bluez_tools

0.1

Generated by Doxygen 1.8.9.1

Mon Apr 4 2016 15:05:16

Contents

Chapter 1

Main Page

bluez_tools is a set of C99 modules aiming to provide an user-friendly upper layer to the well-known Bluetooth library `BlueZ`. Each module targets a different aspect: from the Bluetooth adapter (dongle) management to the creation and management of fully functionnal L2CAP-based servers.

This project is the conclusion of a school project done in may 2014. The goal of this project was to provide a way to implement an indoor positionning system based on Bluetooth LE (Low Energy). In order to do so I created an API for my team to use BlueZ functionnalities with more ease. Since the BlueZ API is not very well documented itself (its documentation lies in the `official Bluetooth core specification`) it was essential to provide a more easy-to-use layer. The bluez_tools modules are the latest representation of this API.

The end-product of the previous school project is given as an example of the bluez_tools use. The main setup was as follow: three Raspberry Pis with Bluetooth dongles attached to them where used as servers receiving RSS↩
I data through L2CAP protocole over Bluetooth from a Bluetooth beacon attached to a person in the room. The computed RSSI data, having been previously calibrated, are then computed to locate the relative coordinates of the beacon inside the room. This school project was created by the ENSIMAG FabLab team : BERTAULD Thomas, BIANCHERI Cédric, BRELOT Lucie and FONT Ludovic.

Key features:

- User-friendly API: the provided modules are a wrapper on the BlueZ library, taking care for you of the error handling and low-level return values, allowing you to concentrate on coding your application.
- Safe: internal mechanicals are in place to ensure as little error cases as possible. For instance, the Bluetooth adapter's HCI is seen as a state machine, allowing you to recover from a potential lockout (say for instance if the dongle stays in the "scan" mode).
- Both low-level and high-level management: functions exist for both low-level management such as sockets management as well as for high-level management such as the creation of a Bluetooth server using the L2CAP protocole.
- Modularity: each module can be used separately (though all rely on the "bt" one) and it's easy to reuse the existing ones to create even higher layers or to enhance the core functionnalities.

Current limitations:

- Non-optimal performances: the use of heavy-load functions such as strings concatenation or snprintf to compute RSSI inquiries and the mechanisms in place to ensure the functionnalities safety can affect the performances of the application.
- The creation of a L2CAP server currently leads to the creation of a thread for each connected client.

Requirements:

- BlueZ v5.2 or higher

Modules overview

Using

Compiling and installing the modules

1. You first need to edit the **headers** to ensure that the modules are configured to suit your needs (especially regarding the timeout set to access the controller or to send an inquiry).
2. Then simply compile the modules using the Makefile provided in the **src** directory. You can specify whether or not to use a cross-compiling toolchain by setting the CROSS_COMPILE flag. You can also decide if you rather want the resulting libraries to be static (.a) or shared (.so) by setting the LIBTYPE flag. The default behavior is to produce a shared library.

"‘sh make "‘ Or

"‘sh make LIBTYPE=so "‘

If everything went well, the library should have been produced in the **lib** folder.

1. You can now install the library and the required headers. The **INSTALL_LIB_PATH** and **INSTALL_H↩ _PATH** flags can be set to provide a custom installation path. The default values are /usr/local/lib and /usr/local/include

"‘sh make install "‘ Or

"‘sh make install INSTALL_LIB_PATH=/home/my/path "‘

You can also re-compile the provided documentation by using the Makefile inside the **doc** directory.

Compiling your code

Now that we have everything ready, we just need to compile our application.

First of all, make sure that the previously generated library can be found by the compiler/linker. Assuming that you're using gcc, you just need to specify the -L option :

"‘sh gcc -L path/to/libraries/ "‘

You will then have to build the host program using the **lbluez_tools** library:

"‘sh gcc -L path/to/libraries/ main.c -lbarelog_logger "‘

Of course, this need to be adapted in case you use another compiler.

Testing

Unit tests

Some modules provide a unit test case that can be performed by enabling the right pre-processor macro. For instance, you can build a test-case for the hci_socket module by defining the **HCI_SOCKET_TEST** macro.

A single test for RSSI measurement is also provided under the **tests** directory.

Build the demo

As previously said, the demo is to run on three L2CAP servers in charge of receiving RSSI values, a beacon in charge of sending those values and a L2CAP client computing all the RSSI values coming from the three servers, thus creating a **triangulation** system. The setup we used was as follow:

- A TI Bluetooth LE beacon was the target to track.

- Three Raspberry Pis equiped with Bluetooth dongles were used as servers, receiving RSSI measures from the beacon.
- A laptop was used as the computation system (client), receiving data from the three servers.
- The data were then sent to a visualization application which is not part of this demo.

To compile the demo simply use the provided makefile:

"‘sh make make server "‘ The first command will build the client's side of the application and the second one the servers side.

Chapter 2

Deprecated List

globalScope$>$ Global close_all_hci_sockets (list_t $**$hci_socket_list)

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 _pretty_print_arg Struct Reference

Data Fields

- size_t **count**
- FILE $*$ **fp**

5.1.1 Detailed Description

Definition at line 771 of file cfuhash.c.

The documentation for this struct was generated from the following file:

- misc/data_struct/cfuhash.c

5.2 bt_device_t Struct Reference

```
#include <bt_device.h>
```

Data Fields

- bt_address_t mac
- bt_address_type_t add_type
- char real_name [BT_NAME_LENGTH]
- char custom_name [BT_NAME_LENGTH]

5.2.1 Detailed Description

Bluetooth Device structure :

Definition at line 65 of file bt_device.h.

5.2.2 Field Documentation

5.2.2.1 bt_address_type_t bt_device_t::add_type

Address type. The following value are allowed :

- 0x00 : Public Device Address (PDA).
- 0x01 : Random Device Address (RDA).
- 0x12 : Unknown address type (personnal code).

Definition at line 73 of file bt_device.h.

5.2.2.2 char bt_device_t::custom_name[BT_NAME_LENGTH]

User-friendly name of the device.

Definition at line 77 of file bt_device.h.

5.2.2.3 bt_address_t bt_device_t::mac

Mac (Public or not) address of the device.

Definition at line 67 of file bt_device.h.

5.2.2.4 char bt_device_t::real_name[BT_NAME_LENGTH]

Real "constructor" name of the device.

Definition at line 75 of file bt_device.h.

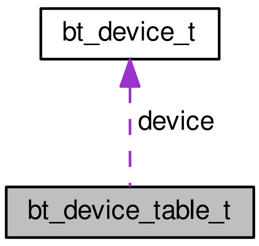
The documentation for this struct was generated from the following file:

- bt/include/bt_device.h

5.3 bt_device_table_t Struct Reference

```
#include <bt_device.h>
```

Collaboration diagram for bt_device_table_t:



Data Fields

- bt_device_t ∗ device
- uint16_t **length**

5.3.1 Detailed Description

Bluetooth devices table structure :

Definition at line 83 of file bt_device.h.

5.3.2 Field Documentation

5.3.2.1 bt_device_t∗ bt_device_table_t::device

Table of registred devices.

Definition at line 85 of file bt_device.h.

The documentation for this struct was generated from the following file:

- bt/include/bt_device.h

5.4 cfuhash_entry Struct Reference

Collaboration diagram for cfuhash_entry:

Data Fields

- void ∗ **key**
- size_t **key_size**
- void ∗ **data**
- size_t **data_size**
- struct cfuhash_entry ∗ **next**

5.4.1 Detailed Description

Definition at line 67 of file cfuhash.c.

The documentation for this struct was generated from the following file:

- misc/data_struct/cfuhash.c

5.5 cfuhash_event_flags Struct Reference

Data Fields

- int **resized**:1
- int **pad**:31

5.5.1 Detailed Description

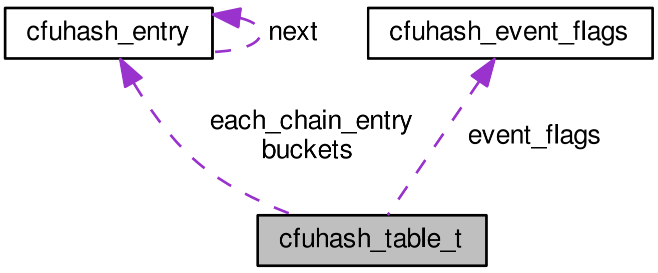
Definition at line 62 of file cfuhash.c.

