

Integración con el dispositivo RedBear Duo

Generated by Doxygen 1.8.13

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	Device_t Struct Reference	5
3.1.1	Detailed Description	5
3.1.2	Member Data Documentation	5
3.1.2.1	addr	5
3.1.2.2	addr_type	6
3.1.2.3	chars [1/2]	6
3.1.2.4	chars [2/2]	6
3.1.2.5	connected_handle	6
3.1.2.6	descriptor	6
3.1.2.7	service [1/2]	6
3.1.2.8	service [2/2]	6

4 File Documentation	7
4.1 ble_central.ino File Reference	7
4.1.1 Function Documentation	10
4.1.1.1 ble_advdata_decode()	10
4.1.1.2 checkAttributePropertyPermission()	11
4.1.1.3 deviceConnectedCallback()	12
4.1.1.4 deviceDisconnectedCallback()	13
4.1.1.5 discoveredCharsCallback()	14
4.1.1.6 discoveredCharsDescriptorsCallback()	16
4.1.1.7 discoveredServiceCallback()	17
4.1.1.8 gattNotifyUpdateCallback()	19
4.1.1.9 gattReadCallback()	20
4.1.1.10 gattReadDescriptorCallback()	21
4.1.1.11 gattReceivedIndicationCallback()	22
4.1.1.12 gattWriteCCCDCallback()	23
4.1.1.13 gattWrittenCallback()	24
4.1.1.14 getThSenseCharacteristicNameByUUID()	25
4.1.1.15 getThSenseDescriptorNameByUUID()	27
4.1.1.16 getThSenseDescriptorValue()	28
4.1.1.17 getThSenseServiceNameByUUID()	30
4.1.1.18 loop()	31
4.1.1.19 menuStateMachine()	32
4.1.1.20 printBLEProfile()	35
4.1.1.21 PrintCharacteristicsAccordingToProperty()	37
4.1.1.22 PrintCharacteristicsAndDescriptorsAccordingToProperty()	38
4.1.1.23 printCharacteristicsNamesFromService()	39
4.1.1.24 printMenuOptions()	40
4.1.1.25 printServiceName()	41
4.1.1.26 printThSenseNotificationValue()	42
4.1.1.27 printThSenseProperties()	43

4.1.1.28	printThSenseValueByHandle()	44
4.1.1.29	readTbSenseData()	47
4.1.1.30	receiveFromSerial()	49
4.1.1.31	reportCallback()	50
4.1.1.32	setup()	52
4.1.2	Variable Documentation	53
4.1.2.1	Acceleration_axis_X	53
4.1.2.2	Acceleration_axis_Y	53
4.1.2.3	Acceleration_axis_Z	54
4.1.2.4	ALight	54
4.1.2.5	Appearance	54
4.1.2.6	Battery_Level	54
4.1.2.7	Buttons	54
4.1.2.8	characteristics_name_gruped_by_service	54
4.1.2.9	chars_index	55
4.1.2.10	conf_completed	55
4.1.2.11	connected_id	55
4.1.2.12	desc_index	55
4.1.2.13	DescrvalidOption	55
4.1.2.14	device	55
4.1.2.15	Device_Name	55
4.1.2.16	Digital_1	55
4.1.2.17	Digital_2	56
4.1.2.18	ECO2	56
4.1.2.19	Field_Strength	56
4.1.2.20	Firmware_Revision	56
4.1.2.21	Hall_Control_Point	56
4.1.2.22	Hall_State	56
4.1.2.23	Hardware_Revision	56
4.1.2.24	Humidity	56

4.1.2.25	indication_lastmills	57
4.1.2.26	Manufacturer_Name	57
4.1.2.27	menuOption	57
4.1.2.28	Model_Number	57
4.1.2.29	n_acceleration_orientation_service	57
4.1.2.30	n_automation_io_service	57
4.1.2.31	n_battery_service	57
4.1.2.32	n_chars	57
4.1.2.33	n_chars_index	58
4.1.2.34	n_device_information_service	58
4.1.2.35	n_env_sensing_service	58
4.1.2.36	n_generic_access_service	58
4.1.2.37	n_generic_attribute_service	58
4.1.2.38	n_hall_effect_service	58
4.1.2.39	n_iq_service	58
4.1.2.40	n_power_management_service	58
4.1.2.41	n_serv	59
4.1.2.42	n_serv_index	59
4.1.2.43	n_serv_sel	59
4.1.2.44	n_user_interface_service	59
4.1.2.45	notification_lastmills	59
4.1.2.46	Orientation_axis_X	59
4.1.2.47	Orientation_axis_Y	59
4.1.2.48	Orientation_axis_Z	59
4.1.2.49	Power_Source	60
4.1.2.50	Pressure	60
4.1.2.51	read_tbsense_timer	60
4.1.2.52	RGB_Leds	60
4.1.2.53	Serial_Number	60
4.1.2.54	serv_index	60

4.1.2.55	services_name	60
4.1.2.56	Sound	61
4.1.2.57	System_ID	61
4.1.2.58	Temperature	61
4.1.2.59	thereIsCharacteristic	61
4.1.2.60	TVOC	61
4.1.2.61	UVindex	61
4.2	define.h File Reference	61
4.2.1	Macro Definition Documentation	64
4.2.1.1	bitRead	64
4.2.1.2	BLE_SCAN_INTERVAL	64
4.2.1.3	BLE_SCAN_TYPE	64
4.2.1.4	BLE_SCAN_WINDOW	64
4.2.1.5	MENU_DEBUG	65
4.2.1.6	NCHAR_MAX	65
4.2.1.7	NDESC_MAX	65
4.2.1.8	NSERV_MAX	65
4.2.1.9	TIMER_DEBUG	65
4.2.2	Typedef Documentation	65
4.2.2.1	stateEnum_t	65
4.2.3	Enumeration Type Documentation	65
4.2.3.1	states	65
4.2.4	Variable Documentation	66
4.2.4.1	accleration_orientation_service_uuid	66
4.2.4.2	automation_io_service_uuid	66
4.2.4.3	battery_service_uuid	66
4.2.4.4	Characteristic_Presentation_Format_uuid	66
4.2.4.5	Client_Characteristic_Configuration_uuid	67
4.2.4.6	device_information_service_uuid	67
4.2.4.7	environmental_sensing_service_uuid	67

4.2.4.8	generic_access_service_uuid	67
4.2.4.9	generic_attribute_service_uuid	67
4.2.4.10	hall_effect_service_uuid	67
4.2.4.11	iaq_service_uuid	67
4.2.4.12	noOfDigitals_uuid	68
4.2.4.13	power_management_service_uuid	68
4.2.4.14	Service0_Characrteristic0_Device_Name_uuid	68
4.2.4.15	Service0_Characrteristic1_Appearance_uuid	68
4.2.4.16	Service1_Characrteristic0_Service_Changed_uuid	68
4.2.4.17	Service2_Characrteristic0_Manufacturer_Name_uuid	68
4.2.4.18	Service2_Characrteristic1_Model_Number_uuid	68
4.2.4.19	Service2_Characrteristic2_Serial_Number_uuid	69
4.2.4.20	Service2_Characrteristic3_Hardware_Revision_uuid	69
4.2.4.21	Service2_Characrteristic4_Firmware_Revision_uuid	69
4.2.4.22	Service2_Characrteristic5_System_ID_uuid	69
4.2.4.23	Service3_Characrteristic0_Battery_Level_uuid	69
4.2.4.24	Service4_Characrteristic0_UV_Index_uuid	69
4.2.4.25	Service4_Characrteristic1_Pressure_uuid	69
4.2.4.26	Service4_Characrteristic2_Temperature_uuid	70
4.2.4.27	Service4_Characrteristic3_Humidity_uuid	70
4.2.4.28	Service4_Characrteristic4_Ambient_Light_uuid	70
4.2.4.29	Service4_Characrteristic5_Sound_Level_uuid	70
4.2.4.30	Service4_Characrteristic6_Control_Point_uuid	70
4.2.4.31	Service5_Characrteristic0_Power_Source_uuid	70
4.2.4.32	Service6_Characrteristic0_ECO2_uuid	70
4.2.4.33	Service6_Characrteristic1_TVOC_uuid	71
4.2.4.34	Service6_Characrteristic2_Control_Point_uuid	71
4.2.4.35	Service7_Characrteristic0_Buttons_uuid	71
4.2.4.36	Service7_Characrteristic1_Leds_uuid	71
4.2.4.37	Service7_Characrteristic2_RGB_Leds_uuid	71
4.2.4.38	Service7_Characrteristic3_Control_Point_uuid	71
4.2.4.39	Service8_Characrteristic0_Digital_1_uuid	71
4.2.4.40	Service8_Characrteristic1_Digital_2_uuid	72
4.2.4.41	Service9_Characrteristic0_Acceleration_uuid	72
4.2.4.42	Service9_Characrteristic1_Orientation_uuid	72
4.2.4.43	Service9_Characrteristic2_Control_Point_uuid	72
4.2.4.44	ServiceA_Characrteristic0_State_uuid	72
4.2.4.45	ServiceA_Characrteristic1_Field_Strength_uuid	72
4.2.4.46	ServiceA_Characrteristic2_Control_Point_uuid	72
4.2.4.47	user_interface_service_uuid	72

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Device_t	Struct to save a BLE device and its related data	5
--------------------------	--	-------------------

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

ble_central.ino	7
define.h	61

Chapter 3

Class Documentation

3.1 Device_t Struct Reference

Struct to save a BLE device and its related data.

```
#include <define.h>
```

Public Attributes

- uint16_t [connected_handle](#)
- uint8_t [addr_type](#)
- bd_addr_t [addr](#)
- struct {
 - gatt_client_service_t [service](#)
 - struct {
 - gatt_client_characteristic_t [chars](#)
 - gatt_client_characteristic_descriptor_t [descriptor](#) [NDESC_MAX]
 - } [chars](#) [NCHAR_MAX]
 - } [service](#) [NSERV_MAX]

3.1.1 Detailed Description

Struct to save a BLE device and its related data.

3.1.2 Member Data Documentation

3.1.2.1 addr

```
bd_addr_t Device_t::addr
```

3.1.2.2 `addr_type`

`uint8_t Device_t::addr_type`

3.1.2.3 `chars` [1/2]

`gatt_client_characteristic_t Device_t::chars`

3.1.2.4 `chars` [2/2]

`struct { ... } Device_t::chars[NCHAR_MAX]`

3.1.2.5 `connected_handle`

`uint16_t Device_t::connected_handle`

3.1.2.6 `descriptor`

`gatt_client_characteristic_descriptor_t Device_t::descriptor[NDESC_MAX]`

3.1.2.7 `service` [1/2]

`gatt_client_service_t Device_t::service`

3.1.2.8 `service` [2/2]

`struct { ... } Device_t::service[NSERV_MAX]`

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

- [define.h](#)

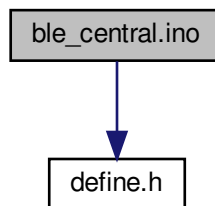
Chapter 4

File Documentation

4.1 ble_central.ino File Reference

```
#include "define.h"
```

Include dependency graph for ble_central.ino:



Functions

- uint32_t [ble_advdata_decode](#) (uint8_t type, uint8_t advdata_len, uint8_t *p_advdata, uint8_t *len, uint8_t *p_field_data)
Find the data given the type in advertising data.
- void [reportCallback](#) (advertisementReport_t *report)
Callback for scanning device.
- void [deviceConnectedCallback](#) (BLEStatus_t status, uint16_t handle)
Callback for the establishment of the BLE connection.
- void [deviceDisconnectedCallback](#) (uint16_t handle)
Callback for the Disconnect procedure.
- static void [discoveredServiceCallback](#) (BLEStatus_t status, uint16_t con_handle, gatt_client_service_t *service)
Callback for handling result of discovering Service.
- static void [discoveredCharsCallback](#) (BLEStatus_t status, uint16_t con_handle, gatt_client_characteristic_t *characteristic)

Callback for handling result of discovering characteristic.

- static void [discoveredCharsDescriptorsCallback](#) (BLEStatus_t status, uint16_t con_handle, gatt_client_t characteristic_descriptor_t *descriptor)

Callback for handling result of discovering Descriptor.

- void [gattReadCallback](#) (BLEStatus_t status, uint16_t con_handle, uint16_t value_handle, uint8_t *value, uint16_t length)

Callback for handling result of reading.

- void [gattWrittenCallback](#) (BLEStatus_t status, uint16_t con_handle)

Callback for handling result of writing.

- void [gattReadDescriptorCallback](#) (BLEStatus_t status, uint16_t con_handle, uint16_t value_handle, uint8_t *value, uint16_t length)

Callback for handling result of reading descriptor.

- void [gattWriteCCCDCallback](#) (BLEStatus_t status, uint16_t con_handle)

Callback for handling result of writing client characteristic configuration.

- void [gattNotifyUpdateCallback](#) (BLEStatus_t status, uint16_t con_handle, uint16_t value_handle, uint8_t *value, uint16_t length)

Callback for handling notify event from remote device.

- void [gattReceivedIndicationCallback](#) (BLEStatus_t status, uint16_t conn_handle, uint16_t value_handle, uint8_t *value, uint16_t length)

Callback for handling Indication event from remote device.

- uint8_t [checkAttributePropertyPermission](#) (uint8_t numService, uint8_t numCharacteristic, uint8_t bitToCheck)

Function to verify the property permission of the attribute of a characteristic.

- char * [getThSenseServiceNameByUUID](#) (uint8_t Service_uuid128[], uint8_t uuid_length)

Function to obtain the service name, of the Thunderboard Sense 2 device, identified by UUID.

- char * [getThSenseCharacteristicNameByUUID](#) (uint8_t Characteristic_uuid128[], uint8_t uuid_length)

Function to obtain the Characteristic name, of the Thunderboard Sense 2 device, identified by UUID.

- char * [getThSenseDescriptorNameByUUID](#) (uint8_t Descriptor_uuid128[])

Function to obtain the Descriptor name, of the Thunderboard Sense 2 device, identified by UUID.

- char * [getThSenseDescriptorValue](#) (uint16_t value_handle, uint8_t *value)

Function to obtain the Descriptor value, of the Thunderboard Sense 2 device, identified by value_handle.

- void [printThSenseValueByHandle](#) (uint16_t value_handle, uint8_t *value, uint16_t length)

Function to print the value of the READ Attribute value of Thunderboard Sense 2 device, identified by value_handle.

- void [printThSenseNotificationValue](#) (uint16_t value_handle, uint8_t *value)

Function to print the value of the Attribute corresponding to a Notification, identified by value_handle.

- void [printThSenseProperties](#) (gatt_client_characteristic_t *characteristic)

Function to print the property permission of a Attribute, of the Thunderboard Sense 2 device, as a string.

- uint8_t [receiveFromSerial](#) (char *message)

Function to recibe the user option in the debug MENU.

- void [printMenuOptions](#) ()

Function to print the MENU options.

- void [printBLEProfile](#) ()

Function to print the BLE profile.

- void [printServiceName](#) (uint8_t numSer)

Function to print the name of a Service.

- void [printCharacteristicsNamesFromService](#) (uint8_t numService)

Function to print all the Characteristic names of the corresponding Service.

- void [PrintCharacteristicsAccordingToProperty](#) (uint8_t numService, uint8_t property)

Function to print the characteristics of a Service according to the Attribute property permission.

- void [PrintCharacteristicsAndDescriptorsAccordingToProperty](#) (uint8_t numSer, uint8_t proper)

Function to print the characteristics and Descriptors of a Service according to the Attribute property permission.

- void `menuStateMachine` ()
This function implement the Satate Machine for the MENU.
- static void `readTbSenseData` (btstack_timer_source_t *ts)
Function to read the Thunderboard Sense 2 sensors data, every 2 s using the btstack_timer.
- void `setup` ()
Setup.
- void `loop` ()
Loop.

Variables

- `Device_t` `device`
- `uint8_t` `n_chars` [NSERV_MAX]
- `uint8_t` `n_chars_index` = 0
- `uint8_t` `n_serv` = 0
- `uint8_t` `serv_index` = 0
- `uint8_t` `chars_index` = 0
- `uint8_t` `desc_index` = 0
- `uint8_t` `n_env_sensing_service` = 0
- `uint8_t` `n_iaq_service` = 0
- `uint8_t` `n_battery_service` = 0
- `uint8_t` `n_generic_access_service` = 0
- `uint8_t` `n_generic_attribute_service` = 0
- `uint8_t` `n_device_information_service` = 0
- `uint8_t` `n_power_management_service` = 0
- `uint8_t` `n_user_interface_service` = 0
- `uint8_t` `n_automation_io_service` = 0
- `uint8_t` `n_accleration_orientation_service` = 0
- `uint8_t` `n_hall_effect_service` = 0
- `bool` `conf_completed` = false
- `unsigned long` `notification_lastmills` = 0
- `unsigned long` `indication_lastmills` = 0
- `static uint16_t` `connected_id` = 0xFFFF
- `uint8_t` `UVindex` = 0
- `uint32_t` `Preasure` = 0
- `int16_t` `Temperature` = 0
- `uint16_t` `Humidity` = 0
- `uint32_t` `ALight` = 0
- `int16_t` `Sound` = 0
- `uint16_t` `ECO2` = 0
- `uint16_t` `TVOC` = 0
- `uint8_t` `Device_Name` [19]
- `uint8_t` `Appearance` = 0
- `uint8_t` `Manufacturer_Name` [19]
- `uint8_t` `Model_Number` [7]
- `uint8_t` `Serial_Number` [3]
- `uint8_t` `Hardware_Revision` [2]
- `uint8_t` `Firmware_Revision` [4]
- `uint8_t` `System_ID` [7]
- `uint8_t` `Digital_1` = 0
- `uint8_t` `Digital_2` = 0
- `uint8_t` `Buttons` = 0
- `uint32_t` `RGB_Leds` = 0

- `uint8_t Battery_Level = 0`
- `uint8_t Power_Source = 0`
- `int16_t Acceleration_axis_X = 0`
- `int16_t Acceleration_axis_Y = 0`
- `int16_t Acceleration_axis_Z = 0`
- `int16_t Orientation_axis_X = 0`
- `int16_t Orientation_axis_Y = 0`
- `int16_t Orientation_axis_Z = 0`
- `uint8_t Hall_State = 0`
- `int32_t Field_Strength = 0`
- `uint16_t Hall_Control_Point = 0`
- `char * services_name [NSERV_MAX] = {(char*)" GENERIC ACCESS", (char*)" GENERIC ATTRIBUTE", (char*)" DEVICE INFORMATION", (char*)" BATTERY", (char*)" ENVIRONMENTAL SENSING", (char*)" POWER SOURCE", (char*)" IAQ SENSING", (char*)" USER INTERFACE", (char*)" AUTOMATION IO", (char*)" ACCLERATION ORIENTATION", (char*)" HALL EFFECT" }`
- `char * caracteristics_name_gruped_by_service [NSERV_MAX][NCHAR_MAX]`
- `stateEnum_t menuOption`
- `uint8_t thereIsCharacteristic = 0`
- `uint8_t DescrvalidOption [3]`
- `static btstack_timer_source_t read_tbsense_timer`
- `uint8_t n_serv_index = 3`
- `uint8_t n_serv_sel [NSERV_MAX] = {0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1}`

4.1.1 Function Documentation

4.1.1.1 `ble_advdata_decode()`

```
uint32_t ble_advdata_decode (
    uint8_t type,
    uint8_t advdata_len,
    uint8_t * p_advdata,
    uint8_t * len,
    uint8_t * p_field_data )
```

Find the data given the type in advertising data.

