Integración con el dispositivo Waspmote

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

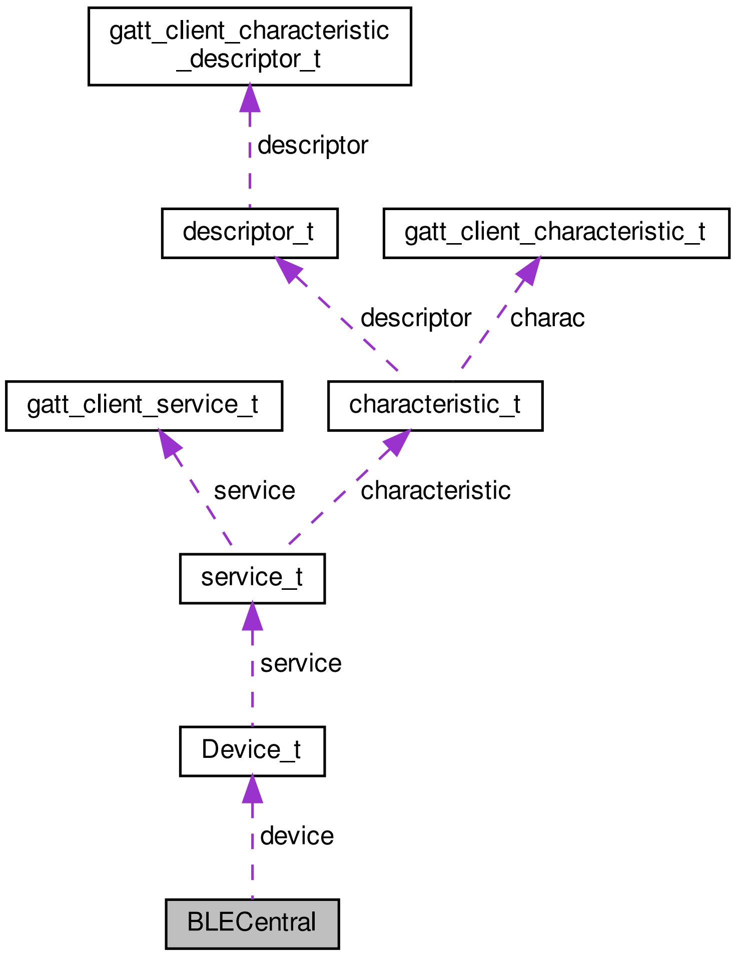
Class Documentation

3.1 BLECentral Class Reference

BLECentral Class.

```
#include <BLECentral.h>
```

Collaboration diagram for BLECentral:



Public Member Functions

- BLECentral ()

  public methods and attributes ////////////
- ∼BLECentral ()
- int8_t turnOnModule (uint8_t socket)
- void turnOffModule ()
- uint8_t bleAdvdataDecode (uint8_t type, uint8_t advdata_len, uint8_t ∗p_advdata, uint8_t ∗len, uint8_t ∗p←_field_data)
- uint8_t scanReport (char ∗nameToSearch)
- void configureScanner (uint8_t txPower, uint8_t discoverMode, uint16_t scanInterval, uint16_t scanWindow, uint8_t scanFilter)
- uint16_t startScanningDevice (char mac[ ])
- uint16_t startScanning (uint8_t time)
- uint16_t connect (char mac[ ])

- uint16_t connectWithSelectedParameters (char mac[ ], uint16_t conn_interval_min, uint16_t conn_interval↩
  _max, uint16_t timeout, uint16_t latency)
- uint16_t disconnect (uint8_t connectionHandle)
- uint8_t discoverServices ()
- uint8_t discoverCharacteristics ()
- uint8_t discoverDescriptors ()
- uint8_t discoverBLEProfile ()
- void printBLEProfile ()
- uint8_t ∗ readAttribute (uint8_t ∗uuid128)
- uint16_t writeAttribute (uint8_t connection, uint8_t ∗uuid128, uint8_t ∗data, uint8_t length)
- uint8_t enableNotification (uint8_t ∗uuid128)
- uint8_t ∗ receiveNotifications ()
- uint8_t getConnectionHandler ()
- uint8_t getConnectionStatus ()

Private Member Functions

- readByGroupCommand_t getDiscoverServiceGroupCommand ()

  private methods ///////////////////////////
- readByGroupCommand_t getDiscoverCharacteristicsCommand ()
- findInformationCommand_t getDiscoverDescriptorsCommand ()
- void newDevice ()
- void newService (uint8_t discoveredService[ ])
- void newCharacteristic (service_t ∗service, uint8_t discoveredCharacteristic[ ])
- void newDescriptor (characteristic_t ∗characteristic, uint8_t discoveredDescriptor[ ])
- void freeDevice ()
- uint16_t uuid16ToHandle (uint16_t uuid16)
- uint16_t uuid128ToHandle (uint8_t ∗uuid128)
- void service_uuid16_to_uuid128 (service_t ∗service)
- void characteristic_uuid16_to_uuid128 (characteristic_t ∗characteristic)

Private Attributes

- Device_t ∗ device

  Variable : Struct to save a BLE device and its data.

3.1.1 Detailed Description

BLECentral Class.

defines all the variables and functions used

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BLECentral()

```
BLECentral::BLECentral ( )
```

public methods and attributes ////////////

class constructor It does nothing

Parameters

void	

Returns

void

```
                         {

}
```

3.1.2.2 ∼BLECentral()

`BLECentral::∼BLECentral ( )`

class Destructor It does nothing

Parameters

void	

Returns

void

```
                          {
}
```

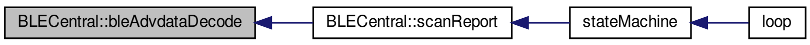
3.1.3 Member Function Documentation

3.1.3.1 bleAdvdataDecode()

```
uint8_t BLECentral::bleAdvdataDecode (
              uint8_t type,
              uint8_t advdata_len,
              uint8_t * p_advdata,
              uint8_t * len,
              uint8_t * p_field_data )
```

```
                          {
    uint8_t index = 1;
    uint8_t field_length, field_type;

    while (index < advdata_len ) {
        field_length = p_advdata[index];
        field_type = p_advdata[index + 1];
        USB.print(F("      - AVD/SR data decoding -> ad_type: "));
        USB.print(field_type, HEX);
        USB.print(F(", length: "));
        USB.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:



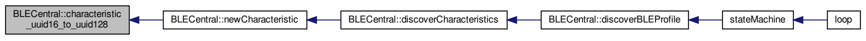
3.1.3.2 characteristic_uuid16_to_uuid128()

```
void BLECentral::characteristic_uuid16_to_uuid128 (
            characteristic_t * characteristic )  [private]
```

```
                                                                                  {

    switch(characteristic->charac.uuid16){
        case 0x2A00 :
            memcpy(characteristic->charac.uuid128,
      Service0_Characrteristic0_Device_Name_uuid, 16);
            break;

        case 0x2A01 :
            memcpy(characteristic->charac.uuid128,
      Service0_Characrteristic1_Appearance_uuid, 16);
            break;

        case 0x2A05 :
            memcpy(characteristic->charac.uuid128,
      Service1_Characrteristic0_Service_Changed_uuid, 16);
            break;

        case 0x2A29 :
            memcpy(characteristic->charac.uuid128,
      Service2_Characrteristic0_Manufacturer_Name_uuid, 16);
            break;

        case 0x2A24 :
            memcpy(characteristic->charac.uuid128,
      Service2_Characrteristic1_Model_Number_uuid, 16);
            break;

        case 0x2A25 :
            memcpy(characteristic->charac.uuid128,
      Service2_Characrteristic2_Serial_Number_uuid, 16);
            break;

        case 0x2A27 :
            memcpy(characteristic->charac.uuid128,
      Service2_Characrteristic3_Hardware_Revision_uuid, 16);
            break;

        case 0x2A26 :
            memcpy(characteristic->charac.uuid128,
      Service2_Characrteristic4_Firmware_Revision_uuid, 16);
```

```
            break;

        case 0x2A23 :
            memcpy(characteristic->charac.uuid128,
      Service2_Characrteristic5_System_ID_uuid, 16);
            break;

        case 0x2A19 :
            memcpy(characteristic->charac.uuid128,
      Service3_Characrteristic0_Battery_Level_uuid, 16);
            break;

        case 0x2A76 :
            memcpy(characteristic->charac.uuid128,
      Service4_Characrteristic0_UV_Index_uuid, 16);
            break;

        case 0x2A6D :
            memcpy(characteristic->charac.uuid128,
      Service4_Characrteristic1_Pressure_uuid, 16);
            break;

        case 0x2A6E :
            memcpy(characteristic->charac.uuid128,
      Service4_Characrteristic2_Temperature_uuid, 16);
            break;

        case 0x2A6F :
            memcpy(characteristic->charac.uuid128,
      Service4_Characrteristic3_Humidity_uuid, 16);
            break;

        case 0x2A56 :
            memcpy(characteristic->charac.uuid128,
      Service8_Characrteristic0_Digital_1_uuid, 16);
            break;
    }
}
```

Here is the caller graph for this function:

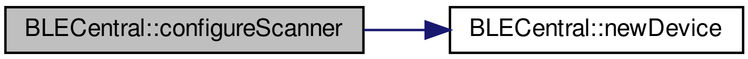

BLECentral::characteristic_uuid16_to_uuid128
BLECentral::newCharacteristic
BLECentral::discoverCharacteristics
BLECentral::discoverBLEProfile
stateMachine
loop

3.1.3.3 configureScanner()

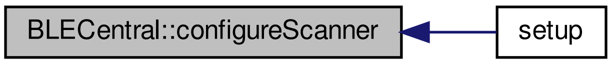
```
void BLECentral::configureScanner (
            uint8_t txPower,
            uint8_t discoverMode,
            uint16_t scanInterval,
            uint16_t scanWindow,
            uint8_t scanFilter )
```

```
                                             {
  BLE.setDiscoverMode(discoverMode);
    BLE.setTXPower(txPower);
    BLE.setScanningParameters(scanInterval, scanWindow, scanningFilter );
  newDevice();//create Device_t struct to store device relate data.
  #if DEBUG >= 1

      BLE.getScanningParameters();
      USB.println(F("____________Central is configure and redy to start____________"));
  #endif

}
```

Here is the call graph for this function:



Here is the caller graph for this function:

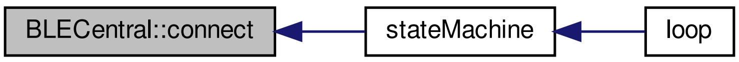


3.1.3.4 connect()

```
uint16_t BLECentral::connect (
              char mac[] )
```

```
                                      {

    uint8_t response;
    USB.println(F("__________Connecting... "));
    response = BLE.connectDirect(mac);
    if (response == 1){
        USB.println(F("__________Connected"));
        USB.print(F("             -connection_handle: "));
        USB.println(BLE.connection_handle, DEC);
        USB.print(F(""));
    }else if (response == 0){
        USB.println(F("Invalid parameters"));
    }else{
        USB.print(F("Connection ERROR = "));
        USB.println(response, DEC);
    }
    return response;
}
```

Here is the caller graph for this function:



3.1.3.5 connectWithSelectedParameters()

```
uint16_t BLECentral::connectWithSelectedParameters (
              char mac[],
              uint16_t conn_interval_min,
              uint16_t conn_interval_max,
              uint16_t timeout,
              uint16_t latency )
```

```
                                    {

    uint16_t response = 0;

    USB.println(F("_________Connecting... "));
    response = BLE.connectDirect(mac, conn_interval_min, conn_interval_max, timeout, latency);//default connectDirect(BLEAddress, 60, 76, 100, 0);
    if (response == 1){
        USB.println(F("_________Connected"));
        USB.print(F("              -connection_handle: "));
        USB.println(BLE.connection_handle, DEC);
        USB.print(F(""));
        return 1;
    }else{
        USB.println(F("NOT Connected"));
        return 0;
    }

}
```

3.1.3.6 disconnect()

```
uint16_t BLECentral::disconnect (
              uint8_t connectionHandle )
```

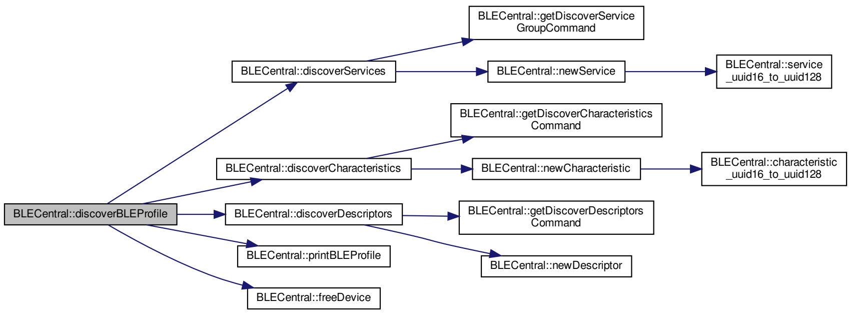
```
                                                        {
    uint16_t response = 0;
    response =  BLE.disconnect(connectionHandle);
    if (response == 0){
        USB.println(F("_________Disconnected"));
    }else if (response == 1){
        USB.print(F("Connection handle is not right"));
    }else{
        USB.print(F("Disconnect, Error = "));
        USB.println(response, HEX);
    }
    return response;
}
```

3.1.3.7 discoverBLEProfile()

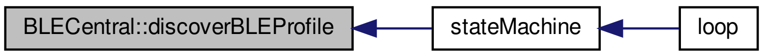
```
uint8_t BLECentral::discoverBLEProfile ( )
```

```
                                       {

    if(discoverServices()){
        if(discoverCharacteristics()){
            if(discoverDescriptors()){
                printBLEProfile();
                return 1;
            }
        }
    }
    freeDevice();
    return 0;

}
```

Here is the call graph for this function:


BLECentral::discoverBLEProfile
BLECentral::discoverServices
BLECentral::getDiscoverService GroupCommand
BLECentral::newService
BLECentral::service _uuid16_to_uuid128
BLECentral::getDiscoverCharacteristics Command
BLECentral::discoverCharacteristics
BLECentral::newCharacteristic
BLECentral::characteristic _uuid16_to_uuid128
BLECentral::discoverDescriptors
BLECentral::getDiscoverDescriptors Command
BLECentral::printBLEProfile
BLECentral::newDescriptor
BLECentral::freeDevice

Here is the caller graph for this function:

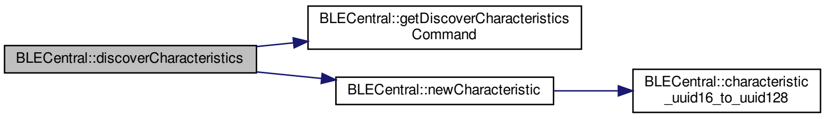


3.1.3.8 discoverCharacteristics()

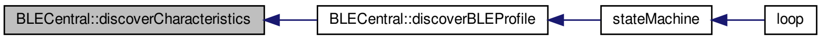
```
uint8_t BLECentral::discoverCharacteristics ( )
```

```
                                           {

    uint16_t event;
    readByGroupCommand_t command;
    command = getDiscoverCharacteristicsCommand();
    USB.println(F("_________Discovering Characteristics... "));
    for(uint8_t numSer = 0; numSer < device->numberOfServices; numSer++){
        command.startFirstAttributeHandle = ((device->service[numSer].service.start_group_handle)+1);
        command.endLastAttributeHandle = ((device->service[numSer].service.end_group_handle)-1);
        BLE.sendCommand((uint8_t *)&command, command.t_length+1);
        BLE.readCommandAnswer();
        event = BLE_EVENT_ATTCLIENT_ATTRIBUTE_VALUE;
        while(event == BLE_EVENT_ATTCLIENT_ATTRIBUTE_VALUE ){
            event = BLE.waitEvent(1000);
            if(event == BLE_EVENT_ATTCLIENT_ATTRIBUTE_VALUE){
                newCharacteristic(&device->service[numSer], BLE.event);
            }else if(event == 0){
                USB.println(F("The connection to the peripheral device has been disconnected"));
                return 0;
            }
        }
    }
    USB.println(F("_________Discovering Characteristics completed"));
    USB.println(F(""));
    return 1;
}
```

Here is the call graph for this function:


BLECentral::discoverCharacteristics
BLECentral::getDiscoverCharacteristicsCommand
BLECentral::newCharacteristic
BLECentral::characteristic_uuid16_to_uuid128

Here is the caller graph for this function:

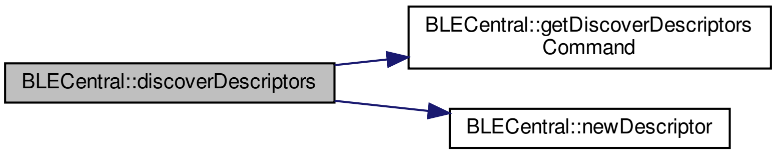

BLECentral::discoverCharacteristics
BLECentral::discoverBLEProfile
stateMachine
loop

3.1.3.9 discoverDescriptors()

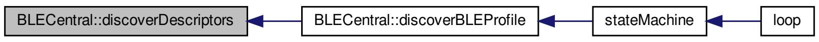
```
uint8_t BLECentral::discoverDescriptors ( )
```

