Integración con el dispositivo RedBear Duo

Generated by Doxygen 1.8.13

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

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:

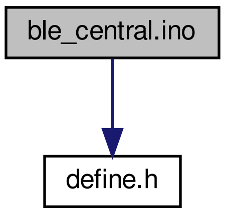
- define.h

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_↩ 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 writting.
- 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 writting 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 bitTo↩ Check)

  Function to verify the property permission of the attribute of a characteristic.
- 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.
- char ∗ getThSenseCaracteristicNameByUUID (uint8_t Caracteristic_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 ATRIBUTE", (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

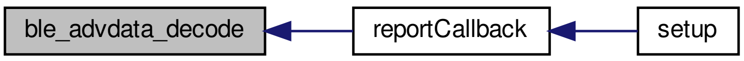
in	*type*	The type of field data.
in	*advdata_len*	Length of advertising data.
in	∗*p_advdata*	The pointer of advertising data.
out	∗*len*	The length of found data.
out	∗*p_field_data*	The pointer of buffer to store field data.

Return values

0	Find the data 1 Not find.

```
                 {
  uint8_t index = 0;
  uint8_t field_length, field_type;

  while (index < advdata_len) {
    field_length = p_advdata[index];
    field_type = p_advdata[index + 1];
    Serial.print("      - AVD/SR data decoding -> ad_type: ");
    Serial.print(field_type, HEX);
    Serial.print(", length: ");
    Serial.println(field_length, HEX);
    if (field_type == type) {
      memcpy(p_field_data, &p_advdata[index + 2], (field_length - 1));
      *len = field_length - 1;
      return 0;
    }
    index += field_length + 1;
  }
  return 1;
}
```

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

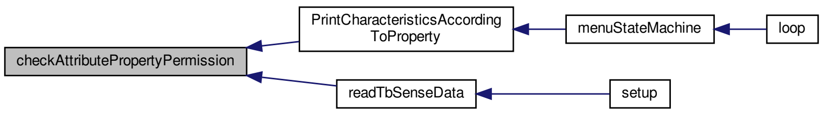
in	*uint8←_t*	numService The Service number
in	*uint8←_t*	numCharacteristic The Caracteristic number
in	*uint8←_t*	bitToCheck The bit to check: 8b –> bit 1 read, bit 3 write, bit 4 notifi, bit 5 indicate , etc.

Return values

1	OK 0 Not OK.

```
{

  if(bitRead(device.service[numService].chars[numCharacteristic].chars.properties,
  bitToCheck)){
    return 1;
  }else{
    return 0;
  }
}
```

Here is the caller graph for this function:


checkAttributePropertyPermission
PrintCharacteristicsAccordingToProperty
menuStateMachine
loop
readTbSenseData
setup

4.1.1.3 deviceConnectedCallback()

```
void deviceConnectedCallback (
            BLEStatus_t status,
            uint16_t handle )
```

Callback for the establishment of the BLE connection.

Parameters

in	*status*	BLE_STATUS_CONNECTION_ERROR or BLE_STATUS_OK.
in	*handle*	Connect handle.

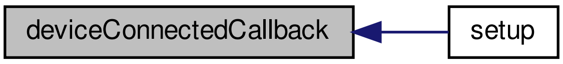
This function updates the connection handle and starts the procedure to discover services.

Return values

None	

```
                                                                   {

  switch (status){

    case BLE_STATUS_OK:
      Serial.println("");
      Serial.println("_____ Device connected");
      // Connect to remote device, start to discover service.
      connected_id = handle;
      device.connected_handle = handle;
      Serial.print("         - Device connected handle: ");
      Serial.println(connected_id);
      // Start to discover service, will report result on discoveredServiceCallback.
      Serial.println("");
      Serial.println("_____ Discovering Service");
      ble.discoverPrimaryServices(handle);
      break;

    default:
      break;
  }
}
```

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	*handle*	Connect handle.

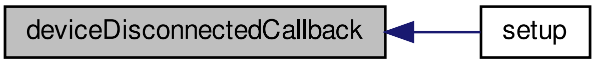
This function updates the connection handle to a not valid value, and restarts the Scanner procedure.

Return values

None	

```
                                                {
  Serial.println("");
  Serial.println("_____ Device disconnected");
  Serial.print("         - Device disconnected handle: ");
  Serial.println(handle,HEX);
  conf_completed = false;
  if (connected_id == handle) {
    Serial.println("");
    Serial.println("_____ BLE Central restart scanning!");
    // Disconnect from remote device, restart to scanning.
    connected_id = 0xFFFF;
    ble.startScanning();
  }
}
```

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	*status*	BLE_STATUS_OK/BLE_STATUS_DONE
in	*con_handle*	
in	*∗characteristic*	Discoverable characteristic.

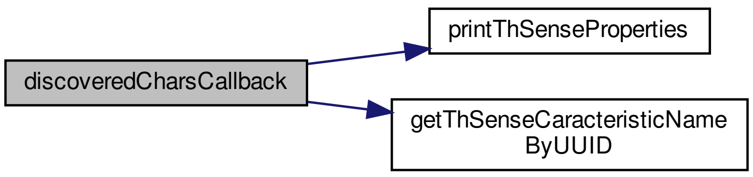
Callback for the handling result of discovering Characteristic, and once discovered, it starts the procedure for discovering descriptor.

Return values

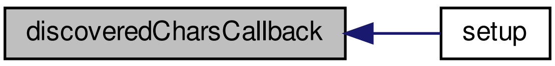
None	

```
                      {
  uint8_t index;
  uint8_t uuid_length = 16;
  char* characteristicName;
  if (status == BLE_STATUS_OK) {   // Found a characteristic.
    Serial.println(" ");
    Serial.print("* Service ");
    Serial.print(serv_index, HEX);
    Serial.print(" - Characteristic ");
    Serial.print(chars_index, HEX);
    Serial.println(" found successfully:");
    Serial.print("   - Characteristic start handle: ");
    Serial.println(characteristic->start_handle, HEX);
    Serial.print("   - Characteristic end handle: ");
    Serial.println(characteristic->end_handle, HEX);
    Serial.print("   - Characteristic value handle: ");
    Serial.println(characteristic->value_handle, HEX);
    Serial.print("   - Characteristic properties: ");
    Serial.print(characteristic->properties, HEX);
    printThSenseProperties(characteristic);
    Serial.print("   - Characteristic uuid16: ");
    Serial.println(characteristic->uuid16, HEX);
    Serial.print("   - Characteristic uuid128 : ");
    for (index = 0; index < 16; index++) {
      Serial.print(characteristic->uuid128[index], HEX);
      Serial.print(" ");
    }
    Serial.println(" ");
   if (chars_index < NCHAR_MAX) {
       characteristicName = getThSenseCaracteristicNameByUUID(
  characteristic->uuid128, uuid_length);
       Serial.println (characteristicName);
       device.service[serv_index].chars[chars_index].chars= *
  characteristic;
       chars_index++;
    }
  }
  else if (status == BLE_STATUS_DONE) {
    n_chars[serv_index] = chars_index;
    Serial.print("*** n_chars for Service ");
    Serial.print(serv_index);
    Serial.print(" = ");
    Serial.println(n_chars[serv_index]);
    serv_index++;
    if (serv_index < n_serv) {
      chars_index=0;
      ble.discoverCharacteristics(device.connected_handle, &
  device.service[serv_index].service);
    }
    else {
      Serial.println("");
      Serial.println("* Discover all Characteristics completed");
      Serial.println("--------------------------------------------------------------------------");
      Serial.println("");
      serv_index = 0;
      chars_index = 0;
      // All characteristics have been found, start to discover descriptors.
      // Result will be reported on discoveredCharsDescriptorsCallback.
      ble.discoverCharacteristicDescriptors(device.connected_handle, &
  device.service[serv_index].chars[chars_index].chars);
    }
  }
}
```

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	*status*	BLE_STATUS_OK/BLE_STATUS_DONE
in	*con_handle*	
in	**descriptor*	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.

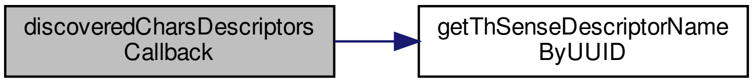
Return values

None	

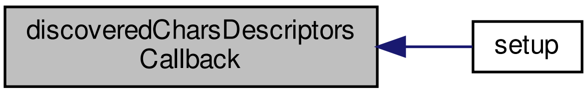
```
                                  {
  uint8_t index;
  char* descriptorName;
  if (status == BLE_STATUS_OK) {   // Found a descriptor.
    Serial.println(" ");
    Serial.print("* Service ");
    Serial.print(serv_index, HEX);
    Serial.print(" - Characteristic ");
    Serial.print(chars_index, 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");
```

```
        Serial.println("--------------------------------------------------------------------------");
        Serial.println("");
        serv_index = 0;
        chars_index =0;
        desc_index = 0;
       conf_completed = true;
      }
    }
  }
}
```

Here is the call graph for this function:


discoveredCharsDescriptors Callback
getThSenseDescriptorName ByUUID

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	*status*	BLE_STATUS_OK/BLE_STATUS_DONE
in	*con_handle*	
in	∗*service*	Discoverable service.

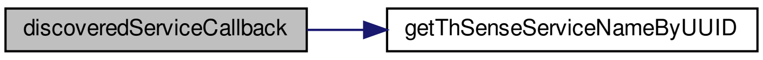
Callback for the handling result of discovering Service, and once discovered, it starts the procedure for discovering characteristic.