Parameters

in	<i>type</i>	The type of field data.
in	<i>advdata_len</i>	Length of advertising data.
in	<i>*p_advdata</i>	The pointer of advertising data.
out	<i>*len</i>	The length of found data.
out	<i>*p_field_data</i>	The pointer of buffer to store field data.

Return values

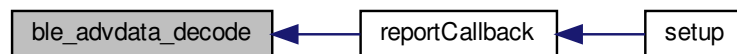
0	Find the data 1 Not find.
---	---------------------------

```

135
136     {
137     uint8_t index = 0;
138     uint8_t field_length, field_type;
139     while (index < advdata_len) {
140         field_length = p_advdata[index];
141         field_type = p_advdata[index + 1];
142         Serial.print("          - AVD/SR data decoding -> ad_type: ");
143         Serial.print(field_type, HEX);
144         Serial.print(", length: ");
145         Serial.println(field_length, HEX);
146         if (field_type == type) {
147             memcpy(p_field_data, &p_advdata[index + 2], (field_length - 1));
148             *len = field_length - 1;
149             return 0;
150         }
151         index += field_length + 1;
152     }
153     return 1;
154 }

```

Here is the caller graph for this function:



4.1.1.2 checkAttributePropertyPermission()

```

uint8_t checkAttributePropertyPermission (
    uint8_t numService,
    uint8_t numCharacteristic,
    uint8_t bitToCheck )

```

Function to verify the property permission of the attribute of a characteristic.

Parameters

in	<i>uint8_t</i>	numService The Service number
in	<i>uint8_t</i>	numCharacteristic The Characteristic number
in	<i>uint8_t</i>	bitToCheck The bit to check: 8b -> bit 1 read, bit 3 write, bit 4 notifi, bit 5 indicate , etc.

Return values

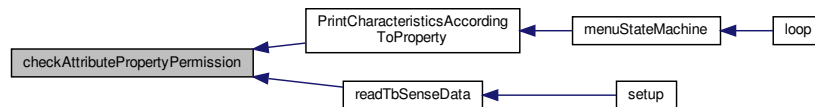
1	OK 0 Not OK.
---	--------------

```

665 {
666   if(bitRead(device.service[numService].chars[numCharacteristic].chars.properties,
        bitToCheck)){
667     return 1;
668   }else{
669     return 0;
670   }
671 }

```

Here is the caller graph for this function:



4.1.1.3 deviceConnectedCallback()

```

void deviceConnectedCallback (
    BLEStatus_t status,
    uint16_t handle )

```

Callback for the establishment of the BLE connection.

Parameters

in	<i>status</i>	BLE_STATUS_CONNECTION_ERROR or BLE_STATUS_OK.
in	<i>handle</i>	Connect handle.

This function updates the connection handle and starts the procedure to discover services.

Return values

<i>None</i>	
-------------	--

```

228                                     {
229
230   switch (status){
231
232     case BLE_STATUS_OK:
233       Serial.println("");
234       Serial.println("____ Device connected");
235       // Connect to remote device, start to discover service.
236       connected_id = handle;
237       device.connected_handle = handle;
238       Serial.print("      - Device connected handle: ");
239       Serial.println(connected_id);
240       // Start to discover service, will report result on discoveredServiceCallback.
241       Serial.println("");
242       Serial.println("____ Discovering Service");
243       ble.discoverPrimaryServices(handle);
244       break;
245

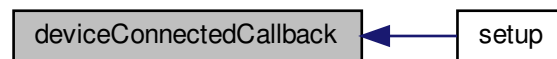
```

```

246     default:
247         break;
248     }
249 }

```

Here is the caller graph for this function:



4.1.1.4 deviceDisconnectedCallback()

```

void deviceDisconnectedCallback (
    uint16_t handle )

```

Callback for the Disconnect procedure.

Parameters

in	<i>handle</i>	Connect handle.
----	---------------	-----------------

This function updates the connection handle to a not valid value, and restarts the Scanner procedure.

Return values

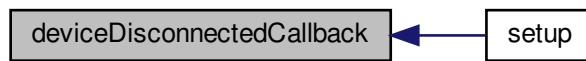
<i>None</i>	
-------------	--

```

260                                     {
261     Serial.println("");
262     Serial.println("_____ Device disconnected");
263     Serial.print("_____ - Device disconnected handle: ");
264     Serial.println(handle, HEX);
265     conf_completed = false;
266     if (connected_id == handle) {
267         Serial.println("");
268         Serial.println("_____ BLE Central restart scanning!");
269         // Disconnect from remote device, restart to scanning.
270         connected_id = 0xFFFF;
271         ble.startScanning();
272     }
273 }

```

Here is the caller graph for this function:



4.1.1.5 discoveredCharsCallback()

```
static void discoveredCharsCallback (
    BLEStatus_t status,
    uint16_t con_handle,
    gatt_client_characteristic_t * characteristic ) [static]
```

Callback for handling result of discovering characteristic.

Parameters

in	<i>status</i>	BLE_STATUS_OK/BLE_STATUS_DONE
in	<i>con_handle</i>	
in	<i>*characteristic</i>	Discoverable characteristic.

Callback for the handling result of discovering Characteristic, and once discovered, it starts the procedure for discovering descriptor.

Return values

<i>None</i>	
-------------	--

```

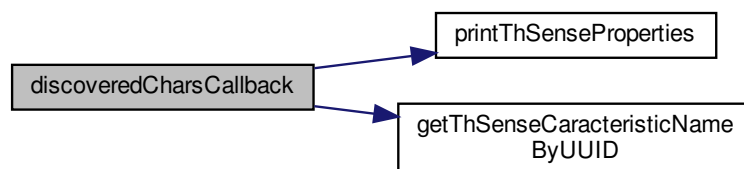
341
342     {
343     uint8_t index;
344     uint8_t uuid_length = 16;
345     char* characteristicName;
346     if (status == BLE_STATUS_OK) { // Found a characteristic.
347         Serial.println(" ");
348         Serial.print("* Service ");
349         Serial.print(serv_index, HEX);
350         Serial.print(" - Characteristic ");
351         Serial.print(chars_index, HEX);
352         Serial.println(" found successfully:");
353         Serial.print(" - Characteristic start handle: ");
354         Serial.println(characteristic->start_handle, HEX);
355         Serial.print(" - Characteristic end handle: ");
356         Serial.println(characteristic->end_handle, HEX);
357         Serial.print(" - Characteristic value handle: ");
358         Serial.println(characteristic->value_handle, HEX);
359         Serial.print(" - Characteristic properties: ");
360         Serial.print(characteristic->properties, HEX);
361         printThSenseProperties(characteristic);
362         Serial.print(" - Characteristic uuid16: ");
363         Serial.println(characteristic->uuid16, HEX);
  
```

```

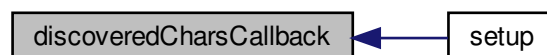
363     Serial.print("    - Characteristic uuid128 : ");
364     for (index = 0; index < 16; index++) {
365         Serial.print(characteristic->uuid128[index], HEX);
366         Serial.print(" ");
367     }
368     Serial.println(" ");
369     if (chars_index < NCHAR_MAX) {
370         characteristicName = getThSenseCharacteristicNameByUUID(
characteristic->uuid128, uuid_length);
371         Serial.println (characteristicName);
372         device.service[serv_index].chars[chars_index].chars= *
characteristic;
373         chars_index++;
374     }
375 }
376 else if (status == BLE_STATUS_DONE) {
377     n_chars[serv_index] = chars_index;
378     Serial.print("*** n_chars for Service ");
379     Serial.print(serv_index);
380     Serial.print(" = ");
381     Serial.println(n_chars[serv_index]);
382     serv_index++;
383     if (serv_index < n_serv) {
384         chars_index=0;
385         ble.discoverCharacteristics(device.connected_handle, &
device.service[serv_index].service);
386     }
387     else {
388         Serial.println("");
389         Serial.println("* Discover all Characteristics completed");
390         Serial.println("-----");
391         Serial.println("");
392         serv_index = 0;
393         chars_index = 0;
394         // All characteristics have been found, start to discover descriptors.
395         // Result will be reported on discoveredCharsDescriptorsCallback.
396         ble.discoverCharacteristicDescriptors(device.connected_handle, &
device.service[serv_index].chars[chars_index].chars);
397     }
398 }
399 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.6 discoveredCharsDescriptorsCallback()

```
static void discoveredCharsDescriptorsCallback (
    BLEStatus_t status,
    uint16_t con_handle,
    gatt_client_characteristic_descriptor_t * descriptor ) [static]
```

Callback for handling result of discovering Descriptor.

Parameters

in	<i>status</i>	BLE_STATUS_OK/BLE_STATUS_DONE
in	<i>con_handle</i>	
in	<i>*descriptor</i>	Discoverable descriptor.

Callback for the handling result of discovering Descriptor, and once discovered, puts the flag `conf_completed` to true, indicating that the initial configuration ends.

Return values

None	
------	--

```
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
{
    uint8_t index;
    char* descriptorName;
    if (status == BLE_STATUS_OK) { // Found a descriptor.
        Serial.println(" ");
        Serial.print("* Service ");
        Serial.print(serv_index, HEX);
        Serial.print(" - Characteristic ");
        Serial.print(chars_index, HEX);
        Serial.print(" - Descriptor ");
        Serial.print(desc_index, HEX);
        Serial.println(" found successfully:");
        Serial.print(" - Descriptor handle: ");
        Serial.println(descriptor->handle, HEX);
        Serial.print(" - Descriptor uuid16: ");
        Serial.println(descriptor->uuid16, HEX);
        Serial.print(" - Descriptor uuid128 : ");
        for (index = 0; index < 16; index++) {
            Serial.print(descriptor->uuid128[index], HEX);
            Serial.print(" ");
        }
        Serial.println(" ");
        if (desc_index < NDESC_MAX) {
            descriptorName = getThSenseDescriptorNameByUUID(descriptor->uuid128);
            Serial.println(descriptorName);
            device.service[serv_index].chars[chars_index].descriptor[
desc_index] = *descriptor;
            desc_index++;
        }
    }
    else if (status == BLE_STATUS_DONE) {
        desc_index = 0;
        chars_index++;
        if (chars_index < n_chars[serv_index]) {
            ble.discoverCharacteristicDescriptors(device.connected_handle, &
device.service[serv_index].chars[chars_index].chars);
        }
        else {
            chars_index = 0;
            serv_index++;
            if (serv_index < n_serv) {
                ble.discoverCharacteristicDescriptors(device.connected_handle, &
device.service[serv_index].chars[chars_index].chars);
            }
            else {
                Serial.println("");
                Serial.println("* Discover all Descriptors completed");
            }
        }
    }
}
```

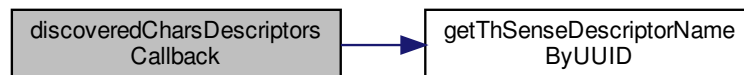


```

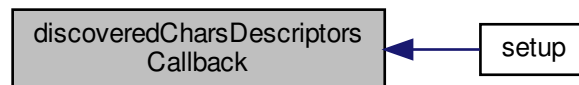
457     Serial.println("-----");
458     Serial.println("");
459     serv_index = 0;
460     chars_index = 0;
461     desc_index = 0;
462     conf_completed = true;
463 }
464 }
465 }
466 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.7 discoveredServiceCallback()

```

static void discoveredServiceCallback (
    BLEStatus_t status,
    uint16_t con_handle,
    gatt_client_service_t * service ) [static]

```

Callback for handling result of discovering Service.

Parameters

in	<i>status</i>	BLE_STATUS_OK/BLE_STATUS_DONE
in	<i>con_handle</i>	
in	<i>*service</i>	Discoverable service.

Callback for the handling result of discovering Service, and once discovered, it starts the procedure for discovering characteristic.

Return values

None	
------	--

```

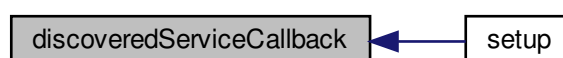
286
287     {
288     uint8_t index;
289     char* serviceName;
290     uint8_t uuid_lenght = 16;
291     if (status == BLE_STATUS_OK) { // Found a service.
292         Serial.println(" ");
293         Serial.print("* Service found successfully ");
294         Serial.print(serv_index, HEX);
295         Serial.println(" :");
296         Serial.print(" - Service start handle: ");
297         Serial.println(service->start_group_handle, HEX);
298         Serial.print(" - Service end handle: ");
299         Serial.println(service->end_group_handle, HEX);
300         Serial.print(" - Service uuid16: ");
301         Serial.println(service->uuid16, HEX);
302         Serial.print(" - Service uuid128 : ");
303         for (index = 0; index < 16; index++) {
304             Serial.print(service->uuid128[index], HEX);
305             Serial.print(" ");
306         }
307         Serial.println(" ");
308         if (serv_index < NSERV_MAX) {
309             serviceName = getThSenseServiceNameByUUID( service->uuid128, uuid_lenght)
310         };
311         Serial.println(serviceName);
312         device.service[serv_index].service= *service;
313         serv_index++;
314     }
315     else if (status == BLE_STATUS_DONE) {
316         Serial.println(" ");
317         n_serv = serv_index;
318         Serial.print("* Discover all Services completed (");
319         Serial.print(n_serv);
320         Serial.println(")");
321         Serial.println("-----");
322         Serial.println("");
323         serv_index = 0;
324         // All service have been found, start to discover characteristics.
325         // Result will be reported on discoveredCharsCallback.
326         Serial.println("_____ Discovering Characteristicd");
327         ble.discoverCharacteristics(device.connected_handle, &
328         device.service[serv_index].service);
329     }
330 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.8 gattNotifyUpdateCallback()

```
void gattNotifyUpdateCallback (
    BLEStatus_t status,
    uint16_t con_handle,
    uint16_t value_handle,
    uint8_t * value,
    uint16_t length )
```

Callback for handling notify event from remote device.

Parameters

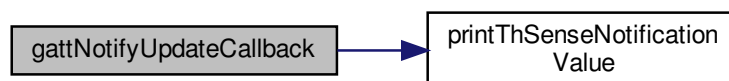
in	<i>status</i>	BLE_STATUS_OK
in	<i>con_handle</i>	
in	<i>value_handle</i>	
in	<i>*value</i>	
in	<i>length</i>	

Return values

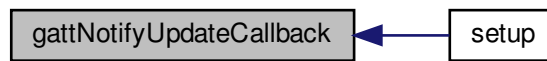
None	
------	--

```
600
601     {
602     uint8_t index;
603     Serial.println(" ");
604     Serial.print("** Received new notification (");
605     Serial.print(millis()- notification_lastmills);
606     Serial.print(" ms) - (");
607     Serial.print(length);
608     Serial.println(" bytes):");
609     notification_lastmills = millis();
610     Serial.print(" - Connection handle: ");
611     Serial.println(con_handle, HEX);
612     Serial.print(" - Characteristic value attribute handle: ");
613     Serial.println(value_handle, HEX);
614     Serial.print(" - Notified value: ");
615     for (index = 0; index < length; index++) {
616         Serial.print(value[index], HEX);
617         Serial.print(" ");
618     }
619     printThSenseNotificationValue( value_handle, value);
620 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.9 gattReadCallback()

```

void gattReadCallback (
    BLEStatus_t status,
    uint16_t con_handle,
    uint16_t value_handle,
    uint8_t * value,
    uint16_t length )
  
```

Callback for handling result of reading.

Parameters

in	<i>status</i>	BLE_STATUS_OK/BLE_STATUS_DONE/BLE_STATUS_OTHER_ERROR
in	<i>con_handle</i>	
in	<i>value_handle</i>	
in	<i>*value</i>	
in	<i>length</i>	

Return values

None	
------	--

```

479
480     {
481     if (status == BLE_STATUS_OK) {
482         Serial.println(" ");
483         Serial.println(" * Read characteristic value successfully:");
484         Serial.print(" - Connection handle: ");
485         Serial.println(con_handle, HEX);
486         Serial.print(" - Characteristic value attribute handle: ");
487         Serial.println(value_handle, HEX);
488         Serial.print(" - Characteristic value : ");
489         for (index = 0; index < length; index++) {
490             Serial.print(value[index], HEX);
491             Serial.print(" ");
492         }
493         Serial.println("");
494         printThSenseValueByHandle( value_handle, value, length);
495     }
496     else if (status != BLE_STATUS_DONE) {
497         Serial.println(" ");
498         Serial.println("! Read characteristic value FAILED");
  
```

```

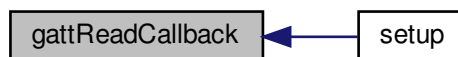
499     Serial.println(" ");
500   }
501 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.10 gattReadDescriptorCallback()

```

void gattReadDescriptorCallback (
    BLEStatus_t status,
    uint16_t con_handle,
    uint16_t value_handle,
    uint8_t * value,
    uint16_t length )

```

Callback for handling result of reading descriptor.

Parameters

in	<i>status</i>	BLE_STATUS_DONE/BLE_STATUS_OTHER_ERROR
in	<i>con_handle</i>	
in	<i>value_handle</i>	
in	<i>*value</i>	
in	<i>length</i>	

Return values

<i>None</i>	
-------------	--

```

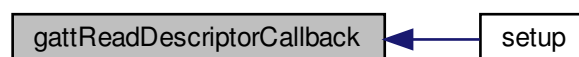
536
537     {
538     uint8_t index;
539     char* enableOrDisableMesage;
540     if(status == BLE_STATUS_OK) {
541         Serial.println("");
542         Serial.print("D --- gattReadDescriptorCallback (");
543         Serial.println(" ");
544         Serial.println("* Read descriptor value successfully:");
545         Serial.print(" - Connection handle: ");
546         Serial.println(con_handle, HEX);
547         Serial.print(" - Descriptor value attribute handle: ");
548         Serial.println(value_handle, HEX);
549         Serial.print(" - Descriptor value : ");
550         for (index = 0; index < length; index++) {
551             Serial.print(value[index], HEX);
552             Serial.print(" ");
553         }
554         Serial.println(" ");
555         enableOrDisableMesage = getThSenseDescriptorValue( value_handle, value);
556         Serial.println(enableOrDisableMesage);
557     }
558     else if (status == !BLE_STATUS_DONE) {
559         Serial.println(" ");
560         Serial.println("! ReadDescriptor FAILED");
561         Serial.println(" ");
562     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.11 gattReceivedIndicationCallback()

```

void gattReceivedIndicationCallback (
    BLEStatus_t status,
    uint16_t conn_handle,
    uint16_t value_handle,
    uint8_t * value,
    uint16_t length )

```

Callback for handling Indication event from remote device.