```
                                     {
uint8_t numSer, numCar;
findInformationCommand_t command;
command = getDiscoverDescriptorsCommand();
uint16_t servEndHandle, carcValueHandle, nexCarcStarHandle;
USB.println(F("_________Discovering Descriptors... "));
for(numSer = 0; numSer < device->numberOfServices; numSer++){
  servEndHandle = device->service[numSer].service.end_group_handle;
  for(numCar = 0; numCar < device->service[numSer].numberOfCharacteristics; numCar++){
    carcValueHandle = device->service[numSer].characteristic[numCar].charac.value_handle;
    if( (numCar+1) < device->service[numSer].numberOfCharacteristics){
      nexCarcStarHandle = device->service[numSer].characteristic[numCar+1].charac.start_handle;
      carcValueHandle = carcValueHandle+1;
      while((carcValueHandle)!= nexCarcStarHandle){
        command.startFirstAttributeHandle = carcValueHandle ;
        command.endLastAttributeHandle = carcValueHandle;
        BLE.sendCommand((uint8_t *)&command, command.t_length+1);
        BLE.readCommandAnswer();
        BLE.waitEvent(1000);
        newDescriptor(&device->service[numSer].characteristic[numCar],BLE.event);
        carcValueHandle = carcValueHandle+1;
      }
    }else if(servEndHandle != 0xFFFF){
      while((carcValueHandle)!= servEndHandle){
        carcValueHandle = carcValueHandle+1;
        command.startFirstAttributeHandle = carcValueHandle ;
        command.endLastAttributeHandle = carcValueHandle;
        BLE.sendCommand((uint8_t *)&command, command.t_length+1);
        BLE.readCommandAnswer();
        BLE.waitEvent(1000);
        newDescriptor(&device->service[numSer].characteristic[numCar],BLE.event);
      }
    }else{
        uint8_t evento;
        evento = BLE_EVENT_ATTCLIENT_FIND_INFORMATION_FOUND;
        while(evento == BLE_EVENT_ATTCLIENT_FIND_INFORMATION_FOUND){
                carcValueHandle = carcValueHandle+1;
                command.startFirstAttributeHandle = carcValueHandle ;
                command.endLastAttributeHandle = carcValueHandle;
                BLE.sendCommand((uint8_t *)&command, command.t_length+1);
                BLE.readCommandAnswer();
                evento = BLE.waitEvent(1000);
                if(evento == BLE_EVENT_ATTCLIENT_FIND_INFORMATION_FOUND){
                  newDescriptor(&device->service[numSer].
    characteristic[numCar],BLE.event);
                }else if(evento == BLE_EVENT_ATTCLIENT_PROCEDURE_COMPLETED){
                  device->service[numSer].service.
    end_group_handle = carcValueHandle -1;
                }
            }
        }
      }
    }
    USB.println(F("_________Discovering Descriptors completed"));
    USB.println(F(""));
    return 1;
}
```

Here is the call graph for this function:


BLECentral::discoverDescriptors
BLECentral::getDiscoverDescriptorsCommand
BLECentral::newDescriptor

Here is the caller graph for this function:


BLECentral::discoverDescriptors
BLECentral::discoverBLEProfile
stateMachine
loop

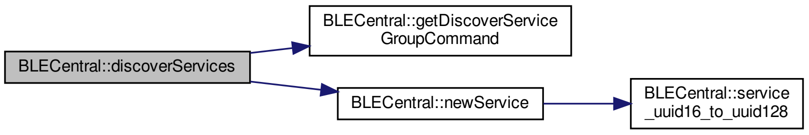
3.1.3.10 discoverServices()

```
uint8_t BLECentral::discoverServices ( )
```

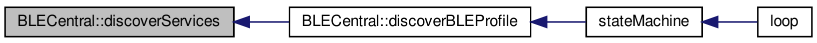
```
                                    {

      uint16_t event = 0;
      readByGroupCommand_t command;
      command = getDiscoverServiceGroupCommand();
      USB.println(F("_________Discovering Services... "));
      BLE.sendCommand((uint8_t *)&command, command.t_length+1);
      BLE.readCommandAnswer();
      while(event != BLE_EVENT_ATTCLIENT_PROCEDURE_COMPLETED){
          event = BLE.waitEvent(1000);
          if(event == BLE_EVENT_ATTCLIENT_GROUP_FOUND){
              newService( BLE.event);
          }else if(event == 0){//No hay evento
              USB.println(F("The connection to the peripheral device has been disconnected"));
```

```
            return 0;
        }
    }
    USB.print(F("__________Discovering Services completed ("));
    USB.print(device->numberOfServices, DEC);
    USB.println(F(")"));
    return 1;
}
```

Here is the call graph for this function:


BLECentral::discoverServices
BLECentral::getDiscoverService GroupCommand
BLECentral::newService
BLECentral::service _uuid16_to_uuid128

Here is the caller graph for this function:

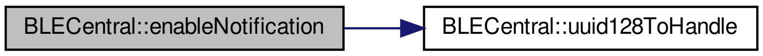


3.1.3.11 enableNotification()

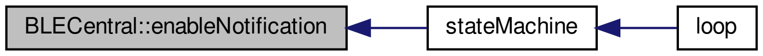
```
uint8_t BLECentral::enableNotification (
            uint8_t * uuid128 )
```

```
                                                         {

    uint16_t response;
    char notify[2] = "1";
    response = BLE.attributeWrite(BLE.connection_handle, (uuid128ToHandle(uuid128) + 1),
  notify);
    if (response == 0){
      USB.println(F("____________Notification enable"));
      USB.print(F("  -For UUID128: "));
      for(uint8_t i=0; i<16; i++){
        USB.print(uuid128[i], HEX);
        USB.print(" ");
      }
    USB.println(F(""));
        return 1;
    }else{
        USB.println(F("____________Failed subscribing"));
        USB.println(F(""));
        return 0;
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:

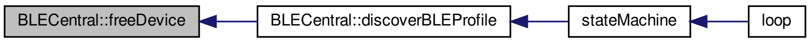


3.1.3.12 freeDevice()

```
void BLECentral::freeDevice ( )  [private]
```

```
                           {
    #if DEBUG >= 1

        USB.print(F("Free Memory(Before freeDevice):"));
        USB.println(freeMemory());
    #endif

    uint8_t numSer;
    uint8_t numCar;
    for(numSer = 0; numSer < device->numberOfServices; numSer++){
        for(numCar = 0; numCar < device->service[numSer].
numberOfCharacteristics; numCar++){
            if(device->service[numSer].characteristic[numCar].
descriptor != NULL){
                free(device->service[numSer].characteristic[numCar].
descriptor);
            }
        }
    }
    for(uint8_t numSer = 0; numSer < device->numberOfServices; numSer++){
        if(device->service[numSer].characteristic != NULL){
                free(device->service[numSer].characteristic);
        }
    }
    if(device->service != NULL){
        free(device->service);
         device->numberOfServices = 0;
         device->service = NULL;
    }
    //~ free(device);
    #if DEBUG >= 1

        USB.print(F("Free Memory(After freeDevice):"));
        USB.println(freeMemory());
    #endif

}
```

Here is the caller graph for this function:



3.1.3.13 getConnectionHandler()

```
uint8_t BLECentral::getConnectionHandler ( )
```

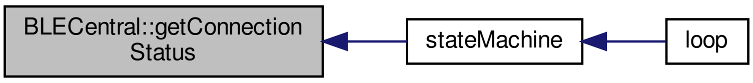
```
                                     {
    return BLE.connection_handle;
}
```

3.1.3.14 getConnectionStatus()

```
uint8_t BLECentral::getConnectionStatus ( )
```

```
                                    {//idea limpiar device aqui usuario no deve tocarlo
    uint8_t status;

    status =  BLE.getStatus(BLE.connection_handle);
    if( status == 0 ) { //Se ha desconectado
       // central.limpiarPerfil();
       USB.print(F("The connection has been disconnected, satus: "));
       USB.println(status, DEC);
    }else if (status == 1){
       USB.print(F("The device is connected, satus: "));
       USB.println(status, DEC);
    }else{
       USB.print(F("The module does not response, satus: "));
       USB.println(status, DEC);
    }
    return status;
}
```

Here is the caller graph for this function:

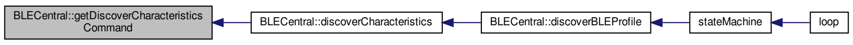


3.1.3.15 getDiscoverCharacteristicsCommand()

```
readByGroupCommand_t BLECentral::getDiscoverCharacteristicsCommand ( )  [private]
```

```
                                                                  {

    readByGroupCommand_t command;
    command.t_length = 12;
    command.messageType = 0;
    command.payloadLenght = 8;
    command.classID = 4;
    command.commandID = 2;
    command.Connectionhandle = BLE.connection_handle;
    command.startFirstAttributeHandle = 0;
    command.endLastAttributeHandle = 0;
    command.uuidLenght = 2;
    command.uuid = 0x2803;
    return command;
}
```

Here is the caller graph for this function:

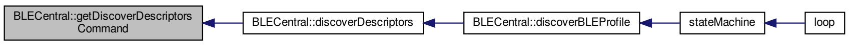

BLECentral::getDiscoverCharacteristics Command
BLECentral::discoverCharacteristics
BLECentral::discoverBLEProfile
stateMachine
loop

3.1.3.16 getDiscoverDescriptorsCommand()

```
findInformationCommand_t BLECentral::getDiscoverDescriptorsCommand ( )  [private]
```

```
                                                                  {
        /*//~ Byte Type Name Description

        //~ 0 0x00 hilen Message type: command

        //~ 1 0x05 lolen Minimum payload length

        //~ 2 0x04 class Message class: Attribute Client

        //~ 3 0x03 method Message ID

        //~ 4 uint8 connection Connection handle

        //~ 5 - 6 uint16 start First attribute handle

        //~ 7 - 8 uint16 end Last attribute handle

        */
        findInformationCommand_t command;
        command.t_length = 9;
        command.messageType = 0;
        command.payloadLenght = 5;
        command.classID = 4;
        command.commandID = 3;
        command.Connectionhandle = BLE.connection_handle;
        return command;
}
```

Here is the caller graph for this function:


BLECentral::getDiscoverDescriptors Command
BLECentral::discoverDescriptors
BLECentral::discoverBLEProfile
stateMachine
loop

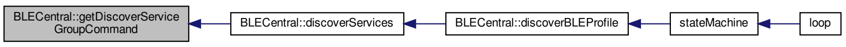
3.1.3.17 getDiscoverServiceGroupCommand()

```
readByGroupCommand_t BLECentral::getDiscoverServiceGroupCommand ( )  [private]
```

private methods ////////////////////////////

```
                                                                {
    /* Byte Type Name Description

    //~ 0 0x00 hilen Message type: command

    //~ 1 0x06 lolen Minimum payload length

    //~ 2 0x04 class Message class: Attribute Client

    //~ 3 0x01 method Message ID

    //~ 4 uint8 connection Connection Handle

    //~ 5 - 6 uint16 start First requested handle number

    //~ 7 - 8 uint16 end Last requested handle number

    //~ 9 uint8array uuid Group UUID to find

    */
    readByGroupCommand_t command;
    command.t_length = 12;
    command.messageType = 0;
    command.payloadLenght = 8;
    command.classID = 4;
    command.commandID = 1;
    command.Connectionhandle = BLE.connection_handle;
    command.startFirstAttributeHandle = 0x0001;
    command.endLastAttributeHandle = 0xFFFF;
    command.uuidLenght = 2;
    command.uuid = 0x2800;
    return command;
}
```

Here is the caller graph for this function:

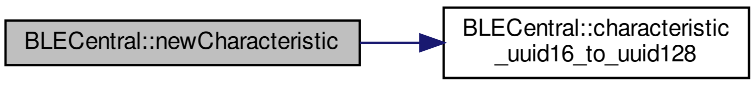

BLECentral::getDiscoverService GroupCommand
BLECentral::discoverServices
BLECentral::discoverBLEProfile
stateMachine
loop

3.1.3.18 newCharacteristic()

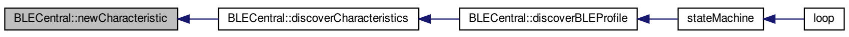
```
void BLECentral::newCharacteristic (
            service_t * service,
            uint8_t discoveredCharacteristic[ ] )  [private]
```

```
                                                                                    {
    /* Attribute Value--attclient

    //~ 0 0x80 hilen Message type: event

    //~ 1 0x05 lolen Minimum payload length

    //~ 2 0x04 class Message class: Attribute Client

    //~ 3 0x05 method Message ID

    //~ 4 uint8 connection Connection handle
    //~ 5 - 6 uint16 atthandle Attribute handle

    //~ 7 uint8 type Attribute type

    //~ 8 uint8array value Attribute value (data)//yo de aqui

    */
    uint8_t contador = 0;
    uint8_t posicion = 0;
    service->numberOfCharacteristics++;
    characteristic_t* tmp = (characteristic_t*)malloc(sizeof(
  characteristic_t)*service->numberOfCharacteristics);
    characteristic_t* basura;
    for(contador = 0; contador <(service->numberOfCharacteristics-1); contador++){
        tmp[contador] = service->characteristic[contador];
    }
    tmp[contador].numberOfDescriptors = 0;
    tmp[contador].descriptor = NULL;
    tmp[contador].charac.start_handle = ((uint16_t)discoveredCharacteristic[6] << 8) |
  discoveredCharacteristic[5];
    tmp[contador].charac.value_handle = ((uint16_t)discoveredCharacteristic[11] << 8) |
  discoveredCharacteristic[10];
    tmp[contador].charac.properties = ((uint8_t)discoveredCharacteristic[9]);
    tmp[contador].charac.uuid16 = 0;
    memset(&tmp[contador].charac.uuid128, 0x00, 16 );
    posicion = ((discoveredCharacteristic[8]) - 3);
    if(posicion == 2){
        tmp[contador].charac.uuid16 = ((uint16_t)discoveredCharacteristic[13] << 8) |
  discoveredCharacteristic[12];
        characteristic_uuid16_to_uuid128(&tmp[contador]);
    }else{
        for(int s = 0; s < 16 ; s++){
            tmp[contador].charac.uuid128[s]= discoveredCharacteristic[27-s];
        }
    }
    basura = service->characteristic;
    service->characteristic = tmp;
    if(basura!= NULL){
        free(basura);
        tmp = NULL;
    }
}
```

Here is the call graph for this function:


BLECentral::newCharacteristic
BLECentral::characteristic_uuid16_to_uuid128

Here is the caller graph for this function:

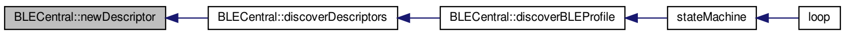

BLECentral::newCharacteristic
BLECentral::discoverCharacteristics
BLECentral::discoverBLEProfile
stateMachine
loop

3.1.3.19 newDescriptor()

```
void BLECentral::newDescriptor (
            characteristic_t * characteristic,
            uint8_t discoveredDescriptor[] )  [private]
```

```
                                                                                               {
    /*  Find Information Found--attclient

    //~ Byte Type Name Description

    //~ 0 0x80 hilen Message type: event

    //~ 1 0x04 lolen Minimum payload length

    //~ 2 0x04 class Message class: Attribute Client

    //~ 3 0x04 method Message ID

    //~ 4 uint8 connection Connection handle

    //~ 5 - 6 uint16 chrhandle Characteristics handle

    //~ 7 uint8array uuid Characteristics type (UUID)

    */
    uint16_t flag1;
    uint8_t contador = 0;
    characteristic->numberOfDescriptors++;
    descriptor_t* tmp = (descriptor_t*)malloc(sizeof(
  descriptor_t)*characteristic->numberOfDescriptors);
    descriptor_t* basura;
    for(contador = 0; contador <(characteristic->numberOfDescriptors-1); contador++){
        tmp[contador] = characteristic->descriptor[contador];
    }
    tmp[contador].descriptor.handle = ((uint16_t)discoveredDescriptor[6] << 8) |
  discoveredDescriptor[5];
    tmp[contador].descriptor.uuid16 = ((uint16_t)discoveredDescriptor[9] << 8) |
  discoveredDescriptor[8];

    basura = characteristic->descriptor;
    characteristic->descriptor = tmp;
    if(basura!= NULL){
        free(basura);
        tmp = NULL;
    }
}
```

Here is the caller graph for this function:

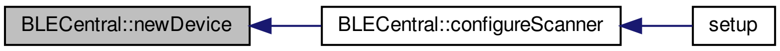

BLECentral::newDescriptor
BLECentral::discoverDescriptors
BLECentral::discoverBLEProfile
stateMachine
loop

3.1.3.20 newDevice()

```
void BLECentral::newDevice ( )  [private]
```

```
                          {
    #if DEBUG >= 1

        USB.print(F("Free Memory(Before New Device):"));
        USB.println(freeMemory());
    #endif

    device = (Device_t*)malloc(sizeof(Device_t));
    device->numberOfServices = 0;
    device->service = NULL;
    USB.println(F("____________Storage to manage new device started____________"));

    #if DEBUG >= 1

        USB.print(F("Free Memory(After New Device):"));
        USB.println(freeMemory());
    #endif

}
```

Here is the caller graph for this function:

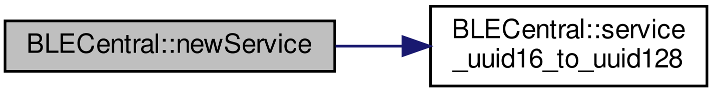


3.1.3.21 newService()