Return values

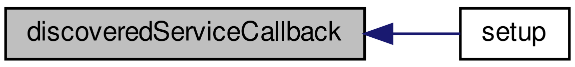
None	

```
    {
uint8_t index;
char* serviceName;
uint8_t uuid_lenght = 16;
if (status == BLE_STATUS_OK) {   // Found a service.
  Serial.println(" ");
  Serial.print("* Service found successfully ");
  Serial.print(serv_index, HEX);
  Serial.println(" :");
  Serial.print("   - Service start handle: ");
  Serial.println(service->start_group_handle, HEX);
  Serial.print("   - Service end handle: ");
  Serial.println(service->end_group_handle, HEX);
  Serial.print("   - Service uuid16: ");
  Serial.println(service->uuid16, HEX);
  Serial.print("   - Service uuid128 : ");
  for (index = 0; index < 16; index++) {
    Serial.print(service->uuid128[index], HEX);
    Serial.print(" ");
  }
  Serial.println(" ");
  if (serv_index < NSERV_MAX) {
     serviceName = getThSenseServiceNameByUUID( service->uuid128, uuid_lenght)
;
     Serial.println(serviceName);
     device.service[serv_index].service= *service;
     serv_index++;
  }
}
else if (status == BLE_STATUS_DONE) {
  Serial.println(" ");
  n_serv = serv_index;
  Serial.print("* Discover all Services completed (");
  Serial.print(n_serv);
  Serial.println(")");
  Serial.println("--------------------------------------------------------------------------");
  Serial.println("");
  serv_index = 0;
  // All sevice have been found, start to discover characteristics.
  // Result will be reported on discoveredCharsCallback.
  Serial.println("_____ Discovering Caracteristicd");
  ble.discoverCharacteristics(device.connected_handle, &
device.service[serv_index].service);
}
}
```

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

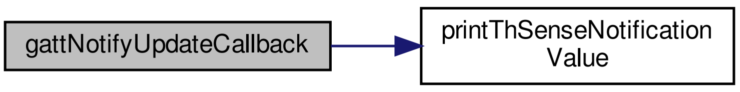
in	*status*	BLE_STATUS_OK
in	*con_handle*	
in	*value_handle*	
in	**value*	
in	*length*	

Return values

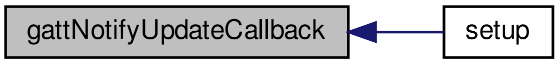
None	

```
                       {
  uint8_t index;
  Serial.println(" ");
  Serial.print("* Received new notification (");
  Serial.print(millis()- notification_lastmills);
  Serial.print(" ms) - (");
  Serial.print(length);
  Serial.println(" bytes):");
  notification_lastmills = millis();
  Serial.print("   - Connection handle: ");
  Serial.println(con_handle, HEX);
  Serial.print("   - Characteristic value attribute handle: ");
  Serial.println(value_handle, HEX);
  Serial.print("   - Notified value: ");
  for (index = 0; index < length; index++) {
    Serial.print(value[index], HEX);
    Serial.print(" ");
  }
    printThSenseNotificationValue( value_handle, value);
}
```

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

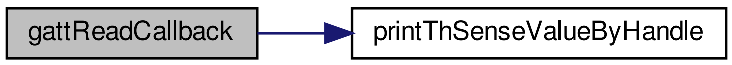
in	*status*	BLE_STATUS_OK/BLE_STATUS_DONE/BLE_STATUS_OTHER_ERROR
in	*con_handle*	
in	*value_handle*	
in	**value*	
in	*length*	

Return values

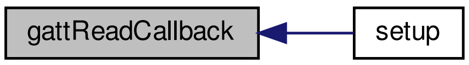
None	

```
                {
  uint8_t index;
  if (status == BLE_STATUS_OK) {
    Serial.println(" ");
    Serial.println("   * Read characteristic value successfully:");
    Serial.print("      - Connection handle: ");
    Serial.println(con_handle, HEX);
    Serial.print("      - Characteristic value attribute handle: ");
    Serial.println(value_handle, HEX);
    Serial.print("      - Characteristic value : ");
    for (index = 0; index < length; index++) {
      Serial.print(value[index], HEX);
      Serial.print(" ");
    }
    Serial.println("");
    printThSenseValueByHandle( value_handle, value, length);
  }
  else if (status != BLE_STATUS_DONE) {
    Serial.println(" ");
    Serial.println("! Read characteristic value FAILED");
    Serial.println(" ");
    }
}
```

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

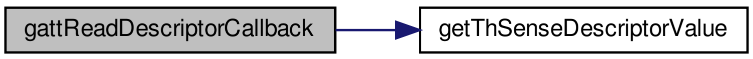
in	*status*	BLE_STATUS_DONE/BLE_STATUS_OTHER_ERROR
in	*con_handle*	
in	*value_handle*	
in	*∗value*	
in	*length*	

Return values

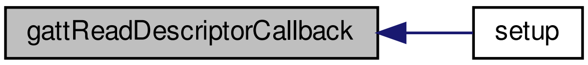
None	

```
                             {
  uint8_t index;
  char* enableOrDisableMesage;
  if(status == BLE_STATUS_OK) {
    Serial.println("");
    Serial.print("D --- gattReadDescriptorCallback (");
    Serial.println(" ");
    Serial.println("* Read descriptor value successfully:");
    Serial.print("   - Connection handle: ");
    Serial.println(con_handle, HEX);
    Serial.print("   - Descriptor value attribute handle: ");
    Serial.println(value_handle, HEX);
    Serial.print("   - Descriptor value : ");
    for (index = 0; index < length; index++) {
      Serial.print(value[index], HEX);
      Serial.print(" ");
    }
    Serial.println(" ");
    enableOrDisableMesage = getThSenseDescriptorValue( value_handle, value);
    Serial.println(enableOrDisableMesage);
  }
  else if (status == !BLE_STATUS_DONE) {
    Serial.println(" ");
    Serial.println("! ReadDescriptor FAILED");
    Serial.println(" ");
  }
}
```

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	*status*	BLE_STATUS_OK
in	*con_handle*	
in	*value_handle*	
in	*∗value*	
in	*length*	

Return values

None	

```
                                  {
  uint8_t index;
  Serial.println(" ");
  Serial.print("Receive new indication:");
  Serial.print(millis()- indication_lastmills);
  Serial.print(" ms) - (");
  Serial.print(length);
  Serial.println(" bytes):");
  indication_lastmills = millis();
  Serial.print("   - Connection handle: ");
  Serial.println(conn_handle, HEX);
  Serial.print("Characteristic value attribute handle: ");
  Serial.println(value_handle, HEX);
  Serial.print("Indicated data: ");
  for (index = 0; index < length; index++) {
    Serial.print(value[index], HEX);
    Serial.print(" ");
  }
  Serial.println(" ");

}
```

Here is the caller graph for this function:

gattReceivedIndicationCallback
setup

4.1.1.12 gattWriteCCCDCallback()

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

Callback for handling result of writting client characteristic configuration.

Parameters

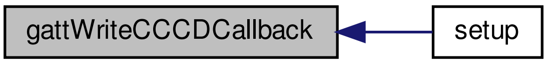
in	*status*	BLE_STATUS_DONE/BLE_STATUS_OTHER_ERROR
in	*con_handle*	

Return values

None	

```
                                                                           {
  Serial.println("");
  Serial.print("D --- gattWriteCCCDCallback (");
  Serial.print(status, HEX);
  Serial.println(")");
  if (status == BLE_STATUS_DONE) {
    Serial.println(" ");
    Serial.println("* Write CCCD value successfully");
    Serial.print("   - Connection handle: ");
    Serial.println(con_handle, HEX);
  }
  else {
    Serial.println(" ");
    Serial.println("! Write CCCD value FAILED");
  }
}
```

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 writting.

Parameters

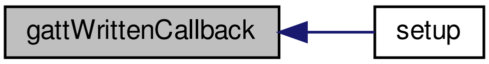
in	*status*	BLE_STATUS_DONE/BLE_STATUS_OTHER_ERROR
in	*con_handle*	

Return values

None	

```
                                                                          {
  if (status == BLE_STATUS_DONE) {
    Serial.println(" ");
    Serial.println("* Write characteristic value done:");
    Serial.print("   - Connection handle: ");
    Serial.println(con_handle, HEX);
  }
  else {
    Serial.println(" ");
    Serial.println("! Write characteristic value FAILED");
    Serial.println(" ");
  }
}
```

Here is the caller graph for this function:



4.1.1.14 getThSenseCaracteristicNameByUUID()

```
char* getThSenseCaracteristicNameByUUID (
            uint8_t Caracteristic_uuid128[],
            uint8_t uuid_length )
```

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

Parameters

in	*Caracteristic_uuid128[]*	The UUID of the Characteristic
in	*uuid_lenght*	The UUID lenght

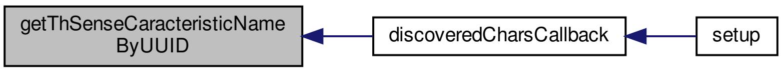
Return values

*char**	thSenseCharacteristicName Characteristic name associated with the UUID that was passed by parameter.

```
                                                                                                    {

  char* thSenseCharacteristicName;