Parameters

in	<i>status</i>	BLE_STATUS_OK
in	<i>con_handle</i>	
in	<i>value_handle</i>	
in	<i>*value</i>	
in	<i>length</i>	

Return values

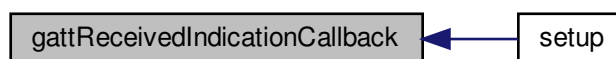
None	
------	--

```

632
633     {
634         uint8_t index;
635         Serial.println(" ");
636         Serial.print("Receive new indication:");
637         Serial.print(millis() - indication_lastmills);
638         Serial.print(" ms) - (");
639         Serial.print(length);
640         Serial.println(" bytes):");
641         indication_lastmills = millis();
642         Serial.print(" - Connection handle: ");
643         Serial.println(conn_handle, HEX);
644         Serial.print("Characteristic value attribute handle: ");
645         Serial.println(value_handle, HEX);
646         Serial.print("Indicated data: ");
647         for (index = 0; index < length; index++) {
648             Serial.print(value[index], HEX);
649             Serial.print(" ");
650         }
651         Serial.println(" ");
652     }

```

Here is the caller graph for this function:



4.1.1.12 gattWriteCCDCallback()

```

void gattWriteCCDCallback (
    BLEStatus_t status,
    uint16_t con_handle )

```

Callback for handling result of writting client characteristic configuration.

Parameters

in	<i>status</i>	BLE_STATUS_DONE/BLE_STATUS_OTHER_ERROR
in	<i>con_handle</i>	

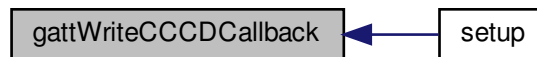
Return values

<i>None</i>	
-------------	--

```

572                                     {
573     Serial.println("");
574     Serial.print("D --- gattWriteCCCDCallback (");
575     Serial.print(status, HEX);
576     Serial.println(")");
577     if (status == BLE_STATUS_DONE) {
578         Serial.println(" ");
579         Serial.println("* Write CCCD value successfully");
580         Serial.print(" - Connection handle: ");
581         Serial.println(con_handle, HEX);
582     }
583     else {
584         Serial.println(" ");
585         Serial.println("! Write CCCD value FAILED");
586     }
587 }
```

Here is the caller graph for this function:

**4.1.1.13 gattWrittenCallback()**

```

void gattWrittenCallback (
    BLEStatus_t status,
    uint16_t con_handle )
```

Callback for handling result of writing.

Parameters

in	<i>status</i>	BLE_STATUS_DONE/BLE_STATUS_OTHER_ERROR
in	<i>con_handle</i>	

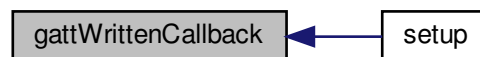
Return values

<i>None</i>	
-------------	--

```

511                                     {
512   if (status == BLE_STATUS_DONE) {
513     Serial.println(" ");
514     Serial.println("* Write characteristic value done:");
515     Serial.print("   - Connection handle: ");
516     Serial.println(con_handle, HEX);
517   }
518   else {
519     Serial.println(" ");
520     Serial.println("! Write characteristic value FAILED");
521     Serial.println(" ");
522   }
523 }
```

Here is the caller graph for this function:



4.1.1.14 getThSenseCharacteristicNameByUUID()

```

char* getThSenseCharacteristicNameByUUID (
    uint8_t Characteristic_uuid128[],
    uint8_t uuid_length )
```

Function to obtain the Characteristic name, of the Thunderboard Sense 2 device, identified by UUID.

Parameters

in	<i>Characteristic_uuid128[]</i>	The UUID of the Characteristic
in	<i>uuid_length</i>	The UUID lenght

Return values

<i>char*</i>	thSenseCharacteristicName Characteristic name associated with the UUID that was passed by parameter.
--------------	--

```

738                                     {
739
740   char* thSenseCharacteristicName;
741 }
```

```

742 if (0x00 == memcmp(Characteristic_uuid128,
Service0_Characteristic0_Device_Name_uuid, uuid_length)) {
743     thSenseCharacteristicName = (char*)" - Device Name Characteristic found successfully";
744 }else if (0x00 == memcmp(Characteristic_uuid128,
Service0_Characteristic1_Appearance_uuid, uuid_length)) {
745     thSenseCharacteristicName = (char*)" - Appearance Characteristic found successfully";
746 }else if (0x00 == memcmp(Characteristic_uuid128,
Service1_Characteristic0_Service_Changed_uuid, uuid_length))
{
747     thSenseCharacteristicName = (char*)" - Service Changed Characteristic found successfully";
748 }else if (0x00 == memcmp(Characteristic_uuid128,
Service2_Characteristic0_Manufacturer_Name_uuid,
uuid_length)) {
749     thSenseCharacteristicName = (char*)" - Manufacturer Name Characteristic found successfully";
750 }else if (0x00 == memcmp(Characteristic_uuid128,
Service2_Characteristic1_Model_Number_uuid, uuid_length)) {
751     thSenseCharacteristicName = (char*)" - Model Number Characteristic found successfully";
752 }else if (0x00 == memcmp(Characteristic_uuid128,
Service2_Characteristic2_Serial_Number_uuid, uuid_length)) {
753     thSenseCharacteristicName = (char*)" - Serial Number Characteristic found successfully";
754 }else if (0x00 == memcmp(Characteristic_uuid128,
Service2_Characteristic3_Hardware_Revision_uuid,
uuid_length)) {
755     thSenseCharacteristicName = (char*)" - Hardware Revision Characteristic found successfully";
756 }else if (0x00 == memcmp(Characteristic_uuid128,
Service2_Characteristic4_Firmware_Revision_uuid,
uuid_length)) {
757     thSenseCharacteristicName = (char*)" - Firmware Revision Characteristic found successfully";
758 }else if (0x00 == memcmp(Characteristic_uuid128,
Service2_Characteristic5_System_ID_uuid, uuid_length)) {
759     thSenseCharacteristicName = (char*)" - System ID Characteristic found successfully";
760 }else if (0x00 == memcmp(Characteristic_uuid128,
Service3_Characteristic0_Battery_Level_uuid, uuid_length)) {
761     thSenseCharacteristicName = (char*)" - Battery Level Characteristic found successfully";
762 }else if (0x00 == memcmp(Characteristic_uuid128,
Service4_Characteristic0_UV_Index_uuid, uuid_length)) {
763     thSenseCharacteristicName = (char*)" - UV Index Characteristic found successfully";
764 }else if (0x00 == memcmp(Characteristic_uuid128,
Service4_Characteristic1_Pressure_uuid, uuid_length)) {
765     thSenseCharacteristicName = (char*)" - Pressure Characteristic found successfully";
766 }else if (0x00 == memcmp(Characteristic_uuid128,
Service4_Characteristic2_Temperature_uuid, uuid_length)) {
767     thSenseCharacteristicName = (char*)" - Temperature Characteristic found successfully";
768 }else if (0x00 == memcmp(Characteristic_uuid128,
Service4_Characteristic3_Humidity_uuid, uuid_length)) {
769     thSenseCharacteristicName = (char*)" - Humidity Characteristic found successfully";
770 }else if (0x00 == memcmp(Characteristic_uuid128,
Service4_Characteristic4_Ambient_Light_uuid, uuid_length)) {
771     thSenseCharacteristicName = (char*)" - Ambient Light Characteristic found successfully";
772 }else if (0x00 == memcmp(Characteristic_uuid128,
Service4_Characteristic5_Sound_Level_uuid, uuid_length)) {
773     thSenseCharacteristicName = (char*)" - Sound Level Characteristic found successfully";
774 }else if (0x00 == memcmp(Characteristic_uuid128,
Service4_Characteristic6_Control_Point_uuid, uuid_length)) {
775     thSenseCharacteristicName = (char*)" - Control Point Characteristic found successfully";
776 }else if (0x00 == memcmp(Characteristic_uuid128,
Service5_Characteristic0_Power_Source_uuid, uuid_length)) {
777     thSenseCharacteristicName = (char*)" - Power Source Characteristic found successfully";
778 }else if (0x00 == memcmp(Characteristic_uuid128,
Service6_Characteristic0_ECO2_uuid, uuid_length)) {
779     thSenseCharacteristicName = (char*)" - ECO2 Characteristic found successfully";
780 }else if (0x00 == memcmp(Characteristic_uuid128,
Service6_Characteristic1_TVOC_uuid, uuid_length)) {
781     thSenseCharacteristicName = (char*)" - TVOC Characteristic found successfully";
782 }else if (0x00 == memcmp(Characteristic_uuid128,
Service6_Characteristic2_Control_Point_uuid, uuid_length)) {
783     thSenseCharacteristicName = (char*)" - Control Point Characteristic found successfully";
784 }else if (0x00 == memcmp(Characteristic_uuid128,
Service7_Characteristic0_Buttons_uuid, uuid_length)) {
785     thSenseCharacteristicName = (char*)" - Buttons Characteristic found successfully";
786 }else if (0x00 == memcmp(Characteristic_uuid128,
Service7_Characteristic1_Leds_uuid, uuid_length)) {
787     thSenseCharacteristicName = (char*)" - Leds Characteristic found successfully";
788 }else if (0x00 == memcmp(Characteristic_uuid128,
Service7_Characteristic2_RGB_Leds_uuid, uuid_length)) {
789     thSenseCharacteristicName = (char*)" - RGB Leds Characteristic found successfully";
790 }else if (0x00 == memcmp(Characteristic_uuid128,
Service7_Characteristic3_Control_Point_uuid, uuid_length)) {
791     thSenseCharacteristicName = (char*)" - Control Point Characteristic found successfully";
792 }else if (0x00 == memcmp(Characteristic_uuid128,
Service8_Characteristic0_Digital_1_uuid, uuid_length)) {
793     thSenseCharacteristicName = (char*)" - Digital 1 Characteristic found successfully";
794 }else if (0x00 == memcmp(Characteristic_uuid128,
Service8_Characteristic1_Digital_2_uuid, uuid_length)) {
795     thSenseCharacteristicName = (char*)" - Digital 2 Characteristic found successfully";
796 }else if (0x00 == memcmp(Characteristic_uuid128,
Service9_Characteristic0_Acceleration_uuid, uuid_length)) {

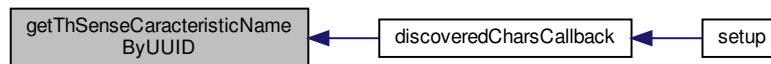
```

```

797     thSenseCharacteristicName = (char*)" - Acceleration Characteristic found successfully";
798 }else if (0x00 == memcmp(Characteristic_uuid128,
Service9_Characteristic1_Orientation_uuid, uuid_length)) {
799     thSenseCharacteristicName = (char*)" - Orientation Characteristic found successfully";
800 }else if (0x00 == memcmp(Characteristic_uuid128,
Service9_Characteristic2_Control_Point_uuid, uuid_length)) {
801     thSenseCharacteristicName = (char*)" - Control Point Characteristic found successfully";
802 }else if (0x00 == memcmp(Characteristic_uuid128,
ServiceA_Characteristic0_State_uuid, uuid_length)) {
803     thSenseCharacteristicName = (char*)" - State Characteristic found successfully";
804 }else if (0x00 == memcmp(Characteristic_uuid128,
ServiceA_Characteristic1_Field_Strength_uuid, uuid_length)) {
805     thSenseCharacteristicName = (char*)" - Field Strength Characteristic found successfully";
806 }else if (0x00 == memcmp(Characteristic_uuid128,
ServiceA_Characteristic2_Control_Point_uuid, uuid_length)) {
807     thSenseCharacteristicName = (char*)" - Control Point Characteristic found successfully";
808 }else {
809     thSenseCharacteristicName = (char*)" _ The Characteristic Name is not define in the Central
device";
810 }
811 return thSenseCharacteristicName;
812 }

```

Here is the caller graph for this function:



4.1.1.15 getThSenseDescriptorNameByUUID()

```

char* getThSenseDescriptorNameByUUID (
    uint8_t Descriptor_uuid128[] )

```

Function to obtain the Descriptor name, of the Thunderboard Sense 2 device, identified by UUID.

Parameters

in	<i>Descriptor_uuid128[]</i>	The UUID128 of the descriptor
----	-----------------------------	-------------------------------

Return values

*char	DescriptorName Descriptor name associated with the UUID that was passed by parameter.
-------	---

```

821                                     {
822
823     char* DescriptorName;
824
825     if (0x00 == memcmp(Descriptor_uuid128,
Client_Characteristic_Configuration_uuid, sizeof(Descriptor_uuid128
))) {
826         DescriptorName = (char*)" - Client_Characteristic_Configuration";
827     }else if (0x00 == memcmp(Descriptor_uuid128,
Characteristic_Presentation_Format_uuid, sizeof(Descriptor_uuid128))
) {

```

```

828     DescriptorName = (char*)" - Characteristic_Presentation_Format" ;
829 }else if (0x00 == memcmp(Descriptor_uuid128, noOfDigitals_uuid, sizeof(
Descriptor_uuid128))) {
830     DescriptorName = (char*)" - noOfDigitals";
831 }else{
832     DescriptorName = (char*)" - The Descriptor Name is not define the in Central device";
833 }
834 return DescriptorName;
835 }

```

Here is the caller graph for this function:



4.1.1.16 getThSenseDescriptorValue()

```

char* getThSenseDescriptorValue (
    uint16_t value_handle,
    uint8_t * value )

```

Function to obtain the Descriptor value, of the Thunderboard Sense 2 device, identified by value_handle.

Parameters

in	<i>value_handle</i>	The value_handle of the Descriptor
in	<i>*value</i>	The value of the Descriptor

Return values

None	
------	--

```

845                                     {
846
847     char* thSenseDescriptorValue;
848
849     if (value_handle == device.service[n_generic_attribute_service].
chars[0].descriptor[0].handle) {
850         if ((value[0] == 0x02) && (value[1] == 0x00)) {
851             thSenseDescriptorValue = (char*)"_____ Indication are enabled for Generic Attribute Service";
852         }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
853             thSenseDescriptorValue = (char*)"_____ Indication are disabled for Generic Attribute Service";
854         }
855     }else if (value_handle == device.service[n_battery_service].chars[0].
descriptor[0].handle) {
856         if ((value[0] == 0x01) && (value[1] == 0x00)) {
857             thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Battery Service";
858         }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
859             thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Battery Service";
860         }
861     }else if (value_handle == device.service[n_env_sensing_service].chars
[6].descriptor[0].handle) {
862         if ((value[0] == 0x02) && (value[1] == 0x00)) {

```

```

863     thSenseDescriptorValue = (char*)"_____ Indication are enabled for Environmental Sensing Service";
864 }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
865     thSenseDescriptorValue = (char*)"_____ Indication are disabled for Environmental Sensing Service";
866 }
867 }else if (value_handle == device.service[n_iaq_service].chars[2].descriptor[0
].handle) {
868     if ((value[0] == 0x02) && (value[1] == 0x00)) {
869         thSenseDescriptorValue = (char*)"_____ Indication are enabled for IAQ Service";
870     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
871         thSenseDescriptorValue = (char*)"_____ Indication are disabled for IAQ Service";
872     }
873 }else if (value_handle == device.service[
n_user_interface_service].chars[1].descriptor[0].handle) {
874     if ((value[0] == 0x02) && (value[1] == 0x00)) {
875         thSenseDescriptorValue = (char*)"_____ Indication are enabled for User Interface Service";
876     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
877         thSenseDescriptorValue = (char*)"_____ Indication are disabled for User Interface Service";
878     }
879 }else if (value_handle == device.service[
n_user_interface_service].chars[2].descriptor[0].handle) {
880     if ((value[0] == 0x02) && (value[1] == 0x00)) {
881         thSenseDescriptorValue = (char*)"_____ Indication are enabled for User Interface Service";
882     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
883         thSenseDescriptorValue = (char*)"_____ Indication are disabled for User Interface Service";
884     }
885 }else if (value_handle == device.service[n_user_interface_service].
chars[3].descriptor[0].handle) {
886     if ((value[0] == 0x02) && (value[1] == 0x00)) {
887         thSenseDescriptorValue = (char*)"_____ Indication are enabled for User Interface Service";
888     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
889         thSenseDescriptorValue = (char*)"_____ Indication are disabled for User Interface Service";
890     }
891 }else if (value_handle == device.service[n_automation_io_service].
chars[0].descriptor[0].handle) {
892     if ((value[0] == 0x01) && (value[1] == 0x00)) {
893         thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Automation IO Service";
894     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
895         thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Automation IO Service";
896     }
897 }else if (value_handle == device.service[
n_acceleration_orientation_service].chars[0].descriptor[0].handle) {
898     if ((value[0] == 0x01) && (value[1] == 0x00)) {
899         thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Acceleration Characteristic";
900     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
901         thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Acceleration Characteristic";
902     }
903 }else if (value_handle == device.service[
n_acceleration_orientation_service].chars[1].descriptor[0].handle) {
904     if ((value[0] == 0x01) && (value[1] == 0x00)) {
905         thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Orientation Characteristic";
906     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
907         thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Orientation Characteristic";
908     }
909 }else if (value_handle == device.service[
n_acceleration_orientation_service].chars[2].descriptor[0].handle) {
910     if ((value[0] == 0x02) && (value[1] == 0x00)) {
911         thSenseDescriptorValue = (char*)"_____ Indication are enabled for User Orientation Acceleration
Characteristic";
912     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
913         thSenseDescriptorValue = (char*)"_____ Indication are disabled for User Orientation Acceleration
Characteristic";
914     }
915 }else if (value_handle == device.service[n_hall_effect_service].chars[0
].descriptor[0].handle) {
916     if ((value[0] == 0x01) && (value[1] == 0x00)) {
917         thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Hall State Characteristic";
918     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
919         thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Hall State Characteristic";
920     }
921 }else if (value_handle == device.service[n_hall_effect_service].chars
[2].descriptor[0].handle) {
922     if ((value[0] == 0x01) && (value[1] == 0x00)) {
923         thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Hall effect Characteristic";
924     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
925         thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Hall effect Characteristic";
926     }
927 }else if (value_handle == device.service[n_hall_effect_service].chars[1
].descriptor[0].handle) {
928     if ((value[0] == 0x01) && (value[1] == 0x00)) {
929         thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Field Strength Characteristic";
930     }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
931         thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Field Strength Characteristic";
932     }
933 }else{
934     thSenseDescriptorValue = (char*)"_____ The Descriptor value is not defined in the Central device";

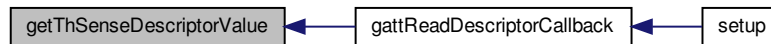
```

```

935     }
936     Serial.println(" ");
937     return thSenseDescriptorValue;
938 }

```

Here is the caller graph for this function:



4.1.1.17 getThSenseServiceNameByUUID()

```

char* getThSenseServiceNameByUUID (
    uint8_t Service_uuid128[],
    uint8_t uuid_lenght )

```

Function to obtain the service name, of the Thunderboard Sense 2 device, identified by UUID.