The documentation for this struct was generated from the following file:

- misc/data_struct/cfuhash.c

5.6 cfuhash_table_t Struct Reference

Collaboration diagram for cfuhash_table_t:



Data Fields

- libcfu_type **type**
- size_t **num_buckets**
- size_t **entries**
- cfuhash_entry ∗∗ **buckets**
- pthread_mutex_t **mutex**
- u_int32_t **flags**
- cfuhash_function_t **hash_func**
- size_t **each_bucket_index**
- cfuhash_entry ∗ **each_chain_entry**
- float **high**
- float **low**
- cfuhash_free_fn_t **free_fn**
- unsigned int **resized_count**
- cfuhash_event_flags **event_flags**

5.6.1 Detailed Description

Definition at line 75 of file cfuhash.c.

The documentation for this struct was generated from the following file:

- misc/data_struct/cfuhash.c

5.7 cfustring_t Struct Reference

Data Fields

- libcfu_type **type**
- size_t **max_size**
- size_t **used_size**
- char ∗ **str**

5.7.1 Detailed Description

Definition at line 60 of file cfustring.c.

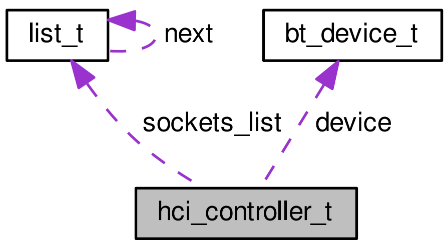
The documentation for this struct was generated from the following file:

- misc/data_struct/cfustring.c

5.8 hci_controller_t Struct Reference

```
#include <hci_controller.h>
```

Collaboration diagram for hci_controller_t:



Data Fields

- bt_device_t device
- list_t ∗ sockets_list
- hci_state_t state
- char interrupted

5.8.1 Detailed Description

hci_controller structure :

Definition at line 76 of file hci_controller.h.

5.8.2 Field Documentation

5.8.2.1 bt_device_t hci_controller_t::device

bt_device corresponding to the physical adapter.

Definition at line 80 of file hci_controller.h.

5.8.2.2 char hci_controller_t::interrupted

Error indicator : if an error occured during a request, the adapter could be stuck in a bad state. By putting this indicator to true (1), we can see that an error occured and then try to solve it (i.e try to put the controller in the default state) by using the

```
hci_resolve_interruption
```

function to be able to use it properly again.

See also

hci_resolve_interruption

Definition at line 100 of file hci_controller.h.

5.8.2.3 list_t∗ hci_controller_t::sockets_list

List of opened sockets on this adapter.

Definition at line 84 of file hci_controller.h.

5.8.2.4 hci_state_t hci_controller_t::state

Current state of the controller.

Definition at line 88 of file hci_controller.h.

The documentation for this struct was generated from the following file:

- hci/include/hci_controller.h

5.9 hci_socket_t Struct Reference

```
#include <hci_socket.h>
```

Data Fields

- int8_t sock
- int8_t dev_id

5.9.1 Detailed Description

HCI socket strucutre.

Definition at line 55 of file hci_socket.h.

5.9.2 Field Documentation

5.9.2.1 int8_t hci_socket_t::dev_id

Bluetooth controller id. NOTE : we can retrieve the bt@ of the controller with the "hci_devba(int dev_id, bt_↩
address_t ∗bdaddr)" function

Definition at line 66 of file hci_socket.h.

5.9.2.2 int8_t hci_socket_t::sock

Socket id. -1 indicates that an error occured.

Definition at line 60 of file hci_socket.h.

The documentation for this struct was generated from the following file:

- hci/include/hci_socket.h

5.10 hci_states_map_t Struct Reference

Data Fields

- uint64_t **mask**
- char ∗ **description**

5.10.1 Detailed Description

Definition at line 30 of file hci_utils.c.

The documentation for this struct was generated from the following file:

- hci/hci_utils.c

5.11 l2cap_client_proxy_t Struct Reference

```
#include <l2cap_server.h>
```

Data Fields

- int8_t conn_id
- l2cap_sockaddr_t rem_addr
- char ∗ buffer

5.11.1 Detailed Description

Structure of a client as seen by the server.

Definition at line 54 of file l2cap_server.h.

5.11.2 Field Documentation

5.11.2.1 char∗ l2cap_client_proxy_t::buffer

Buffer associated with the connection.

Definition at line 66 of file l2cap_server.h.

5.11.2.2 int8_t l2cap_client_proxy_t::conn_id

Id of the established connection between a server and this client.

Definition at line 58 of file l2cap_server.h.

5.11.2.3 l2cap_sockaddr_t l2cap_client_proxy_t::rem_addr

Client's device address.

Definition at line 62 of file l2cap_server.h.

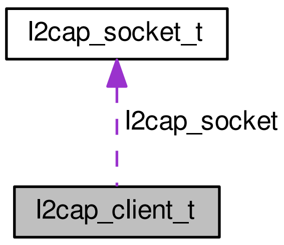
The documentation for this struct was generated from the following file:

- l2cap/include/l2cap_server.h

5.12 l2cap_client_t Struct Reference

```
#include <l2cap_client.h>
```

Collaboration diagram for l2cap_client_t:



Data Fields

- char ∗ buffer
- uint16_t buffer_length
- l2cap_socket_t l2cap_socket
- char connected
- void(∗ treat_buffer)(struct l2cap_client_t client)

  Function used when a packet is received.
- void(∗ send_request)(struct l2cap_client_t client, uint8_t req_type)

  Function used to send a request to a server.

5.12.1 Detailed Description

Structure of a L2CAP client.

Definition at line 43 of file l2cap_client.h.

5.12.2 Field Documentation

5.12.2.1 char∗ l2cap_client_t::buffer

Buffer used to receive L2CAP packets.

Definition at line 47 of file l2cap_client.h.

5.12.2.2 uint16_t l2cap_client_t::buffer_length

Length of receiving buffer.

Definition at line 51 of file l2cap_client.h.

5.12.2.3 char l2cap_client_t::connected

Status of the client (connected or not to a server).

Definition at line 60 of file l2cap_client.h.

5.12.2.4 l2cap_socket_t l2cap_client_t::l2cap_socket

Socket used by the client to connect to a L2CAP server.

Definition at line 56 of file l2cap_client.h.

5.12.2.5 void(∗ l2cap_client_t::send_request) (struct l2cap_client_t client, uint8_t req_type)

Function used to send a request to a server.

Parameters

client	client which desires to send a request.
req_type	this value can be used to discriminate which message to send to the server. The implementation of the communication protocole (i.e of the requests/answers mechanism) is up to the user of the client/server combo. See given example for more details.

Definition at line 75 of file l2cap_client.h.

5.12.2.6 void(∗ l2cap_client_t::treat_buffer) (struct l2cap_client_t client)

Function used when a packet is received.

Parameters

client	client on which to treat the buffer.

Definition at line 65 of file l2cap_client.h.

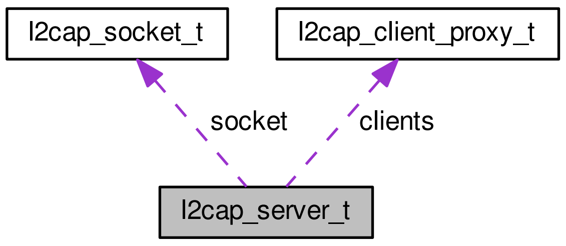
The documentation for this struct was generated from the following file:

- l2cap/include/l2cap_client.h

5.13 l2cap_server_t Struct Reference

```
#include <l2cap_server.h>
```

Collaboration diagram for l2cap_server_t:



Data Fields

- char launched
- l2cap_socket_t socket
- uint16_t buffer_length
- uint8_t max_clients
- l2cap_client_proxy_t ∗ clients
- void(∗ treat_buffer)(struct l2cap_server_t ∗server, uint8_t client_id)

  Function used when a request is received on a client's buffer.
- void(∗ send_response)(struct l2cap_server_t ∗server, uint8_t client_id, uint8_t res_type)

  Function used to send an answer to a client.