```

```
if (0x00 == memcmp(Caracteristic_uuid128, Service0_Characrteristic0_Device_Name_uuid, uuid_length)) {
  thSenseCharacteristicName = (char*)"    - Device Name Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service0_Characrteristic1_Appearance_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Appearance Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service1_Characrteristic0_Service_Changed_uuid, uuid_length))
 {
      thSenseCharacteristicName = (char*)"    - Service Changed Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service2_Characrteristic0_Manufacturer_Name_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Manufacturer Name Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service2_Characrteristic1_Model_Number_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Model Number Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service2_Characrteristic2_Serial_Number_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Serial Number Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service2_Characrteristic3_Hardware_Revision_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Hardware Revision Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service2_Characrteristic4_Firmware_Revision_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Firmware Revision Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service2_Characrteristic5_System_ID_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - System ID Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service3_Characrteristic0_Battery_Level_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Battery Level Characrteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service4_Characrteristic0_UV_Index_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - UV Index Characrteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service4_Characrteristic1_Pressure_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Pressure Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service4_Characrteristic2_Temperature_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Temperature Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service4_Characrteristic3_Humidity_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Humidity Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service4_Characrteristic4_Ambient_Light_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Ambient Light Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service4_Characrteristic5_Sound_Level_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Sound Level Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service4_Characrteristic6_Control_Point_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Control Point Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service5_Characrteristic0_Power_Source_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Power Source Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service6_Characrteristic0_ECO2_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - ECO2 Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service6_Characrteristic1_TVOC_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - TVOC Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service6_Characrteristic2_Control_Point_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Control Point Characrteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service7_Characrteristic0_Buttons_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Buttons Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service7_Characrteristic1_Leds_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Leds Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service7_Characrteristic2_RGB_Leds_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - RGB Leds Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service7_Characrteristic3_Control_Point_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Control Point Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service8_Characrteristic0_Digital_1_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Digital 1 Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service8_Characrteristic1_Digital_2_uuid, uuid_length)) {
      thSenseCharacteristicName = (char*)"    - Digital 2 Characteristic found successfully";
  }else if (0x00 == memcmp(Caracteristic_uuid128, Service9_Characrteristic0_Acceleration_uuid, uuid_length)) {
        thSenseCharacteristicName = (char*)"   - Acceleration Characteristic found successfully";
    }else if (0x00 == memcmp(Caracteristic_uuid128,
   Service9_Characrteristic1_Orientation_uuid, uuid_length)) {
        thSenseCharacteristicName = (char*)"   - Orientation Characteristic found successfully";
    }else if (0x00 == memcmp(Caracteristic_uuid128,
   Service9_Characrteristic2_Control_Point_uuid, uuid_length)) {
        thSenseCharacteristicName = (char*)"   - Control Point Characrteristic found successfully";
    }else if (0x00 == memcmp(Caracteristic_uuid128,
   ServiceA_Characrteristic0_State_uuid, uuid_length)) {
        thSenseCharacteristicName = (char*)"   - State Characteristic found successfully";
    }else if (0x00 == memcmp(Caracteristic_uuid128,
   ServiceA_Characrteristic1_Field_Strength_uuid, uuid_length)) {
        thSenseCharacteristicName = (char*)"   - Field Strength Characteristic found successfully";
    }else if (0x00 == memcmp(Caracteristic_uuid128,
   ServiceA_Characrteristic2_Control_Point_uuid, uuid_length)) {
        thSenseCharacteristicName = (char*)"   - Control Point Characteristic found successfully";
    }else {
        thSenseCharacteristicName = (char*)"   _ The Characteristic Name is not define in the Central
    device";
   }
   return thSenseCharacteristicName;
}
```

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

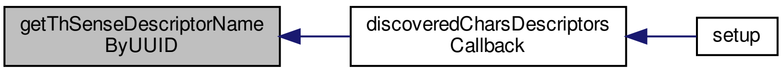
in	*Descriptor_uuid128[ ]*	The UUID128 of the descriptor

Return values

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

```
                                                                   {

   char* DescriptorName;

     if (0x00 == memcmp(Descriptor_uuid128,
   Client_Characteristic_Configuration_uuid, sizeof(Descriptor_uuid128
   ))) {
     DescriptorName = (char*)"   - Client_Characteristic_Configuration";
   }else if (0x00 == memcmp(Descriptor_uuid128,
   Characteristic_Presentation_Format_uuid, sizeof(Descriptor_uuid128))
   ) {
     DescriptorName = (char*)"   - Characteristic_Presentation_Format" ;
   }else if (0x00 == memcmp(Descriptor_uuid128, noOfDigitals_uuid, sizeof(
   Descriptor_uuid128))) {
     DescriptorName = (char*)"   - noOfDigitals";
   }else{
     DescriptorName = (char*)"   - The Descriptor Name is not define the in Central device";
     }
   return DescriptorName;
 }
```

Here is the caller graph for this function:


getThSenseDescriptorName ByUUID
discoveredCharsDescriptors Callback
setup

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	*value_handle*	The value_handle of the Descriptor
in	**value*	The value of the Descriptor

Return values

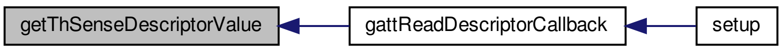
None	

```
                                                                       {

  char* thSenseDescriptorValue;

  if (value_handle == device.service[n_generic_attribute_service].
  chars[0].descriptor[0].handle) {
    if ((value[0] == 0x02) && (value[1] == 0x00)) {
      thSenseDescriptorValue = (char*)"_____ Indication are enabled for Generic Attribute Service";
    }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
      thSenseDescriptorValue = (char*)"_____ Indication are disabled for Generic Attribute Service";
    }
  }else if (value_handle == device.service[n_battery_service].chars[0].
  descriptor[0].handle) {
    if ((value[0] == 0x01) && (value[1] == 0x00)) {
      thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Battery Service";
    }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
      thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Battery Service";
    }
    }else if (value_handle == device.service[n_env_sensing_service].chars
   [6].descriptor[0].handle) {
    if ((value[0] == 0x02) && (value[1] == 0x00)) {
```

```
    thSenseDescriptorValue = (char*)"_____ Indication are enabled for Environmental Sensing Service";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are disabled for Environmental Sensing Service";
  }
  }else if (value_handle == device.service[n_iaq_service].chars[2].descriptor[0].handle) {
  if ((value[0] == 0x02) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are enabled for IAQ Service";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are disabled for IAQ Service";
  }
  }else if (value_handle == device.service[n_user_interface_service].chars[1].descriptor[0].handle) {
  if ((value[0] == 0x02) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are enabled for User Interface Service";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are disabled for User Interface Service";
  }
  }else if (value_handle == device.service[n_user_interface_service].chars[2].descriptor[0].handle) {
  if ((value[0] == 0x02) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are enabled for User Interface Service";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are disabled for User Interface Service";
  }
}else if (value_handle == device.service[n_user_interface_service].chars[3].descriptor[0].handle) {
  if ((value[0] == 0x02) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are enabled for User Interface Service";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are disabled for User Interface Service";
  }
}else if (value_handle == device.service[n_automation_io_service].chars[0].descriptor[0].handle) {
  if ((value[0] == 0x01) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Automation IO Service";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Automation IO Service";
  }
}else if (value_handle == device.service[n_accleration_orientation_service].chars[0].descriptor[0].handle) {
  if ((value[0] == 0x01) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Accleration Characteristic";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Accleration Characteristic";
  }
}else if (value_handle == device.service[n_accleration_orientation_service].chars[1].descriptor[0].handle) {
  if ((value[0] == 0x01) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Orientation Characteristic";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Orientation Characteristic";
  }
  }else if (value_handle == device.service[n_accleration_orientation_service].chars[2].descriptor[0].handle) {
  if ((value[0] == 0x02) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are enabled for User Orientation Acceleration Characteristic";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Indication are disabled for User Orientation Acceleration Characteristic";
  }
}else if (value_handle == device.service[n_hall_effect_service].chars[0].descriptor[0].handle) {
  if ((value[0] == 0x01) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Hall State Characteristic";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Hall State Characteristic";
  }
  }else if (value_handle == device.service[n_hall_effect_service].chars[2].descriptor[0].handle) {
  if ((value[0] == 0x01) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Hall effect Characteristic";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Hall effect Characteristic";
  }
}else if (value_handle == device.service[n_hall_effect_service].chars[1].descriptor[0].handle) {
  if ((value[0] == 0x01) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are enabled for Field Strength Characteristic";
  }else if ((value[0] == 0x00) && (value[1] == 0x00)) {
    thSenseDescriptorValue = (char*)"_____ Notifications are disabled for Field Strength Characteristic";
  }
}else{
  thSenseDescriptorValue = (char*)"_____ The Descriptor value is not defined in the Central device";
```

```
    }
  Serial.println(" ");
  return thSenseDescriptorValue;
}
```

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

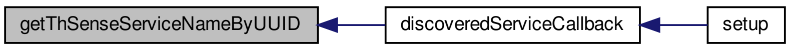
in	*Service_uuid128[]*	The UUID of the Service
in	*uuid_lenght*	The UUID lenght

Return values

char∗	thSenseServiceName Service name associated with the UUID that was passed by parameter.

```
                                                                                 {

  char* thSenseServiceName;

  if (0x00 == memcmp(Service_uuid128, battery_service_uuid, uuid_lenght)) {
    n_battery_service = serv_index;
    thSenseServiceName = (char*)"   - Battery Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, environmental_sensing_service_uuid
  , uuid_lenght)) {
    n_env_sensing_service = serv_index;
    thSenseServiceName = (char*)"   - Environmental Sensing Service found successfully" ;
  }else if (0x00 == memcmp(Service_uuid128, iaq_service_uuid, uuid_lenght)) {
    n_iaq_service = serv_index;
    thSenseServiceName = (char*)"   - IAQ Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, generic_access_service_uuid,
  uuid_lenght)) {
    n_generic_access_service = serv_index;
    thSenseServiceName = (char*)"   - Generic Access Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, generic_attribute_service_uuid,
  uuid_lenght)) {
    n_generic_attribute_service = serv_index;
    thSenseServiceName = (char*)"   - Generic Attribute Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, device_information_service_uuid,
   uuid_lenght)) {
    n_device_information_service = serv_index;
    thSenseServiceName = (char*)"   - Device Information Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, power_management_service_uuid,
  uuid_lenght)) {
    n_power_management_service = serv_index;
    thSenseServiceName = (char*)"    - Power Management Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, user_interface_service_uuid,
  uuid_lenght)) {
    n_user_interface_service = serv_index;
    thSenseServiceName = (char*)"    - User Interface Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, automation_io_service_uuid,
  uuid_lenght)) {
    n_automation_io_service = serv_index;
    thSenseServiceName = (char*)"    - Automation IO Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, accleration_orientation_service_uuid
  , uuid_lenght)) {
    n_accleration_orientation_service =
  serv_index;
    thSenseServiceName = (char*)"    - Accleration Orientation Service found successfully";
  }else if (0x00 == memcmp(Service_uuid128, hall_effect_service_uuid, uuid_lenght))
   {
    n_hall_effect_service = serv_index;
    thSenseServiceName = (char*)"    - Hall Effect Service found successfully";
  }else{
        thSenseServiceName = (char*)"    -  The name of the service is not defined in the Central device";
    }
  return thSenseServiceName;
}
```

