

Integración con el dispositivo Waspnote

Generated by Doxygen 1.8.13

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	BLECentral Class Reference	5
3.1.1	Detailed Description	7
3.1.2	Constructor & Destructor Documentation	7
3.1.2.1	BLECentral()	7
3.1.2.2	~BLECentral()	8
3.1.3	Member Function Documentation	8
3.1.3.1	bleAdvdataDecode()	8
3.1.3.2	characteristic_uuid16_to_uuid128()	9
3.1.3.3	configureScanner()	10
3.1.3.4	connect()	11
3.1.3.5	connectWithSelectedParameters()	12
3.1.3.6	disconnect()	12
3.1.3.7	discoverBLEProfile()	12
3.1.3.8	discoverCharacteristics()	13
3.1.3.9	discoverDescriptors()	14
3.1.3.10	discoverServices()	15
3.1.3.11	enableNotification()	16

3.1.3.12	freeDevice()	17
3.1.3.13	getConnectionHandler()	18
3.1.3.14	getConnectionStatus()	18
3.1.3.15	getDiscoverCharacteristicsCommand()	19
3.1.3.16	getDiscoverDescriptorsCommand()	19
3.1.3.17	getDiscoverServiceGroupCommand()	20
3.1.3.18	newCharacteristic()	20
3.1.3.19	newDescriptor()	21
3.1.3.20	newDevice()	22
3.1.3.21	newService()	23
3.1.3.22	printBLEProfile()	24
3.1.3.23	readAttribute()	25
3.1.3.24	receiveNotifications()	26
3.1.3.25	scanReport()	27
3.1.3.26	service_uuid16_to_uuid128()	28
3.1.3.27	startScanning()	29
3.1.3.28	startScanningDevice()	29
3.1.3.29	turnOffModule()	29
3.1.3.30	turnOnModule()	30
3.1.3.31	uuid128ToHandle()	30
3.1.3.32	uuid16ToHandle()	31
3.1.3.33	writeAttribute()	31
3.1.4	Member Data Documentation	32
3.1.4.1	device	32
3.2	Buffer Class Reference	32
3.2.1	Detailed Description	33
3.2.2	Constructor & Destructor Documentation	33
3.2.2.1	Buffer()	33
3.2.2.2	~Buffer()	33
3.2.3	Member Function Documentation	34

3.2.3.1	clearDataToSend()	34
3.2.3.2	clearNetworkReceivedData()	34
3.2.3.3	getDataToSend()	35
3.2.3.4	getDataToSendSize()	35
3.2.3.5	getNetworkReceivedData()	36
3.2.3.6	putDataToSend()	36
3.2.3.7	putNetworkReceivedData()	37
3.2.4	Member Data Documentation	37
3.2.4.1	dataToSend	37
3.2.4.2	dataToSend_Index	37
3.2.4.3	networkReceivedData	38
3.3	characteristic_t Struct Reference	38
3.3.1	Detailed Description	38
3.3.2	Member Data Documentation	39
3.3.2.1	charac	39
3.3.2.2	descriptor	39
3.3.2.3	numberOfDescriptors	39
3.4	descriptor_t Struct Reference	39
3.4.1	Detailed Description	40
3.4.2	Member Data Documentation	40
3.4.2.1	descriptor	40
3.5	Device_t Struct Reference	40
3.5.1	Detailed Description	41
3.5.2	Member Data Documentation	41
3.5.2.1	connected_handle	41
3.5.2.2	mac	42
3.5.2.3	numberOfServices	42
3.5.2.4	service	42
3.6	findInformationCommand_t Struct Reference	42
3.6.1	Member Data Documentation	42

3.6.1.1	classID	42
3.6.1.2	commandID	43
3.6.1.3	Connectionhandle	43
3.6.1.4	endLastAttributeHandle	43
3.6.1.5	messageType	43
3.6.1.6	payloadLenght	43
3.6.1.7	startFirstAttributeHandle	43
3.6.1.8	t_length	43
3.7	gatt_client_characteristic_descriptor_t Struct Reference	44
3.7.1	Detailed Description	44
3.7.2	Member Data Documentation	44
3.7.2.1	handle	44
3.7.2.2	uuid128	44
3.7.2.3	uuid16	44
3.8	gatt_client_characteristic_t Struct Reference	45
3.8.1	Detailed Description	45
3.8.2	Member Data Documentation	45
3.8.2.1	properties	45
3.8.2.2	start_handle	45
3.8.2.3	uuid128	45
3.8.2.4	uuid16	45
3.8.2.5	value_handle	46
3.9	gatt_client_service_t Struct Reference	46
3.9.1	Detailed Description	46
3.9.2	Member Data Documentation	46
3.9.2.1	end_group_handle	46
3.9.2.2	start_group_handle	46
3.9.2.3	uuid128	47
3.9.2.4	uuid16	47
3.10	LoraWan Class Reference	47

3.10.1 Detailed Description	48
3.10.2 Constructor & Destructor Documentation	48
3.10.2.1 LoraWan()	48
3.10.2.2 ~LoraWan()	48
3.10.3 Member Function Documentation	49
3.10.3.1 configure2ABP()	49
3.10.3.2 configure2OTAA()	49
3.10.3.3 enableOrDisableChannel()	50
3.10.3.4 getAutomaticReply()	50
3.10.3.5 getDowlinkRX1Delay()	51
3.10.3.6 getDowlinkRX2Delay()	51
3.10.3.7 getDowlinkRX2Parameters()	51
3.10.3.8 getDownlinkCounter()	52
3.10.3.9 getGatewayNumber()	52
3.10.3.10 getRetries()	52
3.10.3.11 getTxPower()	53
3.10.3.12 getUplinkCounter()	53
3.10.3.13 joinABP()	53
3.10.3.14 joinOTAA()	54
3.10.3.15 printChannelsStatus()	54
3.10.3.16 printDeviceAddr()	55
3.10.3.17 receiveDowlinkData()	55
3.10.3.18 saveModuleConfig()	56
3.10.3.19 sendConfirmedData()	56
3.10.3.20 sendUnconfirmedData()	57
3.10.3.21 setAdaptativeDataRate()	58
3.10.3.22 setAutomaticReply()	58
3.10.3.23 setBatteryLevelStatus()	59
3.10.3.24 setChannelDataRateRange()	59
3.10.3.25 setChannelDutyCycle()	60

3.10.3.26 setChannelFrequency()	60
3.10.3.27 setDataRateNextTransmission()	61
3.10.3.28 setDowlinkRX1Delay()	61
3.10.3.29 setDowlinkRX2Parameters()	61
3.10.3.30 setRetries()	62
3.10.3.31 setTxPower()	62
3.10.3.32 turnOffModule()	63
3.10.3.33 turnOffModule2()	63
3.10.3.34 turnOnModule()	63
3.11 readByGroupCommand_t Struct Reference	64
3.11.1 Member Data Documentation	64
3.11.1.1 classID	64
3.11.1.2 commandID	65
3.11.1.3 Connectionhandle	65
3.11.1.4 endLastAttributeHandle	65
3.11.1.5 messageType	65
3.11.1.6 payloadLenght	65
3.11.1.7 startFirstAttributeHandle	65
3.11.1.8 t_length	65
3.11.1.9 uuid	65
3.11.1.10 uuidLenght	66
3.12 service_t Struct Reference	66
3.12.1 Detailed Description	67
3.12.2 Member Data Documentation	67
3.12.2.1 characteristic	67
3.12.2.2 numberOfCharacteristics	67
3.12.2.3 service	67
3.13 trama_descriptor_t Struct Reference	67
3.13.1 Detailed Description	67
3.14 trama_grupo_t Struct Reference	67
3.14.1 Detailed Description	67

4 File Documentation	69
4.1 BLECentral.cpp File Reference	69
4.1.1 Detailed Description	69
4.2 BLECentral.h File Reference	70
4.2.1 Detailed Description	71
4.3 Buffer.cpp File Reference	71
4.4 Buffer.h File Reference	72
4.4.1 Macro Definition Documentation	72
4.4.1.1 dataToSend_Size	73
4.4.1.2 ReceivedData_Size	73
4.5 defines.h File Reference	73
4.5.1 Macro Definition Documentation	75
4.5.1.1 BLE_GAP_DISCOVER_OBSERVATION	76
4.5.1.2 BLE_PASSIVE_SCANNING	76
4.5.1.3 DATA_PORT	76
4.5.1.4 DEBUG	76
4.5.1.5 EVENT_PORT	76
4.5.1.6 SCAN_INTERVAL	76
4.5.1.7 SCAN_WINDOW	76
4.5.1.8 SOCKET0	76
4.5.1.9 SOCKET1	77
4.5.1.10 TX_POWER	77
4.5.2 Typedef Documentation	77
4.5.2.1 downlinktypes_t	77
4.5.2.2 stateEnum_t	77
4.5.2.3 UplinkTypes_t	77
4.5.3 Enumeration Type Documentation	77
4.5.3.1 downlinktypes	77
4.5.3.2 states	78
4.5.3.3 uplinkTypes	78

4.5.4	Variable Documentation	79
4.5.4.1	accleration_orientation_service_uuid	79
4.5.4.2	automation_io_service_uuid	79
4.5.4.3	battery_service_uuid	79
4.5.4.4	device_information_service_uuid	79
4.5.4.5	environmental_sensing_service_uuid	79
4.5.4.6	generic_access_service_uuid	80
4.5.4.7	generic_attribute_service_uuid	80
4.5.4.8	hall_effect_service_uuid	80
4.5.4.9	iaq_service_uuid	80
4.5.4.10	power_management_service_uuid	80
4.5.4.11	Service0_Characrteristic0_Device_Name_uuid	80
4.5.4.12	Service0_Characrteristic1_Appearance_uuid	80
4.5.4.13	Service1_Characrteristic0_Service_Changed_uuid	81
4.5.4.14	Service2_Characrteristic0_Manufacturer_Name_uuid	81
4.5.4.15	Service2_Characrteristic1_Model_Number_uuid	81
4.5.4.16	Service2_Characrteristic2_Serial_Number_uuid	81
4.5.4.17	Service2_Characrteristic3_Hardware_Revision_uuid	81
4.5.4.18	Service2_Characrteristic4_Firmware_Revision_uuid	81
4.5.4.19	Service2_Characrteristic5_System_ID_uuid	81
4.5.4.20	Service3_Characrteristic0_Battery_Level_uuid	82
4.5.4.21	Service4_Characrteristic0_UV_Index_uuid	82
4.5.4.22	Service4_Characrteristic1_Pressure_uuid	82
4.5.4.23	Service4_Characrteristic2_Temperature_uuid	82
4.5.4.24	Service4_Characrteristic3_Humidity_uuid	82
4.5.4.25	Service4_Characrteristic4_Ambient_Light_uuid	82
4.5.4.26	Service4_Characrteristic5_Sound_Level_uuid	82
4.5.4.27	Service4_Characrteristic6_Control_Point_uuid	83
4.5.4.28	Service5_Characrteristic0_Power_Source_uuid	83
4.5.4.29	Service6_Characrteristic0_ECO2_uuid	83

4.5.4.30	Service6_Characrteristic1_TVOC_uuid	83
4.5.4.31	Service6_Characrteristic2_Control_Point_uuid	83
4.5.4.32	Service7_Characrteristic0_Buttons_uuid	83
4.5.4.33	Service7_Characrteristic1_Leds_uuid	83
4.5.4.34	Service7_Characrteristic2_RGB_Leds_uuid	84
4.5.4.35	Service7_Characrteristic3_Control_Point_uuid	84
4.5.4.36	Service8_Characrteristic0_Digital_1_uuid	84
4.5.4.37	Service8_Characrteristic1_Digital_2_uuid	84
4.5.4.38	Service9_Characrteristic0_Acceleration_uuid	84
4.5.4.39	Service9_Characrteristic1_Orientation_uuid	84
4.5.4.40	Service9_Characrteristic2_Control_Point_uuid	84
4.5.4.41	ServiceA_Characrteristic0_State_uuid	85
4.5.4.42	ServiceA_Characrteristic1_Field_Strength_uuid	85
4.5.4.43	ServiceA_Characrteristic2_Control_Point_uuid	85
4.5.4.44	user_interface_service_uuid	85
4.6	LoraWan.cpp File Reference	85
4.6.1	Detailed Description	86
4.7	LoraWan.h File Reference	86
4.7.1	Detailed Description	87
4.8	main.pde File Reference	87
4.8.1	Detailed Description	88
4.8.2	Function Documentation	88
4.8.2.1	activateAlarm()	89
4.8.2.2	alarmInterruption()	89
4.8.2.3	disableInterruptionPCINT8()	90
4.8.2.4	enableInterruptionPCINT8()	90
4.8.2.5	ISR()	91
4.8.2.6	loop()	92
4.8.2.7	setup()	93
4.8.2.8	sleep()	94