5.13.1 Detailed Description

Structure of a L2CAP server.

Definition at line 72 of file l2cap_server.h.

5.13.2 Field Documentation

5.13.2.1 uint16_t l2cap_server_t::buffer_length

Length of each client's buffer.

See also

l2cap_client_proxy_t

Definition at line 86 of file l2cap_server.h.

5.13.2.2 l2cap_client_proxy_t∗ l2cap_server_t::clients

List of the connected clients.

Definition at line 94 of file l2cap_server.h.

5.13.2.3 char l2cap_server_t::launched

Indicates if the server has at least one live connection.

Definition at line 76 of file l2cap_server.h.

5.13.2.4 uint8_t l2cap_server_t::max_clients

Maximum number of clients simultaneously treatable.

Definition at line 90 of file l2cap_server.h.

5.13.2.5 void(∗ l2cap_server_t::send_response) (struct l2cap_server_t ∗server, uint8_t client_id, uint8_t res_type)

Function used to send an answer to a client.

Parameters

server	server desiring to answer.
client_id	client to answer to.
res_type	this value can be used to discriminate which message to send to the client. The implementation of the communication protocole (i.e of the requests/answers mechanism) is up to the user of the client/server combo. See given example for more details.

Definition at line 112 of file l2cap_server.h.

5.13.2.6 l2cap_socket_t l2cap_server_t::socket

Socket on which the clients can connect.

Definition at line 80 of file l2cap_server.h.

5.13.2.7 void(∗ l2cap_server_t::treat_buffer) (struct l2cap_server_t ∗server, uint8_t client_id)

Function used when a request is received on a client's buffer.

Parameters

server	server on which to treat the request.
client_id	client to treat.

Definition at line 101 of file l2cap_server.h.

The documentation for this struct was generated from the following file:

- l2cap/include/l2cap_server.h

5.14 l2cap_socket_t Struct Reference

```
#include <l2cap_socket.h>
```

Data Fields

- int8_t sock
- l2cap_sockaddr_t sockaddr

5.14.1 Detailed Description

L2CAP socket strucutre.

Definition at line 50 of file l2cap_socket.h.

5.14.2 Field Documentation

5.14.2.1 int8_t l2cap_socket_t::sock

Socket id. -1 indicates that an error occured during initialization.

Definition at line 55 of file l2cap_socket.h.

5.14.2.2 l2cap_sockaddr_t l2cap_socket_t::sockaddr

Internal socket structure.

Definition at line 59 of file l2cap_socket.h.

The documentation for this struct was generated from the following file:

- l2cap/include/l2cap_socket.h

5.15 libcfu_item_t Struct Reference

Data Fields

- libcfu_type **type**

5.15.1 Detailed Description

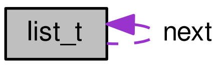
Definition at line 43 of file cfu.c.

The documentation for this struct was generated from the following file:

- misc/data_struct/cfu.c

5.16 list_t Struct Reference

Collaboration diagram for list_t:



Data Fields

- void ∗ **val**
- struct list_t ∗ **next**

5.16.1 Detailed Description

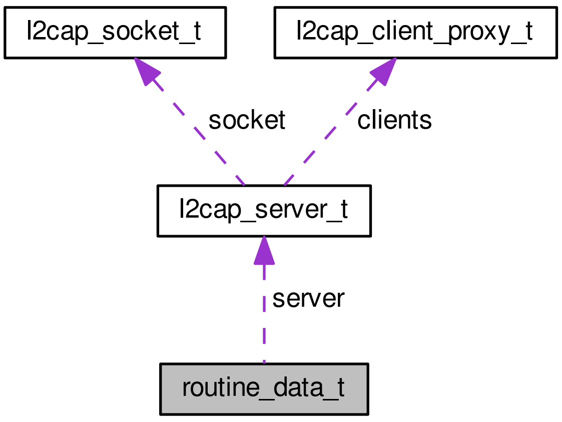
Definition at line 28 of file list.h.

The documentation for this struct was generated from the following file:

- misc/data_struct/include/list.h

5.17 routine_data_t Struct Reference

Collaboration diagram for routine_data_t:



Data Fields

- int16_t **timeout**
- uint8_t **num_client**
- uint16_t **max_req**
- l2cap_server_t ∗ **server**

5.17.1 Detailed Description

Definition at line 38 of file l2cap_server.c.

The documentation for this struct was generated from the following file:

- l2cap/l2cap_server.c

Chapter 6

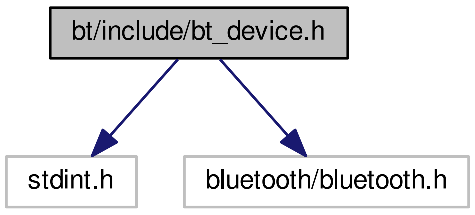
File Documentation

6.1 bt/include/bt_device.h File Reference

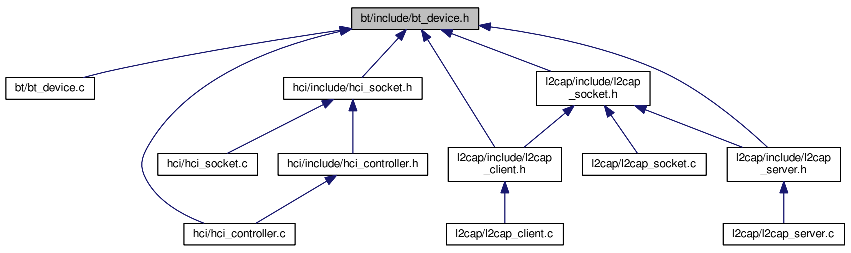
Module bluez_tools.bt.bt_device defining the basics of manipulating BT devices.

```
#include <stdint.h>
#include <bluetooth/bluetooth.h>
```

Include dependency graph for bt_device.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct bt_device_t
- struct bt_device_table_t

Macros

- #define BT_NAME_LENGTH 50

Typedefs

- typedef bdaddr_t bt_address_t

Enumerations

- enum bt_address_type_t { **PUBLIC_DEVICE_ADDRESS** = 0x00, **RANDOM_DEVICE_ADDRESS** = 0x01, **UNKNOWN_ADDRESS_TYPE** = 0x12 }

Functions

- char bt_compare_addresses (const bt_address_t ∗a1, const bt_address_t ∗a2)

  Allows the comparison of two bt mac addresses.
- char bt_already_registered_device (bt_address_t add)

  Allows to know if already encoutered (and stored the information about) a bt device. Please note that a hash table is used to store the bt_device's information.
- bt_device_t ∗ bt_register_device (bt_device_t bt_device)

  Stores a device in the main data structure (currently an hash table). We store the devices by using a couple (@, bt_device). WARNING : there is no possible double-entries (corresponding to two same @) and so, trying to add a device having the same mac @ that another previously stored device will erase that device.
- bt_device_t bt_get_device (bt_address_t add)

  Returns the stored device corresponding to the given address.
- void bt_destroy_device_table (void)

  Function used to destroy and free the structure containing the (@, bt_device) couples. Since the structure will be (re)created by using any function accessing said structure, this function could be used to "reset" the table.
- bt_device_t bt_device_create (bt_address_t mac, bt_address_type_t add_type, const char ∗real_name, const char ∗custom_name)

  Creates a new "bt_device" structure filled with the given parameters. If one (or both) of the.
- void bt_device_display (bt_device_t device)

  Displays the information of a device on the standard output.
- void bt_device_table_display (bt_device_table_t device_table)

  Displays the information of all the devices contained in the given table on the standard output.