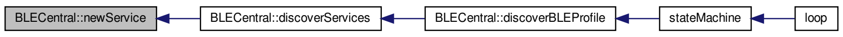
```
void BLECentral::newService (
             uint8_t discoveredService[ ] )  [private]
```

```
                                                    {
    /*  Group Found--attclient

    //~ Byte Type Name Description

    //~ 0 0x80 hilen Message type: event

    //~ 1 0x06 lolen Minimum payload length

    //~ 2 0x04 class Message class: Attribute Client

    //~ 3 0x02 method Message ID

    //~ 4 uint8 connection Connection handle

    //~ 5 - 6 uint16 start Starting handle

    //~ 7 - 8 uint16 end Ending handle

    //~ Note: "end" is a reserved word and in BGScript so "end" cannot be used as

    //~ such.

    //~ 9 uint8array uuid UUID of a service

    //~ Length is 0 if no services are found.

    */
    int contador = 0;
    device->numberOfServices++;
    service_t *tmp = (service_t*)malloc(sizeof(service_t)*
  device->numberOfServices);
    service_t *basura;
    for(contador = 0; contador <(device->numberOfServices-1); contador++){
        tmp[contador] = device->service[contador];
    }
    tmp[contador].numberOfCharacteristics = 0;
    tmp[contador].characteristic = NULL;
    tmp[contador].service.start_group_handle = ((uint16_t)discoveredService[6] <<
  8) | discoveredService[5];
    tmp[contador].service.end_group_handle = ((uint16_t)discoveredService[8] << 8) |
   discoveredService[7];
    tmp[contador].service.uuid16 = 0;
    memset(&tmp[contador].service.uuid128, 0x00, 16 );
    if(discoveredService[9]== 2){
        tmp[contador].service.uuid16 = ((uint16_t)discoveredService[11] << 8) |
  discoveredService[10];
        service_uuid16_to_uuid128(&tmp[contador]);
    }else{
        for(uint8_t indice = 0; indice < 16 ; indice++){
            tmp[contador].service.uuid128[indice] = ((uint16_t)discoveredService[25-indice]);
        }
    }
    basura = device->service;
    device->service = tmp;
    if(basura!= NULL){
        free(basura);
        tmp = NULL;
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:

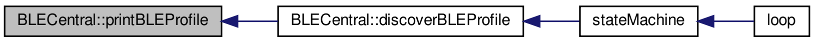


3.1.3.22 printBLEProfile()

```
void BLECentral::printBLEProfile ( )
```

```
                                {
    uint8_t numSer, numCar, numDesc;
    USB.println(F("____________________BLE Profile____________________"));
    USB.println(F(""));
    for(numSer = 0; numSer < device->numberOfServices; numSer++){
        USB.print(F("* Service number "));
        USB.println( numSer, HEX );
        USB.print(F("   - Service start handle: "));
        USB.println(device->service[numSer].service.
start_group_handle, HEX);
        USB.print(F("   - Service end handle: "));
        USB.println(device->service[numSer].service.
end_group_handle, HEX);
        USB.print(F("   - Service uuid16: "));
        USB.println(device->service[numSer].service.uuid16, HEX);
        USB.print(F("   - Service uuid128: "));
        for(int p = 0; p < 16; p++){
            USB.print(device->service[numSer].service.
uuid128[p],HEX);
            USB.print(" ");
        }
        USB.println("");
        USB.print(F("   * Service number of characteristics: "));
        USB.print(device->service[numSer].numberOfCharacteristics, DEC)
;
        USB.println(F(""));
        for(numCar = 0; numCar < device->service[numSer].
numberOfCharacteristics; numCar++){
            USB.print(F("* Service "));
            USB.print( numSer,HEX );
            USB.print(F("   - Characteristic: "));
            USB.println( numCar,HEX );
            USB.print(F("   - Characteristic start handle: "));
            USB.print(device->service[numSer].characteristic[numCar].
charac.start_handle,HEX);
            USB.println("");
            USB.print(F("   - Characteristic value handle: "));
            USB.print(device->service[numSer].characteristic[numCar].
charac.value_handle ,HEX);
            USB.println("");
            USB.print(F("   - Characteristic properties: "));
            USB.print(device->service[numSer].characteristic[numCar].
charac.properties,HEX);
            USB.println("");
            USB.print(F("   - Characteristic uuid16: "));
            USB.print( device->service[numSer].characteristic[numCar].
charac.uuid16,HEX);
            USB.println(F(""));
            USB.print(F("   - Characteristic uuid128: "));
            for(int p = 0; p < 16 ; p++){
            USB.print(device->service[numSer].characteristic[numCar].
charac.uuid128[p],HEX);
            USB.print(" ");
            }
            USB.println(" ");
            for(numDesc = 0; numDesc < device->service[numSer].
characteristic[numCar].numberOfDescriptors; numDesc++){
                USB.print(F("   - Characteristic: "));
                USB.print( numCar,HEX );
                USB.print(F("   - Descriptor: "));
                USB.println( numDesc, HEX );
                USB.print(F("   - Descriptor handle: "));
                USB.println( device->service[numSer].characteristic[numCar].
descriptor[numDesc].descriptor.handle,HEX );
                USB.print(F("   - Descriptor uuid16: "));
                USB.println( device->service[numSer].characteristic[numCar].
descriptor[numDesc].descriptor.uuid16, HEX );
            }
        }
    }
    #if DEBUG >= 1

        USB.print(F("Free Memory(After creating the BLE profile):"));
        USB.println(freeMemory());
        USB.println(F(" "));
    #endif

}
```

Here is the caller graph for this function:

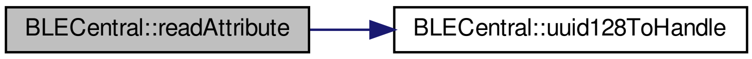


3.1.3.23 readAttribute()

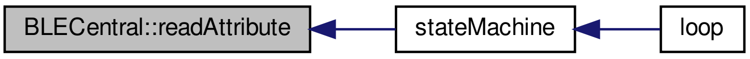
```
uint8_t * BLECentral::readAttribute (
              uint8_t * uuid128 )
```

```
                                                    {
    USB.println(F("BLE Central read Attribute "));
    USB.print(F("  -UUID128: "));
    for(uint8_t i=0; i<16; i++){
      USB.print(uuid128[i], HEX);
      USB.print(" ");
    }
    USB.println(F(""));
    USB.print(F("  -Handle: "));
    USB.println(uuid128ToHandle(uuid128), HEX);
    USB.println(F(""));//probado
    BLE.attributeRead(BLE.connection_handle, uuid128ToHandle(uuid128));
    return BLE.attributeValue;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:

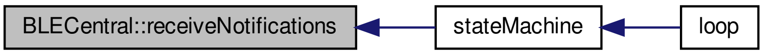


3.1.3.24 receiveNotifications()

```
uint8_t * BLECentral::receiveNotifications ( )
```

```
                                          {
    uint16_t event;
    uint16_t handler;
    USB.println(F("Waiting events..."));
    event = BLE.waitEvent(1000);
    if (event == BLE_EVENT_ATTCLIENT_ATTRIBUTE_VALUE){
        USB.println(F("Notification received"));
            /* attribute value event structure:

             Field:   | Message type | Payload| Msg Class | Method |  Connection | att handle | att type | value |

             Length:  |       1      |    1   |     1     |    1   |      1      |      2     |     8    |   n   |

             Example: |      80      |   05   |     04    |   05   |     00      |   2c 00    |     x    |   n   |*/
        handler = ((uint16_t)BLE.event[6] << 8) | BLE.event[5];
        USB.print(F("  -Attribute with handler "));
        USB.print(handler, DEC);
        USB.println(F(" has changed "));
        USB.print(F("  -Attribute value: "));
        BLE.event[0] = BLE.event[8];
        for(uint8_t i = 0; i < BLE.event[8]; i++){
        USB.printHex(BLE.event[i+9]);
            BLE.event[i+1] = BLE.event[i+9];
        }
        USB.println(F(""));
        return BLE.event;
    }

}
```

Here is the caller graph for this function:

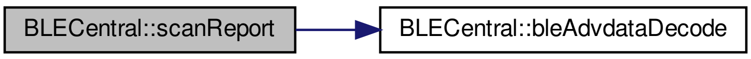


3.1.3.25 scanReport()

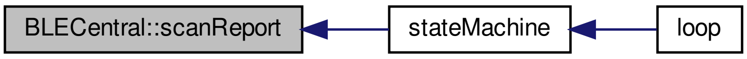
```
uint8_t BLECentral::scanReport (
              char * nameToSearch )
```

```
                                                    {
    USB.println(F(""));
    USB.println(F("* BLE scan report: "));
    USB.print(F("   - Peer device address: "));
    Utils.hex2str(BLE.BLEDev.mac, device->mac, 6);
    USB.println(device->mac);
    USB.print(F("   - RSSI: "));
    USB.print(BLE.BLEDev.rssi, DEC);
    USB.println(F(" dBm "));
    USB.print(F("   - Advertising data packet("));
    USB.print(BLE.BLEDev.advData[0], DEC);
    USB.print(F(" Bytes): "));
    for (int index = 1; index < BLE.BLEDev.advData[0]; index++) {
        USB.print(BLE.BLEDev.advData[index], HEX);
        USB.print(F(" "));
    }
    USB.println(F(" "));

    uint8_t len;
    uint8_t adv_name[31];

    if (0x00 == bleAdvdataDecode(0x09, BLE.BLEDev.advData[0], BLE.BLEDev.advData, &len,
      adv_name)) {
        USB.print(F("  The length of Complete Local Name : "));
        USB.println(len, HEX);
        USB.print(F("  The Complete Local Name is        : "));
        adv_name[len]= 0;
        USB.println((const char *)adv_name);
        USB.println(F(""));
        if (0x00 == memcmp(adv_name, nameToSearch, len)) {
            USB.println(F("* Thunder Sense #02735 found"));
            return 1;
        }
    }
    return 0;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:

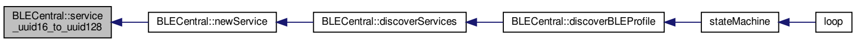


3.1.3.26 service_uuid16_to_uuid128()

```
void BLECentral::service_uuid16_to_uuid128 (
              service_t * service )  [private]
```

```
                                                              {
    switch(service->service.uuid16){
        case 0x1800 :
            memcpy(service->service.uuid128, generic_access_service_uuid, 16);
            break;

        case 0x1801 :
            memcpy(service->service.uuid128, generic_attribute_service_uuid, 16);
            break;

        case 0x180A :
            memcpy(service->service.uuid128, device_information_service_uuid, 16);
            break;

        case 0x180F :
            memcpy(service->service.uuid128, battery_service_uuid, 16);
            break;

        case 0x181A :
            memcpy(service->service.uuid128, environmental_sensing_service_uuid, 16);
            break;

        case 0x1815 :
            memcpy(service->service.uuid128, automation_io_service_uuid, 16);
            break;
    }
}
```

Here is the caller graph for this function:


BLECentral::service _uuid16_to_uuid128
BLECentral::newService
BLECentral::discoverServices
BLECentral::discoverBLEProfile
stateMachine
loop

3.1.3.27 startScanning()

```
uint16_t BLECentral::startScanning (
            uint8_t time )
```

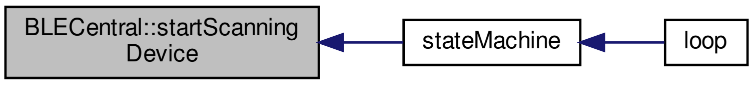
```
                                             {
    uint16_t response = 0;
    response = BLE.scanNetwork(time);
    if(response == 0){
        USB.println(F("__________Device found"));
    }else{
        USB.print(F("Scanner, ERROR = "));
        USB.println(response, DEC);
    }
    return response;
}
```

3.1.3.28 startScanningDevice()

```
uint16_t BLECentral::startScanningDevice (
            char mac[] )
```

```
                                                 {
    uint16_t response = 0;

    response = BLE.scanDevice(mac);//devuelve 1
    if(response == 1){
        USB.println(F("__________Device found"));
    }else if (response == 0){
        USB.println(F("__________Device NOT found"));
    }else{
    USB.print(F("Scanner, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

Here is the caller graph for this function:



3.1.3.29 turnOffModule()

```
void BLECentral::turnOffModule ( )
```

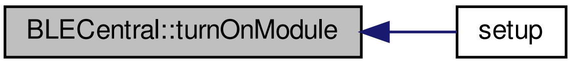
```
                              {
  BLE.OFF();
}
```

3.1.3.30 turnOnModule()

```
int8_t BLECentral::turnOnModule (
             uint8_t socket )
```

```
                                              {
    int8_t response;
    response = BLE.ON(socket);
    if(response == 0){
      USB.println(F("BLE module switch on Ok "));
    }else{
       USB.print(F("BLE module switch, ERROR = "));
       USB.println(response, DEC);
    }
    return response;
 }
```

Here is the caller graph for this function:

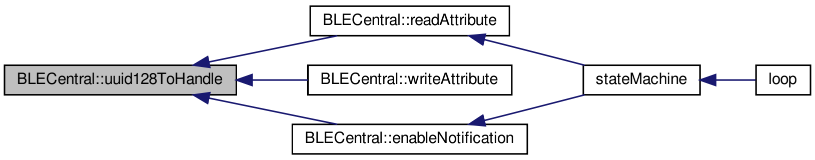


3.1.3.31 uuid128ToHandle()

```
uint16_t BLECentral::uuid128ToHandle (
             uint8_t * uuid128 )  [private]
```

```
                                                     {

   uint8_t numSer;
   uint8_t numCar;
   //uint16_t uuid16;
   //uuid16 =  (((uint16_t)uuid128[2] << 8) + ((uint16_t)(uuid128[3])));

   for(numSer = 0; numSer < device->numberOfServices; numSer++){
     if(0x00 == memcmp(device->service[numSer].service.
      uuid128, uuid128, 16)){//| (device->servicio[numSer].service.uuid16 == uuid16)
       return device->service[numSer].service.
      start_group_handle;
     }
     for(numCar = 0; numCar < device->service[numSer].
      numberOfCharacteristics; numCar++){
       if(0x00 == memcmp(device->service[numSer].characteristic[numCar].
      charac.uuid128, uuid128, 16)){//|
       (device->servicio[numSer].caracteristica[numCar].charac.uuid16 == uuid16)
         return device->service[numSer].characteristic[numCar].
      charac.value_handle;
       }
     }
   }
   return 0;
 }
```

Here is the caller graph for this function:


BLECentral::uuid128ToHandle
BLECentral::readAttribute
BLECentral::writeAttribute
BLECentral::enableNotification
stateMachine
loop

3.1.3.32 uuid16ToHandle()

```
uint16_t BLECentral::uuid16ToHandle (
              uint16_t uuid16 )   [private]
```

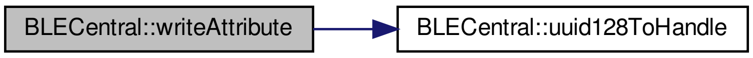
```
                                                  {
   uint8_t numSer;
   uint8_t numCar;
   uint8_t numDesc;
   for(numSer = 0; numSer < device->numberOfServices; numSer++){
       if(device->service[numSer].service.uuid16 == uuid16){
          return device->service[numSer].service.start_group_handle;
       }
       for(numCar = 0; numCar < device->service[numSer].numberOfCharacteristics; numCar++){

           if(device->service[numSer].characteristic[numCar].charac.uuid16 == uuid16){
                return device->service[numSer].characteristic[numCar].charac.value_handle;
           }
           for(numDesc = 0; numDesc < device->service[numSer].characteristic[numCar].numberOfDescriptors; numDesc++){
                if(device->service[numSer].characteristic[numCar].descriptor[numDesc].descriptor.uuid16 == uuid16){
                     return device->service[numSer].characteristic[numCar].descriptor[numDesc].descriptor.handle;
                }
            }
        }
    }
}
```

3.1.3.33 writeAttribute()

```
uint16_t BLECentral::writeAttribute (
              uint8_t connection,
              uint8_t * uuid128,
              uint8_t * data,
              uint8_t length )
```

```
{
    uint16_t response;
    USB.println(F("Writing attribute.. "));
    response = BLE.attributeWrite(connection, uuid128ToHandle(uuid128), data, length);
    if (response == 0){
        USB.println(F("Write attribute OK"));
    }else{
       USB.println(F("Write ERROR = "));
    }
    return response;
}
```

Here is the call graph for this function:



3.1.4 Member Data Documentation

3.1.4.1 device

`Device_t* BLECentral::device [private]`

Variable : Struct to save a BLE device and its data.

For the management of the device by the master

The documentation for this class was generated from the following files:

- BLECentral.h
- BLECentral.cpp

3.2 Buffer Class Reference

Buffer Class.