4.8.2.9	stateMachine()	95
4.8.3	Variable Documentation	99
4.8.3.1	alarmFlag	99
4.8.3.2	APP_EUI	99
4.8.3.3	APP_KEY	99
4.8.3.4	BLE_Disconnected	99
4.8.3.5	bleCentral	99
4.8.3.6	buffer	99
4.8.3.7	DEVICE_EUI	99
4.8.3.8	hours	100
4.8.3.9	lorawan	100
4.8.3.10	MAC	100
4.8.3.11	minutes	100
4.8.3.12	pcint8	100
4.8.3.13	sensorsBitMap	100
4.8.3.14	state	100
Index		101

Chapter 1

Class Index

1.1 Class List

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

BLECentral	
BLECentral Class	5
Buffer	
Buffer Class	32
characteristic_t	
Struct to store a characteristic and its descriptors	38
descriptor_t	
Struct to store descriptors	39
Device_t	
Struct to save a BLE device and its related data	40
findInformationCommand_t	42
gatt_client_characteristic_descriptor_t	
Structure to identificate a descriptor	44
gatt_client_characteristic_t	
Structure to identificate a characteristic	45
gatt_client_service_t	
Structure to identificate a service	46
LoraWan	
LoraWan Class	47
readByGroupCommand_t	64
service_t	
Struct to store a service and its characteristics	66
trama_descriptor_t	
Struct to make command to discover descriptors	67
trama_grupo_t	
Struct to make command to discover services and characteristics	67

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

BLECentral.cpp	Library for managing the Bluetooth low energy module BLE112 as a Central device	69
BLECentral.h	Library for managing the Bluetooth low energy module, BLE112, as a Central device	70
Buffer.cpp	71
Buffer.h	72
defines.h	73
LoraWan.cpp	Library for managing the LoRaWAN module	85
LoraWan.h	Library for managing the LoRaWAN module	86
main.pde	Main file	87

Chapter 3

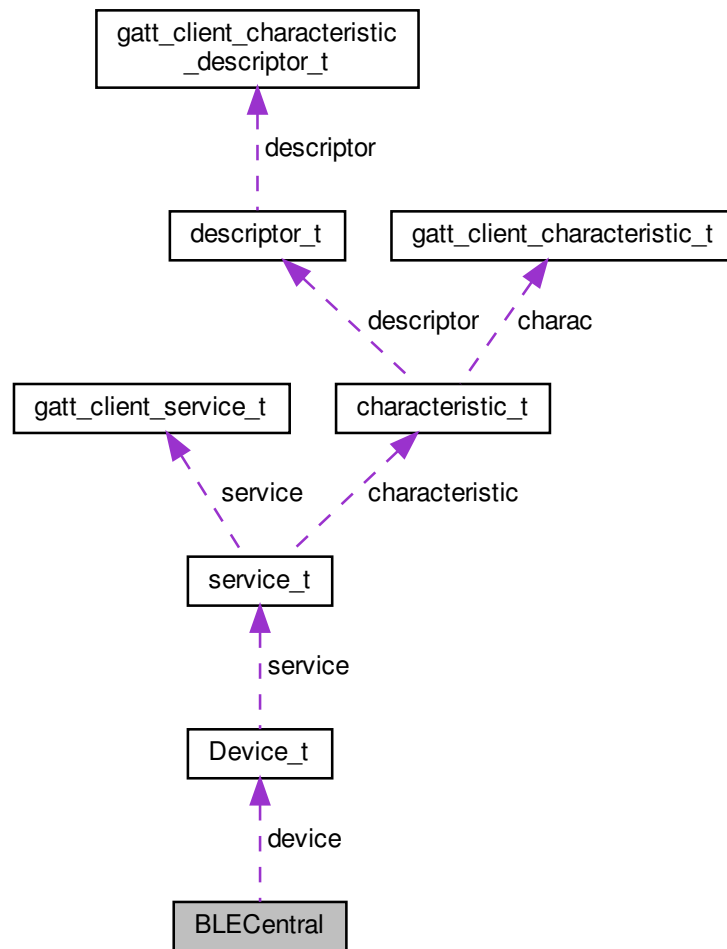
Class Documentation

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

<i>void</i>	
-------------	--

Returns

void

```

27         {
28
29     }
```

3.1.2.2 ~BLECentral()

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

class Destructor It does nothing

Parameters

<i>void</i>	
-------------	--

Returns

void

```

36         {
37     }
```

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 )
```

```

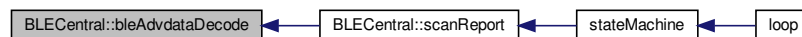
85
86         {
87     uint8_t index = 1;
88     uint8_t field_length, field_type;
89     while (index < advdata_len ) {
90         field_length = p_advdata[index];
```

```

91     field_type = p_advdata[index + 1];
92     USB.print(F("    - AVD/SR data decoding -> ad_type: "));
93     USB.print(field_type, HEX);
94     USB.print(F(", length: "));
95     USB.println(field_length, HEX);
96     if (field_type == type) {
97         memcpy(p_field_data, &p_advdata[index + 2], (field_length - 1));
98         *len = field_length - 1;
99         return 0;
100    }
101    index += field_length + 1;
102 }
103 return 1;
104 }

```

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]

```

```

1076 {
1077
1078     switch(characteristic->charac.uuid16){
1079         case 0x2A00 :
1080             memcpy(characteristic->charac.uuid128,
1081 Service0_Characteristic0_Device_Name_uuid, 16);
1082             break;
1083         case 0x2A01 :
1084             memcpy(characteristic->charac.uuid128,
1085 Service0_Characteristic1_Appearance_uuid, 16);
1086             break;
1087         case 0x2A05 :
1088             memcpy(characteristic->charac.uuid128,
1089 Service1_Characteristic0_Service_Changed_uuid, 16);
1090             break;
1091         case 0x2A29 :
1092             memcpy(characteristic->charac.uuid128,
1093 Service2_Characteristic0_Manufacturer_Name_uuid, 16);
1094             break;
1095         case 0x2A24 :
1096             memcpy(characteristic->charac.uuid128,
1097 Service2_Characteristic1_Model_Number_uuid, 16);
1098             break;
1099         case 0x2A25 :
1100             memcpy(characteristic->charac.uuid128,
1101 Service2_Characteristic2_Serial_Number_uuid, 16);
1102             break;
1103         case 0x2A27 :
1104             memcpy(characteristic->charac.uuid128,
1105 Service2_Characteristic3_Hardware_Revision_uuid, 16);
1106             break;
1107         case 0x2A26 :
1108             memcpy(characteristic->charac.uuid128,
1109 Service2_Characteristic4_Firmware_Revision_uuid, 16);

```

```

1109         break;
1110
1111         case 0x2A23 :
1112             memcpy(characteristic->charac.uuid128,
1113 Service2_Characrteristic5_System_ID_uuid, 16);
1114             break;
1115
1116         case 0x2A19 :
1117             memcpy(characteristic->charac.uuid128,
1118 Service3_Characrteristic0_Battery_Level_uuid, 16);
1119             break;
1120
1121         case 0x2A76 :
1122             memcpy(characteristic->charac.uuid128,
1123 Service4_Characrteristic0_UV_Index_uuid, 16);
1124             break;
1125
1126         case 0x2A6D :
1127             memcpy(characteristic->charac.uuid128,
1128 Service4_Characrteristic1_Pressure_uuid, 16);
1129             break;
1130
1131         case 0x2A6E :
1132             memcpy(characteristic->charac.uuid128,
1133 Service4_Characrteristic2_Temperature_uuid, 16);
1134             break;
1135
1136         case 0x2A6F :
1137             memcpy(characteristic->charac.uuid128,
1138 Service4_Characrteristic3_Humidity_uuid, 16);
1139             break;
1140
1141         case 0x2A56 :
1142             memcpy(characteristic->charac.uuid128,
1143 Service8_Characrteristic0_Digital_1_uuid, 16);
1144             break;
1145     }
1146 }

```

Here is the caller graph for this function:



3.1.3.3 configureScanner()

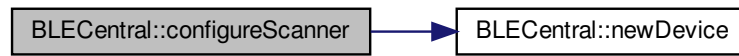
```

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

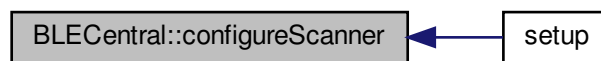
170
171 {
172     BLE.setDiscoverMode(discoverMode);
173     BLE.setTXPower(txPower);
174     BLE.setScanningParameters(scanInterval, scanWindow, scanningFilter );
175     newDevice(); //create Device_t struct to store device relate data.
176     #if DEBUG >= 1
177         BLE.getScanningParameters();
178         USB.println(F("_____Central is configure and redy to start_____"));
179     #endif
180 }

```

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[ ] )

238                                     {
239
240     uint8_t response;
241     USB.println(F("_____Connecting... "));
242     response = BLE.connectDirect(mac);
243     if (response == 1){
244         USB.println(F("_____Connected"));
245         USB.print(F("          -connection_handle: "));
246         USB.println(BLE.connection_handle, DEC);
247         USB.print(F(""));
248     }else if (response == 0){
249         USB.println(F("Invalid parameters"));
250     }else{
251         USB.print(F("Connection ERROR = "));
252         USB.println(response, DEC);
253     }
254     return response;
255 }
  
```

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 )

269         {
270
271     uint16_t response = 0;
272
273     USB.println(F("_____Connecting... "));
274     response = BLE.connectDirect(mac, conn_interval_min, conn_interval_max, timeout, latency); //default
connectDirect(BLEAddress, 60, 76, 100, 0);
275     if (response == 1){
276         USB.println(F("_____Connected"));
277         USB.print(F("          -connection_handle: "));
278         USB.println(BLE.connection_handle, DEC);
279         USB.print(F(""));
280         return 1;
281     }else{
282         USB.println(F("NOT Connected"));
283         return 0;
284     }
285
286 }
```

3.1.3.6 disconnect()

```

uint16_t BLECentral::disconnect (
    uint8_t connectionHandle )

299         {
300     uint16_t response = 0;
301     response = BLE.disconnect(connectionHandle);
302     if (response == 0){
303         USB.println(F("_____Disconnected"));
304     }else if (response == 1){
305         USB.print(F("Connection handle is not right"));
306     }else{
307         USB.print(F("Disconnect, Error = "));
308         USB.println(response, HEX);
309     }
310     return response;
311 }
```

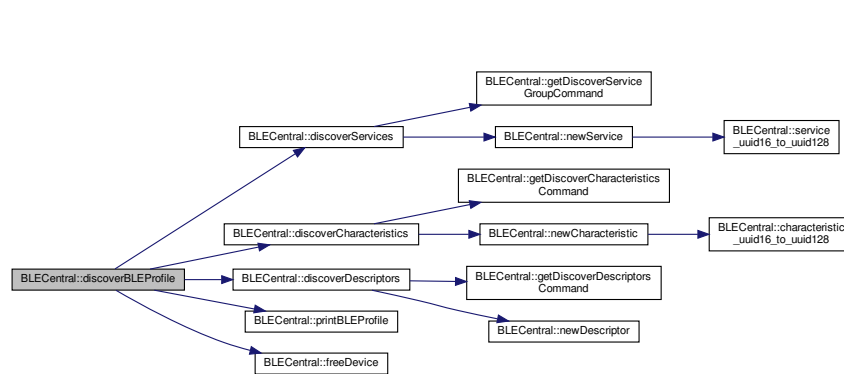
3.1.3.7 discoverBLEProfile()

```

uint8_t BLECentral::discoverBLEProfile ( )

456         {
457
458     if(discoverServices()){
459         if(discoverCharacteristics()){
460             if(discoverDescriptors()){
461                 printBLEProfile();
462                 return 1;
463             }
464         }
465     }
466     freeDevice();
467     return 0;
468
469 }
```