Parameters

in	<i>Service_uuid128[]</i>	The UUID of the Service
in	<i>uuid_lenght</i>	The UUID lenght

Return values

<i>char*</i>	thSenseServiceName Service name associated with the UUID that was passed by parameter.
--------------	--

```

687                                     {
688
689     char* thSenseServiceName;
690
691     if (0x00 == memcmp(Service_uuid128, battery_service_uuid, uuid_lenght)) {
692         n_battery_service = serv_index;
693         thSenseServiceName = (char*)" - Battery Service found successfully";
694     }else if (0x00 == memcmp(Service_uuid128, environmental_sensing_service_uuid,
695         uuid_lenght)) {
696         n_env_sensing_service = serv_index;
697         thSenseServiceName = (char*)" - Environmental Sensing Service found successfully" ;
698     }else if (0x00 == memcmp(Service_uuid128, iaq_service_uuid, uuid_lenght)) {
699         n_iaq_service = serv_index;
700         thSenseServiceName = (char*)" - IAQ Service found successfully";
701     }else if (0x00 == memcmp(Service_uuid128, generic_access_service_uuid,
702         uuid_lenght)) {
703         n_generic_access_service = serv_index;
704         thSenseServiceName = (char*)" - Generic Access Service found successfully";
705     }else if (0x00 == memcmp(Service_uuid128, generic_attribute_service_uuid,
706         uuid_lenght)) {
707         n_generic_attribute_service = serv_index;
708         thSenseServiceName = (char*)" - Generic Attribute Service found successfully";
709     }else if (0x00 == memcmp(Service_uuid128, device_information_service_uuid,
710         uuid_lenght)) {
711         n_device_information_service = serv_index;
712         thSenseServiceName = (char*)" - Device Information Service found successfully";
713     }else if (0x00 == memcmp(Service_uuid128, power_management_service_uuid,
714         uuid_lenght)) {
715         n_power_management_service = serv_index;
716         thSenseServiceName = (char*)" - Power Management Service found successfully";
717     }else {
718         thSenseServiceName = (char*)" - Service not found";
719     }
720
721     return thSenseServiceName;
722 }

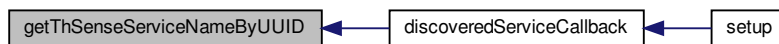
```

```

        uuid_lenght)) {
710     n_power_management_service = serv_index;
711     thSenseServiceName = (char*)" - Power Management Service found successfully";
712 }else if (0x00 == memcmp(Service_uuid128, user_interface_service_uuid,
        uuid_lenght)) {
713     n_user_interface_service = serv_index;
714     thSenseServiceName = (char*)" - User Interface Service found successfully";
715 }else if (0x00 == memcmp(Service_uuid128, automation_io_service_uuid,
        uuid_lenght)) {
716     n_automation_io_service = serv_index;
717     thSenseServiceName = (char*)" - Automation IO Service found successfully";
718 }else if (0x00 == memcmp(Service_uuid128, accleration_orientation_service_uuid
        , uuid_lenght)) {
719     n_accleration_orientation_service =
        serv_index;
720     thSenseServiceName = (char*)" - Accleration Orientation Service found successfully";
721 }else if (0x00 == memcmp(Service_uuid128, hall_effect_service_uuid, uuid_lenght))
        {
722     n_hall_effect_service = serv_index;
723     thSenseServiceName = (char*)" - Hall Effect Service found successfully";
724 }else{
725     thSenseServiceName = (char*)" - The name of the service is not defined in the Central device";
726 }
727 return thSenseServiceName;
728 }

```

Here is the caller graph for this function:



4.1.1.18 loop()

```
void loop ( )
```

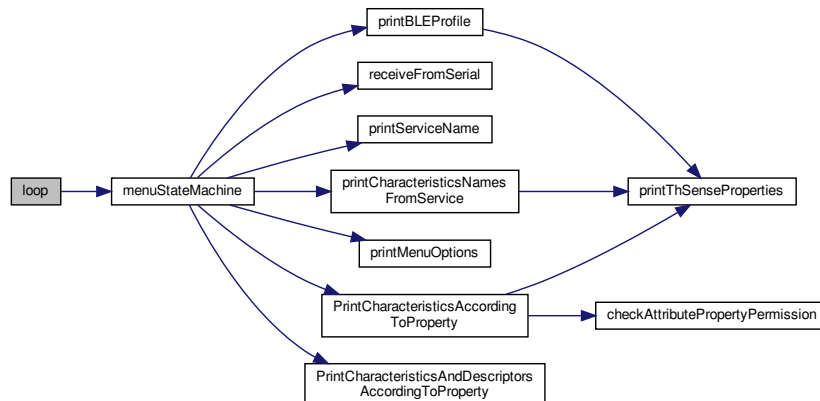
Loop.

```

1818     {
1819
1820     #if MENU_DEBUG >= 1
1821
1822     if (conf_completed == true){
1823         menuStateMachine();
1824     }
1825
1826     #endif
1827 }

```

Here is the call graph for this function:



4.1.1.19 menuStateMachine()

```
void menuStateMachine ( )
```

This function implement the Satate Machine for the MENU.

Parameters

None	States: BLE_CENTRAL_READ_CHARACTERISTIC_VALUE BLE_CENTRAL_READ_DESCRIPTOR_VALUE BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS BLE_CENTRAL_WRITE
------	--

Return values

None	
------	--

```

1463         {
1464
1465     uint8_t numService      = 0;
1466     uint8_t numCharacteristic = 0;
1467     uint8_t enable_disable  = 0;
1468
1469     printBLEProfile();
1470     numService = receiveFromSerial((char*)" Enter number of service: ");
1471     if(numService >= 0 && numService < NSERV_MAX){
1472         Serial.println(numService);
1473         printServiceName(numService);
1474         printCharacteristicsNamesFromService(numService);
1475         printMenuOptions();
1476         menuOption = (stateEnum_t)receiveFromSerial((char*)"Enter the
number corresponding to the menu option to perform: ");
1477         if(menuOption >=0 && menuOption <=4){
1478             Serial.print(menuOption);
1479             Serial.println("");
1480
1481             switch (menuOption){

```



```

1482
1483     case BLE_CENTRAL_READ_CHARACTERISTIC_VALUE:
1484         PrintCharacteristicsAccordingToProperty(numService, 1);
1485         if(thereIsCharacteristic != 0){
1486             Serial.println("");
1487             numCharacteristic = receiveFromSerial((char*)" Enter number of
characteristic: ");
1488             if(numCharacteristic >= 0 && numCharacteristic <
thereIsCharacteristic){
1489                 Serial.print(numCharacteristic);
1490                 Serial.println("");
1491                 ble.readValue(device.connected_handle, &
device.service[numService].chars[numCharacteristic].chars);
1492                 delay(3000);
1493                 //Serial.println("_____LEIDO");
1494                 // Serial.println(thereIsCharacteristic);//DEBUG MIRrararararaaaaaaaaaaaaaaaaaaaaaaa
1495                 // numCharacteristic = receiveFromSerial((char*)"_____Pulse cualquier tecla para volver");
1496             }else{
1497                 Serial.println(" Not a valid option ");
1498                 delay(3000);
1499             }
1500         }else{
1501             Serial.println(" Not characteristic with read properties!!!! ");
1502             delay(3000);
1503         }
1504         break;
1505
1506     case BLE_CENTRAL_READ_DESCRIPTOR_VALUE:
1507         PrintCharacteristicsAndDescriptorsAccordingToProperty
(numService, 1);
1508         if(thereIsCharacteristic != 0){
1509             Serial.println("");
1510             numCharacteristic = receiveFromSerial((char*)" Enter number of
characteristic: ");
1511             if(((device.service[numService].chars[numCharacteristic].descriptor[0].uuid16)== 0
x2902)){
1512                 Serial.print(numCharacteristic);
1513                 Serial.println("");
1514                 ble.readDescriptorValue(device.connected_handle, &
device.service[numService].chars[numCharacteristic].descriptor[0]);
1515                 delay(3000);
1516             }else{
1517                 Serial.println(" Not a valid option ");
1518                 delay(3000);
1519             }
1520         }else{
1521             Serial.println(" They are Not Characteristic Descriptors!!!! ");
1522             delay(3000);
1523         }
1524         break;
1525
1526     case BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS:
1527         PrintCharacteristicsAccordingToProperty(numService, 4);
1528         if(thereIsCharacteristic != 0){
1529             Serial.println("");
1530             numCharacteristic = receiveFromSerial((char*)" Enter number of
characteristic: ");
1531             if(numCharacteristic >= 0 && numCharacteristic <
thereIsCharacteristic){
1532                 Serial.print(numCharacteristic);
1533                 Serial.println("");
1534                 ble.writeClientCharsConfigDescriptor(device.connected_handle, &
device.service[numService].chars[numCharacteristic].chars,
GATT_CLIENT_CHARACTERISTICS_CONFIGURATION_NOTIFICATION);
1535                 enable_disable = 1;
1536                 while(enable_disable){
1537                     Serial.println("Para deshabilitar pulse 0");
1538                     enable_disable = Serial.read() - '0';
1539                     delay(2000);
1540                 }
1541                 ble.writeClientCharsConfigDescriptor(device.connected_handle, &
device.service[numService].chars[numCharacteristic].chars,
GATT_CLIENT_CHARACTERISTICS_CONFIGURATION_NONE);
1542                 delay(3000);
1543             }else{
1544                 Serial.println(" Not a valid option ");
1545                 delay(3000);
1546             }
1547         }else{
1548             Serial.println(" Not Characteristic with notify properties!!!! ");
1549             delay(3000);
1550         }
1551         break;
1552
1553     case BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS:
1554         PrintCharacteristicsAccordingToProperty(numService, 5);
1555         if(thereIsCharacteristic != 0){

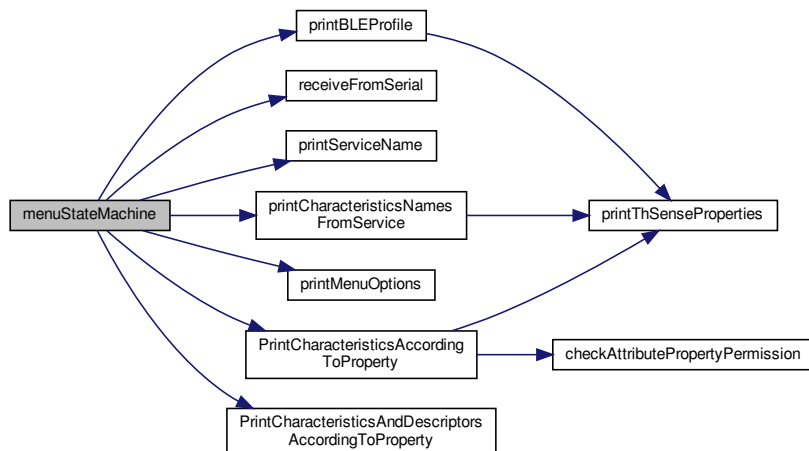
```

```

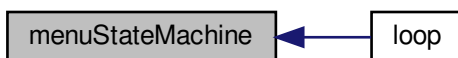
1556         Serial.println("");
1557         numCharacteristic = receiveFromSerial((char*)" Enter number of
characteristic): ");
1558         if(numCharacteristic >= 0 && numCharacteristic <
thereIsCharacteristic){
1559             Serial.print(numCharacteristic);
1560             Serial.println("");
1561             enable_disable = receiveFromSerial((char*)" To enable enter 2, and to
disable enter 3): ");
1562             if(enable_disable == 2){
1563                 ble.writeClientCharsConfigDescriptor(device.
connected_handle, &device.service[numService].chars[numCharacteristic].chars,
GATT_CLIENT_CHARACTERISTICS_CONFIGURATION_INDICATION);
1564                 delay(3000);
1565             }else if(enable_disable == 3){
1566                 ble.writeClientCharsConfigDescriptor(device.
connected_handle, &device.service[numService].chars[numCharacteristic].chars,
GATT_CLIENT_CHARACTERISTICS_CONFIGURATION_NONE);
1567                 delay(3000);
1568             }else{
1569                 Serial.println(" Not a valid option ");
1570                 delay(3000);
1571             }
1572         }else{
1573             Serial.println(" Not a valid option ");
1574             delay(3000);
1575         }
1576     }else{
1577         Serial.println(" Not Characteristic with Indicate properties!!!! ");
1578         delay(3000);
1579     }
1580     break;
1581
1582     case BLE_CENTRAL_WRITE:
1583         uint8_t write_data[20];
1584         uint8_t w_data_length = 0;
1585         uint8_t write_data_temp = 0;
1586         PrintCharacteristicsAccordingToProperty(numService, 3);
1587         if(thereIsCharacteristic != 0){
1588             Serial.println("");
1589             numCharacteristic = receiveFromSerial((char*)" Enter number of
characteristic): ");
1590             if(numCharacteristic >= 0 && numCharacteristic <
thereIsCharacteristic){
1591                 Serial.println(numCharacteristic);
1592                 w_data_length = receiveFromSerial((char*)" Enter write data lenght in
Bytes): ");
1593                 Serial.println(w_data_length);
1594                 for (int i=0; i<w_data_length;i++){
1595                     for(int j=0; j<2;j++){
1596                         write_data_temp = receiveFromSerial((char*)"");
1597                         if(!j){
1598                             write_data[i]= write_data_temp << 4;
1599                         }else{
1600                             write_data[i]|= write_data_temp;
1601                         }
1602                     }
1603                 }
1604                 ble.writeValue(device.connected_handle, device.
service[numService].chars[numCharacteristic].chars.value_handle, w_data_length, write_data);
1605                 delay(3000);
1606             }else{
1607                 Serial.println(" Not a valid option ");
1608                 delay(3000);
1609             }
1610         }else{
1611             Serial.println(" Not Characteristic with Write properties!!!! ");
1612             delay(3000);
1613         }
1614     }break;
1615 }
1616 }
1617 }
1618 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.20 printBLEProfile()

```
void printBLEProfile ( )
```

Function to print the BLE profile.

Parameters

None	
------	--

Return values

None	
------	--

```

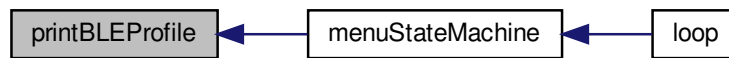
1307
1308 uint8_t numService      = 0;
1309 uint8_t numCharacteristic = 0;
1310 uint8_t numDescriptor   = 0;
1311 uint16_t descriptorUUID = 0;
1312
1313 for( numService = 0; numService < NSERV_MAX; numService++){
1314     Serial.print(" ");
1315     Serial.print(" ");
1316     Serial.print("* Service ");
1317     Serial.print(numService, HEX);
1318     Serial.println(services_name[numService]);
1319     for(numCharacteristic = 0; numCharacteristic < NCHAR_MAX; numCharacteristic++){
1320         if(characteristics_name_gruped_by_service [numService][
numCharacteristic] != NULL){
1321             Serial.print(" ");
1322             Serial.print("- Characteristic ");
1323             Serial.print(numCharacteristic);
1324             Serial.print(characteristics_name_gruped_by_service [numService
] [numCharacteristic]);
1325             printThSenseProperties(&device.service[numService].chars[
numCharacteristic].chars);
1326             for(numDescriptor = 0; numDescriptor < NDESC_MAX; numDescriptor++){
1327                 descriptorUUID = device.service[numService].chars[numCharacteristic].descriptor[
numDescriptor].uuid16;
1328                 switch(descriptorUUID){
1329                     case 0x2902:
1330                         Serial.print(" ");
1331                         Serial.print("- Descriptor");
1332                         Serial.print(numDescriptor);
1333                         Serial.println(" Client Characteristic Configuration ");
1334                         break;
1335
1336                     case 0x2904:
1337                         Serial.print(" ");
1338                         Serial.print("- Descriptor ");
1339                         Serial.print(numDescriptor);
1340                         Serial.println(" Characteristic Presentation Format ");
1341                         break;
1342
1343                     case 0x2909:
1344                         Serial.print(" ");
1345                         Serial.print("- Descriptor");
1346                         Serial.print(numDescriptor);
1347                         Serial.println(" noOfDigitals ");
1348                         break;
1349                 }
1350             }
1351         }
1352     }
1353 }
1354 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.21 PrintCharacteristicsAccordingToProperty()

```
void PrintCharacteristicsAccordingToProperty (
    uint8_t numService,
    uint8_t property )
```

Function to print the characteristics of a Service according to the Attribute property permission.