Here is the caller graph for this function:



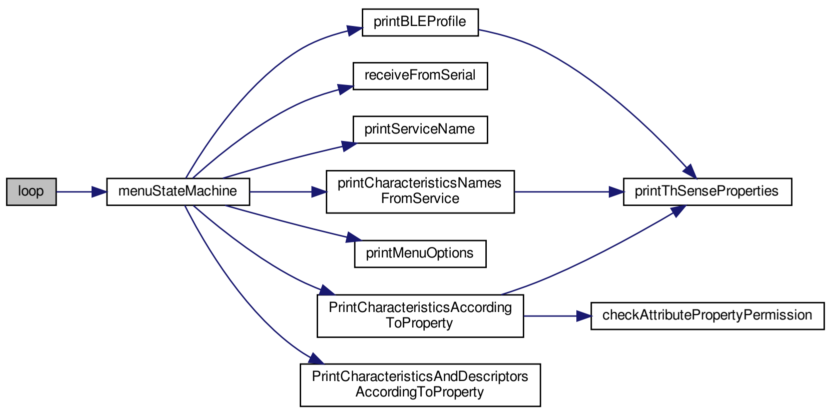
4.1.1.18 loop()

```
void loop ( )
```

Loop.

```
              {

#if MENU_DEBUG >= 1

  if(conf_completed == true){
    menuStateMachine();
  }

#endif
}
```

Here is the call graph for this function:


loop
menuStateMachine
printBLEProfile
receiveFromSerial
printServiceName
printCharacteristicsNames FromService
printMenuOptions
PrintCharacteristicsAccording ToProperty
PrintCharacteristicsAndDescriptors AccordingToProperty
printThSenseProperties
checkAttributePropertyPermission

4.1.1.19 menuStateMachine()

```
void menuStateMachine ( )
```

This function implement the Satate Machine for the MENU.

Parameters

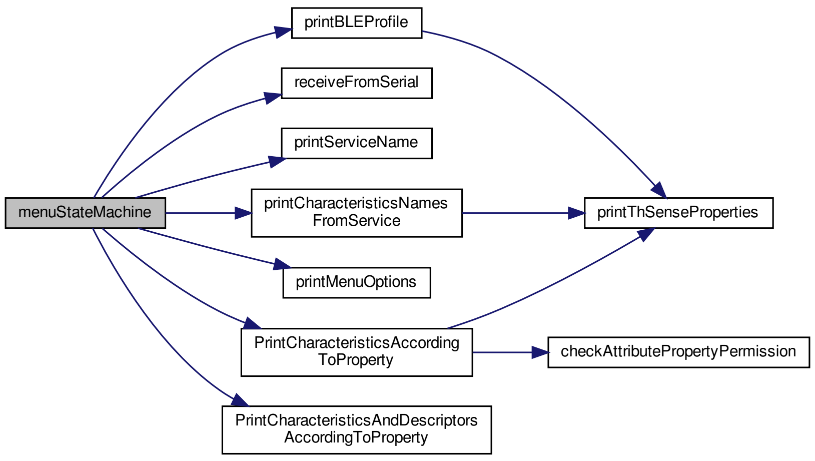
None	States: BLE_CENTRAL_READ_CARACTERISTIC_VALUE BLE_CENTRAL_READ_DESCRIPTOR_VALUE BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS BLE_CENTRAL_WRITE

Return values

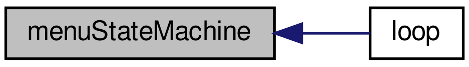
None	

```
                         {

  uint8_t numService        = 0;
  uint8_t numCharacteristic = 0;
  uint8_t enable_disable    = 0;

  printBLEProfile();
  numService = receiveFromSerial((char*)" Enter number of service): ");
  if(numService >= 0 && numService < NSERV_MAX){
    Serial.println(numService);
    printServiceName(numService);
    printCharacteristicsNamesFromService(numService);
    printMenuOptions();
    menuOption = (stateEnum_t)receiveFromSerial((char*)"Enter the
  number corresponding to the menu option to perform: ");
    if(menuOption >=0 && menuOption <=4){
      Serial.print(menuOption);
      Serial.println("");

      switch (menuOption){

        case BLE_CENTRAL_READ_CARACTERISTIC_VALUE:
          PrintCharacteristicsAccordingToProperty(numService, 1);
          if(thereIsCharacteristic != 0){
            Serial.println("");
            numCharacteristic = receiveFromSerial((char*)" Enter number of characteristic): ");
            if(numCharacteristic >= 0 && numCharacteristic < thereIsCharacteristic){
              Serial.print(numCharacteristic);
              Serial.println("");
              ble.readValue(device.connected_handle, &device.service[numService].chars[numCharacteristic].chars);
              delay(3000);
              //Serial.println("_________LEIDO");
             // Serial.println(thereIsCharacteristic);//DEBUG MIRrarararararaaaaaaaaaaaaaaaaaaaaa
             // numCharacteristic = receiveFromSerial((char*)"_________Pulse cualquier tecla para volver");
            }else{
              Serial.println(" Not a valid option ");
              delay(3000);
            }
          }else{
            Serial.println(" Not caracteristic with read properties!!!! ");
            delay(3000);
          }
        break;

        case BLE_CENTRAL_READ_DESCRIPTOR_VALUE:
          PrintCharacteristicsAndDescriptorsAccordingToProperty(numService, 1);
          if(thereIsCharacteristic != 0){
            Serial.println("");
            numCharacteristic = receiveFromSerial((char*)" Enter number of characteristic): ");
            if(((device.service[numService].chars[numCharacteristic].descriptor[0].uuid16)== 0x2902)){
              Serial.print(numCharacteristic);
              Serial.println("");
              ble.readDescriptorValue(device.connected_handle, &device.service[numService].chars[numCharacteristic].descriptor[0]);
              delay(3000);
            }else{
              Serial.println(" Not a valid option ");
              delay(3000);
            }
          }else{
            Serial.println(" They are Not Characteristic Descriptors!!!! ");
            delay(3000);
          }
        break;

        case BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS:
          PrintCharacteristicsAccordingToProperty(numService, 4);
          if(thereIsCharacteristic != 0){
            Serial.println("");
            numCharacteristic = receiveFromSerial((char*)" Enter number of characteristic): ");
            if(numCharacteristic >= 0 && numCharacteristic < thereIsCharacteristic){
              Serial.print(numCharacteristic);
              Serial.println("");
              ble.writeClientCharsConfigDescriptor(device.connected_handle, &device.service[numService].chars[numCharacteristic].chars, GATT_CLIENT_CHARACTERISTICS_CONFIGURATION_NOTIFICATION);
              enable_disable = 1;
              while(enable_disable){
                Serial.println("Para deshabilitar pulse 0");
                enable_disable = Serial.read() - '0';
                delay(2000);
              }
              ble.writeClientCharsConfigDescriptor(device.connected_handle, &device.service[numService].chars[numCharacteristic].chars, GATT_CLIENT_CHARACTERISTICS_CONFIGURATION_NONE);
              delay(3000);
            }else{
              Serial.println(" Not a valid option ");
              delay(3000);
            }
          }else{
            Serial.println(" Not Characteristic with notify properties!!!! ");
            delay(3000);
          }
        break;

        case BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS:
          PrintCharacteristicsAccordingToProperty(numService, 5);
          if(thereIsCharacteristic != 0){
            Serial.println("");
            numCharacteristic = receiveFromSerial((char*)" Enter number of characteristic): ");
            if(numCharacteristic >= 0 && numCharacteristic < thereIsCharacteristic){
              Serial.print(numCharacteristic);
              Serial.println("");
              enable_disable = receiveFromSerial((char*)" To enable enter 2, and to disable enter 3): ");
              if(enable_disable == 2){
                ble.writeClientCharsConfigDescriptor(device.connected_handle, &device.service[numService].chars[numCharacteristic].chars, GATT_CLIENT_CHARACTERISTICS_CONFIGURATION_INDICATION);
                delay(3000);
              }else if(enable_disable == 3){
                ble.writeClientCharsConfigDescriptor(device.connected_handle, &device.service[numService].chars[numCharacteristic].chars, GATT_CLIENT_CHARACTERISTICS_CONFIGURATION_NONE);
                delay(3000);
              }else{
                Serial.println(" Not a valid option ");
                delay(3000);
              }
            }else{
              Serial.println(" Not a valid option ");
              delay(3000);
            }
          }else{
            Serial.println(" Not Characteristic with Indicate properties!!!! ");
            delay(3000);
          }
        break;

        case BLE_CENTRAL_WRITE:
          uint8_t write_data[20];
          uint8_t w_data_length = 0;
          uint8_t write_data_temp = 0;
          PrintCharacteristicsAccordingToProperty(numService, 3);
          if(thereIsCharacteristic != 0){
            Serial.println("");
            numCharacteristic = receiveFromSerial((char*)" Enter number of characteristic): ");
            if(numCharacteristic >= 0 && numCharacteristic < thereIsCharacteristic){
              Serial.println(numCharacteristic);
              w_data_length = receiveFromSerial((char*)" Enter write data lenght in Bytes): ");
              Serial.println(w_data_length);
              for (int i=0; i<w_data_length;i++){
                for(int j=0; j<2;j++){
                  write_data_temp = receiveFromSerial((char*)"");
                  if(!j){
                    write_data[i]= write_data_temp << 4;
                  }else{
                    write_data[i]|= write_data_temp;
                  }
                }
              }
            ble.writeValue(device.connected_handle, device.service[numService].chars[numCharacteristic].chars.value_handle, w_data_length, write_data);
            delay(3000);
          }else{
            Serial.println(" Not a valid option ");
            delay(3000);
          }
        }else{
          Serial.println(" Not Characteristic with Write properties!!!! ");
          delay(3000);
        }
        break;
      }
    }
  }
}
```

Here is the call graph for this function:


printBLEProfile
receiveFromSerial
printServiceName
menuStateMachine
printCharacteristicsNames FromService
printThSenseProperties
printMenuOptions
PrintCharacteristicsAccording ToProperty
checkAttributePropertyPermission
PrintCharacteristicsAndDescriptors AccordingToProperty

Here is the caller graph for this function:



4.1.1.20 printBLEProfile()

```
void printBLEProfile ( )
```

Function to print the BLE profile.