`#include <Buffer.h>`

Public Member Functions

- Buffer ()

    private methods //////////////////////////
- ∼Buffer ()
- void putDataToSend (uint8_t ∗value, uint8_t type)
- uint8_t ∗ getDataToSend ()
- uint8_t getDataToSendSize ()
- void clearDataToSend ()
- void putNetworkReceivedData (char ∗value)
- uint8_t getNetworkReceivedData (uint8_t index)
- void clearNetworkReceivedData ()

Public Attributes

- uint8_t dataToSend_Index
- uint8_t dataToSend [dataToSend_Size]
- uint8_t networkReceivedData [ReceivedData_Size]

3.2.1 Detailed Description

Buffer Class.

defines all the variables and functions used

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Buffer()

```
Buffer::Buffer ( )
```

private methods /////////////////////////

public methods and attributes ////////////

class constructor It does nothing

Parameters

void	

Returns

void

```
{
}
```

3.2.2.2 ∼Buffer()

```
Buffer::∼Buffer ( )
```

class Destructor It does nothing

Parameters

void	

Returns

void

```
                 {
}
```

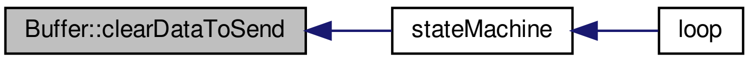
3.2.3 Member Function Documentation

3.2.3.1 clearDataToSend()

```
void Buffer::clearDataToSend ( )
```

```
                           {
    memset(dataToSend, 0x00, sizeof(dataToSend));
    dataToSend_Index = 0;
}
```

Here is the caller graph for this function:

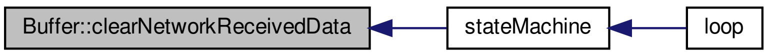


3.2.3.2 clearNetworkReceivedData()

```
void Buffer::clearNetworkReceivedData ( )
```

```
                                    {
    memset(networkReceivedData, 0x00, sizeof(
  networkReceivedData));
}
```

Here is the caller graph for this function:

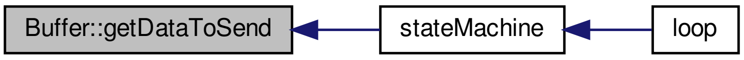


3.2.3.3 getDataToSend()

```
uint8_t * Buffer::getDataToSend ( )
```

```
                                 {

    uint8_t currentIndex = 0;
    uint8_t elemtLenght;
    USB.println(F("________Data stored in the buffer to send:"));
    while(currentIndex < dataToSend_Index){
      USB.print(F(" Element Type: "));
      USB.print( dataToSend[currentIndex++], DEC );
      elemtLenght = dataToSend[currentIndex++];
      USB.print(F(", Element Lenght: "));
      USB.print( elemtLenght, DEC );
      USB.print(F(", Element Data: "));
      for(uint8_t i=0; i<elemtLenght; i++){
        USB.print( dataToSend[currentIndex++], HEX );
        USB.print(F(" "));
      }
      USB.println(F(""));
    }
    return dataToSend;
}
```

Here is the caller graph for this function:

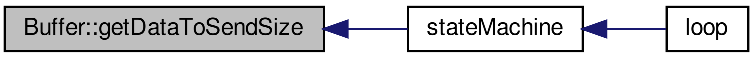


3.2.3.4 getDataToSendSize()

```
uint8_t Buffer::getDataToSendSize ( )
```

```
                                  {
  return dataToSend_Index;
}
```

Here is the caller graph for this function:

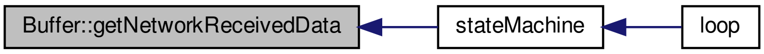


3.2.3.5 getNetworkReceivedData()

```
uint8_t Buffer::getNetworkReceivedData (
             uint8_t index )
```

```
                                                   {
      uint8_t value;
      value = networkReceivedData[index];
      return value;
  }
```

Here is the caller graph for this function:

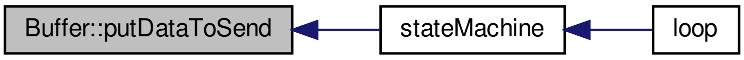


3.2.3.6 putDataToSend()

```
void Buffer::putDataToSend (
             uint8_t * value,
             uint8_t type )
```

```
                                                    {

      uint8_t nextIndex;
      nextIndex = (value[0] + 1);//value[0] contains the size of the data + 1 for the type
      if((nextIndex + dataToSend_Index) < dataToSend_Size){
        dataToSend[dataToSend_Index++]= type;
        memcpy(dataToSend + dataToSend_Index, value, (value[0] + 1));
        dataToSend_Index = dataToSend_Index + nextIndex;
        USB.print("Buffer: data stored correctly, stored data = ");
        for(uint8_t i = 0; i < dataToSend_Index; i++){
          USB.print(dataToSend[i], HEX);
          USB.print(":");
        }
        USB.println("");
        USB.print("  -Index ");
        USB.println(dataToSend_Index, DEC);
        USB.println("");
      }else{
        USB.print("The buffer is full, there is no more space ");
      }
  }
```

Here is the caller graph for this function:

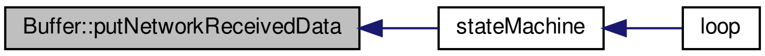


3.2.3.7 putNetworkReceivedData()

```
void Buffer::putNetworkReceivedData (
             char * value )
```

```
                                                 {
    uint8_t type;
    type = (uint8_t)(value[1]-48);
    USB.print("Message type: ");
    USB.println(type, DEC);

    switch(type){
    case 1://Establish the time to send
      networkReceivedData[0] = type;
      networkReceivedData[1] = (uint8_t)((value[3]-48) * 10) + (value[5]-48);//hours
      networkReceivedData[2] = (uint8_t)((value[7]-48) * 10) + (value[9]-48);//minutes
    break;

    case 2://Establish the sensors value to be send
      networkReceivedData[0] = type;
      for(uint8_t i = 0; i < 11; i++){
      networkReceivedData[i+1] = (uint8_t) value[((i*2)+3)] - 48;
      }
    break;

    default:
      networkReceivedData[0] = 0;
    break;
  }
}
```

Here is the caller graph for this function:



3.2.4 Member Data Documentation

3.2.4.1 dataToSend

```
uint8_t Buffer::dataToSend[dataToSend_Size]
```

3.2.4.2 dataToSend_Index

```
uint8_t Buffer::dataToSend_Index
```

3.2.4.3 networkReceivedData

```
uint8_t Buffer::networkReceivedData[ReceivedData_Size]
```

The documentation for this class was generated from the following files:

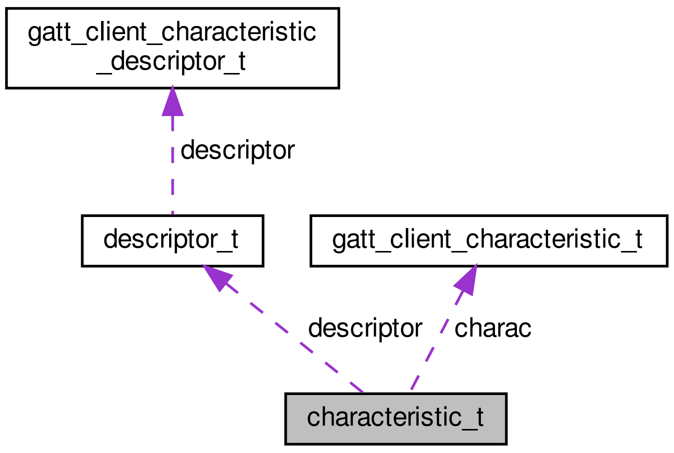
- Buffer.h
- Buffer.cpp

3.3 characteristic_t Struct Reference

Struct to store a characteristic and its descriptors.

```
#include <BLECentral.h>
```

Collaboration diagram for characteristic_t:


gatt_client_characteristic_descriptor_t
descriptor
descriptor_t
gatt_client_characteristic_t
descriptor
charac
characteristic_t

Public Attributes

- gatt_client_characteristic_t charac
- uint8_t numberOfDescriptors
- descriptor_t ∗ descriptor

3.3.1 Detailed Description

Struct to store a characteristic and its descriptors.

3.3.2 Member Data Documentation

3.3.2.1 charac

```
gatt_client_characteristic_t characteristic_t::charac
```

Struct gatt_client_characteristic_t

3.3.2.2 descriptor

```
descriptor_t* characteristic_t::descriptor
```

Pointer to descriptor_t struct

3.3.2.3 numberOfDescriptors

```
uint8_t characteristic_t::numberOfDescriptors
```

Number of descriptors in a characteristic

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

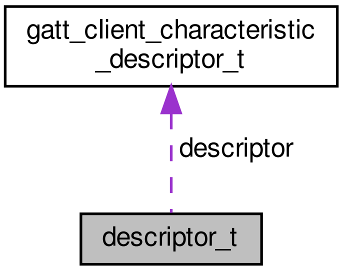
- BLECentral.h

3.4 descriptor_t Struct Reference

Struct to store descriptors.

```
#include <BLECentral.h>
```

Collaboration diagram for descriptor_t:



Public Attributes

- gatt_client_characteristic_descriptor_t descriptor

3.4.1 Detailed Description

Struct to store descriptors.

3.4.2 Member Data Documentation

3.4.2.1 descriptor

```
gatt_client_characteristic_descriptor_t descriptor_t::descriptor
```

Struct gatt_client_characteristic_descriptor_t

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

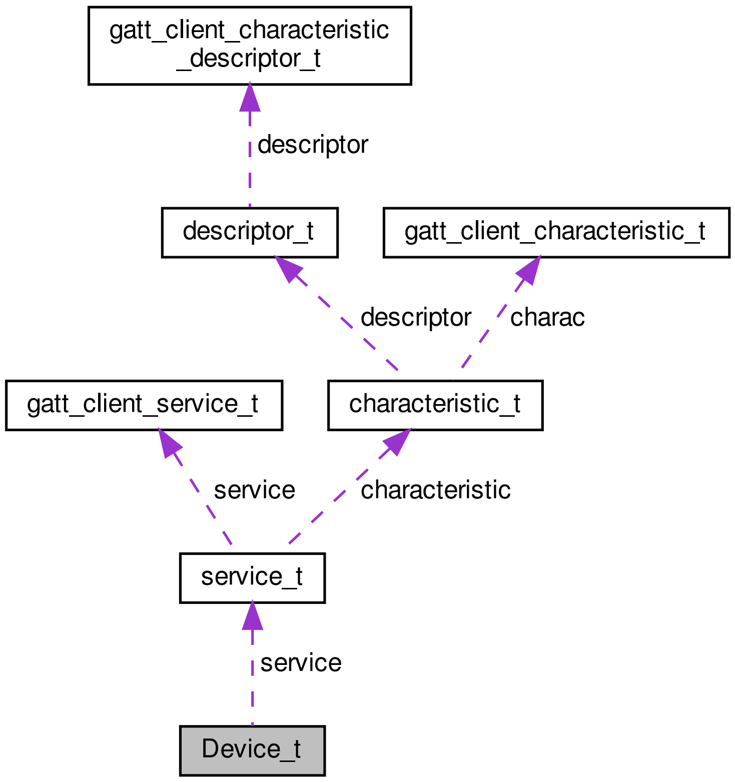
- BLECentral.h

3.5 Device_t Struct Reference

Struct to save a BLE device and its related data.

```
#include <BLECentral.h>
```

Collaboration diagram for Device_t:



Public Attributes

- uint8_t connected_handle
- uint8_t numberOfServices
- char mac [13]
- service_t * service

3.5.1 Detailed Description

Struct to save a BLE device and its related data.

3.5.2 Member Data Documentation

3.5.2.1 connected_handle

```
uint8_t Device_t::connected_handle
```

The connection handle

3.5.2.2 mac

```
char Device_t::mac[13]
```

The device MAC address

3.5.2.3 numberOfServices

```
uint8_t Device_t::numberOfServices
```

Number of service in BLE profile

3.5.2.4 service

```
service_t* Device_t::service
```

Pointer to servicio_t struct

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

- BLECentral.h

3.6 findInformationCommand_t Struct Reference

```
#include <BLECentral.h>
```

Public Attributes

- uint8_t t_length
- uint8_t messageType
- uint8_t payloadLenght
- uint8_t classID
- uint8_t commandID
- uint8_t Connectionhandle
- uint16_t startFirstAttributeHandle
- uint16_t endLastAttributeHandle

3.6.1 Member Data Documentation

3.6.1.1 classID

```
uint8_t findInformationCommand_t::classID
```

Command class ID

3.6.1.2 commandID

```
uint8_t findInformationCommand_t::commandID
```

Command ID

3.6.1.3 Connectionhandle

```
uint8_t findInformationCommand_t::Connectionhandle
```

Connectionhandle

3.6.1.4 endLastAttributeHandle

```
uint16_t findInformationCommand_t::endLastAttributeHandle
```

endLastAttributeHandle

3.6.1.5 messageType

```
uint8_t findInformationCommand_t::messageType
```

The type of command

3.6.1.6 payloadLenght

```
uint8_t findInformationCommand_t::payloadLenght
```

The payloadLenght of the command

3.6.1.7 startFirstAttributeHandle

```
uint16_t findInformationCommand_t::startFirstAttributeHandle
```

startFirstAttributeHandle

3.6.1.8 t_length

```
uint8_t findInformationCommand_t::t_length
```

The total lenght of the command

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

- BLECentral.h

3.7 gatt_client_characteristic_descriptor_t Struct Reference

Structure to identificate a descriptor.

```
#include <BLECentral.h>
```

Public Attributes

- uint16_t handle
- uint16_t uuid16
- uint8_t uuid128 [16]

3.7.1 Detailed Description

Structure to identificate a descriptor.

3.7.2 Member Data Documentation

3.7.2.1 handle

```
uint16_t gatt_client_characteristic_descriptor_t::handle
```

Descriptor handle

3.7.2.2 uuid128

```
uint8_t gatt_client_characteristic_descriptor_t::uuid128[16]
```

Descriptor 128 bits uuid

3.7.2.3 uuid16

```
uint16_t gatt_client_characteristic_descriptor_t::uuid16
```

Descriptor SIG BLE 16 bits uuid

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

- BLECentral.h

3.8 gatt_client_characteristic_t Struct Reference

Structure to identificate a characteristic.

```
#include <BLECentral.h>
```

Public Attributes

- uint16_t start_handle
- uint16_t value_handle
- uint8_t properties
- uint16_t uuid16
- uint8_t uuid128 [16]

3.8.1 Detailed Description

Structure to identificate a characteristic.

3.8.2 Member Data Documentation

3.8.2.1 properties

```
uint8_t gatt_client_characteristic_t::properties
```

Characteristic properties

3.8.2.2 start_handle

```
uint16_t gatt_client_characteristic_t::start_handle
```

Characteristic start handle

3.8.2.3 uuid128

```
uint8_t gatt_client_characteristic_t::uuid128[16]
```

Characteristic 128 bits uuid

3.8.2.4 uuid16

```
uint16_t gatt_client_characteristic_t::uuid16
```

Characteristic SIG BLE 16 bits uuid

3.8.2.5 value_handle

```
uint16_t gatt_client_characteristic_t::value_handle
```

Characteristic value handle

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

- BLECentral.h

3.9 gatt_client_service_t Struct Reference

Structure to identificate a service.

```
#include <BLECentral.h>
```

Public Attributes

- uint16_t start_group_handle
- uint16_t end_group_handle
- uint16_t uuid16
- uint8_t uuid128 [16]

3.9.1 Detailed Description

Structure to identificate a service.

3.9.2 Member Data Documentation

3.9.2.1 end_group_handle

```
uint16_t gatt_client_service_t::end_group_handle
```

Service end group handle

3.9.2.2 start_group_handle

```
uint16_t gatt_client_service_t::start_group_handle
```

Service start group handle

3.9.2.3 uuid128

```
uint8_t gatt_client_service_t::uuid128[16]
```

Service 128 bits uuid

3.9.2.4 uuid16

```
uint16_t gatt_client_service_t::uuid16
```

Service SIG BLE 16 bits uuid

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

- BLECentral.h

3.10 LoraWan Class Reference

LoraWan Class.

```
#include <LoraWan.h>
```

Public Member Functions

- LoraWan ()

    private methods ////////////////////////
- ∼LoraWan ()
- uint8_t turnOnModule (uint8_t socket)
- void turnOffModule ()
- uint8_t turnOffModule2 (uint8_t socket)
- uint8_t setAdaptativeDataRate (char ∗onOff)
- uint8_t setChannelFrequency (uint8_t channel, uint32_t frequency)
- uint8_t setChannelDataRateRange (uint8_t channel, uint8_t drMin, uint8_t drMax)
- uint8_t setChannelDutyCycle (uint8_t channel, uint16_t dutyCycle)
- uint8_t enableOrDisableChannel (uint8_t channel, char ∗onOff)
- uint8_t setTxPower (uint8_t power)
- uint8_t getTxPower ()
- void printChannelsStatus ()
- uint8_t printDeviceAddr ()
- void configure2OTAA (char DEVICE_EUI[ ], char APP_EUI[ ], char APP_KEY [ ])
- void configure2ABP (char DEVICE_EUI[ ], char DEVICE_ADDR[ ], char NWK_SESSION_KEY [ ], char AP←P_SESSION_KEY [ ])
- uint8_t joinOTAA ()
- uint8_t joinABP ()
- uint8_t setRetries (uint8_t retries)
- uint8_t getRetries ()
- uint8_t setAutomaticReply (char ∗onOff)
- uint8_t getAutomaticReply ()
- uint8_t saveModuleConfig ()
- uint8_t setDataRateNextTransmision (uint8_t socket)

- uint8_t sendUnconfirmedData (uint8_t port, uint8_t ∗data, uint8_t len)
- uint8_t sendConfirmedData (uint8_t port, uint8_t ∗data, uint8_t len)
- char ∗ receiveDowlinkData ()
- uint8_t setBatteryLevelStatus ()
- uint32_t getUplinkCounter ()
- uint32_t getDownlinkCounter ()
- uint8_t getGatewayNumber ()
- uint8_t setDowlinkRX1Delay (uint16_t delay)
- uint8_t getDowlinkRX1Delay ()
- uint8_t setDowlinkRX2Parameters (uint8_t datarate, uint32_t frequency)
- uint8_t getDowlinkRX2Delay ()
- uint8_t getDowlinkRX2Parameters (char ∗band)

3.10.1 Detailed Description

LoraWan Class.