Parameters

None	uint8_t numSer The number of the Service
------	--

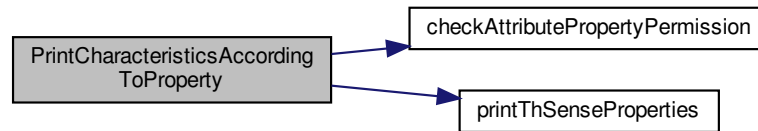
Return values

None	
------	--

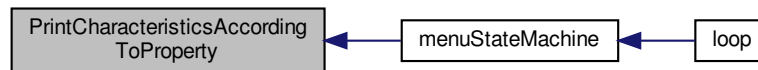
```

1403                                     {
1404
1405     thereIsCharacteristic = 0;
1406
1407     for(int num_Characteristic = 0; num_Characteristic < NCHAR_MAX; num_Characteristic++){
1408         if(characteristics_name_gruped_by_service [numService][
1409 num_Characteristic] != NULL){
1409             if(checkAttributePropertyPermission(numService, num_Characteristic,
1410 property)){
1410                 thereIsCharacteristic++;
1411                 Serial.print(" ");
1412                 Serial.print(num_Characteristic);
1413                 Serial.print(characteristics_name_gruped_by_service [numService
1414 ] [num_Characteristic]);
1414                 printThSenseProperties (&device.service[numService].chars[
1415 num_Characteristic].chars);
1415             }
1416         }
1417     }
1418 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.22 PrintCharacteristicsAndDescriptorsAccordingToProperty()

```

void PrintCharacteristicsAndDescriptorsAccordingToProperty (
    uint8_t numSer,
    uint8_t proper )
  
```

Function to print the characteristics and Descriptors of a Service according to the Attribute property permission.

Parameters

None	
------	--

Return values

None	
------	--

```

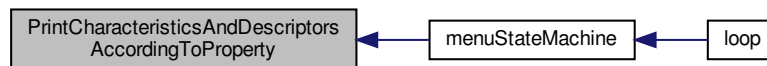
1427                                     {
1428
1429     thereIsCharacteristic = 0;
1430
1431     for(uint8_t numero_Characteristica = 0; numero_Characteristica < NCHAR_MAX; numero_Characteristica+
1432     ){
1433         if(characteristics_name_gruped_by_service [numSer] [
1434         numero_Characteristica] != NULL){
1435             for(uint8_t numero_Descriptor = 0; numero_Descriptor < NDESC_MAX; numero_Descriptor++){
1436                 if((device.service[numSer].chars[numero_Characteristica].descriptor[numero_Descriptor].
1437                 uuid16 & (1 << 13)) != 0){
1438                     Serial.print("      ");
1439                 }
1440             }
1441         }
1442     }
1443 }
  
```

```

1436         Serial.print(numero_Characteristica);
1437         Serial.println(characteristics_name_gruped_by_service [numSer
1438 ] [numero_Characteristica]);
1438         Serial.print("          ");
1439         Serial.print(numero_Descriptor);
1440         Serial.println(" Client Characteristic Configuration ");
1441         thereIsCharacteristic++;
1442     }
1443 }
1444 }
1445 }
1446 }

```

Here is the caller graph for this function:



4.1.1.23 printCharacteristicsNamesFromService()

```

void printCharacteristicsNamesFromService (
    uint8_t numService )

```

Function to print all the Characteristic names of the corresponding Service.

Parameters

<code>uint8_t</code>	numSer The number of the Service
----------------------	----------------------------------

Return values

<i>None</i>	
-------------	--

```

1377         {
1378
1379         for(uint8_t num_Characteristic = 0; num_Characteristic < NCHAR_MAX; num_Characteristic++){
1380             if(characteristics_name_gruped_by_service [numService] [
1381 num_Characteristic] != NULL){
1381                 Serial.print("          ");
1382                 Serial.print(num_Characteristic);
1383                 Serial.print(characteristics_name_gruped_by_service [numService] [
1384 num_Characteristic]);
1384                 printThSenseProperties (&device.service[numService].chars[
1385 num_Characteristic].chars);
1385                 for(uint8_t numDescriptor = 0; numDescriptor < NDESC_MAX; numDescriptor++){
1386                     if((device.service[numService].chars[num_Characteristic].descriptor[numDescriptor].
1387 uuid16 & 1 << 13) != 0){//0x2900 -- UUID IS NOT NULL
1387                         Serial.print("          ");
1388                         Serial.print(numDescriptor);
1389                         Serial.println(" Client Characteristic Configuration ");
1390                     }
1391                 }

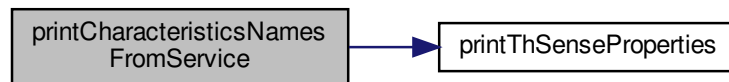
```

```

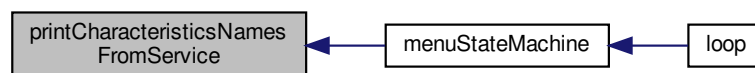
1392     }
1393 }
1394 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.24 printMenuOptions()

```
void printMenuOptions ( )
```

Function to print the MENU options.

Parameters

None	
------	--

Return values

None	
------	--

```

1286     {
1287
1288     Serial.println("");
1289     Serial.println("MENU:");
1290     Serial.println("-----");
1291     Serial.println("0. Leer Atributo");
1292     Serial.println("1. Leer Descriptor");
1293     Serial.println("2. Habilitar/deshabilitar Notificaciones");
1294     Serial.println("3. Habilitar/deshabilitar Indicaciones");
1295     Serial.println("4. Escribir Atributo");

```

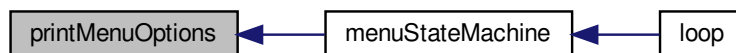


```

1296 Serial.println("-----");
1297 }

```

Here is the caller graph for this function:



4.1.1.25 printServiceName()

```

void printServiceName (
    uint8_t numSer )

```

Function to print the name of a Service.

Parameters

in	<i>uint8_t</i>	numSer The identifier of the Service.
----	----------------	---------------------------------------

Return values

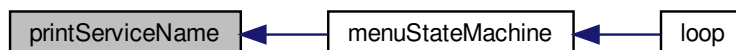
None

```

1363 {
1364
1365 Serial.print(" ");
1366 Serial.print(numSer);
1367 Serial.println(services_name[numSer]);
1368 }

```

Here is the caller graph for this function:



4.1.1.26 printThSenseNotificationValue()

```
void printThSenseNotificationValue (
    uint16_t value_handle,
    uint8_t * value )
```

Function to print the value of the Attribute corresponding to a Notification, identified by value_handle.

Parameters

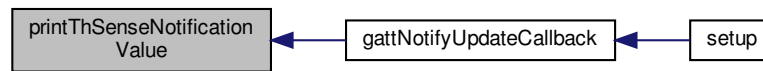
in	<i>value_handle</i>	The handle of the Attribute from which the notification is received
in	<i>*value</i>	Pointer to the value of the attribute from which the notification is received

Return values

None	
------	--

```
1151                                     {
1152
1153     switch(value_handle){
1154
1155         case 0x19:
1156             Battery_Level = value[0];
1157             Serial.print(characteristics_name_gruped_by_service [3][0]);
1158             Serial.print(Battery_Level);
1159             Serial.println(" %");
1160             break;
1161
1162         case 0x58:
1163             Hall_State = value[0];
1164             Serial.print(characteristics_name_gruped_by_service [10][0]);
1165             Serial.print(Hall_State);
1166             break;
1167
1168         case 0x5B :
1169             Field_Strength = int32_t((value[3] << 24) + (value[2] << 16) + (value[1] << 8) + value[0
1170     ]);
1171             Serial.print(characteristics_name_gruped_by_service [10][1]);
1172             Serial.print(Field_Strength);
1173             Serial.println(" uT");
1174             break;
1175
1176         case 0x4E:
1177             Acceleration_axis_X = int16_t((value[1] << 8) + value[0])/1000;
1178             Acceleration_axis_Y = int16_t((value[3] << 8) + value[2])/1000;
1179             Acceleration_axis_Z = int16_t((value[5] << 8) + value[4])/1000;
1180             Serial.println(characteristics_name_gruped_by_service [9][0]);
1181             Serial.print(" Acceleration axis X = ");
1182             Serial.println(Acceleration_axis_X + "g");
1183             Serial.print(" Acceleration axis Y = ");
1184             Serial.println(Acceleration_axis_Y + "g");
1185             Serial.print(" Acceleration axis Z = ");
1186             Serial.println(Acceleration_axis_Z + "g");
1187             break;
1188
1189         case 0x51 :
1190             Orientation_axis_X = int16_t((value[1] << 8) + value[0])/100;
1191             Orientation_axis_Y = int16_t((value[3] << 8) + value[2])/100;
1192             Orientation_axis_Z = int16_t((value[5] << 8) + value[4])/100;
1193             Serial.println(characteristics_name_gruped_by_service [9][0]);
1194             Serial.print(" Orientation axis X = ");
1195             Serial.println(Orientation_axis_X + "°");
1196             Serial.print(" Orientation axis Y = ");
1197             Serial.println(Orientation_axis_Y + "°");
1198             Serial.print(" Orientation axis Z = ");
1199             Serial.println(Orientation_axis_Z + "°");
1200             break;
1201
1202         default:
1203             Serial.println("Characteristic handle value is not define");
1204             break;
1205     }
```

Here is the caller graph for this function:



4.1.1.27 printThSenseProperties()

```
void printThSenseProperties (
    gatt_client_characteristic_t * characteristic )
```

Function to print the property permission of a Attribute, of the Thunderboard Sense 2 device, as a string.

Parameters

in	<i>*gatt_client_characteristic_t</i>	characteristic Pointer to the characteristic struct
----	--------------------------------------	---

Return values

<i>None</i>	
-------------	--

```

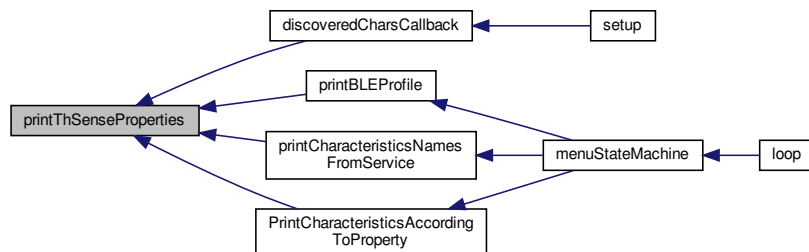
1214                                     {
1215
1216     switch (characteristic->properties){
1217
1218         case 0x02 :
1219             Serial.println("    -Read ");
1220             break;
1221
1222         case 0x0A :
1223             Serial.println("    -Read and Write ");
1224             break;
1225
1226         case 0x20 :
1227             Serial.println("    -Indicate ");
1228             break;
1229
1230         case 0x12 :
1231             Serial.println("    -Read and Notify ");
1232             break;
1233
1234         case 0x28 :
1235             Serial.println("    -Write and Indicate ");
1236             break;
1237
1238         case 0x2A :
1239             Serial.println("    -Read and Write and Indicate ");
1240             break;
1241
1242         case 0x10 :
1243             Serial.println("    -Notify ");
1244             break;
1245
1246         default :
1247             Serial.println("    -Not defined in Central device");
  
```

```

1248         break;
1249     }
1250 }

```

Here is the caller graph for this function:



4.1.1.28 printThSenseValueByHandle()

```

void printThSenseValueByHandle (
    uint16_t value_handle,
    uint8_t * value,
    uint16_t length )

```

Function to print the value of the READ Attribute value of Thunderboard Sense 2 device, identified by value_handle.

Parameters

in	<i>value_handle</i>	The handle of the Attribute value
in	<i>*value</i>	The Attribute value
in	<i>length</i>	The Attribute value length

Return values

<i>None</i>	
-------------	--

```

949     {
950
951     switch (value_handle) {
952
953     case 0x1D:
954         UVindex = value[0];
955         Serial.print("    -");
956         Serial.print(characteristics_name_gruped_by_service [4][0]);
957         Serial.println(UVindex);
958         break;
959
960     case 0x1F:
961         Pressure = uint32_t((value[3] << 24) + (value[2] << 16) + (value[1] << 8) + value[0]) / 1000;
962         Serial.print("    -");
963         Serial.print(characteristics_name_gruped_by_service [4][1]);
964         Serial.print(Pressure);

```

```

965     Serial.println(" hPa");
966     break;
967
968 case 0x21:
969     Temperature = int16_t((value[1] << 8) + value[0]) /100;
970     Serial.print("      -");
971     Serial.print(characteristics_name_gruped_by_service [4][2]);
972     Serial.print(Temperature);
973     Serial.println(" °C");
974     break;
975
976 case 0x23:
977     Humidity = uint16_t((value[1] << 8) + value[0]) /100;
978     Serial.print("      -");
979     Serial.print(characteristics_name_gruped_by_service [4][3]);
980     Serial.print(Humidity);
981     Serial.println(" %");
982     break;
983
984 case 0x25:
985     ALight = uint32_t((value[3] <<24) + (value[2] << 16) + (value[1] << 8) + value[0]) /1000;
986     Serial.print("      -");
987     Serial.print(characteristics_name_gruped_by_service [4][4]);
988     Serial.print(ALight);
989     Serial.println(" Lux");
990     break;
991
992 case 0x27:
993     Sound = int16_t((value[1] << 8) + value[0]) /100;
994     Serial.print("      -");
995     Serial.print(characteristics_name_gruped_by_service [4][5]);
996     Serial.print(Sound);
997     Serial.println(" dB");
998     break;
999
1000 case 0x30:
1001     ECO2 = uint16_t((value[1] << 8) + value[0]);
1002     Serial.print("      -");
1003     Serial.print(characteristics_name_gruped_by_service [6][0]);
1004     Serial.print(ECO2);
1005     Serial.println(" ppm");
1006     break;
1007
1008 case 0x32:
1009     TVOC = uint16_t((value[1] << 8) + value[0]);
1010     Serial.print("      -");
1011     Serial.print(characteristics_name_gruped_by_service [6][1]);
1012     Serial.print(TVOC);
1013     Serial.println(" ppb");
1014     break;
1015
1016 case 0x19:
1017     Battery_Level = value[0];
1018     Serial.print("      -");
1019     Serial.print(characteristics_name_gruped_by_service [3][0]);
1020     Serial.print(Battery_Level);
1021     Serial.println(" %");
1022     break;
1023
1024 case 0x58:
1025     Hall_State = value[0];
1026     Serial.print("      -");
1027     Serial.print(characteristics_name_gruped_by_service [10][0]);
1028     Serial.print(Hall_State);
1029     break;
1030
1031 case 0x5B :
1032     Field_Strength = int32_t((value[3] <<24) + (value[2] << 16) + (value[1] << 8) + value[0]
1033 );
1034     Serial.print("      -");
1035     Serial.print(characteristics_name_gruped_by_service [10][1]);
1036     Serial.print(Field_Strength);
1037     Serial.println(" uT");
1038     break;
1039
1040 case 0x2D :
1041     Power_Source = value[0];
1042     Serial.print("      -");
1043     Serial.print(characteristics_name_gruped_by_service [5][0]);
1044     Serial.print(Power_Source);
1045     if(Power_Source==0x01){
1046         Serial.println("  USB power");
1047     }else{
1048         Serial.println("  CR2032 power");
1049     }
1050     break;

```

```

1051     case 0x03 :
1052         memcpy(Device_Name, value, length);
1053         Serial.print("      -");
1054         Serial.print(characteristics_name_gruped_by_service [0][0]);
1055         Serial.println((const char *)Device_Name);
1056         break;
1057
1058     case 0x05 :
1059         Appearance = value[0];
1060         Serial.print("      -");
1061         Serial.print(characteristics_name_gruped_by_service [0][1]);
1062         Serial.println(Appearance, HEX);
1063         break;
1064
1065     case 0x0C :
1066         memcpy(Manufacturer_Name, value, length);
1067         Serial.print("      -");
1068         Serial.print(characteristics_name_gruped_by_service [2][0]);
1069         Serial.println((const char *)Manufacturer_Name);
1070         break;
1071
1072     case 0x0E :
1073         memcpy(Model_Number, value, length);
1074         Serial.print("      -");
1075         Serial.print(characteristics_name_gruped_by_service [2][1]);
1076         Serial.println((const char *)Model_Number);
1077         break;
1078
1079     case 0x10 :
1080         memcpy(Serial_Number, value, length);
1081         Serial.print("      -");
1082         Serial.print(characteristics_name_gruped_by_service [2][2]);
1083         Serial.println((const char *)Serial_Number);
1084         break;
1085
1086     case 0x12 :
1087         memcpy(Hardware_Revision, value, length);
1088         Serial.print("      -");
1089         Serial.print(characteristics_name_gruped_by_service [2][3]);
1090         Serial.println((const char *)Hardware_Revision);
1091         break;
1092
1093     case 0x14 :
1094         memcpy(Firmware_Revision, value, length);
1095         Serial.print("      -");
1096         Serial.print(characteristics_name_gruped_by_service [2][4]);
1097         Serial.println((const char *)Firmware_Revision);
1098         break;
1099
1100     case 0x16 :
1101         memcpy(System_ID, value, length);
1102         Serial.print("      -");
1103         Serial.print(characteristics_name_gruped_by_service [2][5]);
1104         Serial.println((const char *)System_ID);
1105         break;
1106
1107     case 0x44 :
1108         Digital_1 = value[0];
1109         Serial.print("      -");
1110         Serial.print(characteristics_name_gruped_by_service [2][5]);
1111         Serial.println(Digital_1);
1112         break;
1113
1114     case 0x49 :
1115         Digital_2 = value[0];
1116         Serial.print("      -");
1117         Serial.print(characteristics_name_gruped_by_service [8][0]);
1118         Serial.println(Digital_2);
1119         break;
1120
1121     case 0x38 :
1122         Buttons = value[0];
1123         Serial.print("      -");
1124         Serial.print(characteristics_name_gruped_by_service [8][1]);
1125         Serial.println(Buttons);
1126         break;
1127
1128     case 0x3D :
1129         RGB_Leds = uint32_t((value[0] << 24) + (value[1] << 16) + (value[2] << 8) + value[3]);
1130         Serial.print("      -");
1131         Serial.print(characteristics_name_gruped_by_service [7][2]);
1132         Serial.println(RGB_Leds, HEX);
1133         break;
1134
1135     default:
1136         Serial.print("____ The Characteristic handle value is not defined in the Central device");
1137         break;

```

```

1138     }
1139     Serial.println(" ");
1140     Serial.println("-----");
1141 }

```

Here is the caller graph for this function:



4.1.1.29 readTbSenseData()

```

static void readTbSenseData (
    btstack_timer_source_t * ts ) [static]

```

Function to read the Thunderboard Sense 2 sensors data, every 2 s using the btstack_timer.