6.1.1 Detailed Description

Module bluez_tools.bt.bt_device defining the basics of manipulating BT devices.

This header defines the functions structures and functions used to manage BT devices. Every BT device used inside the application should be registered with this manager. The module keeps track of registered BT devices using a hash table provided by the CFU module.

Author

Thomas Bertauld

Date

03/03/2016

6.1.2 Macro Definition Documentation

6.1.2.1 #define BT_NAME_LENGTH 50

Max length of a stroed name

Definition at line 43 of file bt_device.h.

6.1.3 Typedef Documentation

6.1.3.1 typedef bdaddr_t **bt_address_t**

Wrapper on the bt_address_t type

Definition at line 48 of file bt_device.h.

6.1.4 Enumeration Type Documentation

6.1.4.1 enum **bt_address_type_t**

Addresses types

Definition at line 53 of file bt_device.h.

6.1.5 Function Documentation

6.1.5.1 char bt_already_registered_device (**bt_address_t** *add*)

Allows to know if already encoutered (and stored the information about) a bt device. Please note that a hash table is used to store the bt_device's information.

Parameters

add	the address of the bt device to check.

Returns

1 if the device was already registered. Else returns 0.

Definition at line 42 of file bt_device.c.

6.1.5.2 char bt_compare_addresses (const **bt_address_t** ∗ *a1,* const **bt_address_t** ∗ *a2*)

Allows the comparison of two bt mac addresses.

Parameters

a1	first address to compare.
a2	second address to compare.

Returns

1 if the the two addresses are the same. Else returns 0.

Definition at line 36 of file bt_device.c.

6.1.5.3 bt_device_t bt_device_create (bt_address_t *mac,* bt_address_type_t *add_type,* const char ∗ *real_name,* const char ∗ *custom_name*)

Creates a new "bt_device" structure filled with the given parameters. If one (or both) of the.

```
1 name
```

parameters is NULL, the function will automatically fill the corresponding field with the name "UNKNOWN".

Parameters

mac	address of the device.
add_type	type of the device's address.
real_name	real (constructor given) name of the device.
custom_name	custom user-friendly name.

Returns

the created device structure.

Definition at line 98 of file bt_device.c.

6.1.5.4 void bt_device_display (bt_device_t *device*)

Displays the information of a device on the standard output.

Parameters

device	device to display.

Definition at line 121 of file bt_device.c.

6.1.5.5 void bt_device_table_display (bt_device_table_t *device_table*)

Displays the information of all the devices contained in the given table on the standard output.

Parameters

device_table	table to display.

Definition at line 149 of file bt_device.c.

6.1.5.6 bt_device_t bt_get_device (bt_address_t *add*)

Returns the stored device corresponding to the given address.

Parameters

add	address of the device to retrieve.

Returns

the device corresponding to the given address if any. Returns NULL otherwise.

Definition at line 71 of file bt_device.c.

6.1.5.7 bt_device_t∗ bt_register_device (bt_device_t *bt_device*)

Stores a device in the main data structure (currently an hash table). We store the devices by using a couple (@, bt_device). WARNING : there is no possible double-entries (corresponding to two same @) and so, trying to add a device having the same mac @ that another previously stored device will erase that device.

Parameters

bt_device	the device to register.

Returns

the previously stored device corresponding to the @ associated to the parameter

```
bt_device
```

if any. Returns NULL otherwise.

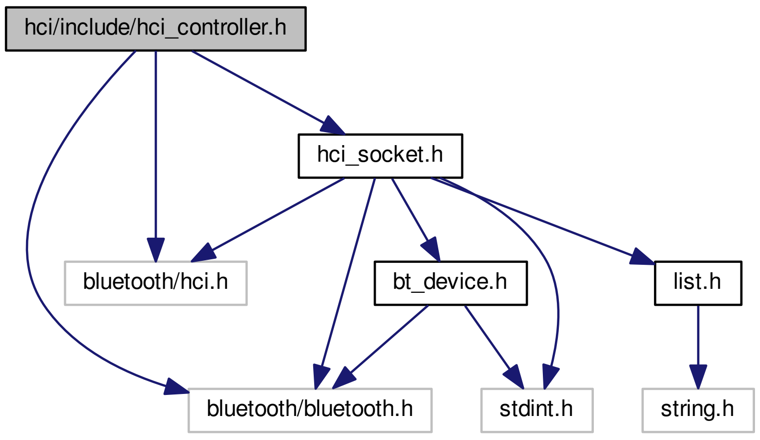
Definition at line 55 of file bt_device.c.

6.2 hci/include/hci_controller.h File Reference

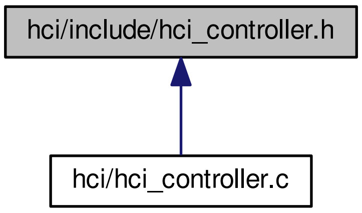
Module bluez_tools.hci.hci_controller bringing an "upper-layer" to the BlueZ's HCI (Host Controller Interface) functions.

```
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include "hci_socket.h"
```

Include dependency graph for hci_controller.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct hci_controller_t

Enumerations

- enum hci_state_t {
HCI_STATE_CLOSED = 0, **HCI_STATE_OPEN** = 1, **HCI_STATE_SCANNING** = 2, **HCI_STATE_ADVE**↩
RTISING = 3,
HCI_STATE_READING = 4, **HCI_STATE_WRITING** = 5 }

  Possible states of the hci_controller. The hci_controller is a state machine. That is to say that a virtual state is associated to each real used state of the adapter. For instance, when an adapter is scanning devices, the state machine will be in the.

Functions

- hci_controller_t hci_open_controller (bt_address_t ∗mac, char ∗name)

  Creates a new hci_controller instance using the adapter given by the.
- int8_t hci_resolve_interruption (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller)

  Tries to resolve a past interruption of a controller in order to put it in the default state to be able to use it properly again. The field.
- int8_t hci_close_controller (hci_controller_t ∗hci_controller)

  Closes and destroys an hci_controller instance. More precisely, it closes all the opened sockets on this controller and frees all the used memory. The.
- hci_socket_t hci_open_socket_controller (hci_controller_t ∗hci_controller)

  Opens a new socket on the given hci_controller. The.
- int8_t hci_close_socket_controller (hci_controller_t ∗hci_controller, hci_socket_t ∗hci_socket)

  Closes one of the opened sockets of an hci_controller. The.
- bt_device_table_t hci_scan_devices (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, uint8_t duration, uint16_t max_rsp, long flags)

  Performs a basic Bluetooth scan to recognize nearby devices. The.
- int8_t hci_compute_device_name (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, bt_device_↩
t ∗bt_device)

  Get the name of a remote device. Send an inquiry to the remote device to get its name. The.
- char ∗ hci_get_RSSI (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, int8_t ∗file_descriptor, bt_↩
address_t ∗mac, uint8_t duration, uint16_t max_rsp)

Performs a RSSI measurement on a remote device. This function is in charge of sending RSSI inquiries and retrieving them to/from the remote device. The computed results can be stored inside a file if needed. Even though the filters apply to the socket are changed during the process, they are reset to their initial values in the end. The.

- char ∗ hci_LE_get_RSSI (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, int8_t ∗file_descriptor, bt_address_t ∗mac, uint16_t max_rsp, uint8_t scan_type, uint16_t scan_interval, uint16_t scan_window, uint8_t own_add_type, uint8_t scan_filter_policy)

  Performs a RSSI measurement on a remote device (LE version). This function is in charge of sending RSSI inquiries and retrieving them to/from the remote device. The computed results can be stored inside a file if needed. Even though the filters apply to the socket are changed during the process, they are reset to their initial values in the end. The.
- int8_t hci_LE_clear_white_list (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller)

  Clears the white list of a Bluetooth adapter. The white list is the list of all devices from which the adapter can take answers while performing LE inquiries. The.
- int8_t hci_LE_add_white_list (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, const bt_device_t bt_device)

  Adds a device to the white list of a Bluetooth adapter. The.
- int8_t hci_LE_rm_white_list (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, const bt_device_↩ t bt_device)

  Removes a device to the white list of a Bluetooth adapter. The.
- int8_t hci_LE_get_white_list_size (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, uint8_t ∗size)

  Reads the size of the white list of a Bluetooth adapter. The.
- int8_t hci_LE_read_local_supported_features (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, uint8_t ∗features)

  Lists the supported LE features of a Bluetooth adapter. Since the official Bluetooth core specification doesn't give any information on this feature, expect that most of the values returned are RFU, this function has no warranty to work well. The.
- int8_t hci_LE_read_supported_states (hci_socket_t ∗hci_socket, hci_controller_t ∗hci_controller, uint64_↩ t ∗states)

  Reads the supported (real) states of a Bluetooth adapter. The.