defines all the variables and functions used

3.10.2 Constructor & Destructor Documentation

3.10.2.1 LoraWan()

```
LoraWan::LoraWan ( )
```

private methods ////////////////////////

public methods ///////////

class constructor It does nothing

Parameters

void	

Returns

void

```
                 {
}
```

3.10.2.2 ∼LoraWan()

```
LoraWan::∼LoraWan ( )
```

class Destructor It does nothing

Parameters

void	

Returns

void

```
                  {
}
```

3.10.3 Member Function Documentation

3.10.3.1 configure2ABP()

```
void LoraWan::configure2ABP (
              char DEVICE_EUI[ ],
              char DEVICE_ADDR[ ],
              char NWK_SESSION_KEY[ ],
              char APP_SESSION_KEY[ ] )
```

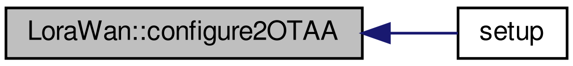
```
            {

    LoRaWAN.setDeviceEUI(DEVICE_EUI);
    LoRaWAN.setDeviceAddr(DEVICE_ADDR);
    LoRaWAN.setNwkSessionKey(NWK_SESSION_KEY);
    LoRaWAN.setAppSessionKey(APP_SESSION_KEY);
}
```

3.10.3.2 configure2OTAA()

```
void LoraWan::configure2OTAA (
              char DEVICE_EUI[ ],
              char APP_EUI[ ],
              char APP_KEY[ ] )
```

```
                                                                              {

    LoRaWAN.setDeviceEUI(DEVICE_EUI);
    LoRaWAN.setAppEUI(APP_EUI);
    LoRaWAN.setAppKey(APP_KEY);
}
```

Here is the caller graph for this function:

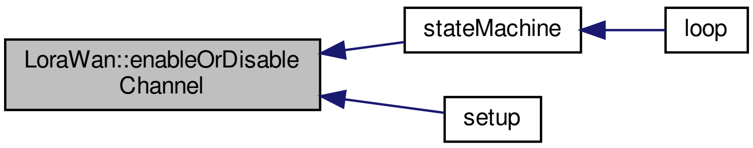


3.10.3.3 enableOrDisableChannel()

```
uint8_t LoraWan::enableOrDisableChannel (
            uint8_t channel,
            char * onOff )
```

```
                                                                    {
    uint8_t response;
    response = LoRaWAN.setChannelStatus(channel, onOff);
    if( response == 0 ){
      USB.println(F("LoRaWAN module Channel status set OK"));
    }else {
      USB.print(F("LoRaWAN module Channel status set, ERROR = "));
      USB.println(response, DEC);
    }
    return response;
}
```

Here is the caller graph for this function:



3.10.3.4 getAutomaticReply()

```
uint8_t LoraWan::getAutomaticReply ( )
```

```
                                  {
  uint8_t response;
  response = LoRaWAN.getAR();
  if( response == 0 ) {
    USB.print(F("LoRaWAN module Get automatic reply status OK. "));
    USB.print(F("LoRaWAN module Automatic reply status: "));
    if (LoRaWAN._ar == true){
      USB.println(F("on"));
    }else{
      USB.println(F("off"));
    }
  }else{
    USB.print(F("LoRaWAN module Get automatic reply status, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

3.10.3.5 getDowlinkRX1Delay()

uint8_t LoraWan::getDowlinkRX1Delay ( )

```
                                   {
  uint8_t response;
  response = LoRaWAN.getRX1Delay();
  if(response == 0){
    return LoRaWAN._rx1Delay;
  }else{
    return response;
  }
}
```

3.10.3.6 getDowlinkRX2Delay()

uint8_t LoraWan::getDowlinkRX2Delay ( )

```
                                   {
  uint8_t response;
  response = LoRaWAN.getRX2Delay();
  if(response == 0){
    return LoRaWAN._rx2Delay;
  }else{
    return response;
  }
}
```

3.10.3.7 getDowlinkRX2Parameters()

uint8_t LoraWan::getDowlinkRX2Parameters (
char * *band*)

```
                                                   {
  uint8_t response;
  response = LoRaWAN.getRX2Parameters(band);
  if(response == 0){
    USB.print(F("LoRaWAN module Dowlink RX2 Parameters, Frequency = "));
    USB.print(LoRaWAN._rx2Frequency, DEC);
    USB.print(F(" ,data rate =  "));
    USB.println(LoRaWAN._rx2DataRate);
    //return LoRaWAN._rx2Frequency;
    //return LoRaWAN._rx2DataRate;
    return response;
  }else{
    return response;
  }
}
```

3.10.3.8 getDownlinkCounter()

```
uint32_t LoraWan::getDownlinkCounter ( )
```

```
                                     {
  uint8_t response;
  response = LoRaWAN.getDownCounter();
  if(response == 0){
    return LoRaWAN._downCounter;
  }else{
    return response;
  }
}
```

3.10.3.9 getGatewayNumber()

```
uint8_t LoraWan::getGatewayNumber ( )
```

```
                                  {
  uint8_t response;
  response = LoRaWAN.getGatewayNumber();
  if(response == 0){
    return LoRaWAN._gwNumber;
  }else{
    return response;
  }
}
```

3.10.3.10 getRetries()

```
uint8_t LoraWan::getRetries ( )
```

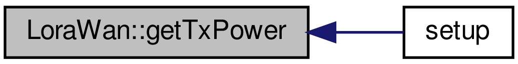
```
                            {
  uint8_t response;
  response = LoRaWAN.getRetries();
  if( response == 0 ) {
    USB.print(F("LoRaWAN module Get Retransmissions for uplink confirmed packet OK. "));
    USB.print(F("TX retries: "));
    USB.println(LoRaWAN._retries, DEC);
  }else{
    USB.print(F("LoRaWAN module Get Retransmissions for uplink confirmed packet, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

3.10.3.11 getTxPower()

```
uint8_t LoraWan::getTxPower ( )
```

```
                            {
  uint8_t response;
  response = LoRaWAN.getPower();
  if( response == 0 ){
    USB.println(F("LoRaWAN module Power level get OK"));
    USB.print(F("  -Power index:"));
    USB.println(LoRaWAN._powerIndex, DEC);
  }else{
    USB.print(F("LoRaWAN module Power level get, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

Here is the caller graph for this function:



3.10.3.12 getUplinkCounter()

```
uint32_t LoraWan::getUplinkCounter ( )
```

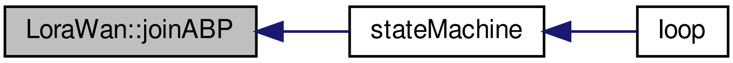
```
                                     {
  uint8_t response;
  response = LoRaWAN.getUpCounter();
  if(response == 0){
    return LoRaWAN._upCounter;
  }else{
    return response;
  }
}
```

3.10.3.13 joinABP()

```
uint8_t LoraWan::joinABP ( )
```

```
                         {
    uint8_t response;
    response = LoRaWAN.joinABP();
    if(response == 0){
        USB.println(F("LoRaWAN module join the network by ABP OK"));
    }else{
        USB.print(F("LoRaWAN module join ABP, ERROR = "));
        USB.println(response, DEC);
    }
    return response;
}
```

Here is the caller graph for this function:

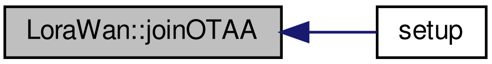


3.10.3.14 joinOTAA()

```
uint8_t LoraWan::joinOTAA ( )
```

```
                         {
    uint8_t response;
    response = LoRaWAN.joinOTAA();
    if(response == 0){
        USB.println(F("LoRaWAN module join the network by OTAA OK"));
    }else{
        USB.print(F("LoRaWAN module join OTAA, ERROR = "));
        USB.println(response, DEC);
    }
    return response;
}
```

Here is the caller graph for this function:

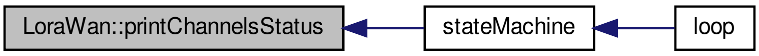


3.10.3.15 printChannelsStatus()

```
void LoraWan::printChannelsStatus ( )
```

```
                                 {
    USB.println(F("\n----------------------------"));
    USB.println(F("LoRaWAN module channels status: "));
    for( int Channel=0; Channel<16; Channel++){
        LoRaWAN.getChannelFreq(Channel);
        LoRaWAN.getChannelDutyCycle(Channel);
        LoRaWAN.getChannelDRRange(Channel);
        LoRaWAN.getChannelStatus(Channel);
        USB.print(F("Channel: "));
        USB.println(Channel);
        USB.print(F("  -Freq: "));
        USB.println(LoRaWAN._freq[Channel]);
        USB.print(F("  -Duty cycle: "));
        USB.println(LoRaWAN._dCycle[Channel]);
        USB.print(F("  -DR min: "));
        USB.println(LoRaWAN._drrMin[Channel], DEC);
        USB.print(F("  -DR max: "));
        USB.println(LoRaWAN._drrMax[Channel], DEC);
        USB.print(F("  -Status: "));
        if (LoRaWAN._status[Channel] == 1){
            USB.println(F("on"));
        }else{
            USB.println(F("off"));
        }
        USB.println(F("----------------------------"));
    }
}
```

Here is the caller graph for this function:



3.10.3.16 printDeviceAddr()

`uint8_t LoraWan::printDeviceAddr ( )`

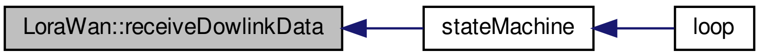
```
                                {
  uint8_t response;
  response = LoRaWAN.getDeviceAddr();
  if( response == 0 ){
    USB.println(F("LoRaWAN DeviceAddr = "));
    USB.println(LoRaWAN._devAddr);
  }else{
    USB.println(F("LoRaWAN DeviceAddr, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

3.10.3.17 receiveDowlinkData()

`char * LoraWan::receiveDowlinkData ( )`

```
                                 {

    USB.print(F("LoRaWAN module there's data on port number "));
    USB.print(LoRaWAN._port,DEC);
    USB.print(F(".\r\n   Data: "));
    USB.println(LoRaWAN._data);
    return LoRaWAN._data;
}
```

Here is the caller graph for this function:

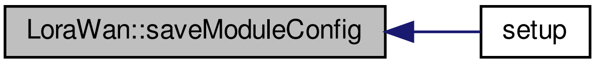


3.10.3.18 saveModuleConfig()

```
uint8_t LoraWan::saveModuleConfig ( )
```

```
                                 {
    uint8_t response;
    response = LoRaWAN.saveConfig();
    if(response == 0){
       USB.println(F("LoRaWAN module saveConfig OK"));
    }else{
        USB.print(F("LoRaWAN module saveConfig, ERROR = "));
        USB.println(response, DEC);
    }
    return response;
}
```

Here is the caller graph for this function:

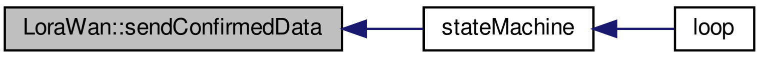


3.10.3.19 sendConfirmedData()

```
uint8_t LoraWan::sendConfirmedData (
              uint8_t port,
              uint8_t * data,
              uint8_t len )
```

```
                                                                                       {
    uint8_t response;
    response = LoRaWAN.sendConfirmed( port, data, len);
    if( response == 0 ) {
        USB.println(F("LoRaWAN module Send Confirmed packet OK"));
        if (LoRaWAN._dataReceived == true){
          return 1;
        }
    }else{
        USB.print(F("LoRaWAN module Send Confirmed packet error = "));
        USB.println(response, DEC);
        //~ '0' if OK
        //~ '1' if error
        //~ '2' if no answer
        //~ '4' if data length error
        //~ '5' if error when sending data
        //~ '6' if module hasn't joined to a network
        //~ '7' if input port parameter error
    }
    return 0;
}
```

Here is the caller graph for this function:

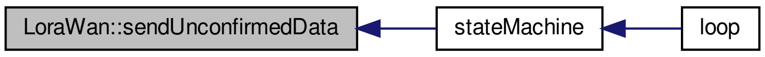


3.10.3.20 sendUnconfirmedData()

```
uint8_t LoraWan::sendUnconfirmedData (
              uint8_t port,
              uint8_t * data,
              uint8_t len )
```

```
                                                                                         {
    uint8_t response;
    response = LoRaWAN.sendUnconfirmed( port, data, len);
    if( response == 0 ) {
        USB.println(F("LoRaWAN module Send Unconfirmed packet OK"));
        if (LoRaWAN._dataReceived == true){
          return 1;
        }
    }else{
        USB.print(F("LoRaWAN module Send Unconfirmed packet ERROR = "));
        USB.println(response, DEC);
        //~ '0' if OK
        //~ '1' if error
        //~ '2' if no answer
        //~ '4' if data length error
        //~ '5' if error when sending data
        //~ '6' if module hasn't joined to a network
        //~ '7' if input port parameter error
    }
    return 0;
}
```

Here is the caller graph for this function:

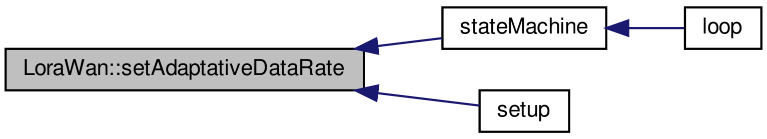


3.10.3.21 setAdaptativeDataRate()

```
uint8_t LoraWan::setAdaptativeDataRate (
               char * onOff )
```

```
                                                   {
    uint8_t response;
    response = LoRaWAN.setADR(onOff);
    if( response == 0 ){
        USB.println(F("LoRaWAN module Adaptive Data Rate OK "));
        USB.print(F("  -ADR:"));
        USB.println(LoRaWAN._adr, DEC);
    }else{
        USB.print(F("LoRaWAN moduleAdaptive Data Rate, ERROR = "));
        USB.println(response, DEC);
    }
    return response;
}
```

Here is the caller graph for this function:

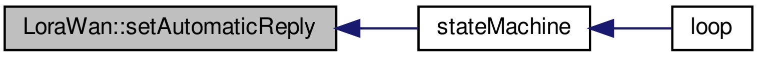


3.10.3.22 setAutomaticReply()

```
uint8_t LoraWan::setAutomaticReply (
               char * onOff )
```

```
                                           {
  uint8_t response;
  response = LoRaWAN.setAR(onOff);
  if( response == 0 ) {
    USB.println(F("LoRaWAN module Set automatic reply status on OK"));
  }else {
    USB.print(F("LoRaWAN module Set automatic reply status on, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

Here is the caller graph for this function:



3.10.3.23 setBatteryLevelStatus()

`uint8_t LoraWan::setBatteryLevelStatus ( )`

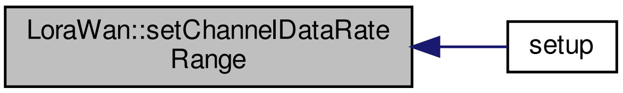
```
                                    {
  uint8_t response;
  response = LoRaWAN.setBatteryLevel();
  if( response == 0 ){
    USB.println(F("LoRaWAN module BatteryLevelStatus set OK. "));
  }else{
    USB.print(F("LoRaWAN module BatteryLevelStatus set, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

3.10.3.24 setChannelDataRateRange()

```
uint8_t LoraWan::setChannelDataRateRange (
              uint8_t channel,
              uint8_t drMin,
              uint8_t drMax )
```

```
                                                                                    {
    uint8_t response;
    response = LoRaWAN.setChannelDRRange(channel, drMin, drMax);
    if( response == 0 ){
        USB.println(F("LoRaWAN module Data Rate range set OK "));
        USB.print(F("  -Data Rate min:"));
        USB.println(LoRaWAN._drrMin[channel], DEC);
        USB.print(F("  -Data Rate max:"));
        USB.println(LoRaWAN._drrMax[channel], DEC);
    }else{
        USB.print(F("LoRaWAN module Data rate range set, ERROR = "));
        USB.println(response, DEC);
    }
    return response;
}
```

Here is the caller graph for this function:



3.10.3.25 setChannelDutyCycle()

```
uint8_t LoraWan::setChannelDutyCycle (
            uint8_t channel,
            uint16_t dutyCycle )
```

```
                                                                  {
    uint8_t response;
    response = LoRaWAN.setChannelDutyCycle(channel, dutyCycle);
    if( response == 0 ){
        USB.println(F("LoRaWAN module Duty Cycle OK. "));
        USB.print(F("Duty Cycle:"));
        USB.println(LoRaWAN._dCycle[channel], DEC);
    }else {
        USB.print(F("LoRaWAN module Duty cycle set, ERROR = "));
        USB.println(response, DEC);
    }
    return response;
}
```

3.10.3.26 setChannelFrequency()

```
uint8_t LoraWan::setChannelFrequency (
            uint8_t channel,
            uint32_t frequency )
```

```
                                                                  {
    uint8_t response;
    response = LoRaWAN.setChannelFreq(channel, frequency);
    if( response == 0 ) {// Check status
      USB.print(F("LoRaWAN module frequency set OK "));
      USB.print(F("Frequency: "));
      USB.print(LoRaWAN._freq[channel]);
      USB.print(F(" for channel: "));
      USB.println(channel, DEC);
    }else{
      USB.print(F("LoRaWAN module frequency set, ERROR = "));
      USB.println(response, DEC);
    }
    return response;
}
```

3.10.3.27 setDataRateNextTransmision()

```
uint8_t LoraWan::setDataRateNextTransmision (
              uint8_t socket )
```

```
                                                           {
    uint8_t respuesta;
   respuesta = LoRaWAN.setDataRate(dataRate);
    if(respuesta == 0){
        USB.println(F("LoRaWAN module Data Rate OK"));
    }else{
        USB.println(F("LoRaWAN module Data Rate ERROR = "));
        USB.println(respuesta, DEC);
    }
    return respuesta;
}
```