Parameters

in	<i>btstack_timer_source_t</i>	*ts Pointer to the BLE stack timer source
----	-------------------------------	---

Return values

None

```

1638                                     {
1639
1640     if ((connected_id != 0xFFFF) && (conf_completed) ) {
1641         Serial.println(" ");
1642         if ((n_serv_index == n_env_sensing_service) && (
1643             n_serv_sel[n_env_sensing_service] == 1)) {
1644             if(checkAttributePropertyPermission(
1645                 n_env_sensing_service, n_chars_index, 1)){
1646                 Serial.print("> Environmental sensing characteristic");
1647                 Serial.print(n_chars_index);
1648                 Serial.println("read: ");
1649                 Serial.print(" - Connection handle: ");
1650                 Serial.println(device.connected_handle, HEX);
1651                 Serial.print(" - Characteristic value attribute handle: ");
1652                 Serial.println(device.service[n_env_sensing_service].chars[
1653                     n_chars_index].chars.value_handle, HEX);
1654                 ble.readValue(device.connected_handle,&device.
1655                     service[n_env_sensing_service].chars[n_chars_index].chars);
1656             }
1657             if (n_chars_index < (n_chars[n_env_sensing_service]-1)){
1658                 n_chars_index++;
1659             }else{
1660                 n_chars_index = 0;
1661                 Serial.println("-----");
1662                 Serial.println("- ENVIRONMENTAL SENSING -");
1663                 Serial.println("-----");
1664                 Serial.print(" - UV Index = ");
1665                 Serial.println(UVindex);

```

```

1662     Serial.print("    - Pressure = ");
1663     Serial.print(Pressure);
1664     Serial.println(" hPa");
1665     Serial.print("    - Temperature = ");
1666     Serial.print(Temperature);
1667     Serial.println(" °C");
1668     Serial.print("    - Humidity = ");
1669     Serial.print(Humidity);
1670     Serial.println(" %");
1671     Serial.print("    - Ambient Light = ");
1672     Serial.print(ALight);
1673     Serial.println(" Lux");
1674     Serial.print("    - Sound Level = ");
1675     Serial.print(Sound);
1676     Serial.println(" dB");
1677     Serial.println("-----");
1678     n_serv_index = 6;
1679 }
1680 }else if ((n_serv_index == n_battery_service) && (
n_serv_sel[n_battery_service] == 1)) {
1681     if(checkAttributePropertyPermission(
n_battery_service, n_chars_index, 1)){
1682         Serial.print("> Battery Level characteristic");
1683         Serial.print(n_chars_index);
1684         Serial.println("_read: ");
1685         Serial.print("    - Connection handle: ");
1686         Serial.println(device.connected_handle, HEX);
1687         Serial.print("    - Characteristic value attribute handle: ");
1688         Serial.println(device.service[n_battery_service].chars[n_chars_index]
.chars.value_handle, HEX);
1689         ble.readValue(device.connected_handle,&device.
service[n_battery_service].chars[n_chars_index].chars);
1690     }
1691     if (n_chars_index < (n_chars[n_battery_service]-1)){
1692         n_chars_index++;
1693     }else{
1694         n_chars_index = 0;
1695         Serial.println("-----");
1696         Serial.println("- Battery Level Service                -");
1697         Serial.println("-----");
1698         Serial.print("    - Battery Level = ");
1699         Serial.print(Battery_Level);
1700         Serial.println(" %");
1701         Serial.println("-----");
1702         n_serv_index = 4;
1703     }
1704 }else if ((n_serv_index == n_hall_effect_service) && (
n_serv_sel[n_hall_effect_service] == 1)) {
1705     if(checkAttributePropertyPermission(
n_hall_effect_service, n_chars_index, 1)){
1706         Serial.print("> Hall Effect characteristic");
1707         Serial.print(n_chars_index);
1708         Serial.println("_read: ");
1709         Serial.print("    - Connection handle: ");
1710         Serial.println(device.connected_handle, HEX);
1711         Serial.print("    - Characteristic value attribute handle: ");
1712         Serial.println(device.service[n_hall_effect_service].chars[
n_chars_index].chars.value_handle, HEX);
1713         ble.readValue(device.connected_handle,&device.
service[n_hall_effect_service].chars[n_chars_index].chars);
1714     }
1715     if (n_chars_index < (n_chars[n_hall_effect_service]-1)){
1716         n_chars_index++;
1717     }else {
1718         n_chars_index = 0;
1719         Serial.println("-----");
1720         Serial.println("- Hall Effect Service                -");
1721         Serial.println("-----");
1722         Serial.print("    - Hall State = ");
1723         Serial.print(Hall_State);
1724         Serial.println(" ");
1725         Serial.print("    - Field Strength = ");
1726         Serial.print(Field_Strength);
1727         Serial.println(" uT");
1728         Serial.println("-----");
1729         n_serv_index = 3;
1730     }
1731 }else if ((n_serv_index == n_iaq_service) && (
n_serv_sel[n_iaq_service] == 1)) {
1732     if(checkAttributePropertyPermission(
n_iaq_service, n_chars_index, 1)){
1733         Serial.print("> IAQ characteristic");
1734         Serial.print(n_chars_index);
1735         Serial.println("_read: ");
1736         Serial.print("    - Connection handle: ");
1737         Serial.println(device.connected_handle, HEX);
1738         Serial.print("    - Characteristic value attribute handle: ");

```

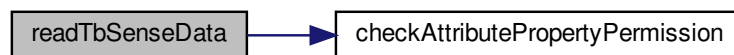


```

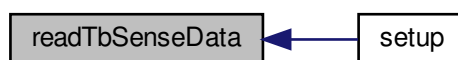
1739     Serial.println(device.service[n_iaq_service].chars[n_chars_index].chars.
value_handle, HEX);
1740     ble.readValue(device.connected_handle,&device.
service[n_iaq_service].chars[n_chars_index].chars);
1741 }
1742 if (n_chars_index < (n_chars[n_iaq_service]-1)){
1743     n_chars_index++;
1744 }else {
1745     n_chars_index = 0;
1746     Serial.println("-----");
1747     Serial.println("- IAQ SENSING -");
1748     Serial.println("-----");
1749     Serial.print(" - ECO2 - Carbon Dioxide = ");
1750     Serial.println(ECO2);
1751     Serial.print(" - TVOC - VOCS = ");
1752     Serial.println(TVOC);
1753     Serial.println("-----");
1754     n_serv_index = 10;
1755 }
1756 }
1757 }
1758 // Restart timer.
1759 ble.setTimer(ts, 2000);
1760 ble.addTimer(ts);
1761 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.30 receiveFromSerial()

```

uint8_t receiveFromSerial (
    char * message )

```

Function to recibe the user option in the debug MENU.

Parameters

in	char*	The message to show the user in the MENU.
----	-------	---

Return values

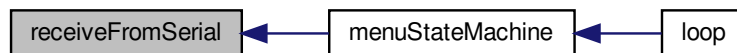
<code>uint8_t</code>	serialValue The recibed user option value.
----------------------	--

```

1267                                     { //receiveFromSerial
1268
1269     uint8_t getSerialValue = 0;
1270     uint8_t serialValue = 0;
1271
1272     Serial.print(message);
1273     while (!(Serial.available() > 0)) Particle.process(); // wait for serial port
1274     getSerialValue = Serial.read();
1275     serialValue = (getSerialValue <= '9')? (getSerialValue - '0'):(getSerialValue - 'A' + 10);
1276     return serialValue;
1277 }

```

Here is the caller graph for this function:



4.1.1.31 reportCallback()

```

void reportCallback (
    advertisementReport_t * report )

```

Callback for scanning device.

Parameters

in	<code>*report</code>	This function report the scanner response, and shearch to the Thunderboard Sense 2 device device to start the connection process.
----	----------------------	---

Return values

None	
------	--

```

165                                     {
166     uint8_t index;
167     Serial.println("");
168     Serial.println("* BLE scan callback: ");
169     Serial.print("    - Advertising event type: ");
170     Serial.println(report->advEventType, HEX);
171     Serial.print("    - Peer device address type: ");
172     Serial.println(report->peerAddrType, HEX);
173     Serial.print("    - Peer device address: ");

```

```

174   for (index = 0; index < 6; index++) {
175       Serial.print(report->peerAddr[index], HEX);
176       Serial.print(" ");
177   }
178   Serial.println(" ");
179   Serial.print("    - RSSI: ");
180   Serial.print(report->rssi, DEC);
181   Serial.println(" dBm ");
182
183   if (report->advEventType == BLE_GAP_ADV_TYPE_SCAN_RSP) {
184       Serial.print("    - Scan response data packet (");
185   }
186   else {
187       Serial.print("    - Advertising data packet (");
188   }
189   Serial.print(report->advDataLen, DEC);
190   Serial.print(" Bytes): ");
191
192   for (index = 0; index < report->advDataLen; index++) {
193       Serial.print(report->advData[index], HEX);
194       Serial.print(" ");
195   }
196   Serial.println(" ");
197
198   uint8_t len;
199   uint8_t adv_name[31];
200
201   if (0x00 == ble_advdata_decode(0x09, report->advDataLen, report->advData, &len,
202   adv_name)) { //Hacer una funcion conectarTh
203       Serial.print("    The length of Complete Local Name : ");
204       Serial.println(len, HEX);
205       Serial.print("    The Complete Local Name is          : ");
206       Serial.println((const char *)adv_name);
207
208       if (0x00 == memcmp(adv_name, "Thunder Sense #02735", len)) {
209           Serial.println("* Thunder Sense #02735 found");
210           ble.stopScanning();
211           device.addr_type = report->peerAddrType;
212           memcpy(device.addr, report->peerAddr, 6);
213           ble.connect(report->peerAddr, {BD_ADDR_TYPE_LE_PUBLIC});
214       }
215   }
216 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.1.32 setup()

```
void setup ( )
```

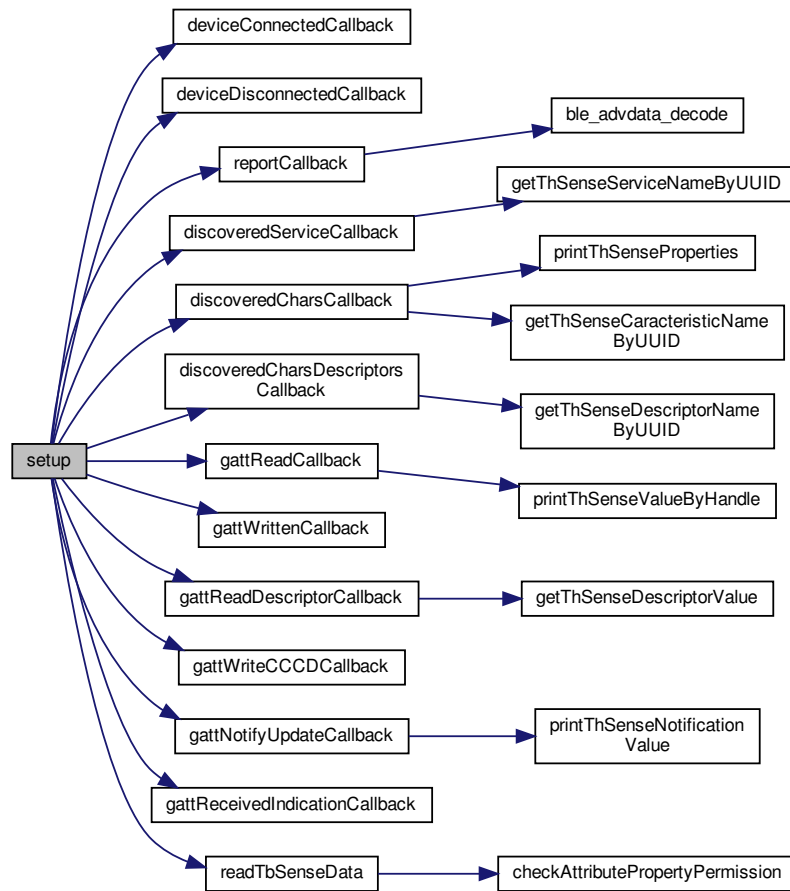
Setup.

```

1770         {
1771
1772     Serial.begin(115200);
1773     delay(5000);
1774     // Open debugger, must befor init()
1775     ble.debugLogger(true);
1776     ble.debugError(true);
1777     //ble.enablePacketLogger();
1778
1779     // Initialize ble_stack
1780     ble.init();
1781
1782     // Register callback functions
1783     ble.onConnectedCallback(deviceConnectedCallback);
1784     ble.onDisconnectedCallback(deviceDisconnectedCallback);
1785     ble.onScanReportCallback(reportCallback);
1786     ble.onServiceDiscoveredCallback(discoveredServiceCallback);
1787     ble.onCharacteristicDiscoveredCallback(discoveredCharsCallback);
1788     ble.onDescriptorDiscoveredCallback(discoveredCharsDescriptorsCallback);
1789     ble.onGattCharacteristicReadCallback(gattReadCallback);
1790     ble.onGattCharacteristicWrittenCallback(gattWrittenCallback);
1791     ble.onGattDescriptorReadCallback(gattReadDescriptorCallback);
1792     ble.onGattWriteClientCharacteristicConfigCallback(gattWriteCCCDCallback);
1793     ble.onGattNotifyUpdateCallback(gattNotifyUpdateCallback);
1794     ble.onGattIndicateUpdateCallback(gattReceivedIndicationCallback);
1795
1796     // Set scan parameters
1797     ble.setScanParams(BLE_SCAN_TYPE, BLE_SCAN_INTERVAL,
1798 BLE_SCAN_WINDOW);
1799
1800     Serial.println("_____RedBear Duo_____");
1801     ble.startScanning();
1802     Serial.println("");
1803     Serial.println("_____ BLE Central start scanning");
1804     delay(1000);
1805     Serial.println("");
1806
1807     #if TIMER_DEBUG >= 1
1808     // set one-shot timer __setup
1809     read_tbsense_timer.process = &readTbSenseData;
1810     ble.setTimer(&read_tbsense_timer, 10000);
1811     ble.addTimer(&read_tbsense_timer);
1812     #endif
1813 }

```

Here is the call graph for this function:



4.1.2 Variable Documentation

4.1.2.1 Acceleration_axis_X

```
int16_t Acceleration_axis_X = 0
```

4.1.2.2 Acceleration_axis_Y

```
int16_t Acceleration_axis_Y = 0
```

4.1.2.3 Acceleration_axis_Z

```
int16_t Acceleration_axis_Z = 0
```

4.1.2.4 ALight

```
uint32_t ALight = 0
```

4.1.2.5 Appearance

```
uint8_t Appearance = 0
```

4.1.2.6 Battery_Level

```
uint8_t Battery_Level = 0
```

4.1.2.7 Buttons

```
uint8_t Buttons = 0
```

4.1.2.8 characteristics_name_gruped_by_service

```
char* characteristics_name_gruped_by_service[NSERV_MAX][NCHAR_MAX]
```

Initial value:

```
= {
    {(char*)" Device_Name ", (char*)" Appearance "},
    {(char*)" Service_Changed "},
    {(char*)" Manufacturer_Name ", (char*)" Serial_Number ", (char*)" Hardware_Revision ", (char*)"
        Firmware_Revision ", (char*)" Model_Number ", (char*)" System_ID "},
    {(char*)" Battery_Level "},
    {(char*)" UV_Index ", (char*)" Pressure ", (char*)" Temperature ", (char*)" Humidity ", (char*)"
        Ambient_Light ", (char*)" Sound_Level ", (char*)" Control_Point "},
    {(char*)" Power_Source "},
    {(char*)" ECO2 ", (char*)" TVOC ", (char*)" Control_Point "},
    {(char*)" Buttons ", (char*)" Leds ", (char*)" RGB_Leds ", (char*)" Control_Point "},
    {(char*)" Digital_1 ", (char*)" Digital_2"},
    {(char*)" Acceleration ", (char*)" Orientation ", (char*)" Control_Point "},
    {(char*)" State ", (char*)" Field_Strength ", (char*)" Control_Point "}
}
```

4.1.2.9 chars_index

```
uint8_t chars_index = 0
```

4.1.2.10 conf_completed

```
bool conf_completed = false
```

4.1.2.11 connected_id

```
uint16_t connected_id = 0xFFFF [static]
```

4.1.2.12 desc_index

```
uint8_t desc_index = 0
```

4.1.2.13 DescrvalidOption

```
uint8_t DescrvalidOption[3]
```

4.1.2.14 device

```
Device_t device
```

4.1.2.15 Device_Name

```
uint8_t Device_Name[19]
```

4.1.2.16 Digital_1

```
uint8_t Digital_1 = 0
```

4.1.2.17 Digital_2

```
uint8_t Digital_2 = 0
```

4.1.2.18 ECO2

```
uint16_t ECO2 = 0
```

4.1.2.19 Field_Strength

```
int32_t Field_Strength = 0
```

4.1.2.20 Firmware_Revision

```
uint8_t Firmware_Revision[4]
```

4.1.2.21 Hall_Control_Point

```
uint16_t Hall_Control_Point = 0
```

4.1.2.22 Hall_State

```
uint8_t Hall_State = 0
```

4.1.2.23 Hardware_Revision

```
uint8_t Hardware_Revision[2]
```

4.1.2.24 Humidity

```
uint16_t Humidity = 0
```


4.1.2.25 indication_lastmills

```
unsigned long indication_lastmills = 0
```

4.1.2.26 Manufacturer_Name

```
uint8_t Manufacturer_Name[19]
```

4.1.2.27 menuOption

```
stateEnum_t menuOption
```

4.1.2.28 Model_Number

```
uint8_t Model_Number[7]
```

4.1.2.29 n_accleration_orientation_service

```
uint8_t n_accleration_orientation_service = 0
```

4.1.2.30 n_automation_io_service

```
uint8_t n_automation_io_service = 0
```

4.1.2.31 n_battery_service

```
uint8_t n_battery_service = 0
```

4.1.2.32 n_chars

```
uint8_t n_chars[NSERV_MAX]
```

4.1.2.33 n_chars_index

```
uint8_t n_chars_index = 0
```

4.1.2.34 n_device_information_service

```
uint8_t n_device_information_service = 0
```

4.1.2.35 n_env_sensing_service

```
uint8_t n_env_sensing_service = 0
```

4.1.2.36 n_generic_access_service

```
uint8_t n_generic_access_service = 0
```

4.1.2.37 n_generic_attribute_service

```
uint8_t n_generic_attribute_service = 0
```