Here is the call graph for this function:



Here is the caller graph for this function:



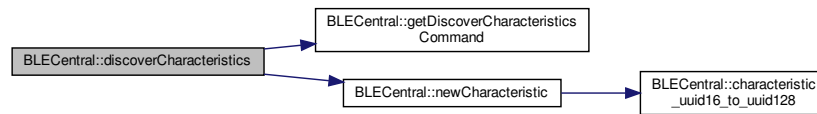
3.1.3.8 discoverCharacteristics()

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

```

354                                     {
355
356     uint16_t event;
357     readByGroupCommand_t command;
358     command = getDiscoverCharacteristicsCommand();
359     USB.println(F("_____Discovering Characteristics... "));
360     for(uint8_t numSer = 0; numSer < device->numberOfServices; numSer++){
361         command.startFirstAttributeHandle = ((device->
362 service[numSer].service.start_group_handle)+1);
363         command.endLastAttributeHandle = ((device->
364 service[numSer].service.end_group_handle)-1);
365         BLE.sendCommand((uint8_t *)&command, command.t_length+1);
366         BLE.readCommandAnswer();
367         event = BLE_EVENT_ATTCLIENT_ATTRIBUTE_VALUE;
368         while(event == BLE_EVENT_ATTCLIENT_ATTRIBUTE_VALUE) {
369             event = BLE.waitEvent(1000);
370             if(event == BLE_EVENT_ATTCLIENT_ATTRIBUTE_VALUE) {
371                 newCharacteristic(&device->service[numSer], BLE.event);
372             } else if(event == 0) {
373                 USB.println(F("The connection to the peripheral device has been disconnected"));
374                 return 0;
375             }
376         }
377     }
378     USB.println(F("_____Discovering Characteristics completed"));
379     USB.println(F(""));
380     return 1;
381 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.9 discoverDescriptors()

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

```

391                                     {
392     uint8_t numSer, numCar;
393     findInformationCommand_t command;
394     command = getDiscoverDescriptorsCommand();
395     uint16_t servEndHandle, carcValueHandle, nexCarcStarHandle;
396     USB.println(F("_____Discovering Descriptors... "));
397     for(numSer = 0; numSer < device->numberOfServices; numSer++) {
398         servEndHandle = device->service[numSer].service.
399         end_group_handle;
400         for(numCar = 0; numCar < device->service[numSer].
401         numberOfCharacteristics; numCar++){
402             carcValueHandle = device->service[numSer].characteristic[numCar].
403             charac.value_handle;
404             if( (numCar+1) < device->service[numSer].
405             numberOfCharacteristics){
406                 nexCarcStarHandle = device->service[numSer].characteristic[numCar+1].
407                 charac.start_handle;
408                 carcValueHandle = carcValueHandle+1;
409                 while((carcValueHandle)!= nexCarcStarHandle){
410                     command.startFirstAttributeHandle = carcValueHandle ;
411                     command.endLastAttributeHandle = carcValueHandle;
412                     BLE.sendCommand((uint8_t *)&command, command.t_length+1);
413                     BLE.readCommandAnswer();
414                     BLE.waitEvent(1000);
415                     newDescriptor(&device->service[numSer].
416                     characteristic[numCar],BLE.event);
417                     carcValueHandle = carcValueHandle+1;
418                 }
419             }else if(servEndHandle != 0xFFFF){
420                 while((carcValueHandle)!= servEndHandle){
421                     carcValueHandle = carcValueHandle+1;
422                     command.startFirstAttributeHandle = carcValueHandle ;
423                     command.endLastAttributeHandle = carcValueHandle;
424                     BLE.sendCommand((uint8_t *)&command, command.t_length+1);
425                     BLE.readCommandAnswer();
426                     BLE.waitEvent(1000);
427                     newDescriptor(&device->service[numSer].
428                     characteristic[numCar],BLE.event);
429                 }
430             }else{
431                 uint8_t evento;
432                 evento = BLE_EVENT_ATTCLIENT_FIND_INFORMATION_FOUND;
433                 while(evento == BLE_EVENT_ATTCLIENT_FIND_INFORMATION_FOUND) {

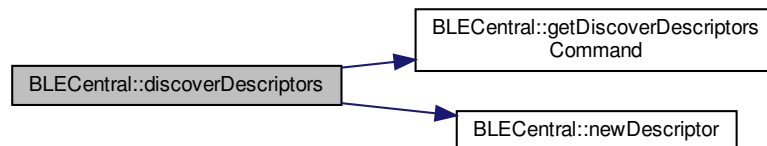
```

```

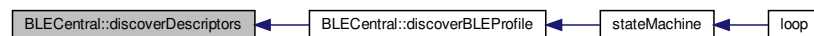
427         carcValueHandle = carcValueHandle+1;
428         command.startFirstAttributeHandle = carcValueHandle ;
429         command.endLastAttributeHandle = carcValueHandle;
430         BLE.sendCommand((uint8_t *)&command, command.t_length+1);
431         BLE.readCommandAnswer();
432         evento = BLE.waitEvent(1000);
433         if(evento == BLE_EVENT_ATTCLIENT_FIND_INFORMATION_FOUND) {
434             newDescriptor (&device->service[numSer] .
characteristic[numCar], BLE.event);
435         }else if(evento == BLE_EVENT_ATTCLIENT_PROCEDURE_COMPLETED) {
436             device->service[numSer].service.
end_group_handle = carcValueHandle -1;
437         }
438     }
439 }
440 }
441 }
442 USB.println(F("_____Discovering Descriptors completed"));
443 USB.println(F(""));
444 return 1;
445 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.10 discoverServices()

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

```

322         {
323
324         uint16_t event = 0;
325         readByGroupCommand_t command;
326         command = getDiscoverServiceGroupCommand();
327         USB.println(F("_____Discovering Services... "));
328         BLE.sendCommand((uint8_t *)&command, command.t_length+1);
329         BLE.readCommandAnswer();
330         while(event != BLE_EVENT_ATTCLIENT_PROCEDURE_COMPLETED) {
331             event = BLE.waitEvent(1000);
332             if(event == BLE_EVENT_ATTCLIENT_GROUP_FOUND) {
333                 newService( BLE.event);
334             }else if(event == 0){//No hay evento
335                 USB.println(F("The connection to the peripheral device has been disconnected"));

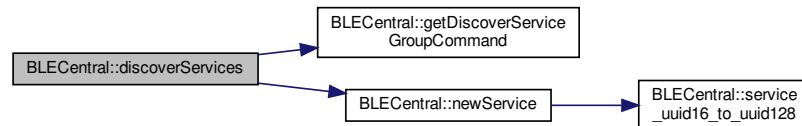
```

```

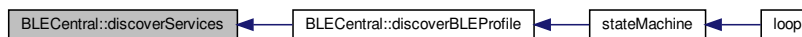
336         return 0;
337     }
338 }
339 USB.print(F("_____Discovering Services completed ("));
340 USB.print(device->numberOfServices, DEC);
341 USB.println(F(")"));
342 return 1;
343 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.11 enableNotification()

```

uint8_t BLECentral::enableNotification (
    uint8_t * uuid128 )

```

```

598         {
599
600     uint16_t response;
601     char notify[2] = "1";
602     response = BLE.attributeWrite(BLE.connection_handle, (uuid128ToHandle(uuid128) + 1),
notify);
603     if (response == 0){
604         USB.println(F("_____Notification enable"));
605         USB.print(F(" -For UUID128: "));
606         for(uint8_t i=0; i<16; i++){
607             USB.print(uuid128[i], HEX);
608             USB.print(" ");
609         }
610         USB.println(F(""));
611         return 1;
612     }else{
613         USB.println(F("_____Failed subscribing"));
614         USB.println(F(""));
615         return 0;
616     }
617 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



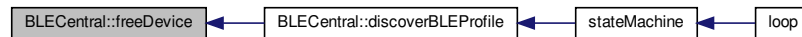
3.1.3.12 freeDevice()

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

```

948         {
949     #if DEBUG >= 1
950         USB.print(F("Free Memory(Before freeDevice):"));
951         USB.println(freeMemory());
952     #endif
953     uint8_t numSer;
954     uint8_t numCar;
955     for(numSer = 0; numSer < device->numberOfServices; numSer++){
956         for(numCar = 0; numCar < device->service[numSer].
957         numberOfCharacteristics; numCar++){
958             if(device->service[numSer].characteristic[numCar].
959             descriptor != NULL){
960                 free(device->service[numSer].characteristic[numCar].
961                 descriptor);
962             }
963         }
964     }
965     for(uint8_t numSer = 0; numSer < device->numberOfServices; numSer++){
966         if(device->service[numSer].characteristic != NULL){
967             free(device->service[numSer].characteristic);
968         }
969     }
970     if(device->service != NULL){
971         free(device->service);
972         device->numberOfServices = 0;
973         device->service = NULL;
974     }
975     //~ free(device);
976     #if DEBUG >= 1
977         USB.print(F("Free Memory(After freeDevice):"));
978         USB.println(freeMemory());
979     #endif
980 }
  
```

Here is the caller graph for this function:



3.1.3.13 getConnectionHandler()

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

```

660                                     {
661     return BLE.connection_handle;
662 }
```

3.1.3.14 getConnectionStatus()

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

```

672                                     { //idea limpiar device aqui usuario no deve tocarlo
673     uint8_t status;
674
675     status = BLE.getStatus(BLE.connection_handle);
676     if( status == 0 ) { //Se ha desconectado
677         // central.limpiarPerfil();
678         USB.print(F("The connection has been disconnected, satus: "));
679         USB.println(status, DEC);
680     } else if (status == 1){
681         USB.print(F("The device is connected, satus: "));
682         USB.println(status, DEC);
683     } else{
684         USB.print(F("The module does not response, satus: "));
685         USB.println(status, DEC);
686     }
687     return status;
688 }
```

Here is the caller graph for this function:



3.1.3.15 getDiscoverCharacteristicsCommand()

```

readByGroupCommand_t BLECentral::getDiscoverCharacteristicsCommand ( ) [private]

734                                     {
735
736     readByGroupCommand_t command;
737     command.t_length = 12;
738     command.messageType = 0;
739     command.payloadLength = 8;
740     command.classID = 4;
741     command.commandID = 2;
742     command.Connectionhandle = BLE.connection_handle;
743     command.startFirstAttributeHandle = 0;
744     command.endLastAttributeHandle = 0;
745     command.uuidLength = 2;
746     command.uuid = 0x2803;
747     return command;
748 }

```

Here is the caller graph for this function:



3.1.3.16 getDiscoverDescriptorsCommand()

```

findInformationCommand_t BLECentral::getDiscoverDescriptorsCommand ( ) [private]

757                                     {
758     /*/~ Byte Type Name Description
759     //~ 0 0x00 hilen Message type: command
760     //~ 1 0x05 lolen Minimum payload length
761     //~ 2 0x04 class Message class: Attribute Client
762     //~ 3 0x03 method Message ID
763     //~ 4 uint8 connection Connection handle
764     //~ 5 - 6 uint16 start First attribute handle
765     //~ 7 - 8 uint16 end Last attribute handle
766
767     */
768     findInformationCommand_t command;
769     command.t_length = 9;
770     command.messageType = 0;
771     command.payloadLength = 5;
772     command.classID = 4;
773     command.commandID = 3;
774     command.Connectionhandle = BLE.connection_handle;
775     return command;
776 }

```

Here is the caller graph for this function:



3.1.3.17 getDiscoverServiceGroupCommand()

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

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

```

702                                     {
703     /* Byte Type Name Description
704     //~ 0 0x00 hilen Message type: command
705     //~ 1 0x06 lolen Minimum payload length
706     //~ 2 0x04 class Message class: Attribute Client
707     //~ 3 0x01 method Message ID
708     //~ 4 uint8 connection Connection Handle
709     //~ 5 - 6 uint16 start First requested handle number
710     //~ 7 - 8 uint16 end Last requested handle number
711     //~ 9 uint8array uuid Group UUID to find
712
713     */
714     readByGroupCommand_t command;
715     command.t_length = 12;
716     command.messageType = 0;
717     command.payloadLenght = 8;
718     command.classID = 4;
719     command.commandID = 1;
720     command.Connectionhandle = BLE.connection_handle;
721     command.startFirstAttributeHandle = 0x0001;
722     command.endLastAttributeHandle = 0xFFFF;
723     command.uuidLenght = 2;
724     command.uuid = 0x2800;
725     return command;
726 }
```