Parameters

None	

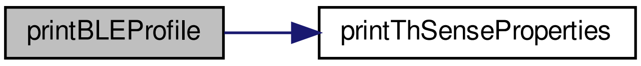
Return values

None	

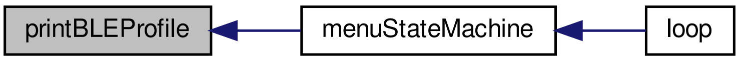
```
                        {
```

```

  uint8_t numService        = 0;
  uint8_t numCharacteristic = 0;
  uint8_t numDescriptor     = 0;
  uint16_t decriptorUUID        = 0;

  for( numService = 0; numService < NSERV_MAX; numService++){
    Serial.print("   ");
    Serial.print("   ");
    Serial.print("* Service ");
    Serial.print(numService, HEX);
    Serial.println(services_name[numService]);
    for(numCharacteristic = 0; numCharacteristic < NCHAR_MAX; numCharacteristic++){
      if(caracteristics_name_gruped_by_service [numService][
numCharacteristic] != NULL){
        Serial.print("      ");
        Serial.print("- Characteristic ");
        Serial.print(numCharacteristic);
        Serial.print(caracteristics_name_gruped_by_service [numService
][numCharacteristic]);
        printThSenseProperties(&device.service[numService].chars[
numCharacteristic].chars);
        for(numDescriptor = 0; numDescriptor < NDESC_MAX; numDescriptor++){
          decriptorUUID = device.service[numService].chars[numCharacteristic].descriptor[
numDescriptor].uuid16;
          switch(decriptorUUID){
            case 0x2902:
              Serial.print("         ");
              Serial.print("- Descriptor");
              Serial.print(numDescriptor);
              Serial.println(" Client Characteristic Configuration ");
            break;

            case 0x2904:
              Serial.print("         ");
              Serial.print("- Descriptor ");
              Serial.print(numDescriptor);
              Serial.println(" Characteristic Presentation Format ");
            break;

            case 0x2909:
              Serial.print("         ");
              Serial.print("- Descriptor");
              Serial.print(numDescriptor);
              Serial.println(" noOfDigitals ");
            break;
          }
        }
      }
    }
  }
}
```

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

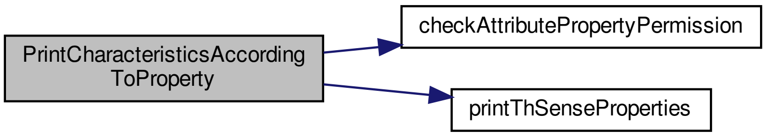
None	uint8_t numSer The number of the Service

Return values

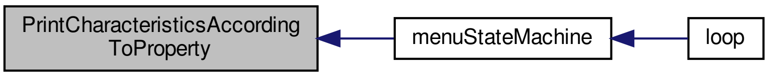
None	

```
                                                                                    {

  thereIsCharacteristic = 0;

  for(int num_Characteristic = 0; num_Characteristic < NCHAR_MAX; num_Characteristic++){
    if(caracteristics_name_gruped_by_service [numService][
num_Characteristic] != NULL){
      if(checkAttributePropertyPermission(numService, num_Characteristic,
property)){
        thereIsCharacteristic++;
        Serial.print("      ");
        Serial.print(num_Characteristic);
        Serial.print(caracteristics_name_gruped_by_service [numService
][num_Characteristic]);
        printThSenseProperties(&device.service[numService].chars[
num_Characteristic].chars);
      }
    }
  }
}
```

Here is the call graph for this function:


PrintCharacteristicsAccording ToProperty
checkAttributePropertyPermission
printThSenseProperties

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	

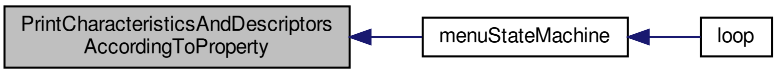
Return values

None	

```
                                                                                           {

 thereIsCharacteristic = 0;

 for(uint8_t numero_Caracteristica = 0; numero_Caracteristica < NCHAR_MAX; numero_Caracteristica+
+){
   if(caracteristics_name_gruped_by_service [numSer][
numero_Caracteristica] != NULL){
     for(uint8_t numero_Descriptor = 0; numero_Descriptor < NDESC_MAX; numero_Descriptor++){
       if((device.service[numSer].chars[numero_Caracteristica].descriptor[numero_Descriptor].
uuid16 & (1 << 13)) != 0){
         Serial.print("      ");
```

```
          Serial.print(numero_Caracteristica);
          Serial.println(caracteristics_name_gruped_by_service [numSer
    ][numero_Caracteristica]);
          Serial.print("           ");
          Serial.print(numero_Descriptor);
          Serial.println(" Client Characteristic Configuration ");
          thereIsCharacteristic++;
        }
      }
    }
  }
}
```

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

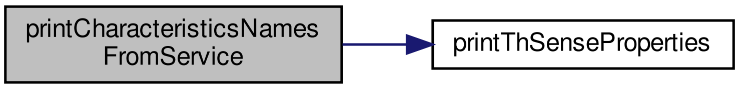
uint8←_t	numSer The number of the Service

Return values

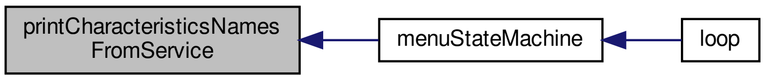
None	

```
                                                              {

  for(uint8_t num_Characteristic = 0; num_Characteristic < NCHAR_MAX; num_Characteristic++){
    if(caracteristics_name_gruped_by_service [numService][
    num_Characteristic] != NULL){
      Serial.print("      ");
      Serial.print(num_Characteristic);
      Serial.print(caracteristics_name_gruped_by_service [numService][
    num_Characteristic]);
      printThSenseProperties(&device.service[numService].chars[
    num_Characteristic].chars);
      for(uint8_t numDescriptor = 0; numDescriptor < NDESC_MAX; numDescriptor++){
        if((device.service[numService].chars[num_Characteristic].descriptor[numDescriptor].
    uuid16 & 1 << 13) != 0){//0x2900 -- UUID IS NOT NULL
          Serial.print("           ");
          Serial.print(numDescriptor);
          Serial.println(" Client Characteristic Configuration ");
        }
      }
    }
  }
}
```

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

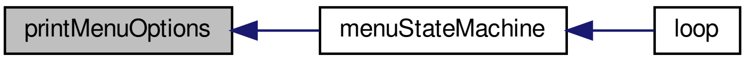
None	

Return values

None	

```
                        {

  Serial.println("");
  Serial.println("MENU:");
  Serial.println("--------------------------------------------------");
  Serial.println("0. Leer Atributo");
  Serial.println("1. Leer Descriptor");
  Serial.println("2. Habilitar/deshabilitar Notificaciones");
  Serial.println("3. Habilitar/deshabilitar Indicaciones");
  Serial.println("4. Escribir Atributo");
  Serial.println("--------------------------------------------------");
}
```

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

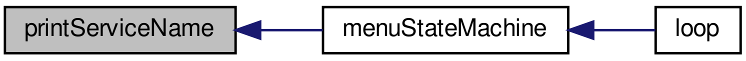
in	*uint8↩_t*	numSer The identifier of the Service.