4.1.2.38 n_hall_effect_service

```
uint8_t n_hall_effect_service = 0
```

4.1.2.39 n_iaq_service

```
uint8_t n_iaq_service = 0
```

4.1.2.40 n_power_management_service

```
uint8_t n_power_management_service = 0
```

4.1.2.41 n_serv

```
uint8_t n_serv = 0
```

4.1.2.42 n_serv_index

```
uint8_t n_serv_index = 3
```

4.1.2.43 n_serv_sel

```
uint8_t n_serv_sel[N SERV_MAX] = {0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1}
```

4.1.2.44 n_user_interface_service

```
uint8_t n_user_interface_service = 0
```

4.1.2.45 notification_lastmills

```
unsigned long notification_lastmills = 0
```

4.1.2.46 Orientation_axis_X

```
int16_t Orientation_axis_X = 0
```

4.1.2.47 Orientation_axis_Y

```
int16_t Orientation_axis_Y = 0
```

4.1.2.48 Orientation_axis_Z

```
int16_t Orientation_axis_Z = 0
```

4.1.2.49 Power_Source

```
uint8_t Power_Source = 0
```

4.1.2.50 Preasure

```
uint32_t Preasure = 0
```

4.1.2.51 read_tbsense_timer

```
btstack_timer_source_t read_tbsense_timer [static]
```

4.1.2.52 RGB_Leds

```
uint32_t RGB_Leds = 0
```

4.1.2.53 Serial_Number

```
uint8_t Serial_Number[3]
```

4.1.2.54 serv_index

```
uint8_t serv_index = 0
```

4.1.2.55 services_name

```
char* services_name[NSERV_MAX] = {(char*)" GENERIC ACCESS", (char*)" GENERIC ATTRIBUTE", (char*)" DEVICE INFORMATION", (char*)" BATTERY", (char*)" ENVIRONMENTAL SENSING", (char*)" POWER SOURCE", (char*)" IAQ SENSING", (char*)" USER INTERFACE", (char*)" AUTOMATION IO", (char*)" ACCLE↵RATION ORIENTATION", (char*)" HALL EFFECT" }
```

4.1.2.56 Sound

```
int16_t Sound = 0
```

4.1.2.57 System_ID

```
uint8_t System_ID[7]
```

4.1.2.58 Temperature

```
int16_t Temperature = 0
```

4.1.2.59 thereIsCharacteristic

```
uint8_t thereIsCharacteristic = 0
```

4.1.2.60 TVOC

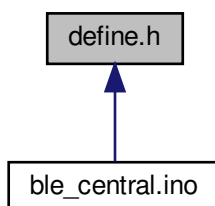
```
uint16_t TVOC = 0
```

4.1.2.61 UVIndex

```
uint8_t UVIndex = 0
```

4.2 define.h File Reference

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



Classes

- struct [Device_t](#)

Struct to save a BLE device and its related data.

Macros

- #define [MENU_DEBUG](#) 1
- #define [TIMER_DEBUG](#) 1
- #define [BLE_SCAN_TYPE](#) 0x00
- #define [BLE_SCAN_INTERVAL](#) 0x0060
- #define [BLE_SCAN_WINDOW](#) 0x0030
- #define [NSERV_MAX](#) 11
- #define [NCHAR_MAX](#) 7
- #define [NDESC_MAX](#) 3
- #define [bitRead](#)(value, bit) (((value) >> (bit)) & 0x01)

Typedefs

- typedef enum [states](#) [stateEnum_t](#)

Enumerations

- enum [states](#) {
[BLE_CENTRAL_READ_CHARACTERISTIC_VALUE](#) = 0, [BLE_CENTRAL_READ_DESCRIPTOR_VALUE](#),
[BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS](#), [BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS](#),
[BLE_CENTRAL_WRITE](#) }

Enum for the diferents states of the debug MENU options.

Variables

- static uint8_t [generic_access_service_uuid](#) [16] = {0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [generic_attribute_service_uuid](#) [16] = {0x00, 0x00, 0x18, 0x01, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [device_information_service_uuid](#) [16] = {0x00, 0x00, 0x18, 0x0A, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [battery_service_uuid](#) [16] = {0x00, 0x00, 0x18, 0x0F, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [environmental_sensing_service_uuid](#) [16] = {0x00, 0x00, 0x18, 0x1A, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [power_management_service_uuid](#) [16] = {0xEC, 0x61, 0xA4, 0x54, 0xED, 0x00, 0xA5, 0xE8, 0xB8, 0xF9, 0xDE, 0x9E, 0xC0, 0x26, 0xEC, 0x51}
- static uint8_t [iaq_service_uuid](#) [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x00, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t [user_interface_service_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x00, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [automation_io_service_uuid](#) [16] = {0x00, 0x00, 0x18, 0x15, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}

- static uint8_t [acceleration_orientation_service_uuid](#) [16] = {0xA4, 0xE6, 0x49, 0xF4, 0x4B, 0xE5, 0x11, 0xE5, 0x88, 0x5D, 0xFE, 0xFF, 0x81, 0x9C, 0xDC, 0x9F}
- static uint8_t [hall_effect_service_uuid](#) [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x00, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}
- static uint8_t [Service0_Characteristic0_Device_Name_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x00, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service0_Characteristic1_Appearance_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x01, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service1_Characteristic0_Service_Changed_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x05, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service2_Characteristic0_Manufacturer_Name_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x29, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service2_Characteristic1_Model_Number_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x24, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service2_Characteristic2_Serial_Number_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x25, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service2_Characteristic3_Hardware_Revision_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x27, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service2_Characteristic4_Firmware_Revision_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x26, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service2_Characteristic5_System_ID_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x23, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service3_Characteristic0_Battery_Level_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x19, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic0_UV_Index_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x76, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic1_Pressure_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x6D, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic2_Temperature_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x6E, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic3_Humidity_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x6F, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic4_Ambient_Light_uuid](#) [16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0xD9, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E}
- static uint8_t [Service4_Characteristic5_Sound_Level_uuid](#) [16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0x02, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E}
- static uint8_t [Service4_Characteristic6_Control_Point_uuid](#) [16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0x03, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E}
- static uint8_t [Service5_Characteristic0_Power_Source_uuid](#) [16] = {0xEC, 0x61, 0xA4, 0x54, 0xED, 0x01, 0xA5, 0xE8, 0xB8, 0xF9, 0xDE, 0x9E, 0xC0, 0x26, 0xEC, 0x51}
- static uint8_t [Service6_Characteristic0_ECO2_uuid](#) [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x01, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t [Service6_Characteristic1_TVOC_uuid](#) [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x02, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t [Service6_Characteristic2_Control_Point_uuid](#) [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x03, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t [Service7_Characteristic0_Buttons_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x01, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [Service7_Characteristic1_Leds_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x02, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [Service7_Characteristic2_RGB_Leds_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x03, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [Service7_Characteristic3_Control_Point_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x04, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [Service8_Characteristic0_Digital_1_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x56, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}

- static uint8_t [Service8_Characteristic1_Digital_2_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x56, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service9_Characteristic0_Acceleration_uuid](#) [16] = {0xC4, 0xC1, 0xF6, 0xE2, 0x4B, 0xE5, 0x11, 0xE5, 0x88, 0x5D, 0xFE, 0xFF, 0x81, 0x9C, 0xDC, 0x9F}
- static uint8_t [Service9_Characteristic1_Orientation_uuid](#) [16] = {0xB7, 0xC4, 0xB6, 0x94, 0xBE, 0xE3, 0x45, 0xDD, 0xBA, 0x9F, 0xF3, 0xB5, 0xE9, 0x94, 0xF4, 0x9A}
- static uint8_t [Service9_Characteristic2_Control_Point_uuid](#) [16] = {0x71, 0xE3, 0x0B, 0x8C, 0x41, 0x31, 0x47, 0x03, 0xB0, 0xA0, 0xB0, 0xBB, 0xBA, 0x75, 0x85, 0x6B}
- static uint8_t [ServiceA_Characteristic0_State_uuid](#) [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x01, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}
- static uint8_t [ServiceA_Characteristic1_Field_Strength_uuid](#) [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x02, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}
- static uint8_t [ServiceA_Characteristic2_Control_Point_uuid](#) [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x03, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}
- static uint8_t [Client_Characteristic_Configuration_uuid](#) [16] = {0x00, 0x00, 0x29, 0x02, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Characteristic_Presentation_Format_uuid](#) [16] = {0x00, 0x00, 0x29, 0x04, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [noOfDigitals_uuid](#) [16] = {0x00, 0x00, 0x29, 0x09, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}

4.2.1 Macro Definition Documentation

4.2.1.1 bitRead

```
#define bitRead(  
    value,  
    bit ) (((value) >> (bit)) & 0x01)
```

4.2.1.2 BLE_SCAN_INTERVAL

```
#define BLE_SCAN_INTERVAL 0x0060
```

4.2.1.3 BLE_SCAN_TYPE

```
#define BLE_SCAN_TYPE 0x00
```

4.2.1.4 BLE_SCAN_WINDOW

```
#define BLE_SCAN_WINDOW 0x0030
```


4.2.1.5 MENU_DEBUG

```
#define MENU_DEBUG 1
```

4.2.1.6 NCHAR_MAX

```
#define NCHAR_MAX 7
```

4.2.1.7 NDESC_MAX

```
#define NDESC_MAX 3
```

4.2.1.8 NSERV_MAX

```
#define NSERV_MAX 11
```

4.2.1.9 TIMER_DEBUG

```
#define TIMER_DEBUG 1
```

4.2.2 Typedef Documentation

4.2.2.1 stateEnum_t

```
typedef enum states stateEnum_t
```

4.2.3 Enumeration Type Documentation

4.2.3.1 states

```
enum states
```

Enum for the diferents states of the debug MENU options.

Enumerator

BLE_CENTRAL_READ_CHARACTERISTIC_VALUE	State BLE_CENTRAL_READ_CHARACTERISTIC_VALUE
BLE_CENTRAL_READ_DESCRIPTOR_VALUE	State BLE_CENTRAL_READ_DESCRIPTOR_VALUE
BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS	State BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS
BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS	State BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS
BLE_CENTRAL_WRITE	State BLE_CENTRAL_WRITE

```

81     {
82         BLE_CENTRAL_READ_CHARACTERISTIC_VALUE = 0,
83         BLE_CENTRAL_READ_DESCRIPTOR_VALUE,
84         BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS,
85         BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS,
86         BLE_CENTRAL_WRITE
87     }stateEnum_t;

```

4.2.4 Variable Documentation

4.2.4.1 accleration_orientation_service_uuid

```
uint8_t accleration_orientation_service_uuid[16] = {0xA4, 0xE6, 0x49, 0xF4, 0x4B, 0xE5, 0x11,
0xE5, 0x88, 0x5D, 0xFE, 0xFF, 0x81, 0x9C, 0xDC, 0x9F} [static]
```

4.2.4.2 automation_io_service_uuid

```
uint8_t automation_io_service_uuid[16] = {0x00, 0x00, 0x18, 0x15, 0x00, 0x00, 0x10, 0x00,
0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.3 battery_service_uuid

```
uint8_t battery_service_uuid[16] = {0x00, 0x00, 0x18, 0x0F, 0x00, 0x00, 0x10, 0x00, 0x80,
0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.4 Characteristic_Presentation_Format_uuid

```
uint8_t Characteristic_Presentation_Format_uuid[16] = {0x00, 0x00, 0x29, 0x04, 0x00, 0x00,
0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.5 Client_Characteristic_Configuration_uuid

```
uint8_t Client_Characteristic_Configuration_uuid[16] = {0x00, 0x00, 0x29, 0x02, 0x00, 0x00,  
0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.6 device_information_service_uuid

```
uint8_t device_information_service_uuid[16] = {0x00, 0x00, 0x18, 0x0A, 0x00, 0x00, 0x10, 0x00,  
0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.7 environmental_sensing_service_uuid

```
uint8_t environmental_sensing_service_uuid[16] = {0x00, 0x00, 0x18, 0x1A, 0x00, 0x00, 0x10,  
0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.8 generic_access_service_uuid

```
uint8_t generic_access_service_uuid[16] = {0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x10, 0x00,  
0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.9 generic_attribute_service_uuid

```
uint8_t generic_attribute_service_uuid[16] = {0x00, 0x00, 0x18, 0x01, 0x00, 0x00, 0x10, 0x00,  
0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.10 hall_effect_service_uuid

```
uint8_t hall_effect_service_uuid[16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x00, 0x4E, 0xC5, 0x99,  
0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F} [static]
```

4.2.4.11 iaq_service_uuid

```
uint8_t iaq_service_uuid[16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x00, 0xEF, 0x33, 0x76, 0xE7,  
0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B} [static]
```

4.2.4.12 noOfDigitals_uuid

```
uint8_t noOfDigitals_uuid[16] = {0x00, 0x00, 0x29, 0x09, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00,
0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.13 power_management_service_uuid

```
uint8_t power_management_service_uuid[16] = {0xEC, 0x61, 0xA4, 0x54, 0xED, 0x00, 0xA5, 0xE8,
0xB8, 0xF9, 0xDE, 0x9E, 0xC0, 0x26, 0xEC, 0x51} [static]
```

4.2.4.14 Service0_Characteristic0_Device_Name_uuid

```
uint8_t Service0_Characteristic0_Device_Name_uuid[16] = {0x00, 0x00, 0x2A, 0x00, 0x00, 0x00,
0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.15 Service0_Characteristic1_Appearance_uuid

```
uint8_t Service0_Characteristic1_Appearance_uuid[16] = {0x00, 0x00, 0x2A, 0x01, 0x00, 0x00,
0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.16 Service1_Characteristic0_Service_Changed_uuid

```
uint8_t Service1_Characteristic0_Service_Changed_uuid[16] = {0x00, 0x00, 0x2A, 0x05, 0x00,
0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.17 Service2_Characteristic0_Manufacturer_Name_uuid

```
uint8_t Service2_Characteristic0_Manufacturer_Name_uuid[16] = {0x00, 0x00, 0x2A, 0x29, 0x00,
0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.18 Service2_Characteristic1_Model_Number_uuid

```
uint8_t Service2_Characteristic1_Model_Number_uuid[16] = {0x00, 0x00, 0x2A, 0x24, 0x00, 0x00,
0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.19 Service2_Characrtteristic2_Serial_Number_uuid

```
uint8_t Service2_Characrtteristic2_Serial_Number_uuid[16] = {0x00, 0x00, 0x2A, 0x25, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.20 Service2_Characrtteristic3_Hardware_Revision_uuid

```
uint8_t Service2_Characrtteristic3_Hardware_Revision_uuid[16] = {0x00, 0x00, 0x2A, 0x27, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.21 Service2_Characrtteristic4_Firmware_Revision_uuid

```
uint8_t Service2_Characrtteristic4_Firmware_Revision_uuid[16] = {0x00, 0x00, 0x2A, 0x26, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.22 Service2_Characrtteristic5_System_ID_uuid

```
uint8_t Service2_Characrtteristic5_System_ID_uuid[16] = {0x00, 0x00, 0x2A, 0x23, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.23 Service3_Characrtteristic0_Battery_Level_uuid

```
uint8_t Service3_Characrtteristic0_Battery_Level_uuid[16] = {0x00, 0x00, 0x2A, 0x19, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.24 Service4_Characrtteristic0_UV_Index_uuid

```
uint8_t Service4_Characrtteristic0_UV_Index_uuid[16] = {0x00, 0x00, 0x2A, 0x76, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.25 Service4_Characrtteristic1_Pressure_uuid

```
uint8_t Service4_Characrtteristic1_Pressure_uuid[16] = {0x00, 0x00, 0x2A, 0x6D, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.26 Service4_Characrteristic2_Temperature_uuid

```
uint8_t Service4_Characrteristic2_Temperature_uuid[16] = {0x00, 0x00, 0x2A, 0x6E, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.27 Service4_Characrteristic3_Humidity_uuid

```
uint8_t Service4_Characrteristic3_Humidity_uuid[16] = {0x00, 0x00, 0x2A, 0x6F, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.28 Service4_Characrteristic4_Ambient_Light_uuid

```
uint8_t Service4_Characrteristic4_Ambient_Light_uuid[16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0xD9, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E} [static]
```

4.2.4.29 Service4_Characrteristic5_Sound_Level_uuid

```
uint8_t Service4_Characrteristic5_Sound_Level_uuid[16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0x02, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E} [static]
```

4.2.4.30 Service4_Characrteristic6_Control_Point_uuid

```
uint8_t Service4_Characrteristic6_Control_Point_uuid[16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0x03, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E} [static]
```

4.2.4.31 Service5_Characrteristic0_Power_Source_uuid

```
uint8_t Service5_Characrteristic0_Power_Source_uuid[16] = {0xEC, 0x61, 0xA4, 0x54, 0xED, 0x01, 0xA5, 0xE8, 0xB8, 0xF9, 0xDE, 0x9E, 0xC0, 0x26, 0xEC, 0x51} [static]
```

4.2.4.32 Service6_Characrteristic0_ECO2_uuid

```
uint8_t Service6_Characrteristic0_ECO2_uuid[16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x01, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B} [static]
```

4.2.4.33 Service6_Characrtteristic1_TVOC_uuid

```
uint8_t Service6_Characrtteristic1_TVOC_uuid[16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x02, 0xEF,
0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B} [static]
```

4.2.4.34 Service6_Characrtteristic2_Control_Point_uuid

```
uint8_t Service6_Characrtteristic2_Control_Point_uuid[16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4,
0x03, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B} [static]
```

4.2.4.35 Service7_Characrtteristic0_Buttons_uuid

```
uint8_t Service7_Characrtteristic0_Buttons_uuid[16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x01,
0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B} [static]
```

4.2.4.36 Service7_Characrtteristic1_Leds_uuid

```
uint8_t Service7_Characrtteristic1_Leds_uuid[16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x02, 0x59,
0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B} [static]
```

4.2.4.37 Service7_Characrtteristic2_RGB_Leds_uuid

```
uint8_t Service7_Characrtteristic2_RGB_Leds_uuid[16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x03,
0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B} [static]
```

4.2.4.38 Service7_Characrtteristic3_Control_Point_uuid

```
uint8_t Service7_Characrtteristic3_Control_Point_uuid[16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6,
0x04, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B} [static]
```

4.2.4.39 Service8_Characrtteristic0_Digital_1_uuid