6.2.1 Detailed Description

Module bluez_tools.hci.hci_controller bringing an "upper-layer" to the BlueZ's HCI (Host Controller Interface) functions.

The goal of this module is to provide more reliable and adaptative functions. All the following functions are using an "hci_controller" structure in order to abstract entirely the BlueZ methods.

In all that follows, we call "adapter" the device on which we want to perform HCI control. It could be for instance a Bluetooth dongle.

Author

Thomas Bertauld

Date

03/03/2016

6.2.2 Enumeration Type Documentation

6.2.2.1 enum hci_state_t

Possible states of the hci_controller. The hci_controller is a state machine. That is to say that a virtual state is associated to each real used state of the adapter. For instance, when an adapter is scanning devices, the state machine will be in the.

```
HCI_STATE_SCANNING
```

state. If you want to add more states, please follow the naming convention "HCI_STATE_..." for convenience. The default state for a controller (idle) is

```
HCI_STATE_OPEN
```

.

Definition at line 65 of file hci_controller.h.

6.2.3 Function Documentation

6.2.3.1 int8_t hci_close_controller (**hci_controller_t** ∗ *hci_controller*)

Closes and destroys an hci_controller instance. More precisely, it closes all the opened sockets on this controller and frees all the used memory. The.

```
hci_controller
```

reference has to be valid.

Parameters

hci_controller	reference on the controller to finalize.

Returns

0 on success, $<$ 0 otherwise.

Definition at line 234 of file hci_controller.c.

6.2.3.2 int8_t hci_close_socket_controller (**hci_controller_t** ∗ *hci_controller,* **hci_socket_t** ∗ *hci_socket*)

Closes one of the opened sockets of an hci_controller. The.

```
hci_controller
```

reference has to be valid (the referenced controller should be opened and initialized). The

```
hci_socket
```

field has to refer to an opened socket on the given controller. WARNING : only this function should be used to close a socket on an hci_controller for the socket to properly be removed from the sockets list of the controller. Using the classic

```
close_hci_socket
```

function will result in the socket still being referenced in the list and could be used (while closed) in another context thus leading to errors.

See also

close_hci_socket

Parameters

hci_controller	controller on which to close the socket.
hci_socket	socket to close.

Returns

0 on success, $<$ 0 otherwise.

Definition at line 270 of file hci_controller.c.

6.2.3.3 int8_t hci_compute_device_name (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* bt_device_t ∗ *bt_device*)

Get the name of a remote device. Send an inquiry to the remote device to get its name. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

.

Parameters

hci_socket	socket to be used to perform the inquiry.
hci_controller	controller emitting the inquiry.
bt_device	device from which the name is to be asked.

Returns

0 on success, in which case the name of the device is successfully stored under its

```
1 real_name
```

field, $<$ 0 if an error occured, in which case its

```
1 real_name
```

becomes

```
1 [UNKNOWN
```

}.

Definition at line 629 of file hci_controller.c.

6.2.3.4 char∗ hci_get_RSSI (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* int8_t ∗ *file_descriptor,* bt_address_t ∗ *mac,* uint8_t *duration,* uint16_t *max_rsp*)

Performs a RSSI measurement on a remote device. This function is in charge of sending RSSI inquiries and retrieving them to/from the remote device. The computed results can be stored inside a file if needed. Even though the filters apply to the socket are changed during the process, they are reset to their initial values in the end. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

. The {} field has to refer to a valid opened hci_controller.

Parameters

hci_socket	socket to be used to send and retrieve RSSI inquiries.
hci_controller	local controller.
file_descriptor	(optional) if set, the RSSI values will be written in the corresponding file.
mac	address of the device from which we want the RSSI values.
duration	duration of the RSSI scan.
max_rsp	maximum RSSI values to receive.

Returns

the computed RSSI values stored in a string.

Definition at line 737 of file hci_controller.c.

6.2.3.5 int8_t hci_LE_add_white_list (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* const bt_device_t *bt_device*)

Adds a device to the white list of a Bluetooth adapter. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

. The {} field has to refer to a valid opened hci_controller.

Parameters

hci_socket	socket used to access the controller's list.
hci_controller	controller from which the list is to be modified.
bt_device	device to add.

Returns

0 upon success, <0 otherwise.

Definition at line 501 of file hci_controller.c.

6.2.3.6 int8_t hci_LE_clear_white_list (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller*)

Clears the white list of a Bluetooth adapter. The white list is the list of all devices from which the adapter can take answers while performing LE inquiries. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

. The {} field has to refer to a valid opened hci_controller.

Parameters

hci_socket	socket used to access the controller's list.
hci_controller	controller from which the list is to be cleared.

Returns

0 upon success, <0 otherwise.

Definition at line 466 of file hci_controller.c.

6.2.3.7 char∗ hci_LE_get_RSSI (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* int8_t ∗ *file_descriptor,* bt_address_t ∗ *mac,* uint16_t *max_rsp,* uint8_t *scan_type,* uint16_t *scan_interval,* uint16_t *scan_window,* uint8_t *own_add_type,* uint8_t *scan_filter_policy*)

Performs a RSSI measurement on a remote device (LE version). This function is in charge of sending RSSI inquiries and retrieving them to/from the remote device. The computed results can be stored inside a file if needed. Even though the filters apply to the socket are changed during the process, they are reset to their initial values in the end. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

. The {} field has to refer to a valid opened hci_controller. For a detailed description of the last five parameters, please refer to the official Bluetooth documentation and the provided examples.

Parameters

hci_socket	socket to be used to send and retrieve RSSI inquiries.
hci_controller	local controller.
file_descriptor	(optional) if set, the RSSI values will be written in the corresponding file.
mac	address of the device from which we want the RSSI values.
duration	duration of the RSSI scan.
max_rsp	maximum RSSI values to receive.
scan_type	

See also

hci_le_set_scan_parameters

Parameters

scan_interval	

See also

hci_le_set_scan_parameters

Parameters

scan_window	

See also

hci_le_set_scan_parameters

Parameters

own_add_type	

See also

hci_le_set_scan_parameters

Parameters

scan_filter_↩ policy	

See also

hci_le_set_scan_parameters

Returns

the computed RSSI values stored in a string.

Definition at line 994 of file hci_controller.c.

6.2.3.8 int8_t hci_LE_get_white_list_size (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* uint8_t ∗ *size*)

Reads the size of the white list of a Bluetooth adapter. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

. The

```
1 hci_controller
```

field has to refer to a valid opened hci_controller.

Parameters

hci_socket	socket used to access the controller's list.
hci_controller	controller from which the list is to be accessed.
size	reference on the size of the list.

Returns

0 upon success, ensuring that the

```
1 size
```

size parameter now contains the size of the white list, <0 otherwise.

Definition at line 593 of file hci_controller.c.

6.2.3.9 int8_t hci_LE_read_local_supported_features (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* uint8_t ∗ *features*)

Lists the supported LE features of a Bluetooth adapter. Since the official Bluetooth core specification doesn't give any information on this feature, expect that most of the values returned are RFU, this function has no warranty to work well. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

. The {} field has to refer to a valid opened hci_controller.

Parameters

hci_socket	socket used to access the controller's list.
hci_controller	controller from which the list is to be accessed.
features	binary mask used to store the list of supported features.