Here is the caller graph for this function:



3.1.3.18 newCharacteristic()

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

```

861                                     {
862     /* Attribute Value--attclient
863     //~ 0 0x80 hilen Message type: event
864     //~ 1 0x05 lolen Minimum payload length
865     //~ 2 0x04 class Message class: Attribute Client
866     //~ 3 0x05 method Message ID
867     //~ 4 uint8 connection Connection handle
868 }
```

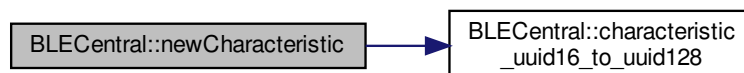


```

868     //~ 5 - 6 uint16 atthandle Attribute handle
869     //~ 7 uint8 type Attribute type
870     //~ 8 uint8array value Attribute value (data)//yo de aqui
871     */
872     uint8_t contador = 0;
873     uint8_t posicion = 0;
874     service->numberOfCharacteristics++;
875     characteristic_t* tmp = (characteristic_t*)malloc(sizeof(
characteristic_t)*service->numberOfCharacteristics);
876     characteristic_t* basura;
877     for(contador = 0; contador < (service->numberOfCharacteristics-1); contador++){
878         tmp[contador] = service->characteristic[contador];
879     }
880     tmp[contador].numberOfDescriptors = 0;
881     tmp[contador].descriptor = NULL;
882     tmp[contador].charac.start_handle = ((uint16_t)discoveredCharacteristic[6] << 8) |
discoveredCharacteristic[5];
883     tmp[contador].charac.value_handle = ((uint16_t)discoveredCharacteristic[11] << 8) |
discoveredCharacteristic[10];
884     tmp[contador].charac.properties = ((uint8_t)discoveredCharacteristic[9]);
885     tmp[contador].charac.uuid16 = 0;
886     memset(&tmp[contador].charac.uuid128, 0x00, 16 );
887     posicion = ((discoveredCharacteristic[8]) - 3);
888     if(posicion == 2){
889         tmp[contador].charac.uuid16 = ((uint16_t)discoveredCharacteristic[13] << 8) |
discoveredCharacteristic[12];
890         characteristic_uuid16_to_uuid128(&tmp[contador]);
891     }else{
892         for(int s = 0; s < 16 ; s++){
893             tmp[contador].charac.uuid128[s]= discoveredCharacteristic[27-s];
894         }
895     }
896     basura = service->characteristic;
897     service->characteristic = tmp;
898     if(basura!= NULL){
899         free(basura);
900         tmp = NULL;
901     }
902 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.19 newDescriptor()

```

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

```

```

912                                     {
913     /* Find Information Found--attclient
914
915     //~ Byte Type Name Description
916
917     //~ 0 0x80 hilen Message type: event
918
919     //~ 1 0x04 lolen Minimum payload length
920
921     //~ 2 0x04 class Message class: Attribute Client
922
923     //~ 3 0x04 method Message ID
924
925     //~ 4 uint8 connection Connection handle
926
927     //~ 5 - 6 uint16 chrhandle Characteristics handle
928
929     //~ 7 uint8array uuid Characteristics type (UUID)
930
931     */
932     uint16_t flag1;
933     uint8_t contador = 0;
934     characteristic->numberOfDescriptors++;
935     descriptor_t* tmp = (descriptor_t*)malloc(sizeof(
936     descriptor_t)*characteristic->numberOfDescriptors);
937     descriptor_t* basura;
938     for(contador = 0; contador <(characteristic->numberOfDescriptors-1); contador++){
939         tmp[contador] = characteristic->descriptor[contador];
940     }
941     tmp[contador].descriptor.handle = ((uint16_t)discoveredDescriptor[6] << 8) |
942     discoveredDescriptor[5];
943     tmp[contador].descriptor.uuid16 = ((uint16_t)discoveredDescriptor[9] << 8) |
944     discoveredDescriptor[8];
945
946     basura = characteristic->descriptor;
947     characteristic->descriptor = tmp;
948     if(basura!= NULL){
949         free(basura);
950         tmp = NULL;
951     }
952 }
953 }

```

Here is the caller graph for this function:



3.1.3.20 newDevice()

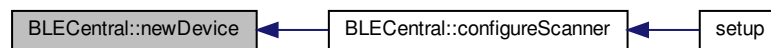
```
void BLECentral::newDevice ( ) [private]
```

```

784                                     {
785     #if DEBUG >= 1
786
787         USB.print(F("Free Memory(Before New Device):"));
788         USB.println(freeMemory());
789     #endif
790
791     device = (Device_t*)malloc(sizeof(Device_t));
792     device->numberOfServices = 0;
793     device->service = NULL;
794     USB.println(F("_____Storage to manage new device started_____"));
795
796     #if DEBUG >= 1
797
798         USB.print(F("Free Memory(After New Device):"));
799         USB.println(freeMemory());
800     #endif
801 }

```

Here is the caller graph for this function:



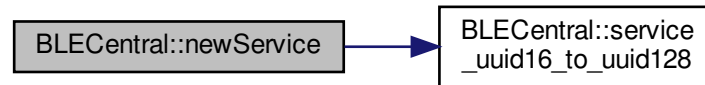
3.1.3.21 newService()

```

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

808                                     {
809     /* Group Found--attclient
810     //~ Byte Type Name Description
811     //~ 0 0x80 hilen Message type: event
812     //~ 1 0x06 lolen Minimum payload length
813     //~ 2 0x04 class Message class: Attribute Client
814     //~ 3 0x02 method Message ID
815     //~ 4 uint8 connection Connection handle
816     //~ 5 - 6 uint16 start Starting handle
817     //~ 7 - 8 uint16 end Ending handle
818     //~ Note: "end" is a reserved word and in BGScript so "end" cannot be used as
819     //~ such.
820     //~ 9 uint8array uuid UUID of a service
821     //~ Length is 0 if no services are found.
822     */
823     int contador = 0;
824     device->numberOfServices++;
825     service_t *tmp = (service_t*)malloc(sizeof(service_t)*
device->numberOfServices);
826     service_t *basura;
827     for(contador = 0; contador < (device->numberOfServices-1); contador++){
828         tmp[contador] = device->service[contador];
829     }
830     tmp[contador].numberOfCharacteristics = 0;
831     tmp[contador].characteristic = NULL;
832     tmp[contador].service.start_group_handle = ((uint16_t)discoveredService[6] <<
8) | discoveredService[5];
833     tmp[contador].service.end_group_handle = ((uint16_t)discoveredService[8] << 8) |
discoveredService[7];
834     tmp[contador].service.uuid16 = 0;
835     memset(&tmp[contador].service.uuid128, 0x00, 16 );
836     if(discoveredService[9]== 2){
837         tmp[contador].service.uuid16 = ((uint16_t)discoveredService[11] << 8) |
discoveredService[10];
838         service_uuid16_to_uuid128(&tmp[contador]);
839     }else{
840         for(uint8_t indice = 0; indice < 16 ; indice++){
841             tmp[contador].service.uuid128[indice] = ((uint16_t)discoveredService[25-indice]);
842         }
843     }
844     basura = device->service;
845     device->service = tmp;
846     if(basura!= NULL){
847         free(basura);
848         tmp = NULL;
849     }
850 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.22 printBLEProfile()

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

```

477         {
478     uint8_t numSer, numCar, numDesc;
479     USB.println(F("_____BLE Profile_____"));
480     USB.println(F(""));
481     for(numSer = 0; numSer < device->numberOfServices; numSer++){
482         USB.print(F("* Service number "));
483         USB.println( numSer, HEX );
484         USB.print(F("    - Service start handle: "));
485         USB.println(device->service[numSer].service.
start_group_handle, HEX);
486         USB.print(F("    - Service end handle: "));
487         USB.println(device->service[numSer].service.
end_group_handle, HEX);
488         USB.print(F("    - Service uuid16: "));
489         USB.println(device->service[numSer].service.uuid16, HEX);
490         USB.print(F("    - Service uuid128: "));
491         for(int p = 0; p < 16; p++){
492             USB.print(device->service[numSer].service.
uuid128[p], HEX);
493             USB.print(" ");
494         }
495         USB.println("");
496         USB.print(F("    * Service number of characteristics: "));
497         USB.print(device->service[numSer].numberOfCharacteristics, DEC)
;
498         USB.println(F(""));
499         for(numCar = 0; numCar < device->service[numSer].
numberOfCharacteristics; numCar++){
500             USB.print(F("* Service "));
501             USB.print( numSer, HEX );
502             USB.print(F("    - Characteristic: "));
503             USB.println( numCar, HEX );
504             USB.print(F("    - Characteristic start handle: "));
505             USB.print(device->service[numSer].characteristic[numCar].
charac.start_handle, HEX);
506             USB.println("");
507             USB.print(F("    - Characteristic value handle: "));
508             USB.print(device->service[numSer].characteristic[numCar].
charac.value_handle , HEX);
509             USB.println("");

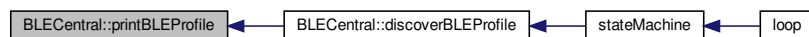
```

```

510         USB.print(F(" - Characteristic properties: "));
511         USB.print(device->service[numSer].characteristic[numCar].
charac.properties, HEX);
512         USB.println("");
513         USB.print(F(" - Characteristic uuid16: "));
514         USB.print(device->service[numSer].characteristic[numCar].
charac.uuid16, HEX);
515         USB.println(F(""));
516         USB.print(F(" - Characteristic uuid128: "));
517         for(int p = 0; p < 16 ; p++){
518             USB.print(device->service[numSer].characteristic[numCar].
charac.uuid128[p], HEX);
519             USB.print(" ");
520         }
521         USB.println(" ");
522         for(numDesc = 0; numDesc < device->service[numSer].
characteristic[numCar].numberOfDescriptors; numDesc++){
523             USB.print(F(" - Characteristic: "));
524             USB.print(numCar, HEX );
525             USB.print(F(" - Descriptor: "));
526             USB.println(numDesc, HEX );
527             USB.print(F(" - Descriptor handle: "));
528             USB.println(device->service[numSer].characteristic[numCar].
descriptor[numDesc].descriptor.handle, HEX );
529             USB.print(F(" - Descriptor uuid16: "));
530             USB.println(device->service[numSer].characteristic[numCar].
descriptor[numDesc].descriptor.uuid16, HEX );
531         }
532     }
533 }
534 #if DEBUG >= 1
535     USB.print(F("Free Memory(After creating the BLE profile:"));
536     USB.println(freeMemory());
537     USB.println(F(" "));
538 #endif
539 }

```

Here is the caller graph for this function:



3.1.3.23 readAttribute()

```

uint8_t * BLECentral::readAttribute (
    uint8_t * uuid128 )

```

```

548     {
549         USB.println(F("BLE Central read Attribute "));
550         USB.print(F(" -UUID128: "));
551         for(uint8_t i=0; i<16; i++){
552             USB.print(uuid128[i], HEX);
553             USB.print(" ");
554         }
555         USB.println(F(""));
556         USB.print(F(" -Handle: "));
557         USB.println(uuid128ToHandle(uuid128), HEX);
558         USB.println(F("")); //probado
559         BLE.attributeRead(BLE.connection_handle, uuid128ToHandle(uuid128));
560         return BLE.attributeValue;
561     }
562 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



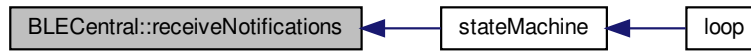
3.1.3.24 receiveNotifications()

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