```
uint8_t Service8_Characrtteristic0_Digital_1_uuid[16] = {0x00, 0x00, 0x2A, 0x56, 0x00, 0x00,
0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.40 Service8_Characrtteristic1_Digital_2_uuid

```
uint8_t Service8_Characrtteristic1_Digital_2_uuid[16] = {0x00, 0x00, 0x2A, 0x56, 0x00, 0x00,
0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB} [static]
```

4.2.4.41 Service9_Characrtteristic0_Acceleration_uuid

```
uint8_t Service9_Characrtteristic0_Acceleration_uuid[16] = {0xC4, 0xC1, 0xF6, 0xE2, 0x4B, 0xE5,
0x11, 0xE5, 0x88, 0x5D, 0xFE, 0xFF, 0x81, 0x9C, 0xDC, 0x9F} [static]
```

4.2.4.42 Service9_Characrtteristic1_Orientation_uuid

```
uint8_t Service9_Characrtteristic1_Orientation_uuid[16] = {0xB7, 0xC4, 0xB6, 0x94, 0xBE, 0xE3,
0x45, 0xDD, 0xBA, 0x9F, 0xF3, 0xB5, 0xE9, 0x94, 0xF4, 0x9A} [static]
```

4.2.4.43 Service9_Characrtteristic2_Control_Point_uuid

```
uint8_t Service9_Characrtteristic2_Control_Point_uuid[16] = {0x71, 0xE3, 0x0B, 0x8C, 0x41,
0x31, 0x47, 0x03, 0xB0, 0xA0, 0xB0, 0xBB, 0xBA, 0x75, 0x85, 0x6B} [static]
```

4.2.4.44 ServiceA_Characrtteristic0_State_uuid

```
uint8_t ServiceA_Characrtteristic0_State_uuid[16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x01, 0x4E,
0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F} [static]
```

4.2.4.45 ServiceA_Characrtteristic1_Field_Strength_uuid

```
uint8_t ServiceA_Characrtteristic1_Field_Strength_uuid[16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F,
0x02, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F} [static]
```

4.2.4.46 ServiceA_Characrtteristic2_Control_Point_uuid

```
uint8_t ServiceA_Characrtteristic2_Control_Point_uuid[16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F,
0x03, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F} [static]
```

4.2.4.47 user_interface_service_uuid

```
uint8_t user_interface_service_uuid[16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x00, 0x59, 0xF3,
0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B} [static]
```


Index

ALight
 ble_central.ino, [54](#)
Acceleration_axis_X
 ble_central.ino, [53](#)
Acceleration_axis_Y
 ble_central.ino, [53](#)
Acceleration_axis_Z
 ble_central.ino, [53](#)
acceleration_orientation_service_uuid
 define.h, [66](#)
addr
 Device_t, [5](#)
addr_type
 Device_t, [5](#)
Appearance
 ble_central.ino, [54](#)
automation_io_service_uuid
 define.h, [66](#)

BLE_SCAN_INTERVAL
 define.h, [64](#)
BLE_SCAN_TYPE
 define.h, [64](#)
BLE_SCAN_WINDOW
 define.h, [64](#)
Battery_Level
 ble_central.ino, [54](#)
battery_service_uuid
 define.h, [66](#)
bitRead
 define.h, [64](#)
ble_advdata_decode
 ble_central.ino, [10](#)
ble_central.ino, [7](#)
 ALight, [54](#)
 Acceleration_axis_X, [53](#)
 Acceleration_axis_Y, [53](#)
 Acceleration_axis_Z, [53](#)
 Appearance, [54](#)
 Battery_Level, [54](#)
 ble_advdata_decode, [10](#)
 Buttons, [54](#)
 characteristics_name_grouped_by_service, [54](#)
 chars_index, [54](#)
 checkAttributePropertyPermission, [11](#)
 conf_completed, [55](#)
 connected_id, [55](#)
 desc_index, [55](#)
 DescrvalidOption, [55](#)
 device, [55](#)
 Device_Name, [55](#)
 deviceConnectedCallback, [12](#)
 deviceDisconnectedCallback, [13](#)
 Digital_1, [55](#)
 Digital_2, [55](#)
 discoveredCharsCallback, [14](#)
 discoveredCharsDescriptorsCallback, [15](#)
 discoveredServiceCallback, [17](#)
 ECO2, [56](#)
 Field_Strength, [56](#)
 Firmware_Revision, [56](#)
 gattNotifyUpdateCallback, [19](#)
 gattReadCallback, [20](#)
 gattReadDescriptorCallback, [21](#)
 gattReceivedIndicationCallback, [22](#)
 gattWriteCCCDCallback, [23](#)
 gattWrittenCallback, [24](#)
 getThSenseCharacteristicNameByUUID, [25](#)
 getThSenseDescriptorNameByUUID, [27](#)
 getThSenseDescriptorValue, [28](#)
 getThSenseServiceNameByUUID, [30](#)
 Hall_Control_Point, [56](#)
 Hall_State, [56](#)
 Hardware_Revision, [56](#)
 Humidity, [56](#)
 indication_lastmills, [56](#)
 loop, [31](#)
 Manufacturer_Name, [57](#)
 menuOption, [57](#)
 menuStateMachine, [32](#)
 Model_Number, [57](#)
 n_acceleration_orientation_service, [57](#)
 n_automation_io_service, [57](#)
 n_battery_service, [57](#)
 n_chars, [57](#)
 n_chars_index, [57](#)
 n_device_information_service, [58](#)
 n_env_sensing_service, [58](#)
 n_generic_access_service, [58](#)
 n_generic_attribute_service, [58](#)
 n_hall_effect_service, [58](#)
 n_iaq_service, [58](#)
 n_power_management_service, [58](#)
 n_serv, [58](#)
 n_serv_index, [59](#)
 n_serv_sel, [59](#)
 n_user_interface_service, [59](#)
 notification_lastmills, [59](#)
 Orientation_axis_X, [59](#)

- Orientation_axis_Y, 59
- Orientation_axis_Z, 59
- Power_Source, 59
- Pressure, 60
- printBLEProfile, 35
- PrintCharacteristicsAccordingToProperty, 37
- PrintCharacteristicsAndDescriptorsAccordingToProperty, 38
- printCharacteristicsNamesFromService, 39
- printMenuOptions, 40
- printServiceName, 41
- printThSenseNotificationValue, 41
- printThSenseProperties, 43
- printThSenseValueByHandle, 44
- RGB_Leds, 60
- read_tbsense_timer, 60
- readTbSenseData, 47
- receiveFromSerial, 49
- reportCallback, 50
- Serial_Number, 60
- serv_index, 60
- services_name, 60
- setup, 51
- Sound, 60
- System_ID, 61
- TVOC, 61
- Temperature, 61
- therelsCharacteristic, 61
- UVindex, 61
- Buttons
 - ble_central.ino, 54
- characteristics_name_grouped_by_service
 - ble_central.ino, 54
- Characteristic_Presentation_Format_uuid
 - define.h, 66
- chars
 - Device_t, 6
- chars_index
 - ble_central.ino, 54
- checkAttributePropertyPermission
 - ble_central.ino, 11
- Client_Characteristic_Configuration_uuid
 - define.h, 66
- conf_completed
 - ble_central.ino, 55
- connected_handle
 - Device_t, 6
- connected_id
 - ble_central.ino, 55
- define.h, 61
 - acceleration_orientation_service_uuid, 66
 - automation_io_service_uuid, 66
 - BLE_SCAN_INTERVAL, 64
 - BLE_SCAN_TYPE, 64
 - BLE_SCAN_WINDOW, 64
 - battery_service_uuid, 66
 - bitRead, 64
 - Characteristic_Presentation_Format_uuid, 66
 - Client_Characteristic_Configuration_uuid, 66
 - device_information_service_uuid, 67
 - environmental_sensing_service_uuid, 67
 - generic_access_service_uuid, 67
 - generic_attribute_service_uuid, 67
 - hall_effect_service_uuid, 67
 - iaq_service_uuid, 67
 - MENU_DEBUG, 64
 - NCHAR_MAX, 65
 - NDESC_MAX, 65
 - NSERV_MAX, 65
 - noOfDigitals_uuid, 67
 - power_management_service_uuid, 68
 - Service0_Characteristic0_Device_Name_uuid, 68
 - Service0_Characteristic1_Appearance_uuid, 68
 - Service1_Characteristic0_Service_Changed_uuid, 68
 - Service2_Characteristic0_Manufacturer_Name_uuid, 68
 - Service2_Characteristic1_Model_Number_uuid, 68
 - Service2_Characteristic2_Serial_Number_uuid, 68
 - Service2_Characteristic3_Hardware_Revision_uuid, 69
 - Service2_Characteristic4_Firmware_Revision_uuid, 69
 - Service2_Characteristic5_System_ID_uuid, 69
 - Service3_Characteristic0_Battery_Level_uuid, 69
 - Service4_Characteristic0_UV_Index_uuid, 69
 - Service4_Characteristic1_Pressure_uuid, 69
 - Service4_Characteristic2_Temperature_uuid, 69
 - Service4_Characteristic3_Humidity_uuid, 70
 - Service4_Characteristic4_Ambient_Light_uuid, 70
 - Service4_Characteristic5_Sound_Level_uuid, 70
 - Service4_Characteristic6_Control_Point_uuid, 70
 - Service5_Characteristic0_Power_Source_uuid, 70
 - Service6_Characteristic0_ECO2_uuid, 70
 - Service6_Characteristic1_TVOC_uuid, 70
 - Service6_Characteristic2_Control_Point_uuid, 71
 - Service7_Characteristic0_Buttons_uuid, 71
 - Service7_Characteristic1_Leds_uuid, 71
 - Service7_Characteristic2_RGB_Leds_uuid, 71
 - Service7_Characteristic3_Control_Point_uuid, 71
 - Service8_Characteristic0_Digital_1_uuid, 71
 - Service8_Characteristic1_Digital_2_uuid, 71
 - Service9_Characteristic0_Acceleration_uuid, 72
 - Service9_Characteristic1_Orientation_uuid, 72
 - Service9_Characteristic2_Control_Point_uuid, 72
 - ServiceA_Characteristic0_State_uuid, 72
 - ServiceA_Characteristic1_Field_Strength_uuid, 72
 - ServiceA_Characteristic2_Control_Point_uuid, 72
 - stateEnum_t, 65
 - states, 65

- TIMER_DEBUG, 65
- user_interface_service_uuid, 72
- desc_index
 - ble_central.ino, 55
- descriptor
 - Device_t, 6
- DescrvalidOption
 - ble_central.ino, 55
- device
 - ble_central.ino, 55
- Device_Name
 - ble_central.ino, 55
- device_information_service_uuid
 - define.h, 67
- Device_t, 5
 - addr, 5
 - addr_type, 5
 - chars, 6
 - connected_handle, 6
 - descriptor, 6
 - service, 6
- deviceConnectedCallback
 - ble_central.ino, 12
- deviceDisconnectedCallback
 - ble_central.ino, 13
- Digital_1
 - ble_central.ino, 55
- Digital_2
 - ble_central.ino, 55
- discoveredCharsCallback
 - ble_central.ino, 14
- discoveredCharsDescriptorsCallback
 - ble_central.ino, 15
- discoveredServiceCallback
 - ble_central.ino, 17
- ECO2
 - ble_central.ino, 56
- environmental_sensing_service_uuid
 - define.h, 67
- Field_Strength
 - ble_central.ino, 56
- Firmware_Revision
 - ble_central.ino, 56
- gattNotifyUpdateCallback
 - ble_central.ino, 19
- gattReadCallback
 - ble_central.ino, 20
- gattReadDescriptorCallback
 - ble_central.ino, 21
- gattReceivedIndicationCallback
 - ble_central.ino, 22
- gattWriteCCCDCallback
 - ble_central.ino, 23
- gattWrittenCallback
 - ble_central.ino, 24
- generic_access_service_uuid
 - define.h, 67
- generic_attribute_service_uuid
 - define.h, 67
- getThSenseCharacteristicNameByUUID
 - ble_central.ino, 25
- getThSenseDescriptorNameByUUID
 - ble_central.ino, 27
- getThSenseDescriptorValue
 - ble_central.ino, 28
- getThSenseServiceNameByUUID
 - ble_central.ino, 30
- Hall_Control_Point
 - ble_central.ino, 56
- Hall_State
 - ble_central.ino, 56
- hall_effect_service_uuid
 - define.h, 67
- Hardware_Revision
 - ble_central.ino, 56
- Humidity
 - ble_central.ino, 56
- iaq_service_uuid
 - define.h, 67
- indication_lastmills
 - ble_central.ino, 56
- loop
 - ble_central.ino, 31
- MENU_DEBUG
 - define.h, 64
- Manufacturer_Name
 - ble_central.ino, 57
- menuOption
 - ble_central.ino, 57
- menuStateMachine
 - ble_central.ino, 32
- Model_Number
 - ble_central.ino, 57
- n_acceleration_orientation_service
 - ble_central.ino, 57
- n_automation_io_service
 - ble_central.ino, 57
- n_battery_service
 - ble_central.ino, 57
- n_chars
 - ble_central.ino, 57
- n_chars_index
 - ble_central.ino, 57
- n_device_information_service
 - ble_central.ino, 58
- n_env_sensing_service
 - ble_central.ino, 58
- n_generic_access_service
 - ble_central.ino, 58
- n_generic_attribute_service

- ble_central.ino, 58
- n_hall_effect_service
 - ble_central.ino, 58
- n_iq_service
 - ble_central.ino, 58
- n_power_management_service
 - ble_central.ino, 58
- n_serv
 - ble_central.ino, 58
- n_serv_index
 - ble_central.ino, 59
- n_serv_sel
 - ble_central.ino, 59
- n_user_interface_service
 - ble_central.ino, 59
- NCHAR_MAX
 - define.h, 65
- NDESC_MAX
 - define.h, 65
- NSERV_MAX
 - define.h, 65
- noOfDigitals_uuid
 - define.h, 67
- notification_lastmills
 - ble_central.ino, 59
- Orientation_axis_X
 - ble_central.ino, 59
- Orientation_axis_Y
 - ble_central.ino, 59
- Orientation_axis_Z
 - ble_central.ino, 59
- Power_Source
 - ble_central.ino, 59
- power_management_service_uuid
 - define.h, 68
- Preasure
 - ble_central.ino, 60
- printBLEProfile
 - ble_central.ino, 35
- PrintCharacteristicsAccordingToProperty
 - ble_central.ino, 37
- PrintCharacteristicsAndDescriptorsAccordingToProperty
 - ble_central.ino, 38
- printCharacteristicsNamesFromService
 - ble_central.ino, 39
- printMenuOptions
 - ble_central.ino, 40
- printServiceName
 - ble_central.ino, 41
- printThSenseNotificationValue
 - ble_central.ino, 41
- printThSenseProperties
 - ble_central.ino, 43
- printThSenseValueByHandle
 - ble_central.ino, 44
- RGB_Leds
 - ble_central.ino, 60
- read_tbsense_timer
 - ble_central.ino, 60
- readTbSenseData
 - ble_central.ino, 47
- receiveFromSerial
 - ble_central.ino, 49
- reportCallback
 - ble_central.ino, 50
- Serial_Number
 - ble_central.ino, 60
- serv_index
 - ble_central.ino, 60
- service
 - Device_t, 6
- Service0_Characrteristic0_Device_Name_uuid
 - define.h, 68
- Service0_Characrteristic1_Appearance_uuid
 - define.h, 68
- Service1_Characrteristic0_Service_Changed_uuid
 - define.h, 68
- Service2_Characrteristic0_Manufacturer_Name_uuid
 - define.h, 68
- Service2_Characrteristic1_Model_Number_uuid
 - define.h, 68
- Service2_Characrteristic2_Serial_Number_uuid
 - define.h, 68
- Service2_Characrteristic3_Hardware_Revision_uuid
 - define.h, 69
- Service2_Characrteristic4_Firmware_Revision_uuid
 - define.h, 69
- Service2_Characrteristic5_System_ID_uuid
 - define.h, 69
- Service3_Characrteristic0_Battery_Level_uuid
 - define.h, 69
- Service4_Characrteristic0_UV_Index_uuid
 - define.h, 69
- Service4_Characrteristic1_Pressure_uuid
 - define.h, 69
- Service4_Characrteristic2_Temperature_uuid
 - define.h, 69
- Service4_Characrteristic3_Humidity_uuid
 - define.h, 70
- Service4_Characrteristic4_Ambient_Light_uuid
 - define.h, 70
- Service4_Characrteristic5_Sound_Level_uuid
 - define.h, 70
- Service4_Characrteristic6_Control_Point_uuid
 - define.h, 70
- Service5_Characrteristic0_Power_Source_uuid
 - define.h, 70
- Service6_Characrteristic0_ECO2_uuid
 - define.h, 70
- Service6_Characrteristic1_TVOC_uuid
 - define.h, 70
- Service6_Characrteristic2_Control_Point_uuid
 - define.h, 71

Service7_Characrteristic0_Buttons_uuid
define.h, [71](#)

Service7_Characrteristic1_Leds_uuid
define.h, [71](#)

Service7_Characrteristic2_RGB_Leds_uuid
define.h, [71](#)

Service7_Characrteristic3_Control_Point_uuid
define.h, [71](#)

Service8_Characrteristic0_Digital_1_uuid
define.h, [71](#)

Service8_Characrteristic1_Digital_2_uuid
define.h, [71](#)

Service9_Characrteristic0_Acceleration_uuid
define.h, [72](#)

Service9_Characrteristic1_Orientation_uuid
define.h, [72](#)

Service9_Characrteristic2_Control_Point_uuid
define.h, [72](#)

ServiceA_Characrteristic0_State_uuid
define.h, [72](#)

ServiceA_Characrteristic1_Field_Strength_uuid
define.h, [72](#)

ServiceA_Characrteristic2_Control_Point_uuid
define.h, [72](#)

services_name
ble_central.ino, [60](#)

setup
ble_central.ino, [51](#)

Sound
ble_central.ino, [60](#)

stateEnum_t
define.h, [65](#)

states
define.h, [65](#)

System_ID
ble_central.ino, [61](#)

TIMER_DEBUG
define.h, [65](#)

TVOC
ble_central.ino, [61](#)

Temperature
ble_central.ino, [61](#)

thereIsCharacteristic
ble_central.ino, [61](#)

UVindex
ble_central.ino, [61](#)

user_interface_service_uuid
define.h, [72](#)