BT_UUID_GATT_HRES_H_VAL

GATT Characteristic High Resolution Height UUID Value.

BT_UUID_GATT_HRES_H

GATT Characteristic High Resolution Height.

BT_UUID_GATT_MID_NAME_VAL

GATT Characteristic Middle Name UUID Value.

BT_UUID_GATT_MID_NAME

GATT Characteristic Middle Name.

BT_UUID_GATT_STRDLEN_VAL

GATT Characteristic Stride Length UUID Value.

BT_UUID_GATT_STRDLEN

GATT Characteristic Stride Length.

BT_UUID_GATT_HANDEDNESS_VAL

GATT Characteristic Handedness UUID Value.

BT_UUID_GATT_HANDEDNESS

GATT Characteristic Handedness.

BT_UUID_GATT_DEVICE_WP_VAL

GATT Characteristic Device Wearing Position UUID Value.

BT_UUID_GATT_DEVICE_WP

GATT Characteristic Device Wearing Position.

BT_UUID_GATT_4ZHRL_VAL

GATT Characteristic Four Zone Heart Rate Limit UUID Value.

BT_UUID_GATT_4ZHRL

GATT Characteristic Four Zone Heart Rate Limit.

BT_UUID_GATT_HIET_VAL

GATT Characteristic High Intensity Exercise Threshold UUID Value.

BT_UUID_GATT_HIET

GATT Characteristic High Intensity Exercise Threshold.

BT_UUID_GATT_AG_VAL

GATT Characteristic Activity Goal UUID Value.

BT_UUID_GATT_AG

GATT Characteristic Activity Goal.

BT_UUID_GATT_SIN_VAL

GATT Characteristic Sedentary Interval Notification UUID Value.

BT_UUID_GATT_SIN

GATT Characteristic Sedentary Interval Notification.

BT_UUID_GATT_CI_VAL

GATT Characteristic Caloric Intake UUID Value.

BT_UUID_GATT_CI

GATT Characteristic Caloric Intake.

BT_UUID_GATT_TMAPR_VAL

GATT Characteristic TMAP Role UUID Value.

BT_UUID_GATT_TMAPR

GATT Characteristic TMAP Role.

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_SIRK_VAL

Set Identity Resolving Key value.

BT_UUID_CSIS_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_GATT_EDKM_VAL

GATT Characteristic Encrypted Data Key Material UUID Value.

BT_UUID_GATT_EDKM

GATT Characteristic Encrypted Data Key Material.

BT_UUID_GATT_AE32_VAL

GATT Characteristic Apparent Energy 32 UUID Value.

BT_UUID_GATT_AE32

GATT Characteristic Apparent Energy 32.

BT_UUID_GATT_AP_VAL

GATT Characteristic Apparent Power UUID Value.

BT_UUID_GATT_AP

GATT Characteristic Apparent Power.

BT_UUID_GATT_CO2CONC_VAL

GATT Characteristic CO2 Concentration UUID Value.

BT_UUID_GATT_CO2CONC

GATT Characteristic CO2 Concentration.

BT_UUID_GATT_COS_VAL

GATT Characteristic Cosine of the Angle UUID Value.

BT_UUID_GATT_COS

GATT Characteristic Cosine of the Angle.

BT_UUID_GATT_DEVTF_VAL

GATT Characteristic Device Time Feature UUID Value.

BT_UUID_GATT_DEVTF

GATT Characteristic Device Time Feature.

BT_UUID_GATT_DEVTP_VAL

GATT Characteristic Device Time Parameters UUID Value.

BT_UUID_GATT_DEVTP

GATT Characteristic Device Time Parameters.

BT_UUID_GATT_DEVT_VAL

GATT Characteristic Device Time UUID Value.

BT_UUID_GATT_DEVT

GATT Characteristic String.

BT_UUID_GATT_DEVTCP_VAL

GATT Characteristic Device Time Control Point UUID Value.

BT_UUID_GATT_DEVTCP

GATT Characteristic Device Time Control Point.

BT_UUID_GATT_TCLD_VAL

GATT Characteristic Time Change Log Data UUID Value.

BT_UUID_GATT_TCLD

GATT Characteristic Time Change Log Data.

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_GATT_E32_VAL

GATT Characteristic Energy 32 UUID Value.

BT_UUID_GATT_E32

GATT Characteristic Energy 32.

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_GATT_CTEE_VAL

GATT Characteristic Constant Tone Extension Enable UUID Value.

BT_UUID_GATT_CTEE

GATT Characteristic Constant Tone Extension Enable.

BT_UUID_GATT_ACTEML_VAL

GATT Characteristic Advertising Constant Tone Extension Minimum Length UUID Value.

BT_UUID_GATT_ACTEML

GATT Characteristic Advertising Constant Tone Extension Minimum Length.

BT_UUID_GATT_ACTEMTC_VAL

GATT Characteristic Advertising Constant Tone Extension Minimum Transmit Count UUID Value.