3.10.3.28 setDowlinkRX1Delay()

```
uint8_t LoraWan::setDowlinkRX1Delay (
              uint16_t delay )
```

```
                                                 {
  uint8_t response;
  response = LoRaWAN.setRX1Delay(delay);
  if(response == 0){
    return LoRaWAN._gwNumber;
  }else{
    return response;
  }
}
```

3.10.3.29 setDowlinkRX2Parameters()

```
uint8_t LoraWan::setDowlinkRX2Parameters (
              uint8_t datarate,
              uint32_t frequency )
```

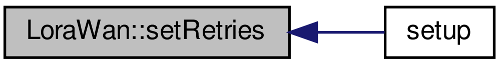
```
                                                                                {
  uint8_t response;
  response = LoRaWAN.setRX2Parameters(datarate, frequency);
  if(response == 0){
    return LoRaWAN._gwNumber;
  }else{
    return response;
  }
}
```

3.10.3.30 setRetries()

```
uint8_t LoraWan::setRetries (
              uint8_t retries )
```

```
                                          {
  uint8_t response;
  response = LoRaWAN.setRetries(retries);
  if( response == 0 ) {
    USB.println(F("LoRaWAN module Set Retransmissions for uplink confirmed packet OK"));
  }else{
    USB.print(F("LoRaWAN module Set Retransmissions for uplink confirmed packet, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

Here is the caller graph for this function:



3.10.3.31 setTxPower()

```
uint8_t LoraWan::setTxPower (
              uint8_t power )
```

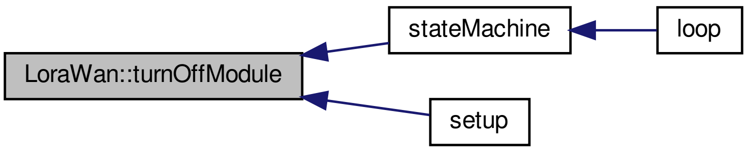
```
                                        {
  uint8_t response;
  response = LoRaWAN.setPower(power);
  if( response == 0 ){
    USB.println(F("LoRaWAN module Power level set OK"));
  }else{
    USB.print(F("LoRaWAN module Power level set, ERROR = "));
    USB.println(response, DEC);
  }
  return response;
}
```

3.10.3.32 turnOffModule()

```
void LoraWan::turnOffModule ( )
```

```
                             {
      Utils.muxOFF1();
      USB.println(F("LoRaWAN module switch off "));
}
```

Here is the caller graph for this function:



3.10.3.33 turnOffModule2()

```
uint8_t LoraWan::turnOffModule2 (
              uint8_t socket )
```

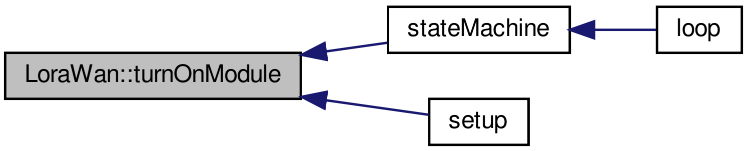
```
                                              {
      uint8_t response;
      response = LoRaWAN.OFF(socket);
      if(response == 0){
         USB.println(F("LoRaWAN module switch off ok"));
      }else{
          USB.print(F("LoRaWAN module switch ERROR = "));
          USB.println(response, DEC);
      }
      return response;
}
```

3.10.3.34 turnOnModule()

```
uint8_t LoraWan::turnOnModule (
              uint8_t socket )
```

```
                                            {

      uint8_t response;
      response = LoRaWAN.ON(socket);
      if(response == 0){
          USB.println(F("LoRaWAN module switch on Ok "));
      }else{
          USB.print(F("LoRaWAN turnOnModule(),ERROR = "));
          USB.println(response, DEC);
      }
      return response;
}
```

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- LoraWan.h
- LoraWan.cpp

3.11 readByGroupCommand_t Struct Reference

```
#include <BLECentral.h>
```

Public Attributes

- uint8_t t_length
- uint8_t messageType
- uint8_t payloadLenght
- uint8_t classID
- uint8_t commandID
- uint8_t Connectionhandle
- uint16_t startFirstAttributeHandle
- uint16_t endLastAttributeHandle
- uint8_t uuidLenght
- uint16_t uuid

3.11.1 Member Data Documentation

3.11.1.1 classID

```
uint8_t readByGroupCommand_t::classID
```

Command class ID

3.11.1.2 commandID

```
uint8_t readByGroupCommand_t::commandID
```

Command ID

3.11.1.3 Connectionhandle

```
uint8_t readByGroupCommand_t::Connectionhandle
```

Connectionhandle

3.11.1.4 endLastAttributeHandle

```
uint16_t readByGroupCommand_t::endLastAttributeHandle
```

endLastAttributeHandle

3.11.1.5 messageType

```
uint8_t readByGroupCommand_t::messageType
```

The type of command

3.11.1.6 payloadLenght

```
uint8_t readByGroupCommand_t::payloadLenght
```

The payloadLenght of the command

3.11.1.7 startFirstAttributeHandle

```
uint16_t readByGroupCommand_t::startFirstAttributeHandle
```

startFirstAttributeHandle

3.11.1.8 t_length

```
uint8_t readByGroupCommand_t::t_length
```

The total lenght of the command

3.11.1.9 uuid

```
uint16_t readByGroupCommand_t::uuid
```

uuid

3.11.1.10 uuidLenght

```
uint8_t readByGroupCommand_t::uuidLenght
```

uuidLenght

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

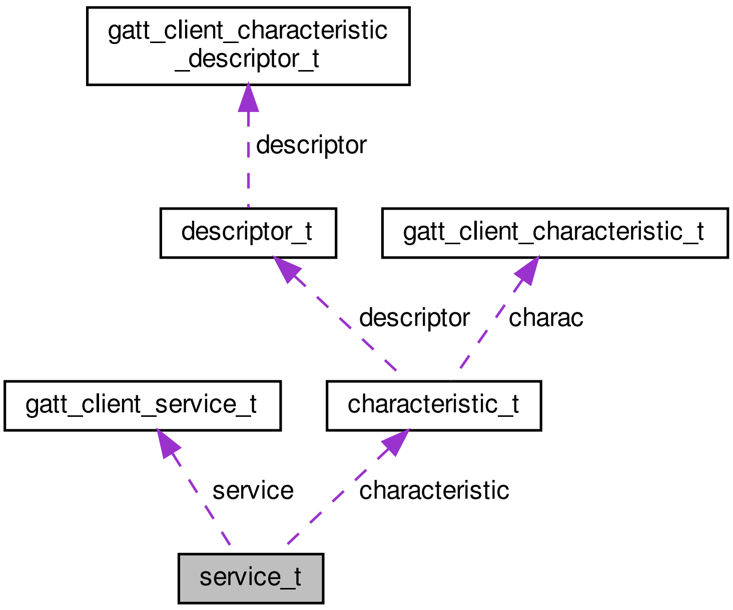
- BLECentral.h

3.12 service_t Struct Reference

Struct to store a service and its characteristics.

```
#include <BLECentral.h>
```

Collaboration diagram for service_t:



Public Attributes

- gatt_client_service_t service
- uint8_t numberOfCharacteristics
- characteristic_t ∗ characteristic

3.12.1 Detailed Description

Struct to store a service and its characteristics.

3.12.2 Member Data Documentation

3.12.2.1 characteristic

`characteristic_t* service_t::characteristic`

Pointer to characteristic_t struct

3.12.2.2 numberOfCharacteristics

`uint8_t service_t::numberOfCharacteristics`

Number of characteristic in a service

3.12.2.3 service

`gatt_client_service_t service_t::service`

Struct gatt_client_service_t

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

- BLECentral.h

3.13 trama_descriptor_t Struct Reference

Struct to make command to discover descriptors.

`#include <BLECentral.h>`

3.13.1 Detailed Description

Struct to make command to discover descriptors.

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

- BLECentral.h

3.14 trama_grupo_t Struct Reference

Struct to make command to discover services and characteristics.

`#include <BLECentral.h>`

3.14.1 Detailed Description

Struct to make command to discover services and characteristics.

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

- BLECentral.h

Chapter 4

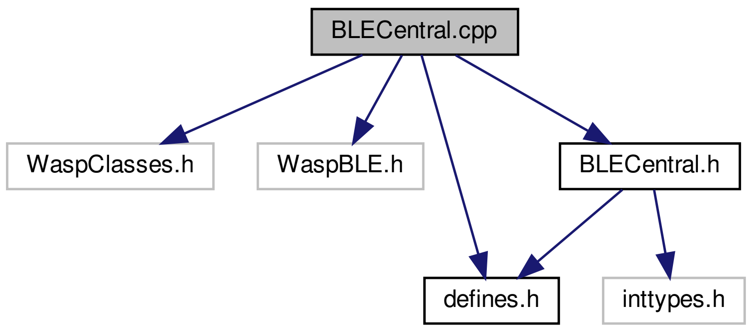
File Documentation

4.1 BLECentral.cpp File Reference

Library for managing the Bluetooth low energy module BLE112 as a Central device.

```
#include <WaspClasses.h>
#include <WaspBLE.h>
#include "defines.h"
#include "BLECentral.h"
```

Include dependency graph for BLECentral.cpp:



4.1.1 Detailed Description

Library for managing the Bluetooth low energy module BLE112 as a Central device.

Date

05/11/2018

Author

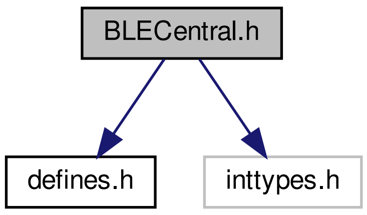
Alejandro Piñan Roescher

4.2 BLECentral.h File Reference

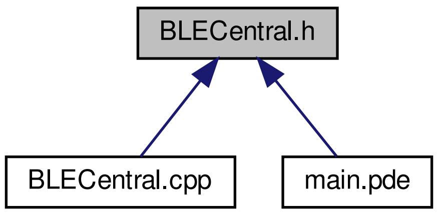
Library for managing the Bluetooth low energy module, BLE112, as a Central device.

```
#include "defines.h"
#include <inttypes.h>
```

Include dependency graph for BLECentral.h:



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



Classes

- struct gatt_client_service_t

  Structure to identificate a service.
- struct gatt_client_characteristic_t

  Structure to identificate a characteristic.
- struct gatt_client_characteristic_descriptor_t

  Structure to identificate a descriptor.
- struct descriptor_t

  Struct to store descriptors.
- struct characteristic_t

  Struct to store a characteristic and its descriptors.
- struct service_t

Struct to store a service and its characteristics.

- struct Device_t

  Struct to save a BLE device and its related data.

- struct readByGroupCommand_t
- struct findInformationCommand_t
- class BLECentral

  BLECentral Class.

4.2.1 Detailed Description

Library for managing the Bluetooth low energy module, BLE112, as a Central device.

Date

05/11/2018

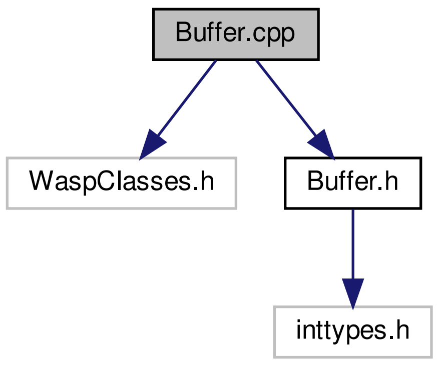
Author

Alejandro Piñan Roescher

4.3 Buffer.cpp File Reference

```
#include <WaspClasses.h>
#include "Buffer.h"
```

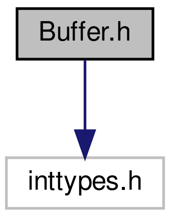
Include dependency graph for Buffer.cpp:



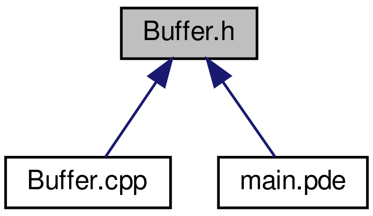
4.4 Buffer.h File Reference

```
#include <inttypes.h>
```

Include dependency graph for Buffer.h:



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



Classes

- class Buffer

    Buffer Class.

Macros

- #define dataToSend_Size 60
- #define ReceivedData_Size 30

4.4.1 Macro Definition Documentation

4.4.1.1 dataToSend_Size

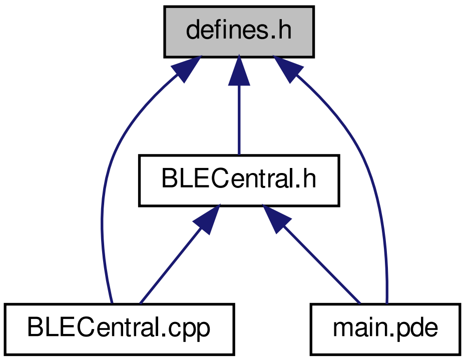
```
#define dataToSend_Size 60
```

4.4.1.2 ReceivedData_Size

```
#define ReceivedData_Size 30
```

4.5 defines.h File Reference

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



Macros

- #define DEBUG 1
- #define TX_POWER 10
- #define SCAN_INTERVAL 96
- #define SCAN_WINDOW 48
- #define BLE_GAP_DISCOVER_OBSERVATION 2
- #define BLE_PASSIVE_SCANNING 0
- #define SOCKET0 0
- #define SOCKET1 1
- #define EVENT_PORT 1
- #define DATA_PORT 3

Typedefs

- typedef enum states stateEnum_t
- typedef enum uplinkTypes UplinkTypes_t
- typedef enum downlinktypes downlinktypes_t

Enumerations

- enum states {
  BLE_SCANNING = 0, BLE_CONNECT, DISCOVER_BLE_PROFILE, ENABLE_BLE_NOTIFICATIONS,
  ENABLE_INTERRUPTIONS, SLEEP, WAKE_UP_AND_CKECK, LORAWAN_SEND_UPLINK,
  LORAWAN_RECEIVE_DOWNLINK }
- enum uplinkTypes {
  BLE_DISCONNECT_TYPE, UV_INDEX_TYPE, PRESSURE_TYPE, TEMPERATURE_TYPE,
  AMBIENT_LIGHT_TYPE, SOUND_LEVEL_TYPE, HUMIDITY_TYPE, BATTERY_LEVEL_TYPE,
  ECO2_TYPE, TVOC_TYPE, HALL_STATE_TYPE, FIELD_STRENGHT_TYPE }
- enum downlinktypes { ERROR_TYPE, CONFIGURE_TIME_TYPE, CONFIGURE_SELECTED_SENSOR↩
  S_TYPE }

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 automation_io_service_uuid [16] = {0x00, 0x00, 0x18, 0x15, 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 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}

4.5.1 Macro Definition Documentation

4.5.1.1 BLE_GAP_DISCOVER_OBSERVATION

```
#define BLE_GAP_DISCOVER_OBSERVATION 2
```

4.5.1.2 BLE_PASSIVE_SCANNING

```
#define BLE_PASSIVE_SCANNING 0
```

4.5.1.3 DATA_PORT

```
#define DATA_PORT 3
```

Port associated with the data values sent by the LoRa module

4.5.1.4 DEBUG

```
#define DEBUG 1
```

4.5.1.5 EVENT_PORT

```
#define EVENT_PORT 1
```

Port associated with the notification of the device

4.5.1.6 SCAN_INTERVAL

```
#define SCAN_INTERVAL 96
```

4.5.1.7 SCAN_WINDOW

```
#define SCAN_WINDOW 48
```

4.5.1.8 SOCKET0

```
#define SOCKET0 0
```

4.5.1.9 SOCKET1

```
#define SOCKET1 1
```

4.5.1.10 TX_POWER

```
#define TX_POWER 10
```

4.5.2 Typedef Documentation

4.5.2.1 downlinktypes_t

```
typedef enum downlinktypes downlinktypes_t
```

4.5.2.2 stateEnum_t

```
typedef enum states stateEnum_t
```

4.5.2.3 UplinkTypes_t

```
typedef enum uplinkTypes UplinkTypes_t
```

4.5.3 Enumeration Type Documentation

4.5.3.1 downlinktypes

```
enum downlinktypes
```

Enumerator

ERROR_TYPE	type ERROR_TYPE
CONFIGURE_TIME_TYPE	type CONFIGURE_TIME_TYPE
CONFIGURE_SELECTED_SENSORS_TYPE	type CONFIGURE_SELECTED_SENSORS_TYPE

```
                        {
    ERROR_TYPE,
    CONFIGURE_TIME_TYPE,
    CONFIGURE_SELECTED_SENSORS_TYPE
}downlinktypes_t;
```

4.5.3.2 states

`enum states`

Enumerator

BLE_SCANNING	State BLE_SCANNING
BLE_CONNECT	State BLE_CONNECT
DISCOVER_BLE_PROFILE	State DISCOVER_BLE_PROFILE
ENABLE_BLE_NOTIFICATIONS	State ENABLE_BLE_NOTIFICATIONS
ENABLE_INTERRUPTIONS	State ENABLE_INTERRUPTIONS
SLEEP	State SLEEP
WAKE_UP_AND_CKECK	State WAKE_UP_AND_CKECK
LORAWAN_SEND_UPLINK	State LORAWAN_SEND_UPLINK
LORAWAN_RECEIVE_DOWNLINK	State LORAWAN_RECEIVE_DOWNLINK