Return values

None	

```
                                     {

  Serial.print("   ");
  Serial.print(numSer);
  Serial.println(services_name[numSer]);
}
```

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

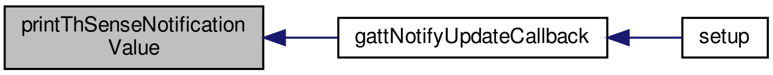
in	*value_handle*	The handle of the Attribute from which the notification is received
in	∗*value*	Pointer to the value of the attribute from which the notification is received

Return values

None	

```
                                                                          {

  switch(value_handle){

    case 0x19:
      Battery_Level = value[0];
      Serial.print(caracteristics_name_gruped_by_service [3][0]);
      Serial.print(Battery_Level);
      Serial.println(" %");
      break;

    case 0x58:
      Hall_State = value[0];
      Serial.print(caracteristics_name_gruped_by_service [10][0]);
      Serial.print(Hall_State);
      break;

    case 0x5B :
      Field_Strength = int32_t((value[3] <<24) + (value[2] << 16) + (value[1] << 8) + value[0
]);
      Serial.print(caracteristics_name_gruped_by_service [10][1]);
      Serial.print(Field_Strength);
      Serial.println(" uT");
      break;

    case 0x4E:
      Acceleration_axis_X = int16_t((value[1] << 8) + value[0])/1000;
      Acceleration_axis_Y = int16_t((value[3] << 8) + value[2])/1000;
      Acceleration_axis_Z = int16_t((value[5] << 8) + value[4])/1000;
      Serial.println(caracteristics_name_gruped_by_service [9][0]);
      Serial.print(" Acceleration axis X = ");
      Serial.println(Acceleration_axis_X + "g");
      Serial.print(" Acceleration axis Y = ");
      Serial.println(Acceleration_axis_Y + "g");
      Serial.print(" Acceleration axis Z = ");
      Serial.println(Acceleration_axis_Z + "g");
      break;

    case 0x51 :
      Orientation_axis_X = int16_t((value[1] << 8) + value[0])/100;
      Orientation_axis_Y = int16_t((value[3] << 8) + value[2])/100;
      Orientation_axis_Z = int16_t((value[5] << 8) + value[4])/100;
      Serial.println(caracteristics_name_gruped_by_service [9][0]);
      Serial.print(" Orientation axis X = ");
      Serial.println(Orientation_axis_X + "°");
      Serial.print(" Orientation axis Y = ");
      Serial.println(Orientation_axis_Y + "°");
      Serial.print(" Orientation axis Z = ");
      Serial.println(Orientation_axis_Z + "°");
      break;

    default:
      Serial.println("Caracteristic handle value is not define");
      break;
  }
}
```

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	*∗gatt_client_characteristic_t*	characteristic Pointer to the characteristic struct

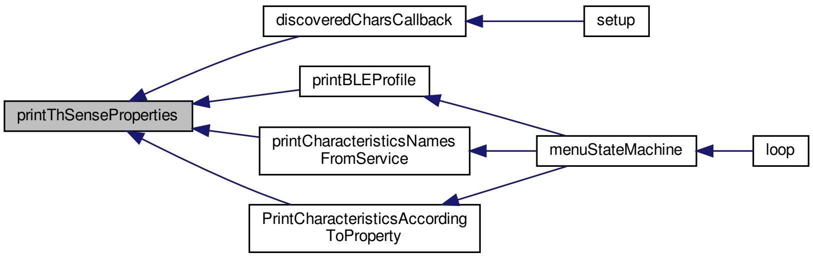
Return values

None	

```
                                                                            {

  switch (characteristic->properties){

    case 0x02 :
      Serial.println("   -Read ");
      break;

    case 0x0A :
      Serial.println("   -Read and Write ");
      break;

    case 0x20 :
      Serial.println("   -Indicate ");
      break;

    case 0x12 :
      Serial.println("   -Read and Notify ");
      break;

    case 0x28 :
      Serial.println("   -Write and Indicate ");
      break;

    case 0x2A :
      Serial.println("   -Read and Write and Indicate ");
      break;

    case 0x10 :
      Serial.println("   -Notify ");
      break;

    default  :
      Serial.println("   -Not defined in Central device");
```

```
        break;
    }
}
```

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	*value_handle*	The handle of the Attribute value
in	∗*value*	The Attribute value
in	*length*	The Attribute value length

Return values

None	

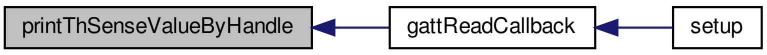
```
                                                                                   {

  switch (value_handle){

    case 0x1D:
      UVindex = value[0];
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [4][0]);
      Serial.println(UVindex);
      break;

    case 0x1F:
      Preasure = uint32_t((value[3] <<24) + (value[2] << 16) + (value[1] << 8) + value[0]) /1000;
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [4][1]);
      Serial.print(Preasure);
```

```
        Serial.println(" hPa");
        break;

      case 0x21:
        Temperature = int16_t((value[1] << 8) + value[0]) /100;
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [4][2]);
        Serial.print(Temperature);
        Serial.println(" °C");
        break;

      case 0x23:
        Humidity = uint16_t((value[1] << 8) + value[0]) /100;
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [4][3]);
        Serial.print(Humidity);
        Serial.println(" %");
        break;

      case 0x25:
        ALight = uint32_t((value[3] <<24) + (value[2] << 16) + (value[1] << 8) + value[0]) /1000;
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [4][4]);
        Serial.print(ALight);
        Serial.println(" Lux");
        break;

      case 0x27:
        Sound = int16_t((value[1] << 8) + value[0]) /100;
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [4][5]);
        Serial.print(Sound);
        Serial.println(" dB");
        break;

      case 0x30:
        ECO2 = uint16_t((value[1] << 8) + value[0]);
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [6][0]);
        Serial.print(ECO2);
        Serial.println(" ppm");
        break;

      case 0x32:
        TVOC = uint16_t((value[1] << 8) + value[0]);
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [6][1]);
        Serial.print(TVOC);
        Serial.println(" ppb");
        break;

      case 0x19:
        Battery_Level = value[0];
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [3][0]);
        Serial.print(Battery_Level);
        Serial.println(" %");
        break;

      case 0x58:
        Hall_State = value[0];
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [10][0]);
        Serial.print(Hall_State);
        break;

      case 0x5B :
        Field_Strength = int32_t((value[3] <<24) + (value[2] << 16) + (value[1] << 8) + value[0
]);
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [10][1]);
        Serial.print(Field_Strength);
        Serial.println(" uT");
      break;

      case 0x2D :
        Power_Source = value[0];
        Serial.print("      -");
        Serial.print(caracteristics_name_gruped_by_service [5][0]);
        Serial.print(Power_Source);
        if(Power_Source==0x01){
          Serial.println("  USB power");
        }else{
          Serial.println("  CR2032 power");
        }
        break;

```

```
    case 0x03 :
      memcpy(Device_Name, value, length);
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [0][0]);
      Serial.println((const char *)Device_Name);
      break;

    case 0x05 :
      Appearance = value[0];
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [0][1]);
      Serial.println(Appearance, HEX);
      break;

    case 0x0C :
      memcpy(Manufacturer_Name, value, length);
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [2][0]);
      Serial.println((const char *)Manufacturer_Name);
      break;

    case 0x0E :
      memcpy(Model_Number, value, length);
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [2][1]);
      Serial.println((const char *)Model_Number);
    break;

    case 0x10 :
      memcpy(Serial_Number, value, length);
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [2][2]);
      Serial.println((const char *)Serial_Number);
      break;

    case 0x12 :
      memcpy(Hardware_Revision, value, length);
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [2][3]);
      Serial.println((const char *)Hardware_Revision);
    break;

    case 0x14 :
      memcpy(Firmware_Revision, value, length);
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [2][4]);
      Serial.println((const char *)Firmware_Revision);
      break;

    case 0x16 :
      memcpy(System_ID, value, length);
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [2][5]);
      Serial.println((const char *)System_ID);
      break;

    case 0x44 :
      Digital_1 = value[0];
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [2][5]);
      Serial.println(Digital_1);
      break;

    case 0x49 :
      Digital_2 = value[0];
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [8][0]);
      Serial.println(Digital_2);
      break;

    case 0x38 :
      Buttons = value[0];
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [8][1]);
      Serial.println(Buttons);
      break;

    case 0x3D :
      RGB_Leds = uint32_t((value[0] <<24) + (value[1] << 16) + (value[2] << 8) + value[3]);
      Serial.print("      -");
      Serial.print(caracteristics_name_gruped_by_service [7][2]);
      Serial.println(RGB_Leds, HEX);
      break;

    default:
      Serial.print("____ The Caracteristic handle value is not defined in the Central device");
      break;
```

```
  }
  Serial.println(" ");
  Serial.println("--------------------------------------------------------------------------");
}
```

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

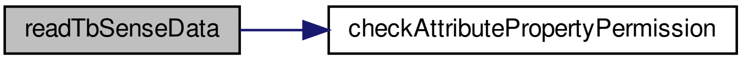
in	*btstack_timer_← source_t*	∗ts Pointer to the BLE stack timer source