BT_UUID_GATT_ACTEMTC

GATT Characteristic Advertising Constant Tone Extension Minimum Transmit Count.

BT_UUID_GATT_ACTETD_VAL

GATT Characteristic Advertising Constant Tone Extension Transmit Duration UUID Value.

BT_UUID_GATT_ACTETD

GATT Characteristic Advertising Constant Tone Extension Transmit Duration.

BT_UUID_GATT_ACTEI_VAL

GATT Characteristic Advertising Constant Tone Extension Interval UUID Value.

BT_UUID_GATT_ACTEI

GATT Characteristic Advertising Constant Tone Extension Interval.

BT_UUID_GATT_ACTEP_VAL

GATT Characteristic Advertising Constant Tone Extension PHY UUID Value.

BT_UUID_GATT_ACTEP

GATT Characteristic Advertising Constant Tone Extension PHY.

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 Control Service Mute value.

BT_UUID_MICS_MUTE

Microphone 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_GATT_NH4CONC_VAL

GATT Characteristic Ammonia Concentration UUID Value.

BT_UUID_GATT_NH4CONC

GATT Characteristic Ammonia Concentration.

BT_UUID_GATT_COCONC_VAL

GATT Characteristic Carbon Monoxide Concentration UUID Value.

BT_UUID_GATT_COCONC

GATT Characteristic Carbon Monoxide Concentration.

BT_UUID_GATT_CH4CONC_VAL

GATT Characteristic Methane Concentration UUID Value.

BT_UUID_GATT_CH4CONC

GATT Characteristic Methane Concentration.

BT_UUID_GATT_NO2CONC_VAL

GATT Characteristic Nitrogen Dioxide Concentration UUID Value.

BT_UUID_GATT_NO2CONC

GATT Characteristic Nitrogen Dioxide Concentration.

BT_UUID_GATT_NONCH4CONC_VAL

GATT Characteristic Non-Methane Volatile Organic Compounds Concentration UUID Value.

BT_UUID_GATT_NONCH4CONC

GATT Characteristic Non-Methane Volatile Organic Compounds Concentration.

BT_UUID_GATT_O3CONC_VAL

GATT Characteristic Ozone Concentration UUID Value.

BT_UUID_GATT_O3CONC

GATT Characteristic Ozone Concentration.

BT_UUID_GATT_PM1CONC_VAL

GATT Characteristic Particulate Matter - PM1 Concentration UUID Value.

BT_UUID_GATT_PM1CONC

GATT Characteristic Particulate Matter - PM1 Concentration.

BT_UUID_GATT_PM25CONC_VAL

GATT Characteristic Particulate Matter - PM2.5 Concentration UUID Value.

BT_UUID_GATT_PM25CONC

GATT Characteristic Particulate Matter - PM2.5 Concentration.

BT_UUID_GATT_PM10CONC_VAL

GATT Characteristic Particulate Matter - PM10 Concentration UUID Value.

BT_UUID_GATT_PM10CONC

GATT Characteristic Particulate Matter - PM10 Concentration.

BT_UUID_GATT_SO2CONC_VAL

GATT Characteristic Sulfur Dioxide Concentration UUID Value.

BT_UUID_GATT_SO2CONC

GATT Characteristic Sulfur Dioxide Concentration.

BT_UUID_GATT_SF6CONC_VAL

GATT Characteristic Sulfur Hexafluoride Concentration UUID Value.

BT_UUID_GATT_SF6CONC

GATT Characteristic Sulfur Hexafluoride Concentration.

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_GATT_FSTR64_VAL

GATT Characteristic Fixed String 64 UUID Value.

BT_UUID_GATT_FSTR64

GATT Characteristic Fixed String 64.

BT_UUID_GATT_HITEMP_VAL

GATT Characteristic High Temperature UUID Value.

BT_UUID_GATT_HITEMP

GATT Characteristic High Temperature.

BT_UUID_GATT_HV_VAL

GATT Characteristic High Voltage UUID Value.

BT_UUID_GATT_HV

GATT Characteristic High Voltage.

BT_UUID_GATT_LD_VAL

GATT Characteristic Light Distribution UUID Value.

BT_UUID_GATT_LD

GATT Characteristic Light Distribution.

BT_UUID_GATT_LO_VAL

GATT Characteristic Light Output UUID Value.

BT_UUID_GATT_LO

GATT Characteristic Light Output.

BT_UUID_GATT_LST_VAL

GATT Characteristic Light Source Type UUID Value.

BT_UUID_GATT_LST

GATT Characteristic Light Source Type.

BT_UUID_GATT_NOISE_VAL

GATT Characteristic Noise UUID Value.

BT_UUID_GATT_NOISE

GATT Characteristic Noise.

BT_UUID_GATT_RRCCTP_VAL

GATT Characteristic Relative Runtime in a Correlated Color Temperature Range UUID Value.