Returns

0 upon success, ensuring that the

```
1 features
```

parameter now contains the list of all supported LE features, <0 otherwise.

Definition at line 339 of file hci_controller.c.

6.2.3.10 int8_t hci_LE_read_supported_states (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* uint64_t ∗ *states*)

Reads the supported (real) states of a Bluetooth adapter. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

. The {} field has to refer to a valid opened hci_controller. On success, the function should return a non-negative value and the variable pointed by

```
1 states
```

should have been set to an unsigned int representing a binary filter of the available states. For an user-friendly view, the states contained in this binary filter can be displayed on the standard output by using the function

```
1 hci_display_LE_states
```

from the

```
1 hci_utils
```

module.

See also

hci_display_LE_states

Parameters

hci_socket	socket used to access the controller's list.
hci_controller	controller from which the list is to be accessed.
states	binary mask used to store the list of supported states.

Returns

0 upon success, ensuring that the

```
states
```

parameter now contains the list of all supported LE states, <0 otherwise.

Definition at line 402 of file hci_controller.c.

6.2.3.11 int8_t hci_LE_rm_white_list (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* const bt_device_t *bt_device*)

Removes a device to the white list of a Bluetooth adapter. The.

```
hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
hci_controller
```

. The {} field has to refer to a valid opened hci_controller.

Parameters

hci_socket	socket used to access the controller's list.
hci_controller	controller from which the list is to be modified.
bt_device	device to remove.

Returns

0 upon success, <0 otherwise.

Definition at line 547 of file hci_controller.c.

6.2.3.12 hci_controller_t hci_open_controller (bt_address_t ∗ *mac,* char ∗ *name*)

Creates a new hci_controller instance using the adapter given by the.

```
mac
```

reference. If this reference is NULL, we take the first available adapter in the system. The field

```
name
```

is used to describe in an user-friendly way the adapter. If NULL then the name of the adapter will be "UNKNOWN".

Parameters

mac	address of the adapter to use.
name	user-friendly name used to describe the adapter.

Returns

the newly created instance of the adapter's hci_controller.

Definition at line 207 of file hci_controller.c.

6.2.3.13 hci_socket_t hci_open_socket_controller (hci_controller_t ∗ *hci_controller*)

Opens a new socket on the given hci_controller. The.

```
hci_controller
```

reference has to be valid. WARNING : only this function should be used to open a socket on an hci_controller for the socket to properly be added to the sockets list of the controller. Indeed, if you use the classic

```
open_hci_socket
```

function, the socket will not be added to the controller and it could result to an UNSTABLE system state when using a socket non-created with this function.

See also

open_hci_socket

Parameters

hci_controller	reference on the controller on which the socket is to be opened.

Returns

the newly created instance of the socket. If an error occured during the creation of the socket, its

```
sock
```

field is set to -1 and the socket is not added to the sockets list on the controller.

Definition at line 247 of file hci_controller.c.

6.2.3.14 int8_t hci_resolve_interruption (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller*)

Tries to resolve a past interruption of a controller in order to put it in the default state to be able to use it properly again. The field.

```
hci_socket
```

must refers to either NULL, so a new socket is opened on the given controller, or a valid opened socket on the given controller. The field

```
hci_controller
```

has to be a valid reference on an opened hci_controller.

Parameters

hci_socket	a reference on either a valid opened socket on the controller or NULL. In the later case, a new one is opened.
hci_controller	a valid reference on a hci_controller.

Returns

0 if the interruption has been resolved, a value < 0 otherwise.

Definition at line 294 of file hci_controller.c.

6.2.3.15 bt_device_table_t hci_scan_devices (hci_socket_t ∗ *hci_socket,* hci_controller_t ∗ *hci_controller,* uint8_t *duration,* uint16_t *max_rsp,* long *flags*)

Performs a basic Bluetooth scan to recognize nearby devices. The.

```
1 hci_socket
```

field can either be a valid opened socket on a valid Bluetooth adapter or NULL, in which case a new socket is opened on the given

```
1 hci_controller
```

. The {} field has to refer to a valid opened hci_controller.

Parameters

hci_socket	socket to be used to perform the scan.
hci_controller	controller emitting the scan.
duration	the scan will last at most `1 duration` ∗1,28s.
max_rsp	max number of devices to be scanned.
flags	inquiry flags.

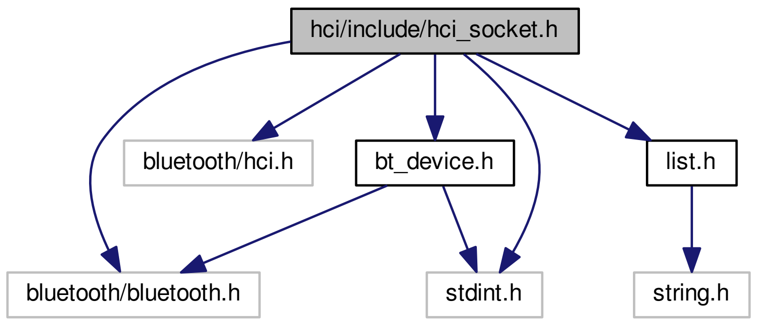
Definition at line 667 of file hci_controller.c.

6.3 hci/include/hci_socket.h File Reference

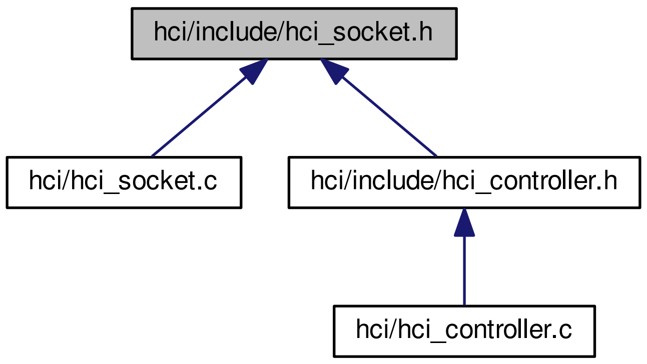
Module bluez_tools.hci.hci_socket bringing functions to deal with hci_sockets. Those sockets are "intern" sockets and are only used to communicate with a LOCAL bt adapter (typically a bt dongle).

```
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <stdint.h>
#include "list.h"
#include "bt_device.h"
```

Include dependency graph for hci_socket.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct hci_socket_t

Functions

- hci_socket_t open_hci_socket (bt_address_t ∗controller)

  Opens an hci_socket on the given controller. If the controller address is NULL, the first available controller on the system is taken. On success, the returned socket must have its both fields ("sock" and "dev_id") non-negative.
- void close_hci_socket (hci_socket_t ∗hci_socket)

  Closes a previsouly opened hci_socket. If the given socket's reference is invalid or if the socket is already closed, this function should print a warning message on the standard error output.
- void close_all_hci_sockets (list_t ∗∗hci_socket_list)

Closes all the hci_sockets stored in a list. Because this function calls "close_hci_socket", if one of the references stored in the list is invalid or if one of the contained sockets was already closed, a warning message should be displayed on the standard error output.

- int8_t get_hci_socket_filter (hci_socket_t hci_socket, struct hci_filter ∗old_flt)

  Retrieves the current socket filter applied to the given hci_socket. Upon success the filter referenced by the.

- int8_t set_hci_socket_filter (hci_socket_t hci_socket, struct hci_filter ∗flt)

  Sets the filter of the given hci_socket by using the given filter's reference. The reference has to be a valid one.

- void display_hci_socket_list (list_t ∗hci_socket_list)

  Displays all hci_sockets stored in an hci_socket list on the standard output. The list's reference has to be a valid one.

6.3.1 Detailed Description

Module bluez_tools.hci.hci_socket bringing functions to deal with hci_sockets. Those sockets are "intern" sockets and are only used to communicate with a LOCAL bt adapter (typically a bt dongle).