```

626                                     {
627     uint16_t event;
628     uint16_t handler;
629     USB.println(F("Waiting events..."));
630     event = BLE.waitEvent(1000);
631     if (event == BLE_EVENT_ATTCLIENT_ATTRIBUTE_VALUE){
632         USB.println(F("Notification received"));
633         /* attribute value event structure:
634
635         value |
636
637         Length: |      1      |      1      |      1      |      1      |      1      |      2      |      8      |
638         n      |
639
640         Example: |      80      |      05      |      04      |      05      |      00      |      2c 00      |      x      |
641         n      | */
642         handler = ((uint16_t)BLE.event[6] << 8) | BLE.event[5];
643         USB.print(F(" -Attribute with handler "));
644         USB.print(handler, DEC);
645         USB.println(F(" has changed "));
646         USB.print(F(" -Attribute value: "));
647         BLE.event[0] = BLE.event[8];
648         for(uint8_t i = 0; i < BLE.event[8]; i++){
649             USB.printHex(BLE.event[i+9]);
650             BLE.event[i+1] = BLE.event[i+9];
651         }
652         USB.println(F(""));
653         return BLE.event;
654     }
655 }
  
```

Here is the caller graph for this function:



3.1.3.25 scanReport()

```

uint8_t BLECentral::scanReport (
    char * nameToSearch )

{
    114
    115     USB.println(F(""));
    116     USB.println(F("* BLE scan report: "));
    117     USB.print(F(" - Peer device address: "));
    118     Utils.hex2str(BLE.BLEDev.mac, device->mac, 6);
    119     USB.println(device->mac);
    120     USB.print(F(" - RSSI: "));
    121     USB.print(BLE.BLEDev.rssi, DEC);
    122     USB.println(F(" dBm "));
    123     USB.print(F(" - Advertising data packet("));
    124     USB.print(BLE.BLEDev.advData[0], DEC);
    125     USB.print(F(" Bytes): "));
    126     for (int index = 1; index < BLE.BLEDev.advData[0]; index++) {
    127         USB.print(BLE.BLEDev.advData[index], HEX);
    128         USB.print(F(" "));
    129     }
    130     USB.println(F(" "));
    131
    132     uint8_t len;
    133     uint8_t adv_name[31];
    134
    135     if (0x00 == bleAdvdataDecode(0x09, BLE.BLEDev.advData[0], BLE.BLEDev.advData, &len,
    adv_name)) {
    136         USB.print(F(" The length of Complete Local Name : "));
    137         USB.println(len, HEX);
    138         USB.print(F(" The Complete Local Name is : "));
    139         adv_name[len] = 0;
    140         USB.println((const char *)adv_name);
    141         USB.println(F(""));
    142         if (0x00 == memcmp(adv_name, nameToSearch, len)) {
    143             USB.println(F("* Thunder Sense #02735 found"));
    144             return 1;
    145         }
    146     }
    147     return 0;
    148 }
  
```

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]
```

```

1041                                     {
1042     switch(service->service.uuid16){
1043     case 0x1800 :
1044         memcpy(service->service.uuid128,
1045 generic_access_service_uuid, 16);
1046         break;
1047     case 0x1801 :
1048         memcpy(service->service.uuid128,
1049 generic_attribute_service_uuid, 16);
1050         break;
1051     case 0x180A :
1052         memcpy(service->service.uuid128,
1053 device_information_service_uuid, 16);
1054         break;
1055     case 0x180F :
1056         memcpy(service->service.uuid128, battery_service_uuid, 16);
1057         break;
1058     case 0x181A :
1059         memcpy(service->service.uuid128,
1060 environmental_sensing_service_uuid, 16);
1061         break;
1062     case 0x1815 :
1063         memcpy(service->service.uuid128,
1064 automation_io_service_uuid, 16);
1065         break;
1066     }
1067 }
```

Here is the caller graph for this function:



3.1.3.27 startScanning()

```

uint16_t BLECentral::startScanning (
    uint8_t time )

216
217     {
218     uint16_t response = 0;
219     response = BLE.scanNetwork(time);
220     if(response == 0){
221         USB.println(F("_____Device found"));
222     }else{
223         USB.print(F("Scanner, ERROR = "));
224         USB.println(response, DEC);
225     }
226     return response;
227 }

```

3.1.3.28 startScanningDevice()

```

uint16_t BLECentral::startScanningDevice (
    char mac[] )

192
193     {
194     uint16_t response = 0;
195     response = BLE.scanDevice(mac); //devuelve 1
196     if(response == 1){
197         USB.println(F("_____Device found"));
198     }else if (response == 0){
199         USB.println(F("_____Device NOT found"));
200     }else{
201         USB.print(F("Scanner, ERROR = "));
202         USB.println(response, DEC);
203     }
204     return response;
205 }

```

Here is the caller graph for this function:



3.1.3.29 turnOffModule()

```

void BLECentral::turnOffModule ( )

68
69     {
70     BLE.OFF();
71 }

```

3.1.3.30 turnOnModule()

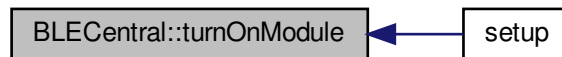
```

int8_t BLECentral::turnOnModule (
    uint8_t socket )

{
49
50     int8_t response;
51     response = BLE.ON(socket);
52     if(response == 0){
53         USB.println(F("BLE module switch on Ok "));
54     }else{
55         USB.print(F("BLE module switch, ERROR = "));
56         USB.println(response, DEC);
57     }
58     return response;
59 }

```

Here is the caller graph for this function:



3.1.3.31 uuid128ToHandle()

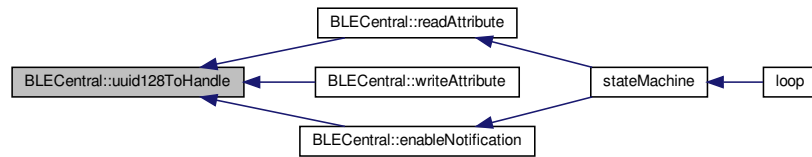
```

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

{
1014
1015
1016     uint8_t numSer;
1017     uint8_t numCar;
1018     //uint16_t uuid16;
1019     //uuid16 = (((uint16_t)uuid128[2] << 8) + ((uint16_t)(uuid128[3])));
1020
1021     for(numSer = 0; numSer < device->numberOfServices; numSer++){
1022         if(0x00 == memcmp(device->service[numSer].service.
1023             uuid128, uuid128, 16)){//| (device->servicio[numSer].service.uuid16 == uuid16)
1024             return device->service[numSer].service.
1025                 start_group_handle;
1026         }
1027         for(numCar = 0; numCar < device->service[numSer].
1028             numberOfCharacteristics; numCar++){
1029             if(0x00 == memcmp(device->service[numSer].characteristic[numCar].
1030                 charac.uuid128, uuid128, 16)){//|
1031                 (device->servicio[numSer].caracteristica[numCar].charac.uuid16 == uuid16)
1032                 return device->service[numSer].characteristic[numCar].
1033                     charac.value_handle;
1034         }
1035     }
1036     return 0;
1037 }

```

Here is the caller graph for this function:



3.1.3.32 uuid16ToHandle()

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

```

986                                     {
987     uint8_t numSer;
988     uint8_t numCar;
989     uint8_t numDesc;
990     for(numSer = 0; numSer < device->numberOfServices; numSer++){
991         if(device->service[numSer].service.uuid16 == uuid16){
992             return device->service[numSer].service.
start_group_handle;
993         }
994         for(numCar = 0; numCar < device->service[numSer].
numberOfCharacteristics; numCar++){
995             if(device->service[numSer].characteristic[numCar].
charac.uuid16 == uuid16){
996                 return device->service[numSer].characteristic[numCar].
charac.value_handle;
997             }
998             for(numDesc = 0; numDesc < device->service[numSer].
characteristic[numCar].numberOfDescriptors; numDesc++){
1000                 if(device->service[numSer].characteristic[numCar].
descriptor[numDesc].descriptor.uuid16 == uuid16){
1001                     return device->service[numSer].characteristic[numCar].
descriptor[numDesc].descriptor.handle;
1002                 }
1003             }
1004         }
1005     }
1006 }
```

3.1.3.33 writeAttribute()

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

```

577                                     {
578     uint16_t response;
579     USB.println(F("Writing attribute.. "));
580     response = BLE.attributeWrite(connection, uuid128ToHandle(uuid128), data, length);
581     if (response == 0){
582         USB.println(F("Write attribute OK"));
583     }else{
584         USB.println(F("Write ERROR = "));
585     }
586     return response;
587 }

```

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

<code>void</code>	
-------------------	--

Returns

`void`

```
25         {
26
27     }
```

3.2.2.2 `~Buffer()`

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

class Destructor It does nothing

Parameters

<code>void</code>	
-------------------	--

Returns`void`

```
34         {  
35     }
```

3.2.3 Member Function Documentation

3.2.3.1 clearDataToSend()

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

```
117     {  
118     memset(dataToSend, 0x00, sizeof(dataToSend));  
119     dataToSend_Index = 0;  
120 }
```

Here is the caller graph for this function:



3.2.3.2 clearNetworkReceivedData()

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

```
185     {  
186     memset(networkReceivedData, 0x00, sizeof(  
187     networkReceivedData));
```

Here is the caller graph for this function:



3.2.3.3 getDataToSend()

```

uint8_t * Buffer::getDataToSend ( )

79         {
80
81     uint8_t currentIndex = 0;
82     uint8_t elemtLenght;
83     USB.println(F("_____Data stored in the buffer to send:"));
84     while(currentIndex < dataToSend_Index){
85         USB.print(F(" Element Type: "));
86         USB.print( dataToSend[currentIndex++], DEC );
87         elemtLenght = dataToSend[currentIndex++];
88         USB.print(F(", Element Lenght: "));
89         USB.print( elemtLenght, DEC );
90         USB.print(F(", Element Data: "));
91         for(uint8_t i=0; i<elemtLenght; i++){
92             USB.print( dataToSend[currentIndex++], HEX );
93             USB.print(F(" "));
94         }
95         USB.println(F(""));
96     }
97     return dataToSend;
98 }

```

Here is the caller graph for this function:



3.2.3.4 getDataToSendSize()

```

uint8_t Buffer::getDataToSendSize ( )

106         {
107     return dataToSend_Index;
108 }

```

Here is the caller graph for this function:



3.2.3.5 getNetworkReceivedData()

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

```
172                                     {
173     uint8_t value;
174     value = networkReceivedData[index];
175     return value;
176 }
```

Here is the caller graph for this function:



3.2.3.6 putDataToSend()

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

```
50                                     {
51
52     uint8_t nextIndex;
53     nextIndex = (value[0] + 1); //value[0] contains the size of the data + 1 for the type
54     if((nextIndex + dataToSend_Index) < dataToSend_Size){
55         dataToSend[dataToSend_Index++] = type;
56         memcpy(dataToSend + dataToSend_Index, value, (value[0] + 1));
57         dataToSend_Index = dataToSend_Index + nextIndex;
58         USB.print("Buffer: data stored correctly, stored data = ");
59         for(uint8_t i = 0; i < dataToSend_Index; i++){
60             USB.print(dataToSend[i], HEX);
61             USB.print(":");
62         }
63         USB.println("");
64         USB.print("  -Index ");
65         USB.println(dataToSend_Index, DEC);
66         USB.println("");
67     }else{
68         USB.print("The buffer is full, there is no more space ");
69     }
70 }
```

Here is the caller graph for this function:



3.2.3.7 putNetworkReceivedData()

```

void Buffer::putNetworkReceivedData (
    char * value )

140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
    {
        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:

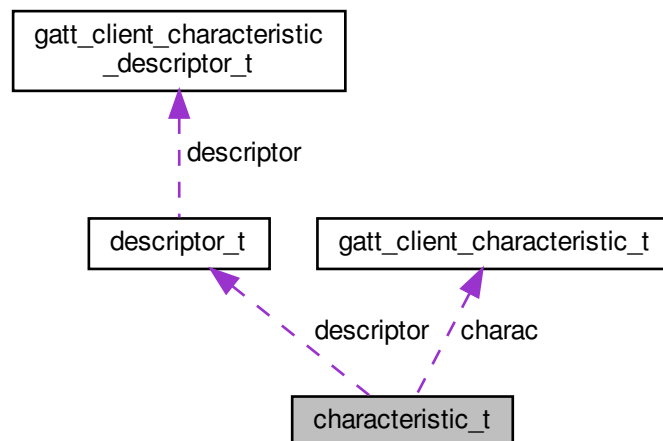
- [Buffer.h](#)
- [Buffer.cpp](#)

3.3 characteristic_t Struct Reference

Struct to store a characteristic and its descriptors.

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

Collaboration diagram for characteristic_t:



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:

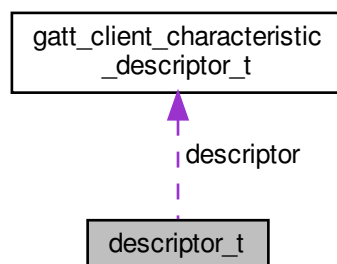
- [BLECentral.h](#)

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:

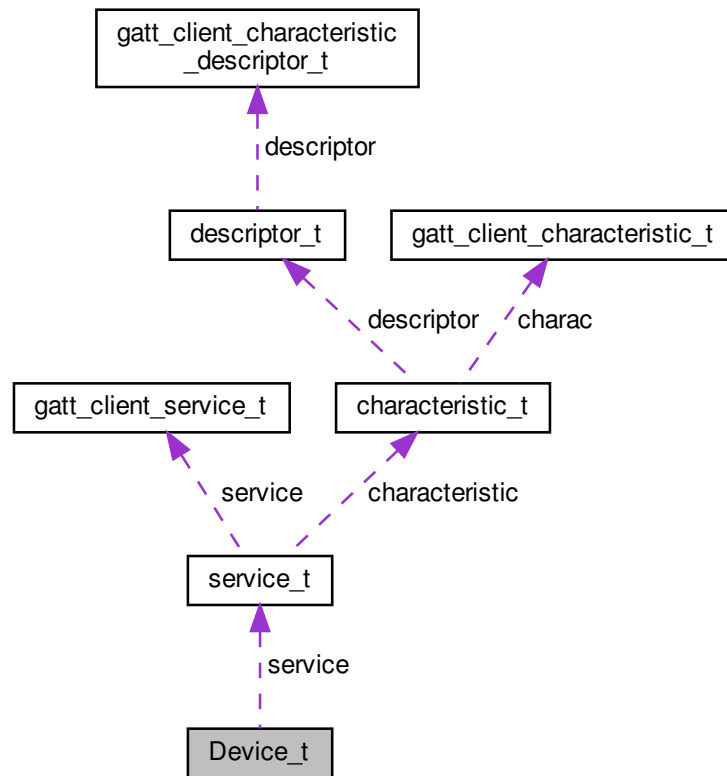
- [BLECentral.h](#)

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 [APP_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 setDataRateNextTransmission](#) (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

<i>void</i>	
-------------	--

Returns

`void`

```
30         {
31     }
```

3.10.2.2 ~LoraWan()

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

class Destructor It does nothing

Parameters

<i>void</i>	
-------------	--

Returns

void

```
38         {  
39     }
```

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[ ] )  
  
383     {  
384  
385         LoRaWAN.setDeviceEUI(DEVICE_EUI);  
386         LoRaWAN.setDeviceAddr(DEVICE_ADDR);  
387         LoRaWAN.setNwkSessionKey(NWK_SESSION_KEY);  
388         LoRaWAN.setAppSessionKey(APP_SESSION_KEY);  
389     }
```

3.10.3.2 configure2OTAA()

```
void LoraWan::configure2OTAA (  
    char  DEVICE_EUI[ ],  
    char  APP_EUI[ ],  
    char  APP_KEY[ ] )  
  
361     {  
362  
363         LoRaWAN.setDeviceEUI(DEVICE_EUI);  
364         LoRaWAN.setAppEUI(APP_EUI);  
365         LoRaWAN.setAppKey(APP_KEY);  
366     }
```

Here is the caller graph for this function:



3.10.3.3 enableOrDisableChannel()

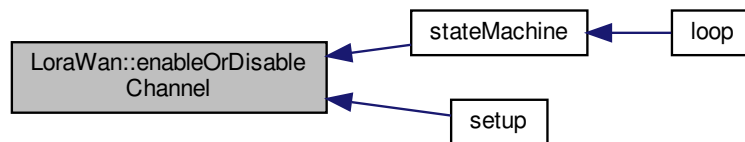
```

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

225                                     {
226     uint8_t response;
227     response = LoRaWAN.setChannelStatus(channel, onOff);
228     if( response == 0 ){
229         USB.println(F("LoRaWAN module Channel status set OK"));
230     }else {
231         USB.print(F("LoRaWAN module Channel status set, ERROR = "));
232         USB.println(response, DEC);
233     }
234     return response;
235 }

```

Here is the caller graph for this function:



3.10.3.4 getAutomaticReply()

```

uint8_t LoraWan::getAutomaticReply ( )

518                                     {
519     uint8_t response;
520     response = LoRaWAN.getAR();
521     if( response == 0 ) {
522         USB.print(F("LoRaWAN module Get automatic reply status OK. "));
523         USB.print(F("LoRaWAN module Automatic reply status: "));
524         if (LoRaWAN._ar == true){
525             USB.println(F("on"));
526         }else{
527             USB.println(F("off"));
528         }
529     }else{
530         USB.print(F("LoRaWAN module Get automatic reply status, ERROR = "));
531         USB.println(response, DEC);
532     }
533     return response;
534 }

```

3.10.3.5 getDowlinkRX1Delay()

```
uint8_t LoraWan::getDowlinkRX1Delay ( )
```

```
776                                     {
777     uint8_t response;
778     response = LoRaWAN.getRX1Delay();
779     if(response == 0){
780         return LoRaWAN._rx1Delay;
781     }else{
782         return response;
783     }
784 }
```

3.10.3.6 getDowlinkRX2Delay()

```
uint8_t LoraWan::getDowlinkRX2Delay ( )
```

```
827                                     {
828     uint8_t response;
829     response = LoRaWAN.getRX2Delay();
830     if(response == 0){
831         return LoRaWAN._rx2Delay;
832     }else{
833         return response;
834     }
835 }
```

3.10.3.7 getDowlinkRX2Parameters()

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

```
849                                     {
850     uint8_t response;
851     response = LoRaWAN.getRX2Parameters(band);
852     if(response == 0){
853         USB.print(F("LoRaWAN module Dowlink RX2 Parameters, Frequency = "));
854         USB.print(LoRaWAN._rx2Frequency, DEC);
855         USB.print(F(" ,data rate = "));
856         USB.println(LoRaWAN._rx2DataRate);
857         //return LoRaWAN._rx2Frequency;
858         //return LoRaWAN._rx2DataRate;
859         return response;
860     }else{
861         return response;
862     }
863 }
```

3.10.3.8 getDownlinkCounter()

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

```
713     {
714     uint8_t response;
715     response = LoRaWAN.getDownCounter();
716     if(response == 0){
717         return LoRaWAN._downCounter;
718     }else{
719         return response;
720     }
721 }
```

3.10.3.9 getGatewayNumber()

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

```
734     {
735     uint8_t response;
736     response = LoRaWAN.getGatewayNumber();
737     if(response == 0){
738         return LoRaWAN._gwNumber;
739     }else{
740         return response;
741     }
742 }
```

3.10.3.10 getRetries()

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

```
469     {
470     uint8_t response;
471     response = LoRaWAN.getRetries();
472     if( response == 0 ) {
473         USB.print(F("LoRaWAN module Get Retransmissions for uplink confirmed packet OK. "));
474         USB.print(F("TX  retries: "));
475         USB.println(LoRaWAN._retries, DEC);
476     }else{
477         USB.print(F("LoRaWAN module Get Retransmissions for uplink confirmed packet, ERROR = "));
478         USB.println(response, DEC);
479     }
480     return response;
481 }
```

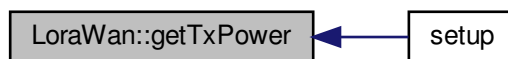

3.10.3.11 getTxPower()

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

```

278         {
279     uint8_t response;
280     response = LoRaWAN.getPower();
281     if( response == 0 ){
282         USB.println(F("LoRaWAN module Power level get OK"));
283         USB.print(F("  -Power index:"));
284         USB.println(LoRaWAN._powerIndex, DEC);
285     }else{
286         USB.print(F("LoRaWAN module Power level get, ERROR = "));
287         USB.println(response, DEC);
288     }
289     return response;
290 }
```

Here is the caller graph for this function:



3.10.3.12 getUplinkCounter()

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

```

694         {
695     uint8_t response;
696     response = LoRaWAN.getUpCounter();
697     if(response == 0){
698         return LoRaWAN._upCounter;
699     }else{
700         return response;
701     }
702 }
```

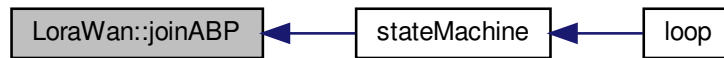
3.10.3.13 joinABP()

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

```

425         {
426     uint8_t response;
427     response = LoRaWAN.joinABP();
428     if(response == 0){
429         USB.println(F("LoRaWAN module join the network by ABP OK"));
430     }else{
431         USB.print(F("LoRaWAN module join ABP, ERROR = "));
432         USB.println(response, DEC);
433     }
434     return response;
435 }
```

Here is the caller graph for this function:



3.10.3.14 joinOTAA()

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

```

402     {
403     uint8_t response;
404     response = LoRaWAN.joinOTAA();
405     if(response == 0){
406         USB.println(F("LoRaWAN module join the network by OTAA OK"));
407     }else{
408         USB.print(F("LoRaWAN module join OTAA, ERROR = "));
409         USB.println(response, DEC);
410     }
411     return response;
412 }
```

Here is the caller graph for this function:



3.10.3.15 printChannelsStatus()

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

```

299     {
300     USB.println(F("\n-----"));
301     USB.println(F("LoRaWAN module channels status: "));
302     for( int Channel=0; Channel<16; Channel++){
303         LoRaWAN.getChannelFreq(Channel);
304         LoRaWAN.getChannelDutyCycle(Channel);
305         LoRaWAN.getChannelDRRange(Channel);
306         LoRaWAN.getChannelStatus(Channel);
307         USB.print(F("Channel: "));
```

```

308     USB.println(Channel);
309     USB.print(F("  -Freq: "));
310     USB.println(LoRaWAN._freq[Channel]);
311     USB.print(F("  -Duty cycle: "));
312     USB.println(LoRaWAN._dCycle[Channel]);
313     USB.print(F("  -DR min: "));
314     USB.println(LoRaWAN._drMin[Channel], DEC);
315     USB.print(F("  -DR max: "));
316     USB.println(LoRaWAN._drMax[Channel], DEC);
317     USB.print(F("  -Status: "));
318     if (LoRaWAN._status[Channel] == 1){
319         USB.println(F("on"));
320     }else{
321         USB.println(F("off"));
322     }
323     USB.println(F("-----"));
324 }
325 }

```

Here is the caller graph for this function:



3.10.3.16 printDeviceAddr()

```
uint8_t LoraWan::printDeviceAddr ( )
```

```

334     {
335     uint8_t response;
336     response = LoRaWAN.getDeviceAddr();
337     if( response == 0 ){
338         USB.println(F("LoRaWAN DeviceAddr = "));
339         USB.println(LoRaWAN._devAddr);
340     }else{
341         USB.println(F("LoRaWAN DeviceAddr, ERROR = "));
342         USB.println(response, DEC);
343     }
344     return response;
345 }

```

3.10.3.17 receiveDowlinkData()

```
char * LoraWan::receiveDowlinkData ( )
```

```

655     {
656
657     USB.print(F("LoRaWAN module there's data on port number "));
658     USB.print(LoRaWAN._port, DEC);
659     USB.print(F(".\r\n  Data: "));
660     USB.println(LoRaWAN._data);
661     return LoRaWAN._data;
662 }

```

Here is the caller graph for this function:



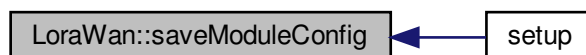
3.10.3.18 saveModuleConfig()

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

```

546         {
547     uint8_t response;
548     response = LoRaWAN.saveConfig();
549     if(response == 0){
550         USB.println(F("LoRaWAN module saveConfig OK"));
551     }else{
552         USB.print(F("LoRaWAN module saveConfig, ERROR = "));
553         USB.println(response, DEC);
554     }
555     return response;
556 }
```