BT_UUID_GATT_RRCCTR

GATT Characteristic Relative Runtime in a Correlated Color Temperature Range.

BT_UUID_GATT_TIM_S32_VAL

GATT Characteristic Time Second 32 UUID Value.

BT_UUID_GATT_TIM_S32

GATT Characteristic Time Second 32.

BT_UUID_GATT_VOCCONC_VAL

GATT Characteristic VOC Concentration UUID Value.

BT_UUID_GATT_VOCCONC

GATT Characteristic VOC Concentration.

BT_UUID_GATT_VF_VAL

GATT Characteristic Voltage Frequency UUID Value.

BT_UUID_GATT_VF

GATT Characteristic Voltage Frequency.

BT_UUID_BAS_BATTERY_CRIT_STATUS_VAL

BAS Characteristic Battery Critical Status UUID Value.

BT_UUID_BAS_BATTERY_CRIT_STATUS

BAS Characteristic Battery Critical Status.

BT_UUID_BAS_BATTERY_HEALTH_STATUS_VAL

BAS Characteristic Battery Health Status UUID Value.

BT_UUID_BAS_BATTERY_HEALTH_STATUS

BAS Characteristic Battery Health Status.

BT_UUID_BAS_BATTERY_HEALTH_INF_VAL

BAS Characteristic Battery Health Information UUID Value.

BT_UUID_BAS_BATTERY_HEALTH_INF

BAS Characteristic Battery Health Information.

BT_UUID_BAS_BATTERY_INF_VAL

BAS Characteristic Battery Information UUID Value.

BT_UUID_BAS_BATTERY_INF

BAS Characteristic Battery Information.

BT_UUID_BAS_BATTERY_LEVEL_STATUS_VAL

BAS Characteristic Battery Level Status UUID Value.

BT_UUID_BAS_BATTERY_LEVEL_STATUS

BAS Characteristic Battery Level Status.

BT_UUID_BAS_BATTERY_TIME_STATUS_VAL

BAS Characteristic Battery Time Status UUID Value.

BT_UUID_BAS_BATTERY_TIME_STATUS

BAS Characteristic Battery Time Status.

BT_UUID_GATT_ESD_VAL

GATT Characteristic Estimated Service Date UUID Value.

BT_UUID_GATT_ESD

GATT Characteristic Estimated Service Date.

BT_UUID_BAS_BATTERY_ENERGY_STATUS_VAL

BAS Characteristic Battery Energy Status UUID Value.

BT_UUID_BAS_BATTERY_ENERGY_STATUS

BAS Characteristic Battery Energy Status.

BT_UUID_GATT_SL_VAL

GATT Characteristic LE GATT Security Levels UUID Value.

BT_UUID_GATT_SL

GATT Characteristic LE GATT Security Levels.

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_GMAS_VAL

Gaming Service UUID value.

BT_UUID_GMAS

Common Audio Service.

BT_UUID_GMAP_ROLE_VAL

Gaming Audio Profile Role UUID value.

BT_UUID_GMAP_ROLE

Gaming Audio Profile Role.

BT_UUID_GMAP_UGG_FEAT_VAL

Gaming Audio Profile Unicast Game Gateway Features UUID value.

BT_UUID_GMAP_UGG_FEAT

Gaming Audio Profile Unicast Game Gateway Features.

BT_UUID_GMAP_UGT_FEAT_VAL

Gaming Audio Profile Unicast Game Terminal Features UUID value.

BT_UUID_GMAP_UGT_FEAT

Gaming Audio Profile Unicast Game Terminal Features.

BT_UUID_GMAP_BGS_FEAT_VAL

Gaming Audio Profile Broadcast Game Sender Features UUID value.

BT_UUID_GMAP_BGS_FEAT

Gaming Audio Profile Broadcast Game Sender Features.

BT_UUID_GMAP_BGR_FEAT_VAL

Gaming Audio Profile Broadcast Game Receiver Features UUID value.

BT_UUID_GMAP_BGR_FEAT

Gaming Audio Profile Broadcast Game Receiver Features.

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_WSP_VAL

BT_UUID_WSP

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

Returns

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.

uint16_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.

1.17 services

1.17.1 HTTP Proxy Service (HPS)

1.17.1.1 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**

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**

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>

1.17.2 Health Thermometer Service (HTS)

1.17.2.1 API Reference

group **bt_hts**

Health Thermometer Service (HTS)

[Experimental] Users should note that the APIs can change as a part of ongoing development.

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**

enumerator **hts_toe**

enumerator **hts_tympanum**

Functions

void **bt_hts_indicate**(void)

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>

1.17.3 Internet Protocol Support Profile (IPSP)

1.17.3.1 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 receiiving 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.

int **ipsp_listen**(void)

Setup an IPSP Server.

Returns

Zero in case of success and error code in case of error.

1.17.4 Proximity Reporter (PXR)

1.17.4.1 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

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.

Returns

Number of bytes read.

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.

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.

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