```
                 {
    BLE_SCANNING = 0,
    BLE_CONNECT,
    DISCOVER_BLE_PROFILE,
    ENABLE_BLE_NOTIFICATIONS,
    ENABLE_INTERRUPTIONS,
    SLEEP,
    WAKE_UP_AND_CKECK,
    LORAWAN_SEND_UPLINK,
    LORAWAN_RECEIVE_DOWNLINK
}stateEnum_t;
```

4.5.3.3 uplinkTypes

`enum uplinkTypes`

Enumerator

BLE_DISCONNECT_TYPE	type BLE_DISCONNECT
UV_INDEX_TYPE	type UV_INDEX
PRESSURE_TYPE	type PRESSURE
TEMPERATURE_TYPE	type TEMPERATURE
AMBIENT_LIGHT_TYPE	type AMBIENT_LIGHT
SOUND_LEVEL_TYPE	type SOUND_LEVEL
HUMIDITY_TYPE	type HUMIDITY
BATTERY_LEVEL_TYPE	type BATTERY_LEVEL
ECO2_TYPE	type ECO2
TVOC_TYPE	type TVOC
HALL_STATE_TYPE	type HALL_STATE
FIELD_STRENGHT_TYPE	type FIELD_STRENGHT

```
                        {
      BLE_DISCONNECT_TYPE,
      UV_INDEX_TYPE,
      PRESSURE_TYPE,
      TEMPERATURE_TYPE,
      AMBIENT_LIGHT_TYPE,
      SOUND_LEVEL_TYPE,
      HUMIDITY_TYPE,
      BATTERY_LEVEL_TYPE,
      ECO2_TYPE,
      TVOC_TYPE,
      HALL_STATE_TYPE,
      FIELD_STRENGHT_TYPE
   }UplinkTypes_t;
```

4.5.4 Variable Documentation

4.5.4.1 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]`

4.5.4.2 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]`

4.5.4.3 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]`

4.5.4.4 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]`

4.5.4.5 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]`

4.5.4.6 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]
```

4.5.4.7 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]
```

4.5.4.8 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]
```

4.5.4.9 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]
```

4.5.4.10 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]
```

4.5.4.11 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]
```

4.5.4.12 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]
```

4.5.4.13 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]
```

4.5.4.14 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]
```

4.5.4.15 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]
```

4.5.4.16 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]
```

4.5.4.17 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]
```

4.5.4.18 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]
```

4.5.4.19 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]
```

4.5.4.20 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]
```

4.5.4.21 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]
```

4.5.4.22 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]
```

4.5.4.23 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]
```

4.5.4.24 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]
```

4.5.4.25 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]
```

4.5.4.26 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]
```

4.5.4.27 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]
```

4.5.4.28 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]
```

4.5.4.29 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]
```

4.5.4.30 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]
```

4.5.4.31 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]
```

4.5.4.32 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]
```

4.5.4.33 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]
```

4.5.4.34 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]
```

4.5.4.35 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]
```

4.5.4.36 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]
```

4.5.4.37 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]
```

4.5.4.38 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]
```

4.5.4.39 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]
```

4.5.4.40 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]
```

4.5.4.41 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]
```

4.5.4.42 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]
```

4.5.4.43 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]
```

4.5.4.44 user_interface_service_uuid

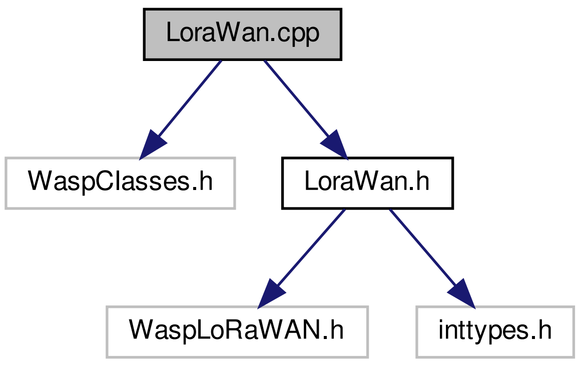
```
uint8_t user_interface_service_uuid[16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x00, 0x59, 0xF3,
0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B} [static]
```

4.6 LoraWan.cpp File Reference

Library for managing the LoRaWAN module.

```
#include <WaspClasses.h>
#include "LoraWan.h"
```

Include dependency graph for LoraWan.cpp:



4.6.1 Detailed Description

Library for managing the LoRaWAN module.

Date

05/11/2018

Author

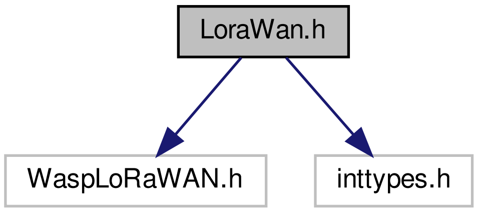
Alejandro Piñan Roescher

4.7 LoraWan.h File Reference

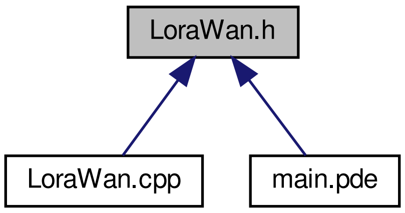
Library for managing the LoRaWAN module.

```
#include <WaspLoRaWAN.h>
#include <inttypes.h>
```

Include dependency graph for LoraWan.h:



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



Classes

- class LoraWan

    LoraWan Class.

4.7.1 Detailed Description

Library for managing the LoRaWAN module.

Date

05/11/2018

Author

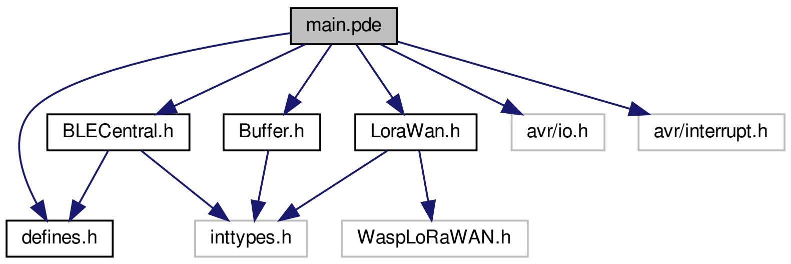
Alejandro Piñan Roescher

4.8 main.pde File Reference

main file

```
#include "BLECentral.h"
#include "LoraWan.h"
#include "Buffer.h"
#include "defines.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

Include dependency graph for main.pde:



Functions

- void activateAlarm (uint8_t hours, uint8_t minutes)
- void alarmInterruption ()

    ISR to handle the waspmote Alarm.
- void enableInterruptionPCINT8 ()

    Enables the PCINT8 interrupt, which corresponds to the RX pin of socket 0 where the BLE module is connected.
- void disableInterruptionPCINT8 ()

    Disables the PCINT8 interrupt, which corresponds to the RX pin of socket0 where the BLE module is connected.
- ISR (PCINT1_vect)

    ISR to handle the PCINT8(Pin PE0(RXD0/PCINT8/PDI)–> connected to Waspmote Socket0 (RXD0 BLE module))
- void sleep ()

    turn waspmote in sleep mode(SLEEP_MODE_PWR_DOWN)
- void stateMachine ()

    different states of the BLE-LoraWAN node
- void setup ()
- void loop ()

    while(true)

Variables

- volatile stateEnum_t state
- volatile uint8_t alarmFlag = 0
- volatile uint8_t pcint8 = 0
- char MAC [13] = "000b57a90aaf"
- char DEVICE_EUI [ ] = "00D994825A4CEA00"
- char APP_EUI [ ] = "0000000000000000"
- char APP_KEY [ ] = "170957426080C62A29819A7D656368E4"
- uint8_t hours
- uint8_t minutes
- uint8_t sensorsBitMap [12]
- uint8_t BLE_Disconnected [2] = {0x01, 0x01}
- BLECentral bleCentral = BLECentral()
- LoraWan lorawan = LoraWan()
- Buffer buffer = Buffer()

4.8.1 Detailed Description

main file

Date

05/11/2018

Author

Alejandro Piñan Roescher

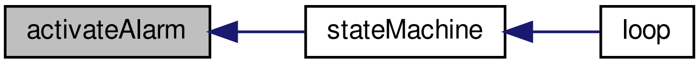
4.8.2 Function Documentation

4.8.2.1 activateAlarm()

```
void activateAlarm (
            uint8_t hours,
            uint8_t minutes )
```

```
                                                {
    uint8_t response;
    response = RTC.setAlarm1(0,hours,minutes,0,RTC_OFFSET,RTC_ALM1_MODE4);//RTC_OFFSET-->
    'time' is added to the actual time read from RTC
    if(response == 0){
      USB.print(F("RTC Alarm 1 is set in OFFSET MODE, for hours = "));
      USB.print(hours, DEC);
      USB.print(F(" ,minutes = "));
      USB.println(minutes, DEC);
    }else{
      USB.print(F("RTC Alarm 1 error, Incorrect input parameters"));
    }
}
```

Here is the caller graph for this function:



4.8.2.2 alarmInterruption()

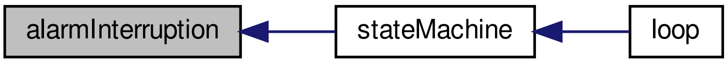
```
void alarmInterruption ( )
```

ISR to handle the waspmote Alarm.

Parameters


```
                         {
     alarmFlag = 1;
}
```

Here is the caller graph for this function:



4.8.2.3 disableInterruptionPCINT8()

```
void disableInterruptionPCINT8 ( )
```

Disables the PCINT8 interrupt, which corresponds to the RX pin of socket0 where the BLE module is connected.

Parameters

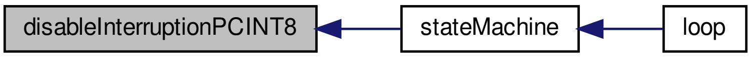
void	

Return values

void	

```
                                  {
        PCICR &= (0 << PCIE1);
}
```

Here is the caller graph for this function:



4.8.2.4 enableInterruptionPCINT8()

```
void enableInterruptionPCINT8 ( )
```

Enables the PCINT8 interrupt, which corresponds to the RX pin of socket 0 where the BLE module is connected.

Parameters

void	

Return values

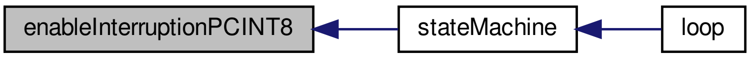
void	PCMSK1 – Pin Change Mask Register 1: Each PCINT15:8-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT15:8 isset and the PCIE1 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT15:8 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

PCIFR – Pin Change Interrupt Flag Register: When a logic change on any PCINT15:8 pin triggers an interrupt request, PCIF1 becomes set (one). If the I-bit inSREG and the PCIE1 bit in PCICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flagis cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

PCICR – Pin Change Interrupt Control Register: When the PCIE1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 1 isenabled. Any change on any enabled PCINT15:8 pin will cause an interrupt. The corresponding interrupt of PinChange Interrupt Request is executed from the PCI1 Interrupt Vector. PCINT15:8 pins are enabled individually bythe PCMSK1 Register.

```
                                {
  USB.println(F("Enabled PCINT8 interruption to receive notifications from the BLE module"));
  PCMSK1 |= (1 << PCINT8);
  PCIFR  |= (1 << PCIF1);
  PCICR  |= (1 << PCIE1);
}
```

Here is the caller graph for this function:



4.8.2.5 ISR()

```
ISR (
            PCINT1_vect   )
```

ISR to handle the PCINT8(Pin PE0(RXD0/PCINT8/PDI)–> connected to Waspmote Socket0 (RXD0 BLE module))

Parameters


```
                {
    pcint8++;
}
```

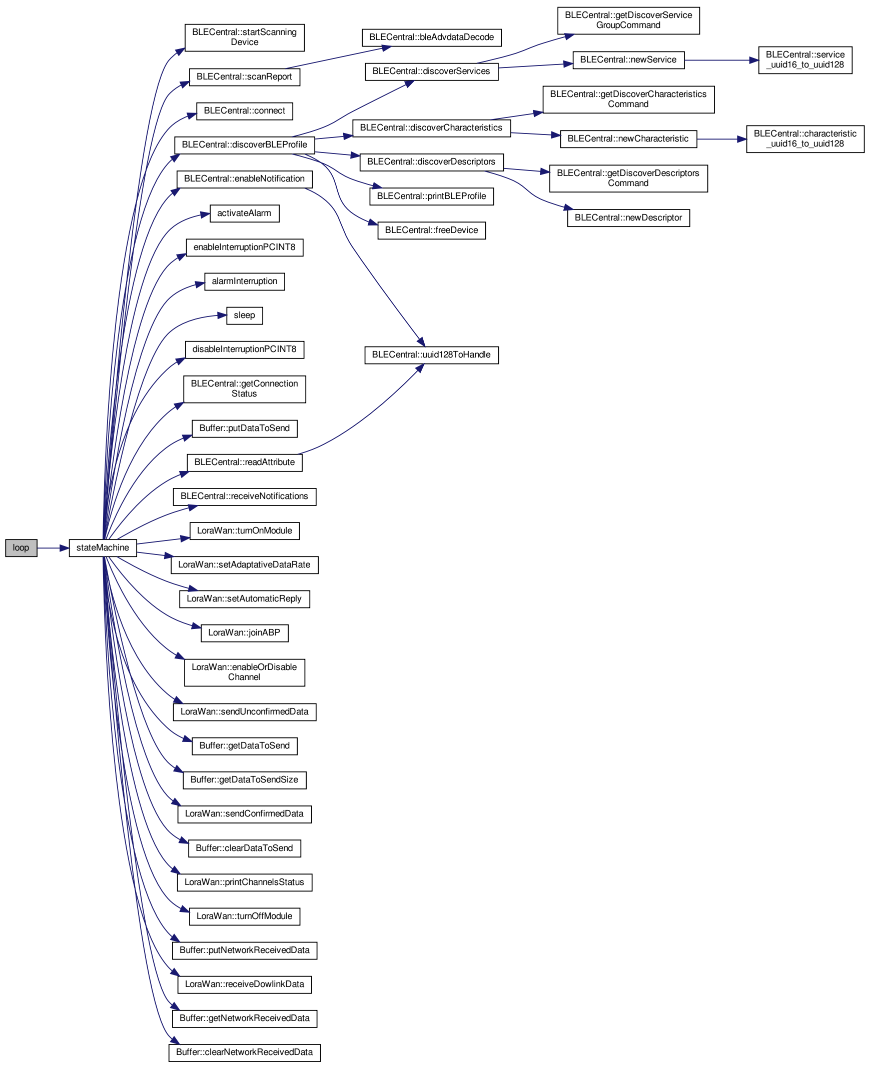
4.8.2.6 loop()

```
loop ( )
```

while(true)

```
            {

    stateMachine();
}
```

Here is the call graph for this function:

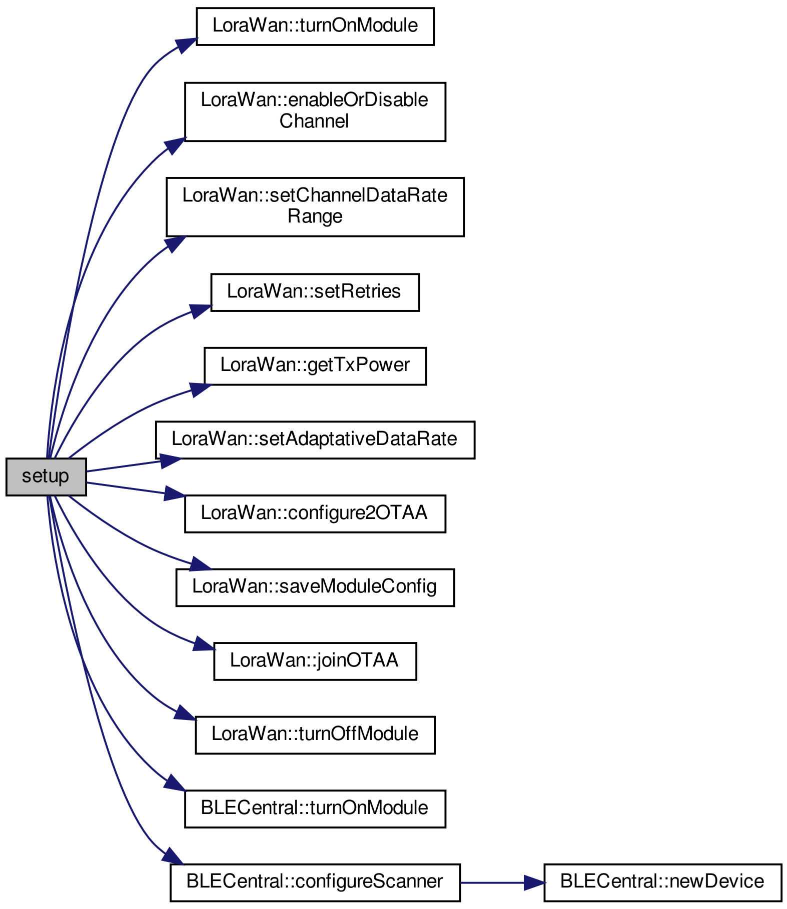

loop
stateMachine
BLECentral::startScanning Device
BLECentral::scanReport
BLECentral::connect
BLECentral::discoverBLEProfile
BLECentral::enableNotification
activateAlarm
enableInterruptionPCINT8
alarmInterruption
sleep
disableInterruptionPCINT8
BLECentral::getConnection Status
Buffer::putDataToSend
BLECentral::readAttribute
BLECentral::receiveNotifications
LoraWan::turnOnModule
LoraWan::setAdaptativeDataRate
LoraWan::setAutomaticReply
LoraWan::joinABP
LoraWan::enableOrDisable Channel
LoraWan::sendUnconfirmedData
Buffer::getDataToSend
Buffer::getDataToSendSize
LoraWan::sendConfirmedData
Buffer::clearDataToSend
LoraWan::printChannelsStatus
LoraWan::turnOffModule
Buffer::putNetworkReceivedData
LoraWan::receiveDowlinkData
Buffer::getNetworkReceivedData
Buffer::clearNetworkReceivedData
BLECentral::bleAdvdataDecode
BLECentral::discoverServices
BLECentral::discoverCharacteristics
BLECentral::discoverDescriptors
BLECentral::printBLEProfile
BLECentral::freeDevice
BLECentral::uuid128ToHandle
BLECentral::getDiscoverService GroupCommand
BLECentral::newService
BLECentral::getDiscoverCharacteristics Command
BLECentral::newCharacteristic
BLECentral::getDiscoverDescriptors Command
BLECentral::newDescriptor
BLECentral::service _uuid16_to_uuid128
BLECentral::characteristic _uuid16_to_uuid128