Return values

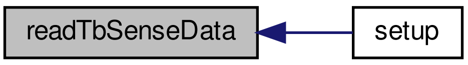
None	

```
                                                         {

  if ((connected_id != 0xFFFF) && (conf_completed) ) {
    Serial.println(" ");
    if ((n_serv_index == n_env_sensing_service) && (
n_serv_sel[n_env_sensing_service] == 1)) {
      if(checkAttributePropertyPermission(
n_env_sensing_service, n_chars_index, 1)){
        Serial.print("> Environmental sensing characteristic");
        Serial.print(n_chars_index);
        Serial.println("_read: ");
        Serial.print("    - Connection handle: ");
        Serial.println(device.connected_handle, HEX);
        Serial.print("    - Characteristic value attribute handle: ");
        Serial.println(device.service[n_env_sensing_service].chars[
n_chars_index].chars.value_handle, HEX);
        ble.readValue(device.connected_handle,&device.
service[n_env_sensing_service].chars[n_chars_index].chars);
      }
      if (n_chars_index < (n_chars[n_env_sensing_service]-1)){
        n_chars_index++;
      }else{
        n_chars_index = 0;
        Serial.println("--------------------------------------------------------------------------");
        Serial.println("- ENVIRONMENTAL SENSING                                                 -");
        Serial.println("--------------------------------------------------------------------------");
        Serial.print("    - UV Index = ");
        Serial.println(UVindex);
        Serial.print("    - Preasure = ");
        Serial.print(Preasure);
        Serial.println(" hPa");
        Serial.print("    - Temperature = ");
        Serial.print(Temperature);
        Serial.println(" °C");
        Serial.print("    - Humidity = ");
        Serial.print(Humidity);
        Serial.println(" %");
        Serial.print("    - Ambient Light = ");
        Serial.print(ALight);
        Serial.println(" Lux");
        Serial.print("    - Sound Level = ");
        Serial.print(Sound);
        Serial.println(" dB");
        Serial.println("--------------------------------------------------------------------------");
        n_serv_index = 6;
      }
    }else if ((n_serv_index == n_battery_service) && (n_serv_sel[n_battery_service] == 1)) {
      if(checkAttributePropertyPermission(n_battery_service, n_chars_index, 1)){
        Serial.print("> Battery Level characteristic");
        Serial.print(n_chars_index);
        Serial.println("_read: ");
        Serial.print("    - Connection handle: ");
        Serial.println(device.connected_handle, HEX);
        Serial.print("    - Characteristic value attribute handle: ");
        Serial.println(device.service[n_battery_service].chars[n_chars_index].chars.value_handle, HEX);
        ble.readValue(device.connected_handle,&device.service[n_battery_service].chars[n_chars_index].chars);
      }
      if (n_chars_index < (n_chars[n_battery_service]-1)){
        n_chars_index++;
      }else{
        n_chars_index = 0;
        Serial.println("--------------------------------------------------------------------------");
        Serial.println("- Battery Level Service                                                 -");
        Serial.println("--------------------------------------------------------------------------");
        Serial.print("    - Battery Level = ");
        Serial.print(Battery_Level);
        Serial.println(" %");
        Serial.println("--------------------------------------------------------------------------");
        n_serv_index = 4;
      }
    }else if ((n_serv_index == n_hall_effect_service) && (n_serv_sel[n_hall_effect_service] == 1)) {
      if(checkAttributePropertyPermission(n_hall_effect_service, n_chars_index, 1)){
        Serial.print("> Hall Effect characteristic");
        Serial.print(n_chars_index);
        Serial.println("_read: ");
        Serial.print("    - Connection handle: ");
        Serial.println(device.connected_handle, HEX);
        Serial.print("    - Characteristic value attribute handle: ");
        Serial.println(device.service[n_hall_effect_service].chars[n_chars_index].chars.value_handle, HEX);
        ble.readValue(device.connected_handle,&device.service[n_hall_effect_service].chars[n_chars_index].chars);
      }
      if (n_chars_index < (n_chars[n_hall_effect_service]-1)){
        n_chars_index++;
      }else {
        n_chars_index = 0;
        Serial.println("--------------------------------------------------------------------------");
        Serial.println("- Hall Effect Service                                                 -");
        Serial.println("--------------------------------------------------------------------------");
        Serial.print("    - Hall State = ");
        Serial.print(Hall_State);
        Serial.println(" ");
        Serial.print("    - Field Strength = ");
        Serial.print(Field_Strength);
        Serial.println(" uT");
        Serial.println("--------------------------------------------------------------------------");
        n_serv_index = 3;
      }
    }else if ((n_serv_index == n_iaq_service) && (n_serv_sel[n_iaq_service] == 1)) {
      if(checkAttributePropertyPermission(n_iaq_service, n_chars_index, 1)){
        Serial.print("> IAQ characteristic");
        Serial.print(n_chars_index);
        Serial.println("_read: ");
        Serial.print("    - Connection handle: ");
        Serial.println(device.connected_handle, HEX);
        Serial.print("    - Characteristic value attribute handle: ");
        Serial.println(device.service[n_iaq_service].chars[n_chars_index].chars.
    value_handle, HEX);
        ble.readValue(device.connected_handle,&device.
    service[n_iaq_service].chars[n_chars_index].chars);
      }
      if (n_chars_index < (n_chars[n_iaq_service]-1)){
        n_chars_index++;
      }else {
        n_chars_index = 0;
        Serial.println("--------------------------------------------------------------------------");
        Serial.println("- IAQ SENSING                                                  -");
        Serial.println("--------------------------------------------------------------------------");
        Serial.print("    - ECO2 - Carbon Dioxide = ");
        Serial.println(ECO2);
        Serial.print("    - TVOC - VOCS = ");
        Serial.println(TVOC);
        Serial.println("--------------------------------------------------------------------------");
        n_serv_index = 10;
      }
    }
  }
  // Restart timer.
  ble.setTimer(ts, 2000);
  ble.addTimer(ts);
}
```

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

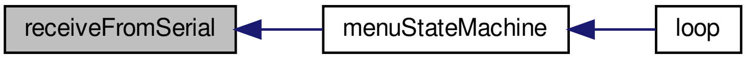
in	*char∗*	The message to show the user in the MENU.

Return values

uint8←↩ _t	serialValue The recibed user option value.

```
                                  {//receiveFromSerial

  uint8_t getSerialValue = 0;
  uint8_t serialValue = 0;

  Serial.print(message);
  while (!(Serial.available() > 0)) Particle.process(); // wait for serial port
  getSerialValue = Serial.read();
  serialValue = (getSerialValue <= '9')? (getSerialValue - '0'):(getSerialValue - 'A' + 10);
  return serialValue;
}
```

Here is the caller graph for this function:



4.1.1.31 reportCallback()

```
void reportCallback (
            advertisementReport_t * report )
```

Callback for scanning device.

Parameters

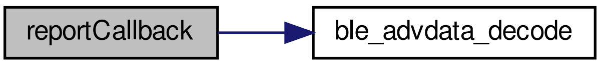
in	**report*	This function report the scanner response, and shearch to the Thunderboard Sense 2 device device to start the connection process.

Return values

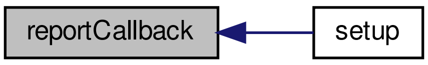
None	

```
                                              {
  uint8_t index;
  Serial.println("");
  Serial.println("* BLE scan callback: ");
  Serial.print("   - Advertising event type: ");
  Serial.println(report->advEventType, HEX);
  Serial.print("   - Peer device address type: ");
  Serial.println(report->peerAddrType, HEX);
  Serial.print("   - Peer device address: ");
  for (index = 0; index < 6; index++) {
    Serial.print(report->peerAddr[index], HEX);
    Serial.print(" ");
  }
  Serial.println(" ");
  Serial.print("   - RSSI: ");
  Serial.print(report->rssi, DEC);
  Serial.println(" dBm ");

  if (report->advEventType == BLE_GAP_ADV_TYPE_SCAN_RSP) {
    Serial.print("   - Scan response data packet (");
  }
  else {
    Serial.print("   - Advertising data packet(");
  }
  Serial.print(report->advDataLen, DEC);
  Serial.print(" Bytes): ");

  for (index = 0; index < report->advDataLen; index++) {
    Serial.print(report->advData[index], HEX);
    Serial.print(" ");
  }
  Serial.println(" ");

  uint8_t len;
  uint8_t adv_name[31];

  if (0x00 == ble_advdata_decode(0x09, report->advDataLen, report->advData, &len,
  adv_name)) {//Hacer una funcion conectarTh
    Serial.print("  The length of Complete Local Name : ");
    Serial.println(len, HEX);
    Serial.print("  The Complete Local Name is        : ");
    Serial.println((const char *)adv_name);

    if (0x00 == memcmp(adv_name, "Thunder Sense #02735", len)) {
      Serial.println("* Thunder Sense #02735 found");
      ble.stopScanning();
      device.addr_type = report->peerAddrType;
      memcpy(device.addr, report->peerAddr, 6);

      ble.connect(report->peerAddr, {BD_ADDR_TYPE_LE_PUBLIC});
    }
  }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



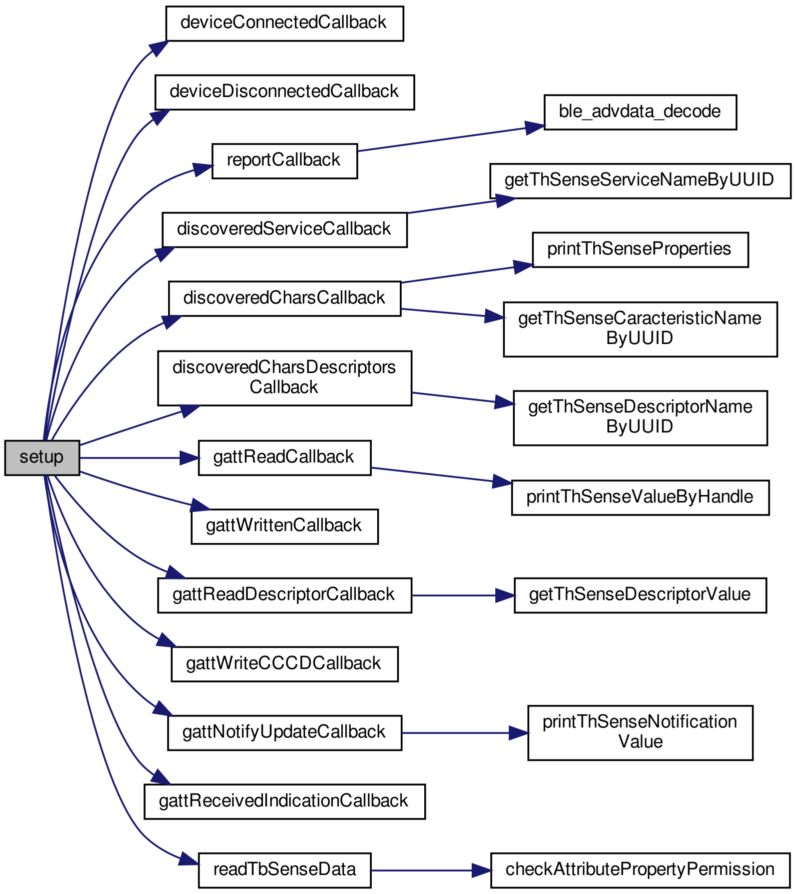
4.1.1.32 setup()

```
void setup ( )
```

Setup.