Author

Thomas Bertauld

Date

03/03/2016

6.3.2 Function Documentation

6.3.2.1 void close_all_hci_sockets (list_t ∗∗ *hci_socket_list*)

Closes all the hci_sockets stored in a list. Because this function calls "close_hci_socket", if one of the references stored in the list is invalid or if one of the contained sockets was already closed, a warning message should be displayed on the standard error output.

Deprecated

Parameters

hci_socket_list	reference on a sockets list.

Definition at line 76 of file hci_socket.c.

6.3.2.2 void close_hci_socket (hci_socket_t ∗ *hci_socket*)

Closes a previsouly opened hci_socket. If the given socket's reference is invalid or if the socket is already closed, this function should print a warning message on the standard error output.

Parameters

hci_socket	socket to close.

Definition at line 63 of file hci_socket.c.

6.3.2.3 void display_hci_socket_list (list_t ∗ *hci_socket_list*)

Displays all hci_sockets stored in an hci_socket list on the standard output. The list's reference has to be a valid one.

Parameters

hci_socket_list	list to display.

Definition at line 126 of file hci_socket.c.

6.3.2.4 int8_t get_hci_socket_filter (hci_socket_t *hci_socket,* struct hci_filter ∗ *old_flt*)

Retrieves the current socket filter applied to the given hci_socket. Upon success the filter referenced by the.

```
1 old_flt
```

parameter is filled with the current filter applied to the given socket and the function returns 0. If an error occured, the filter has not been retrieved and -1 is returned.

Parameters

hci_socket	the socket from which the filter is to be retrieved.
old_flt	reference on the retrieved filter.

Returns

0 upon success.

Definition at line 102 of file hci_socket.c.

6.3.2.5 hci_socket_t open_hci_socket (bt_address_t ∗ *controller*)

Opens an hci_socket on the given controller. If the controller address is NULL, the first available controller on the system is taken. On success, the returned socket must have its both fields ("sock" and "dev_id") non-negative.

Parameters

controller	address of the controller on which the socket is to be opened.

Returns

the newly created socket. Upon succes, both its fiels should be non-negative.

Definition at line 33 of file hci_socket.c.

6.3.2.6 int8_t set_hci_socket_filter (hci_socket_t *hci_socket,* struct hci_filter ∗ *flt*)

Sets the filter of the given hci_socket by using the given filter's reference. The reference has to be a valid one.

Parameters

hci_socket	socket on which the filter is to be set.
flt	filter to apply.

Returns

upon success, the filter has been set on the socket and the function returns 0. If an error occured, the filter has bot been set and -1 is returned.

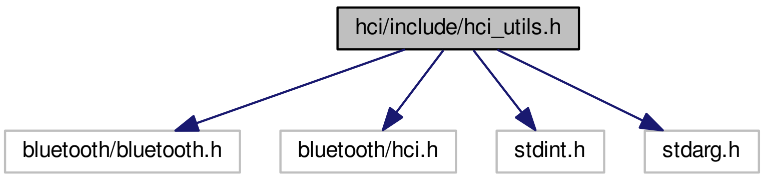
Definition at line 115 of file hci_socket.c.

6.4 hci/include/hci_utils.h File Reference

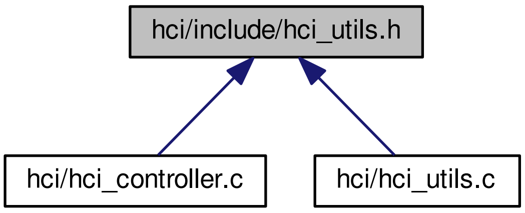
Module bluez_tools.hci.hci_utils bringing functions to deal with internal informations of an hci_controller.

```
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <stdint.h>
#include <stdarg.h>
```

Include dependency graph for hci_utils.h:



This graph shows which files directly or indirectly include this file:



Functions

- void hci_compute_filter (struct hci_filter ∗flt,...)

  Computes an hci_filter using the given event types. hci_filters are used to filter out only the events one is interested in. For a list of complete suported events, please refer to.
- void hci_display_LE_supported_states (uint64_t states)

  Displays on the standard output the (real) supported states of an hci_controller. The parameter.

6.4.1 Detailed Description

Module bluez_tools.hci.hci_utils bringing functions to deal with internal informations of an hci_controller.

Author

Thomas Bertauld

Date

03/03/2016

6.4.2 Function Documentation

6.4.2.1 void hci_compute_filter (struct hci_filter ∗ *flt*, *...*)

Computes an hci_filter using the given event types. hci_filters are used to filter out only the events one is interested in. For a list of complete suported events, please refer to.

```
include/net/bluetooth/hci.h
```

.

Parameters

flt	reference on the filter to compute;
...	the events to be enabled to pass through the computed filter.

Definition at line 69 of file hci_utils.c.

6.4.2.2 void hci_display_LE_supported_states (uint64_t *states*)

Displays on the standard output the (real) supported states of an hci_controller. The parameter.

```
states
```

is a binary mask giving the supported states of a controller. This filter has to have previously been retrieved from a real hci_controller by using the function

```
hci_LE_read_supported_states
```

in the

```
hci_controller
```

module.

See also

hci_LE_read_supported_states

Parameters

states	mask representing the supported states.

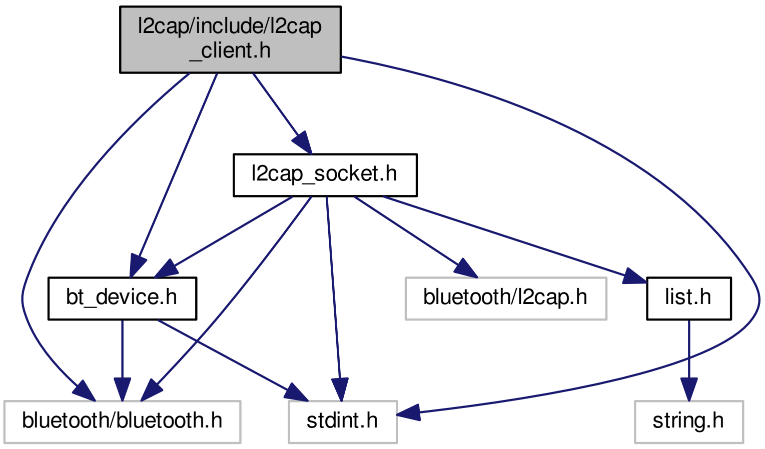
Definition at line 93 of file hci_utils.c.

6.5 l2cap/include/l2cap_client.h File Reference

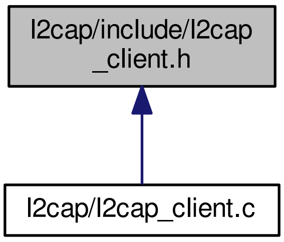
Module bluez_tools.l2cap.l2cap_client giving the basis to create a generic client using the L2CAP communication protocole.

```
#include "l2cap_socket.h"
#include <bluetooth/bluetooth.h>
#include <stdint.h>
#include "bt_device.h"
```

Include dependency graph for l2cap_client.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct l2cap_client_t

Functions

- int8_t l2cap_client_create (l2cap_client_t ∗client, bt_address_t ∗server_add, uint16_t port, uint16_t buffer_↩
length, void(∗treat_buffer_func)(l2cap_client_t client), void(∗send_request_func)(l2cap_client_t client, uint8↩
_t req_type))

  Creates a new L2CAP client.
- int8_t l2cap_client_connect (l2cap_client_t ∗client)

Connects a previously initialized client to its server.

- int8_t l2cap_client_send (l2cap_client_t ∗client, int16_t timeout, uint8_t req_type)

  Sends a request to the server. The client must have been previously successfully connected to said server. Once the request has been sent, the timeout value is used to wait for the server to answer the request. If the answer arrives before the timeout is reached, the client then calls its.
- void l2cap_client_close (l2cap_client_t ∗client)

  Destroys a client. WARNING: this function DOES NOT terminate the connection with the server, it only frees the memory taken by the client and closes its socket. It may be something to work on ? Moreover this function does not perform any kind of checking on the given client. Use with caution.