4.8.2.7 setup()

```
void setup ( )
```

```
            {
    USB.println(F("_______Starting setup"));
    USB.println(F(""));
    USB.println(F("_______Starting LoRaWAN module configuration"));
    state = BLE_SCANNING;//Initial state of the State Machine (BLE-LoRaWAN Node)
    hours = 0;//Initial hour and minute, 00:02, to receive the sensors data
    minutes = 2;
    memset(sensorsBitMap, 0x01, sizeof(sensorsBitMap));//By default all sensors values to send
    lorawan.turnOnModule(SOCKET1);
    //Obliged because the nanoGateway, Lopy4, has a single channel(Waspmote has 0..15 channels, but only 0,1,2,are enabled by default)
    lorawan.enableOrDisableChannel(1, "off");
    lorawan.enableOrDisableChannel(2, "off");
    lorawan.setChannelDataRateRange(0, 5, 5);//We use the channel 0 --->frec=868100000 and date rate=5-->sf=7(Lopy4, has a single data rate)
    lorawan.setRetries(2);//Number of retries for the send with confirmation(Used to send hall sensor events)
    //~ lorawan.setTxPower(4);
    lorawan.getTxPower();
    lorawan.setAdaptativeDataRate("off");//This parameter cannot be stored in the module's EEPROM using the saveConfig() function
    lorawan.configure2OTAA(DEVICE_EUI, APP_EUI, APP_KEY);
    lorawan.saveModuleConfig();
    lorawan.joinOTAA();
    lorawan.turnOffModule();
    USB.println(F("_______LoRaWAN module configuration completed"));
    USB.println(F(""));
    USB.println(F("_______BLE module configuration"));
    bleCentral.turnOnModule(SOCKET0);
    bleCentral.configureScanner(BLE_GAP_DISCOVER_OBSERVATION, TX_POWER, SCAN_INTERVAL, SCAN_WINDOW, BLE_PASSIVE_SCANNING);
    USB.println(F("_______Finished setup"));
    USB.println(F(""));
}
```

Here is the call graph for this function:



4.8.2.8 sleep()

```
void sleep ( )
```

turn waspmote in sleep mode(SLEEP_MODE_PWR_DOWN)

Parameters

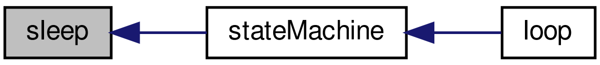
void	

Return values

void	This function turn waspmote in sleep mode(SLEEP_MODE_PWR_DOWN)

```
            {
    /* There are five different sleep modes in order of power saving:
      SLEEP_MODE_IDLE - the lowest power saving mode
      SLEEP_MODE_ADC
      SLEEP_MODE_PWR_SAVE
      SLEEP_MODE_STANDBY
      SLEEP_MODE_EXT_STANDBY
      SLEEP_MODE_PWR_DOWN - the highest power saving mode
    */
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_mode();//Put the device into sleep mode, taking care of setting the SE bit before, and clearing
   it afterwards
}
```

Here is the caller graph for this function:



4.8.2.9 stateMachine()

```
void stateMachine ( )
```

different states of the BLE-LoraWAN node

Parameters

void	

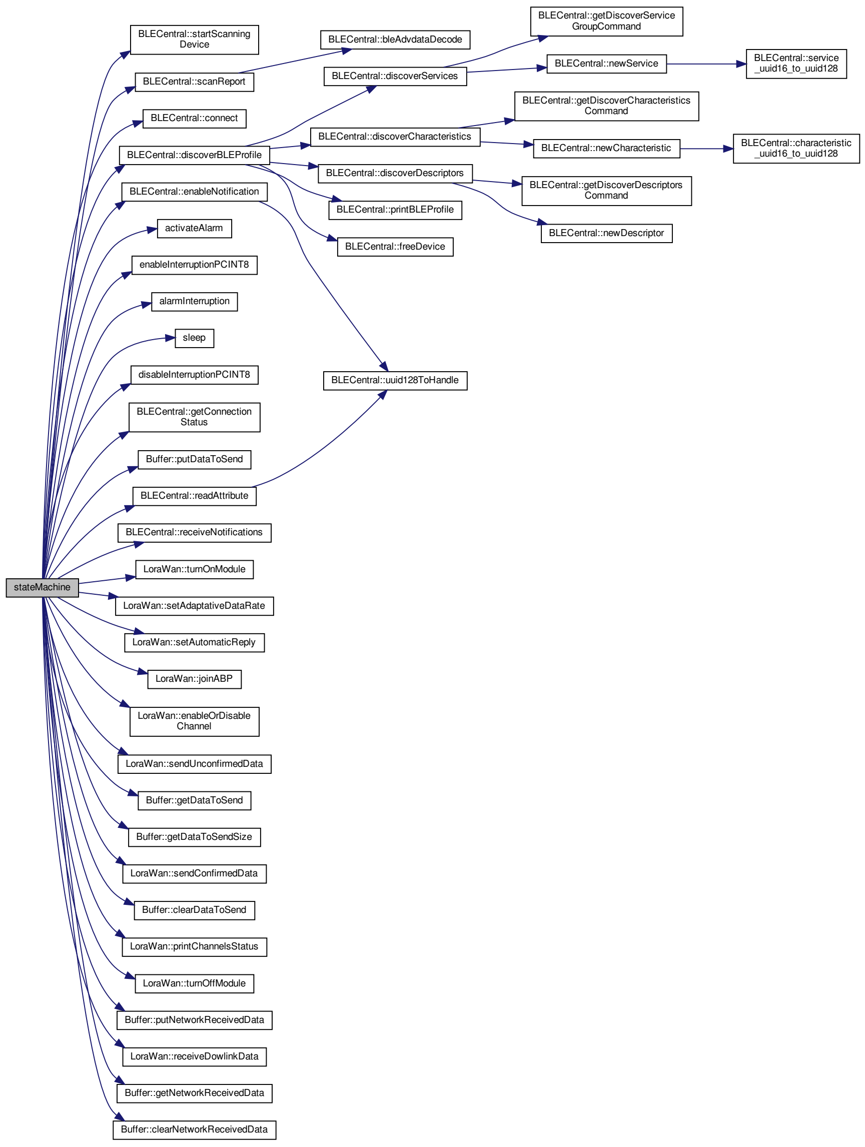
Return values

void	This function implement the sate machine of the BLE-LoraWAN node

```
                    {

    uint8_t response = 0;

    switch(state){

        case BLE_SCANNING:
            #if DEBUG >= 1
                USB.println(F("State: BLE_SCANNING"));
            #endif
            response = bleCentral.startScanningDevice(
      MAC);
            if(response){
                response = bleCentral.scanReport("Thunder Sense #02735");
                if (response){
                    state = BLE_CONNECT;
                }
            }
            break;

        case BLE_CONNECT:
            #if DEBUG >= 1
                USB.println(F("State: BLE_CONNECT"));
            #endif
            response = bleCentral.connect(MAC);
            if (response){
                state = DISCOVER_BLE_PROFILE;
                delay(1000);
            }else{
                state = BLE_SCANNING;
            }
            break;

        case DISCOVER_BLE_PROFILE:
            #if DEBUG >= 1
                USB.println(F("State: DISCOVER_BLE_PROFILE"));
            #endif
            response = bleCentral.discoverBLEProfile();
            if (response){
              state = ENABLE_BLE_NOTIFICATIONS;
            }else{
                state = BLE_SCANNING;
            }
            break;

        case ENABLE_BLE_NOTIFICATIONS:
            #if DEBUG >= 1
                USB.println(F("State: ENABLE_BLE_NOTIFICATIONS"));
            #endif
            bleCentral.enableNotification(ServiceA_Characrteristic0_State_uuid);
            state = ENABLE_INTERRUPTIONS;
            break;

        case ENABLE_INTERRUPTIONS:
            #if DEBUG >= 1
                USB.println(F("State: ENABLE_INTERRUPTIONS"));
            #endif
            activateAlarm(hours, minutes);
            enableInterruptionPCINT8();
            attachInterrupt(RTC_INT, alarmInterruption, 1);
            state = SLEEP;
            break;

        case SLEEP:
            #if DEBUG >= 1
                USB.println(F("State: SLEEP"));
            #endif
            alarmFlag = 0;
            sleep();
            USB.println(F("Waspmote wake up"));
            disableInterruptionPCINT8();
            state = WAKE_UP_AND_CKECK;
            break;

        case WAKE_UP_AND_CKECK:
            #if DEBUG >= 1
                USB.println(F("State: WAKE_UP_AND_CKECK"));
            #endif
            if(bleCentral.getConnectionStatus() != 1){//The BLE connection has been disconnected
                buffer.putDataToSend(BLE_Disconnected, BLE_DISCONNECT_TYPE);
            }else if( alarmFlag == 1){//Attend the Alarm, the established time has been met
                if(sensorsBitMap[1]==1)
                    buffer.putDataToSend(bleCentral.readAttribute(Service4_Characrteristic0_UV_Index_uuid), UV_INDEX_TYPE);
                if(sensorsBitMap[2]==1)
                    buffer.putDataToSend(bleCentral.readAttribute(Service4_Characrteristic1_Pressure_uuid), PRESSURE_TYPE);
                if(sensorsBitMap[3]==1)
                    buffer.putDataToSend(bleCentral.readAttribute(Service4_Characrteristic2_Temperature_uuid), TEMPERATURE_TYPE);
                if(sensorsBitMap[4]==1)
                    buffer.putDataToSend(bleCentral.readAttribute(Service4_Characrteristic4_Ambient_Light_uuid), AMBIENT_LIGHT_TYPE);
                if(sensorsBitMap[5]==1)
                    buffer.putDataToSend(bleCentral.readAttribute(Service4_Characrteristic5_Sound_Level_uuid), SOUND_LEVEL_TYPE);
                if(sensorsBitMap[6]==1)
                    buffer.putDataToSend(bleCentral.readAttribute(Service4_Characrteristic3_Humidity_uuid),
   HUMIDITY_TYPE);
                if(sensorsBitMap[7]==1)
                    buffer.putDataToSend(bleCentral.
  readAttribute(Service3_Characrteristic0_Battery_Level_uuid
  ), BATTERY_LEVEL_TYPE);
                if(sensorsBitMap[8]==1)
                    buffer.putDataToSend(bleCentral.
  readAttribute(Service6_Characrteristic0_ECO2_uuid),
  ECO2_TYPE);
                if(sensorsBitMap[9]==1)
                    buffer.putDataToSend(bleCentral.
  readAttribute(Service6_Characrteristic1_TVOC_uuid),
  TVOC_TYPE);
                if(sensorsBitMap[10]==1)
                    buffer.putDataToSend(bleCentral.
  readAttribute(ServiceA_Characrteristic0_State_uuid),
  HALL_STATE_TYPE);
                if(sensorsBitMap[11]==1)
                    buffer.putDataToSend(bleCentral.
  readAttribute(ServiceA_Characrteristic1_Field_Strength_uuid
  ), FIELD_STRENGHT_TYPE);
            }else{//Attend the Hall sensor notification
                buffer.putDataToSend(bleCentral.
  receiveNotifications(), HALL_STATE_TYPE);
            }
            state = LORAWAN_SEND_UPLINK;
            break;

        case LORAWAN_SEND_UPLINK:
            #if DEBUG >= 1
                USB.println(F("State: LORAWAN_SEND_UPLINK"));
            #endif
            lorawan.turnOnModule(SOCKET1);
            lorawan.setAdaptativeDataRate("off");
            lorawan.setAutomaticReply("on");
            lorawan.joinABP();
            lorawan.enableOrDisableChannel(1, "off");
            lorawan.enableOrDisableChannel(2, "off");
            if(alarmFlag == 1) {
              response = lorawan.sendUnconfirmedData(
  DATA_PORT, buffer.getDataToSend(), buffer.
  getDataToSendSize());
            }else{
              response = lorawan.sendConfirmedData(
  EVENT_PORT, buffer.getDataToSend(), buffer.
  getDataToSendSize());
            }
            if(response == 1){
              state = LORAWAN_RECEIVE_DOWNLINK;
            }else{
              state = ENABLE_INTERRUPTIONS;
            }
            buffer.clearDataToSend();
            lorawan.printChannelsStatus();
            lorawan.turnOffModule();
            USB.println(F(""));
            break;

        case LORAWAN_RECEIVE_DOWNLINK:
            #if DEBUG >= 1
                USB.println(F("State: LORAWAN_RECEIVE_DOWNLINK"));
            #endif
            buffer.putNetworkReceivedData( lorawan.
  receiveDowlinkData());
            if(buffer.getNetworkReceivedData(0) ==
  CONFIGURE_TIME_TYPE){
              hours = buffer.getNetworkReceivedData(1);
              minutes = buffer.getNetworkReceivedData(2);
              USB.print(F("Hours received: "));
              USB.println(hours,DEC);
              USB.print(F("Minutes received: "));
              USB.println(minutes,DEC);
            }else if(buffer.getNetworkReceivedData(0) ==
  CONFIGURE_SELECTED_SENSORS_TYPE){
               USB.println(F("Selected sensors = "));
              for(uint8_t i = 0; i < 12; i++){
                sensorsBitMap[i] = buffer.
  getNetworkReceivedData(i);
                USB.print(sensorsBitMap[i],DEC);
                USB.print(F(":"));
              }
              USB.println(F(""));
            }
            buffer.clearNetworkReceivedData();
            state = ENABLE_INTERRUPTIONS;
            break;
    }
```

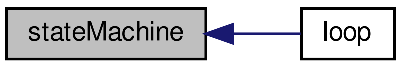
```
316 }
```

Here is the call graph for this function:



stateMachine
BLECentral::startScanning Device
BLECentral::scanReport
BLECentral::connect
BLECentral::discoverBLEProfile
BLECentral::enableNotification
activateAlarm
enableInterruptionPCINT8
alarmInterruption
sleep
disableInterruptionPCINT8
BLECentral::getConnection Status
Buffer::putDataToSend
BLECentral::readAttribute
BLECentral::receiveNotifications
LoraWan::turnOnModule
LoraWan::setAdaptativeDataRate
LoraWan::setAutomaticReply
LoraWan::joinABP
LoraWan::enableOrDisable Channel
LoraWan::sendUnconfirmedData
Buffer::getDataToSend
Buffer::getDataToSendSize
LoraWan::sendConfirmedData
Buffer::clearDataToSend
LoraWan::printChannelsStatus
LoraWan::turnOffModule
Buffer::putNetworkReceivedData
LoraWan::receiveDowlinkData
Buffer::getNetworkReceivedData
Buffer::clearNetworkReceivedData
BLECentral::bleAdvdataDecode
BLECentral::discoverServices
BLECentral::discoverCharacteristics
BLECentral::discoverDescriptors
BLECentral::printBLEProfile
BLECentral::freeDevice
BLECentral::uuid128ToHandle
BLECentral::getDiscoverService GroupCommand
BLECentral::newService
BLECentral::getDiscoverCharacteristics Command
BLECentral::newCharacteristic
BLECentral::getDiscoverDescriptors Command
BLECentral::newDescriptor
BLECentral::service _uuid16_to_uuid128
BLECentral::characteristic _uuid16_to_uuid128

Here is the caller graph for this function:



4.8.3 Variable Documentation

4.8.3.1 alarmFlag

```
volatile uint8_t alarmFlag = 0
```

variable for the alarmInterruption

4.8.3.2 APP_EUI

```
char APP_EUI[ ] = "0000000000000000"
```

Loraserver APP identifier

4.8.3.3 APP_KEY

```
char APP_KEY[ ] = "170957426080C62A29819A7D656368E4"
```

APP Key to The Things Network and loraserver

4.8.3.4 BLE_Disconnected

```
uint8_t BLE_Disconnected[2] = {0x01, 0x01}
```

4.8.3.5 bleCentral

```
BLECentral bleCentral = BLECentral()
```

4.8.3.6 buffer

```
Buffer buffer = Buffer()
```

4.8.3.7 DEVICE_EUI

```
char DEVICE_EUI[ ] = "00D994825A4CEA00"
```

Device Identifier

4.8.3.8 hours

```
uint8_t hours
```

4.8.3.9 lorawan

```
LoraWan lorawan = LoraWan()
```

4.8.3.10 MAC

```
char MAC[13] = "000b57a90aaf"
```

MAC of the device to search

4.8.3.11 minutes

```
uint8_t minutes
```

4.8.3.12 pcint8

```
volatile uint8_t pcint8 = 0
```

variable for the PCINT8 ISR

4.8.3.13 sensorsBitMap

```
uint8_t sensorsBitMap[12]
```

4.8.3.14 state

```
volatile stateEnum_t state
```

variable for the state machine

Index