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 )
```

```

625                                     {
626     uint8_t response;
627     response = LoRaWAN.sendConfirmed( port, data, len);
628     if( response == 0 ) {
629         USB.println(F("LoRaWAN module Send Confirmed packet OK"));
630         if (LoRaWAN._dataReceived == true){
631             return 1;
632         }
633     }else{
634         USB.print(F("LoRaWAN module Send Confirmed packet error = "));
635         USB.println(response, DEC);
636         //~ '0' if OK
637         //~ '1' if error
638         //~ '2' if no answer
639         //~ '4' if data length error
640         //~ '5' if error when sending data
641         //~ '6' if module hasn't joined to a network
642         //~ '7' if input port parameter error
643     }
644     return 0;
645 }

```

Here is the caller graph for this function:



3.10.3.20 sendUnconfirmedData()

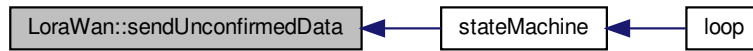
```

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

593                                     {
594     uint8_t response;
595     response = LoRaWAN.sendUnconfirmed( port, data, len);
596     if( response == 0 ) {
597         USB.println(F("LoRaWAN module Send Unconfirmed packet OK"));
598         if (LoRaWAN._dataReceived == true){
599             return 1;
600         }
601     }else{
602         USB.print(F("LoRaWAN module Send Unconfirmed packet ERROR = "));
603         USB.println(response, DEC);
604         //~ '0' if OK
605         //~ '1' if error
606         //~ '2' if no answer
607         //~ '4' if data length error
608         //~ '5' if error when sending data
609         //~ '6' if module hasn't joined to a network
610         //~ '7' if input port parameter error
611     }
612     return 0;
613 }

```

Here is the caller graph for this function:



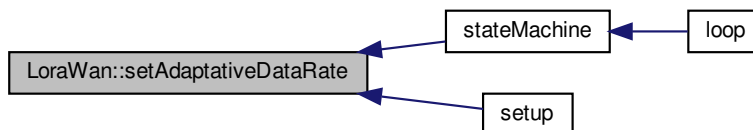
3.10.3.21 setAdaptativeDataRate()

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

```

111                                     {
112     uint8_t response;
113     response = LoRaWAN.setADR(onOff);
114     if( response == 0 ){
115         USB.println(F("LoRaWAN module Adaptive Data Rate OK "));
116         USB.print(F("  -ADR:"));
117         USB.println(LoRaWAN._adr, DEC);
118     }else{
119         USB.print(F("LoRaWAN module Adaptive Data Rate, ERROR = "));
120         USB.println(response, DEC);
121     }
122     return response;
123 }
```

Here is the caller graph for this function:



3.10.3.22 setAutomaticReply()

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

```

495                                     {
496     uint8_t response;
497     response = LoRaWAN.setAR(onOff);
498     if( response == 0 ) {
499         USB.println(F("LoRaWAN module Set automatic reply status on OK"));
500     }else {
501         USB.print(F("LoRaWAN module Set automatic reply status on, ERROR = "));
502         USB.println(response, DEC);
503     }
504     return response;
505 }

```

Here is the caller graph for this function:



3.10.3.23 setBatteryLevelStatus()

```
uint8_t LoraWan::setBatteryLevelStatus ( )
```

```

674                                     {
675     uint8_t response;
676     response = LoRaWAN.setBatteryLevel();
677     if( response == 0 ){
678         USB.println(F("LoRaWAN module BatteryLevelStatus set OK. "));
679     }else{
680         USB.print(F("LoRaWAN module BatteryLevelStatus set, ERROR = "));
681         USB.println(response, DEC);
682     }
683     return response;
684 }

```

3.10.3.24 setChannelDataRateRange()

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

```

169                                     {
170     uint8_t response;
171     response = LoRaWAN.setChannelDRRange(channel, drMin, drMax);
172     if( response == 0 ){
173         USB.println(F("LoRaWAN module Data Rate range set OK "));
174         USB.print(F(" -Data Rate min:"));
175         USB.println(LoRaWAN._drMin[channel], DEC);
176         USB.print(F(" -Data Rate max:"));
177         USB.println(LoRaWAN._drMax[channel], DEC);
178     }else{
179         USB.print(F("LoRaWAN module Data rate range set, ERROR = "));
180         USB.println(response, DEC);
181     }
182     return response;
183 }

```

Here is the caller graph for this function:



3.10.3.25 setChannelDutyCycle()

```

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

198                                     {
199     uint8_t response;
200     response = LoRaWAN.setChannelDutyCycle(channel, dutyCycle);
201     if( response == 0 ){
202         USB.println(F("LoRaWAN module Duty Cycle OK. "));
203         USB.print(F("Duty Cycle:"));
204         USB.println(LoRaWAN._dCycle[channel], DEC);
205     }else {
206         USB.print(F("LoRaWAN module Duty cycle set, ERROR = "));
207         USB.println(response, DEC);
208     }
209     return response;
210 }
  
```

3.10.3.26 setChannelFrequency()

```

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

139                                     {
140     uint8_t response;
141     response = LoRaWAN.setChannelFreq(channel, frequency);
142     if( response == 0 ) { // Check status
143         USB.print(F("LoRaWAN module frequency set OK "));
144         USB.print(F("Frequency: "));
145         USB.print(LoRaWAN._freq[channel]);
146         USB.print(F(" for channel: "));
147         USB.println(channel, DEC);
148     }else{
149         USB.print(F("LoRaWAN module frequency set, ERROR = "));
150         USB.println(response, DEC);
151     }
152     return response;
153 }
  
```


3.10.3.27 setDataRateNextTransmission()

```
uint8_t LoraWan::setDataRateNextTransmission (
    uint8_t socket )

571                                     {
572     uint8_t respuesta;
573     respuesta = LoRaWAN.setDataRate(dataRate);
574     if(respuesta == 0){
575         USB.println(F("LoRaWAN module Data Rate OK"));
576     }else{
577         USB.println(F("LoRaWAN module Data Rate ERROR = "));
578         USB.println(respuesta, DEC);
579     }
580     return respuesta;
581 }
```

3.10.3.28 setDowlinkRX1Delay()

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

755                                     {
756     uint8_t response;
757     response = LoRaWAN.setRX1Delay(delay);
758     if(response == 0){
759         return LoRaWAN._gwNumber;
760     }else{
761         return response;
762     }
763 }
```

3.10.3.29 setDowlinkRX2Parameters()

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

806                                     {
807     uint8_t response;
808     response = LoRaWAN.setRX2Parameters(datarate, frequency);
809     if(response == 0){
810         return LoRaWAN._gwNumber;
811     }else{
812         return response;
813     }
814 }
```

3.10.3.30 setRetries()

```

uint8_t LoraWan::setRetries (
    uint8_t retries )

448                                     {
449     uint8_t response;
450     response = LoRaWAN.setRetries(retries);
451     if( response == 0 ) {
452         USB.println(F("LoRaWAN module Set Retransmissions for uplink confirmed packet OK"));
453     }else{
454         USB.print(F("LoRaWAN module Set Retransmissions for uplink confirmed packet, ERROR = "));
455         USB.println(response, DEC);
456     }
457     return response;
458 }

```

Here is the caller graph for this function:



3.10.3.31 setTxPower()

```

uint8_t LoraWan::setTxPower (
    uint8_t power )

254                                     {
255     uint8_t response;
256     response = LoRaWAN.setPower(power);
257     if( response == 0 ){
258         USB.println(F("LoRaWAN module Power level set OK"));
259     }else{
260         USB.print(F("LoRaWAN module Power level set, ERROR = "));
261         USB.println(response, DEC);
262     }
263     return response;
264 }

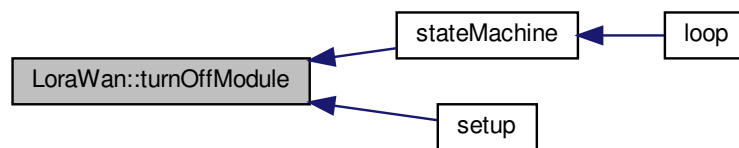
```

3.10.3.32 turnOffModule()

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

```
71         {
72     Uti1s.muxOFF1();
73     USB.println(F("LoRaWAN module switch off "));
74 }
```

Here is the caller graph for this function:



3.10.3.33 turnOffModule2()

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

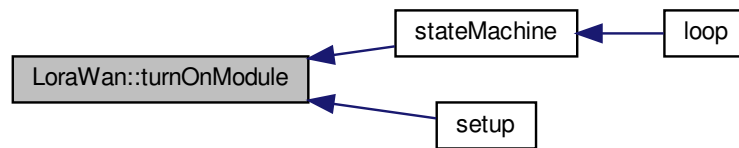
```
86         {
87     uint8_t response;
88     response = LoRaWAN.OFF(socket);
89     if(response == 0){
90         USB.println(F("LoRaWAN module switch off ok"));
91     }else{
92         USB.print(F("LoRaWAN module switch ERROR = "));
93         USB.println(response, DEC);
94     }
95     return response;
96 }
```

3.10.3.34 turnOnModule()

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

```
50         {
51
52     uint8_t response;
53     response = LoRaWAN.ON(socket);
54     if(response == 0){
55         USB.println(F("LoRaWAN module switch on Ok "));
56     }else{
57         USB.print(F("LoRaWAN turnOnModule(),ERROR = "));
58         USB.println(response, DEC);
59     }
60     return response;
61 }
```

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 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:

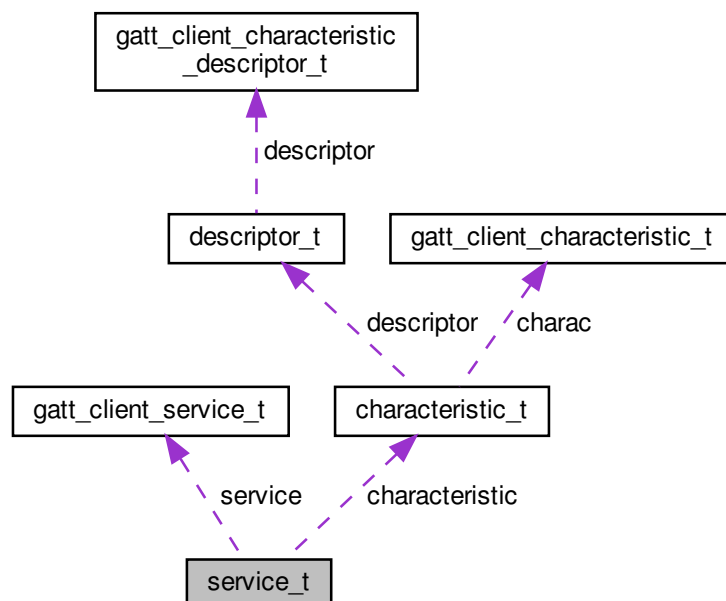
- [BLECentral.h](#)

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

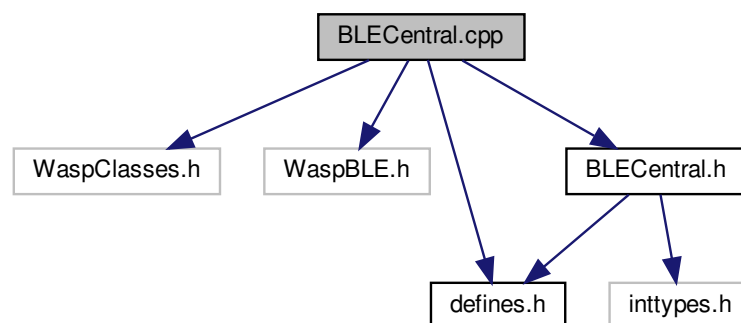
File Documentation

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

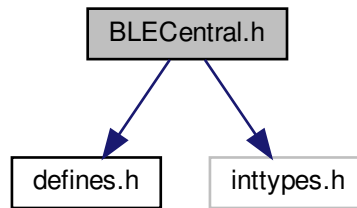
Alejandro Piñan Roescher

4.2 BLECentral.h File Reference

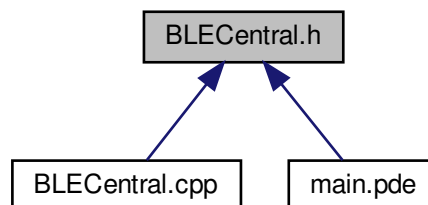
Library for managing the Bluetooth low energy module, BLE112, as a Central device.