6.5.1 Detailed Description

Module bluez_tools.l2cap.l2cap_client giving the basis to create a generic client using the L2CAP communication protocole.

Author

Thomas Bertauld

Date

03/03/2016

6.5.2 Function Documentation

6.5.2.1 void l2cap_client_close (l2cap_client_t ∗ *client*)

Destroys a client. WARNING: this function DOES NOT terminate the connection with the server, it only frees the memory taken by the client and closes its socket. It may be something to work on ? Moreover this function does not perform any kind of checking on the given client. Use with caution.

Parameters

client	the client to destroy.

Definition at line 173 of file l2cap_client.c.

6.5.2.2 int8_t l2cap_client_connect (l2cap_client_t ∗ *client*)

Connects a previously initialized client to its server.

Parameters

client	to connect.

Returns

0 on success, a value $<$ 0 otherwise. Note that in case of success, the field client.connected will also be set to 1.

Definition at line 97 of file l2cap_client.c.

6.5.2.3 int8_t l2cap_client_create (l2cap_client_t ∗ *client,* bt_address_t ∗ *server_add,* uint16_t *port,* uint16_t *buffer_length,* void(∗)(l2cap_client_t client) *treat_buffer_func,* void(∗)(l2cap_client_t client, uint8_t req_type) *send_request_func*)

Creates a new L2CAP client.

Parameters

client	reference on the client to initialize.
server_add	address of the server on which the client is to be later connected.
port	server's port on which to connect later.
buffer_length	desired length for the receiving buffer. This buffer will handle the server's answers.
treat_buffer_func	function to use when a packet is received by the client. If NULL, a default function, simply displaying the received packet on the standard output, will be used.
send_request↩ _func	function to use to send request to the server. If NULL, a default function, simply sending the string "Request echo.", will be used.

Returns

0 on success, a value $<$ 0 otherwise.

Definition at line 57 of file l2cap_client.c.

6.5.2.4 int8_t l2cap_client_send (l2cap_client_t ∗ *client,* int16_t *timeout,* uint8_t *req_type*)

Sends a request to the server. The client must have been previously successfully connected to said server. Once the request has been sent, the timeout value is used to wait for the server to answer the request. If the answer arrives before the timeout is reached, the client then calls its.

```
treat_buffer_func
```

function.

Parameters

client	client willing to send a request.
timeout	timeout for the server's answer to arrive.
req_type	type of the request to send.

Returns

0 in case of success, a value $<$ 0 otherwise.

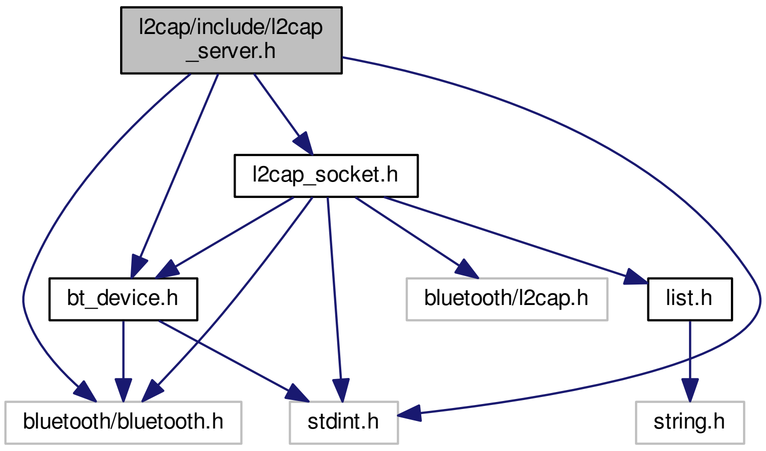
Definition at line 109 of file l2cap_client.c.

6.6 l2cap/include/l2cap_server.h File Reference

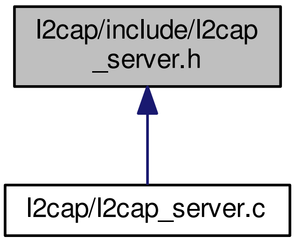
Module bluez_tools.l2cap.l2cap_server giving the basis to create a generic server using the L2CAP communication protocole.

```
#include "l2cap_socket.h"
#include "bt_device.h"
#include <bluetooth/bluetooth.h>
#include <stdint.h>
```

Include dependency graph for l2cap_server.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct l2cap_client_proxy_t
- struct l2cap_server_t

Macros

- #define L2CAP_SERVER_UNIVERSAL_STOP "STOP"

Functions

- int8_t l2cap_server_create (l2cap_server_t ∗server, bt_address_t ∗adapter, uint16_t port, uint8_t max←_clients, uint16_t buffer_length, void(∗treat_buffer_func)(l2cap_server_t ∗server, uint8_t num_client), void(∗send_response_func)(l2cap_server_t ∗server, uint8_t num_client, uint8_t res_type))

Creates a new L2CAP server.

- int8_t l2cap_server_launch (l2cap_server_t ∗server, int16_t timeout, uint16_t max_req)

  Launches a server. Launches a server by making it listen on its socket. WARNING: for now, for each connection accepted, a NEW thread is created to handle the client. If this is not the behavior you expect, please modify the implementation.
- void l2cap_server_close (l2cap_server_t ∗server)

  Destroys a server. WARNING: this function DOES NOT terminate the connections with the clients, it only frees the memory taken by the server and closes its socket. It may be something to work on ? Moreover this function does not perform any kind of checking on the given server. Use with caution.

6.6.1 Detailed Description

Module bluez_tools.l2cap.l2cap_server giving the basis to create a generic server using the L2CAP communication protocole.

Module bluez_tools.l2cap.l2cap_socket giving the basis to manage L2CAP sockets.

Author

Thomas Bertauld

Date

03/03/2016

6.6.2 Macro Definition Documentation

6.6.2.1 #define L2CAP_SERVER_UNIVERSAL_STOP "STOP"

Generic request to receive from a client to indicate the end of the communication.

Definition at line 45 of file l2cap_server.h.

6.6.3 Function Documentation

6.6.3.1 void l2cap_server_close (l2cap_server_t ∗ *server*)

Destroys a server. WARNING: this function DOES NOT terminate the connections with the clients, it only frees the memory taken by the server and closes its socket. It may be something to work on ? Moreover this function does not perform any kind of checking on the given server. Use with caution.

Parameters

server	the server to destroy.

Definition at line 265 of file l2cap_server.c.

6.6.3.2 int8_t l2cap_server_create (l2cap_server_t ∗ *server,* bt_address_t ∗ *adapter,* uint16_t *port,* uint8_t *max_clients,* uint16_t *buffer_length,* void(∗)(l2cap_server_t ∗server, uint8_t num_client) *treat_buffer_func,* void(∗)(l2cap_server_t ∗server, uint8_t num_client, uint8_t res_type) *send_response_func*)

Creates a new L2CAP server.

Parameters

server	reference on the server to initialize.
adapter	address of the adapter the server will use to communicate.
port	port on which the connections will be allowed.
max_clients	number of clients to be treated simultaneously.
buffer_length	desired length for the receiving buffer on each clients. Those buffers will handle the clients requests.
treat_buffer_func	function to use when a request is received from a client. If NULL, a default function, simply displaying the received packet on the standard output, will be used.
send_↩ response_func	function to use to send an answer to a client. If NULL, a default function, simply sending the string "Response echo.", will be used.

Returns

0 on success, a value $<$ 0 otherwise.

Definition at line 62 of file l2cap_server.c.

6.6.3.3 int8_t l2cap_server_launch (l2cap_server_t $*$ *server,* int16_t *timeout,* uint16_t *max_req*)

Launches a server. Launches a server by making it listen on its socket. WARNING: for now, for each connection accepted, a NEW thread is created to handle the client. If this is not the behavior you expect, please modify the implementation.

Parameters

server	server to launch.
timeout	timeout for a client's request to arrive.
max_req	maximum number of requests to handle for each client. A value of -1 indicates that no limit will be set.

Returns

0 on success, a value $<$ 0 otherwise.

Definition at line 181 of file l2cap_server.c.

Index