```
            {

 Serial.begin(115200);
 delay(5000);
 // Open debugger, must befor init()
 ble.debugLogger(true);
 ble.debugError(true);
 //ble.enablePacketLogger();

 // Initialize ble_stack
 ble.init();

 // Register callback functions
 ble.onConnectedCallback(deviceConnectedCallback);
 ble.onDisconnectedCallback(deviceDisconnectedCallback);
 ble.onScanReportCallback(reportCallback);
 ble.onServiceDiscoveredCallback(discoveredServiceCallback);
 ble.onCharacteristicDiscoveredCallback(discoveredCharsCallback);
 ble.onDescriptorDiscoveredCallback(discoveredCharsDescriptorsCallback);
 ble.onGattCharacteristicReadCallback(gattReadCallback);
 ble.onGattCharacteristicWrittenCallback(gattWrittenCallback);
 ble.onGattDescriptorReadCallback(gattReadDescriptorCallback);
 ble.onGattWriteClientCharacteristicConfigCallback(gattWriteCCCDCallback);
 ble.onGattNotifyUpdateCallback(gattNotifyUpdateCallback);
 ble.onGattIndicateUpdateCallback(gattReceivedIndicationCallback);

 // Set scan parameters
 ble.setScanParams(BLE_SCAN_TYPE, BLE_SCAN_INTERVAL,
BLE_SCAN_WINDOW);

 Serial.println("_____________________RedBear Duo_____________________:");
 ble.startScanning();
 Serial.println("");
 Serial.println("_____ BLE Central start scanning");
 delay(1000);
 Serial.println("");

 #if TIMER_DEBUG >= 1
   // set one-shot timer __setup
   read_tbsense_timer.process = &readTbSenseData;
   ble.setTimer(&read_tbsense_timer, 10000);
   ble.addTimer(&read_tbsense_timer);
 #endif
}
```

Here is the call graph for this function:


setup
deviceConnectedCallback
deviceDisconnectedCallback
reportCallback
ble_advdata_decode
discoveredServiceCallback
getThSenseServiceNameByUUID
discoveredCharsCallback
printThSenseProperties
getThSenseCaracteristicName ByUUID
discoveredCharsDescriptors Callback
getThSenseDescriptorName ByUUID
gattReadCallback
printThSenseValueByHandle
gattWrittenCallback
gattReadDescriptorCallback
getThSenseDescriptorValue
gattWriteCCCDCallback
gattNotifyUpdateCallback
printThSenseNotification Value
gattReceivedIndicationCallback
readTbSenseData
checkAttributePropertyPermission

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 caracteristics_name_gruped_by_service

```
char* caracteristics_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[NSERV_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 ATRIBUTE", (char*)"
DEVICE INFORMATION", (char*)" BATTERY", (char*)" ENVIRONMENTAL SENSING", (char*)" POWER SOUR←
CE", (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

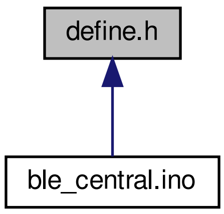
```
uint16_t TVOC = 0
```

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_CARACTERISTIC_VALUE = 0, BLE_CENTRAL_READ_DESCRIPTOR_VALUE, BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS, BLE_CENTRAL_ENABLE_DISABLE_INDICATI←
  ONS,
  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 accleration_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_Characrteristic0_Device_Name_uuid [16] = {0x00, 0x00, 0x2A, 0x00, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service0_Characrteristic1_Appearance_uuid [16] = {0x00, 0x00, 0x2A, 0x01, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service1_Characrteristic0_Service_Changed_uuid [16] = {0x00, 0x00, 0x2A, 0x05, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service2_Characrteristic0_Manufacturer_Name_uuid [16] = {0x00, 0x00, 0x2A, 0x29, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service2_Characrteristic1_Model_Number_uuid [16] = {0x00, 0x00, 0x2A, 0x24, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service2_Characrteristic2_Serial_Number_uuid [16] = {0x00, 0x00, 0x2A, 0x25, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service2_Characrteristic3_Hardware_Revision_uuid [16] = {0x00, 0x00, 0x2A, 0x27, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service2_Characrteristic4_Firmware_Revision_uuid [16] = {0x00, 0x00, 0x2A, 0x26, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service2_Characrteristic5_System_ID_uuid [16] = {0x00, 0x00, 0x2A, 0x23, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service3_Characrteristic0_Battery_Level_uuid [16] = {0x00, 0x00, 0x2A, 0x19, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service4_Characrteristic0_UV_Index_uuid [16] = {0x00, 0x00, 0x2A, 0x76, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service4_Characrteristic1_Pressure_uuid [16] = {0x00, 0x00, 0x2A, 0x6D, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service4_Characrteristic2_Temperature_uuid [16] = {0x00, 0x00, 0x2A, 0x6E, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service4_Characrteristic3_Humidity_uuid [16] = {0x00, 0x00, 0x2A, 0x6F, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static 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 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 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 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 uint8_t Service6_Characrteristic0_ECO2_uuid [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x01, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t Service6_Characrteristic1_TVOC_uuid [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x02, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t Service6_Characrteristic2_Control_Point_uuid [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x03, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t Service7_Characrteristic0_Buttons_uuid [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x01, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t Service7_Characrteristic1_Leds_uuid [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x02, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t Service7_Characrteristic2_RGB_Leds_uuid [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x03, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t Service7_Characrteristic3_Control_Point_uuid [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x04, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t Service8_Characrteristic0_Digital_1_uuid [16] = {0x00, 0x00, 0x2A, 0x56, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}

- static uint8_t Service8_Characrteristic1_Digital_2_uuid [16] = {0x00, 0x00, 0x2A, 0x56, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t Service9_Characrteristic0_Acceleration_uuid [16] = {0xC4, 0xC1, 0xF6, 0xE2, 0x4B, 0xE5, 0x11, 0xE5, 0x88, 0x5D, 0xFE, 0xFF, 0x81, 0x9C, 0xDC, 0x9F}
- static uint8_t Service9_Characrteristic1_Orientation_uuid [16] = {0xB7, 0xC4, 0xB6, 0x94, 0xBE, 0xE3, 0x45, 0xDD, 0xBA, 0x9F, 0xF3, 0xB5, 0xE9, 0x94, 0xF4, 0x9A}
- static uint8_t Service9_Characrteristic2_Control_Point_uuid [16] = {0x71, 0xE3, 0x0B, 0x8C, 0x41, 0x31, 0x47, 0x03, 0xB0, 0xA0, 0xB0, 0xBB, 0xBA, 0x75, 0x85, 0x6B}
- static uint8_t ServiceA_Characrteristic0_State_uuid [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x01, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}
- static uint8_t ServiceA_Characrteristic1_Field_Strength_uuid [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x02, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}
- static uint8_t ServiceA_Characrteristic2_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_CARACTERISTIC_VALUE	State BLE_CENTRAL_READ_CARACTERISTIC_VALUE
BLE_CENTRAL_READ_DESCRIPTOR_VALUE	State BLE_CENTRAL_READ_DESCRIPTOR_VALUE
BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATI↩ ONS	State BLE_CENTRAL_ENABLE_DISABLE_NOTIFI↩ CATIONS
BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS	State BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS
BLE_CENTRAL_WRITE	State BLE_CENTRAL_WRITE

```
                 {
    BLE_CENTRAL_READ_CARACTERISTIC_VALUE = 0,
    BLE_CENTRAL_READ_DESCRIPTOR_VALUE,
    BLE_CENTRAL_ENABLE_DISABLE_NOTIFICATIONS,
    BLE_CENTRAL_ENABLE_DISABLE_INDICATIONS,
    BLE_CENTRAL_WRITE
}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_Characrteristic0_Device_Name_uuid

```
uint8_t Service0_Characrteristic0_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_Characrteristic1_Appearance_uuid

```
uint8_t Service0_Characrteristic1_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_Characrteristic0_Service_Changed_uuid

```
uint8_t Service1_Characrteristic0_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_Characrteristic0_Manufacturer_Name_uuid

```
uint8_t Service2_Characrteristic0_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_Characrteristic1_Model_Number_uuid

```
uint8_t Service2_Characrteristic1_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_Characrteristic2_Serial_Number_uuid

```
uint8_t Service2_Characrteristic2_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_Characrteristic3_Hardware_Revision_uuid

```
uint8_t Service2_Characrteristic3_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_Characrteristic4_Firmware_Revision_uuid

```
uint8_t Service2_Characrteristic4_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_Characrteristic5_System_ID_uuid

```
uint8_t Service2_Characrteristic5_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_Characrteristic0_Battery_Level_uuid

```
uint8_t Service3_Characrteristic0_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_Characrteristic0_UV_Index_uuid

```
uint8_t Service4_Characrteristic0_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_Characrteristic1_Pressure_uuid

```
uint8_t Service4_Characrteristic1_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, 0x←
D9, 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_Characrteristic1_TVOC_uuid

```
uint8_t Service6_Characrteristic1_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_Characrteristic2_Control_Point_uuid

```
uint8_t Service6_Characrteristic2_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_Characrteristic0_Buttons_uuid

```
uint8_t Service7_Characrteristic0_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_Characrteristic1_Leds_uuid

```
uint8_t Service7_Characrteristic1_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_Characrteristic2_RGB_Leds_uuid

```
uint8_t Service7_Characrteristic2_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_Characrteristic3_Control_Point_uuid

```
uint8_t Service7_Characrteristic3_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_Characrteristic0_Digital_1_uuid

```
uint8_t Service8_Characrteristic0_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_Characrteristic1_Digital_2_uuid

```
uint8_t Service8_Characrteristic1_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_Characrteristic0_Acceleration_uuid

```
uint8_t Service9_Characrteristic0_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_Characrteristic1_Orientation_uuid

```
uint8_t Service9_Characrteristic1_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_Characrteristic2_Control_Point_uuid

```
uint8_t Service9_Characrteristic2_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_Characrteristic0_State_uuid

```
uint8_t ServiceA_Characrteristic0_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_Characrteristic1_Field_Strength_uuid

```
uint8_t ServiceA_Characrteristic1_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_Characrteristic2_Control_Point_uuid

```
uint8_t ServiceA_Characrteristic2_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