```
#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

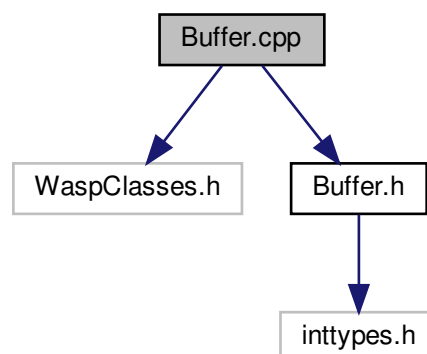
05/11/2018

Author

Alejandro Piñan Roescher

4.3 Buffer.cpp File Reference

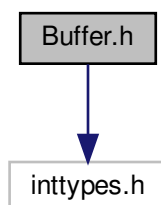
```
#include <WaspClasses.h>
#include "Buffer.h"
Include dependency graph for Buffer.cpp:
```



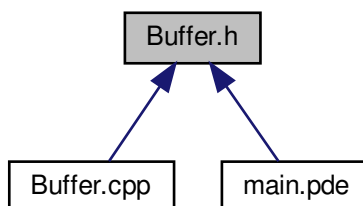
4.4 Buffer.h File Reference

```
#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

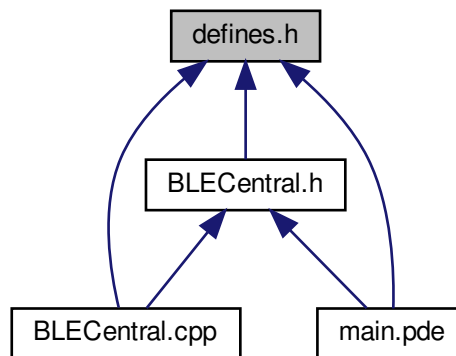
```
#define dataToSend_Size 60
```

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_Characteristic0_Battery_Level_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x19, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic0_UV_Index_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x76, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic1_Pressure_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x6D, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic2_Temperature_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x6E, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic3_Humidity_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x6F, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service4_Characteristic4_Ambient_Light_uuid](#) [16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0xD9, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E}
- static uint8_t [Service4_Characteristic5_Sound_Level_uuid](#) [16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0x02, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E}
- static uint8_t [Service4_Characteristic6_Control_Point_uuid](#) [16] = {0xC8, 0x54, 0x69, 0x13, 0xBF, 0x03, 0x45, 0xEB, 0x8D, 0xDE, 0x9F, 0x87, 0x54, 0xF4, 0xA3, 0x2E}
- static uint8_t [Service5_Characteristic0_Power_Source_uuid](#) [16] = {0xEC, 0x61, 0xA4, 0x54, 0xED, 0x01, 0xA5, 0xE8, 0xB8, 0xF9, 0xDE, 0x9E, 0xC0, 0x26, 0xEC, 0x51}
- static uint8_t [Service6_Characteristic0_ECO2_uuid](#) [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x01, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t [Service6_Characteristic1_TVOC_uuid](#) [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x02, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t [Service6_Characteristic2_Control_Point_uuid](#) [16] = {0xEF, 0xD6, 0x58, 0xAE, 0xC4, 0x03, 0xEF, 0x33, 0x76, 0xE7, 0x91, 0xB0, 0x00, 0x19, 0x10, 0x3B}
- static uint8_t [Service7_Characteristic0_Buttons_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x01, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [Service7_Characteristic1_Leds_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x02, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [Service7_Characteristic2_RGB_Leds_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x03, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [Service7_Characteristic3_Control_Point_uuid](#) [16] = {0xFC, 0xB8, 0x9C, 0x40, 0xC6, 0x04, 0x59, 0xF3, 0x7D, 0xC3, 0x5E, 0xCE, 0x44, 0x4A, 0x40, 0x1B}
- static uint8_t [Service8_Characteristic0_Digital_1_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x56, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service8_Characteristic1_Digital_2_uuid](#) [16] = {0x00, 0x00, 0x2A, 0x56, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFB}
- static uint8_t [Service9_Characteristic0_Acceleration_uuid](#) [16] = {0xC4, 0xC1, 0xF6, 0xE2, 0x4B, 0xE5, 0x11, 0xE5, 0x88, 0x5D, 0xFE, 0xFF, 0x81, 0x9C, 0xDC, 0x9F}
- static uint8_t [Service9_Characteristic1_Orientation_uuid](#) [16] = {0xB7, 0xC4, 0xB6, 0x94, 0xBE, 0xE3, 0x45, 0xDD, 0xBA, 0x9F, 0xF3, 0xB5, 0xE9, 0x94, 0xF4, 0x9A}
- static uint8_t [Service9_Characteristic2_Control_Point_uuid](#) [16] = {0x71, 0xE3, 0x0B, 0x8C, 0x41, 0x31, 0x47, 0x03, 0xB0, 0xA0, 0xB0, 0xBB, 0xBA, 0x75, 0x85, 0x6B}
- static uint8_t [ServiceA_Characteristic0_State_uuid](#) [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x01, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}
- static uint8_t [ServiceA_Characteristic1_Field_Strength_uuid](#) [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x02, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}
- static uint8_t [ServiceA_Characteristic2_Control_Point_uuid](#) [16] = {0xF5, 0x98, 0xDB, 0xC5, 0x2F, 0x03, 0x4E, 0xC5, 0x99, 0x36, 0xB3, 0xD1, 0xAA, 0x4F, 0x95, 0x7F}

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

```

59         {
60             ERROR_TYPE,
61             CONFIGURE_TIME_TYPE,
62             CONFIGURE_SELECTED_SENSORS_TYPE
63 }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

```

22     {
23         BLE_SCANNING = 0,
24         BLE_CONNECT,
25         DISCOVER_BLE_PROFILE,
26         ENABLE_BLE_NOTIFICATIONS,
27         ENABLE_INTERRUPTIONS,
28         SLEEP,
29         WAKE_UP_AND_CKECK,
30         LORAWAN_SEND_UPLINK,
31         LORAWAN_RECEIVE_DOWNLINK
32 }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

```
39     {
40     BLE_DISCONNECT_TYPE,
41     UV_INDEX_TYPE,
42     PRESSURE_TYPE,
43     TEMPERATURE_TYPE,
44     AMBIENT_LIGHT_TYPE,
45     SOUND_LEVEL_TYPE,
46     HUMIDITY_TYPE,
47     BATTERY_LEVEL_TYPE,
48     ECO2_TYPE,
49     TVOC_TYPE,
50     HALL_STATE_TYPE,
51     FIELD_STRENGTH_TYPE
52 }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_Characteristic0_Device_Name_uuid

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

4.5.4.12 Service0_Characteristic1_Appearance_uuid

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

4.5.4.13 Service1_Characrtteristic0_Service_Changed_uuid

```
uint8_t Service1_Characrtteristic0_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_Characrtteristic0_Manufacturer_Name_uuid

```
uint8_t Service2_Characrtteristic0_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_Characrtteristic1_Model_Number_uuid

```
uint8_t Service2_Characrtteristic1_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_Characrtteristic2_Serial_Number_uuid

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

4.5.4.17 Service2_Characrtteristic3_Hardware_Revision_uuid

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

4.5.4.18 Service2_Characrtteristic4_Firmware_Revision_uuid

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

4.5.4.19 Service2_Characrtteristic5_System_ID_uuid

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

4.5.4.20 Service3_Characrtteristic0_Battery_Level_uuid

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

4.5.4.21 Service4_Characrtteristic0_UV_Index_uuid

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

4.5.4.22 Service4_Characrtteristic1_Pressure_uuid

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

4.5.4.23 Service4_Characrtteristic2_Temperature_uuid

```
uint8_t Service4_Characrtteristic2_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_Characrtteristic3_Humidity_uuid

```
uint8_t Service4_Characrtteristic3_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_Characrtteristic4_Ambient_Light_uuid

```
uint8_t Service4_Characrtteristic4_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_Characrtteristic5_Sound_Level_uuid

```
uint8_t Service4_Characrtteristic5_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_Characrtteristic6_Control_Point_uuid

```
uint8_t Service4_Characrtteristic6_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_Characrtteristic0_Power_Source_uuid

```
uint8_t Service5_Characrtteristic0_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_Characrtteristic0_ECO2_uuid

```
uint8_t Service6_Characrtteristic0_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_Characrtteristic1_TVOC_uuid

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

4.5.4.31 Service6_Characrtteristic2_Control_Point_uuid

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

4.5.4.32 Service7_Characrtteristic0_Buttons_uuid

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

4.5.4.33 Service7_Characrtteristic1_Leds_uuid

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

4.5.4.34 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]
```

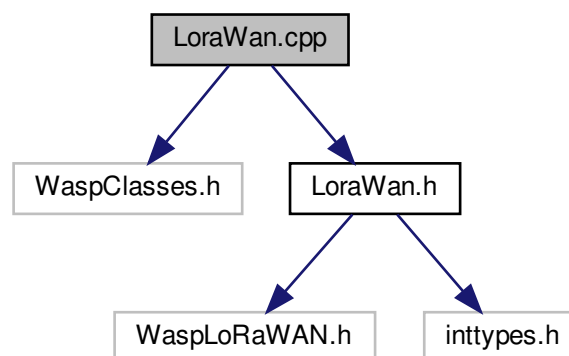
4.6 LoraWan.cpp File Reference

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

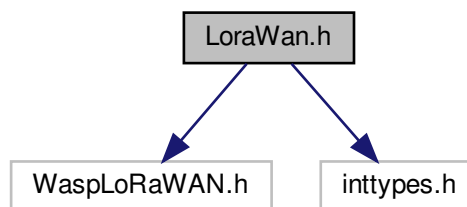
4.7 LoraWan.h File Reference

Library for managing the LoRaWAN module.

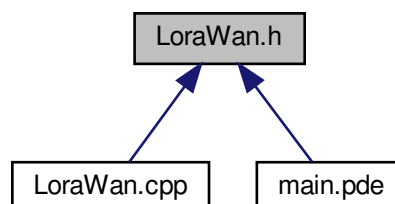
```
#include <WaspLoRaWAN.h>
```

```
#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

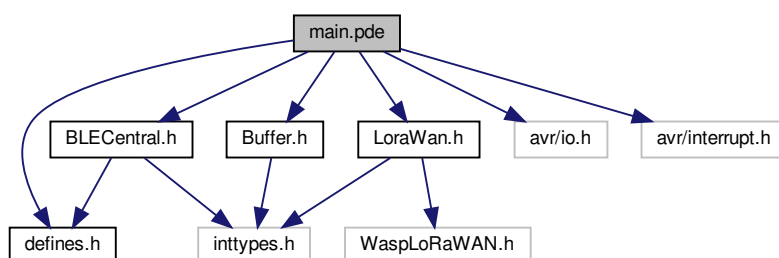
Alejandro Piñan Roescher

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 Waspote 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()

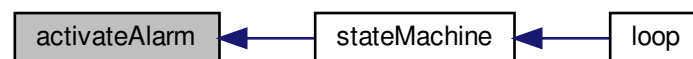
```

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

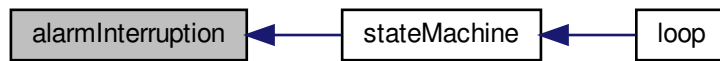
--	--

```

75     {
76         alarmFlag = 1;
77     }

```

Here is the caller graph for this function:



4.8.2.3 disableInterruptPCINT8()

```
void disableInterruptPCINT8 ( )
```

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

Parameters

<i>void</i>	
-------------	--

Return values

<i>void</i>	
-------------	--

```

114         {
115     PCICR &= (0 << PCIE1);
116 }
  
```

Here is the caller graph for this function:



4.8.2.4 enableInterruptPCINT8()

```
void enableInterruptPCINT8 ( )
```

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

Parameters

<i>void</i>	
-------------	--

Return values

<i>void</i>	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 is set 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 in SREG and the PCIE1 bit in PCICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is 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 is enabled. Any change on any enabled PCINT15:8 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PC11 Interrupt Vector. PCINT15:8 pins are enabled individually by the PCMSK1 Register.

```

101      {
102      USB.println(F("Enabled PCINT8 interruption to receive notifications from the BLE module"));
103      PCMSK1 |= (1 << PCINT8);
104      PCIFR  |= (1 << PCIF1);
105      PCICR  |= (1 << PCIE1);
106  }
```

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

--	--

```

126         {
127         pcint8++;
128     }

```

4.8.2.6 loop()

```

loop ( )

```

```

while(true)

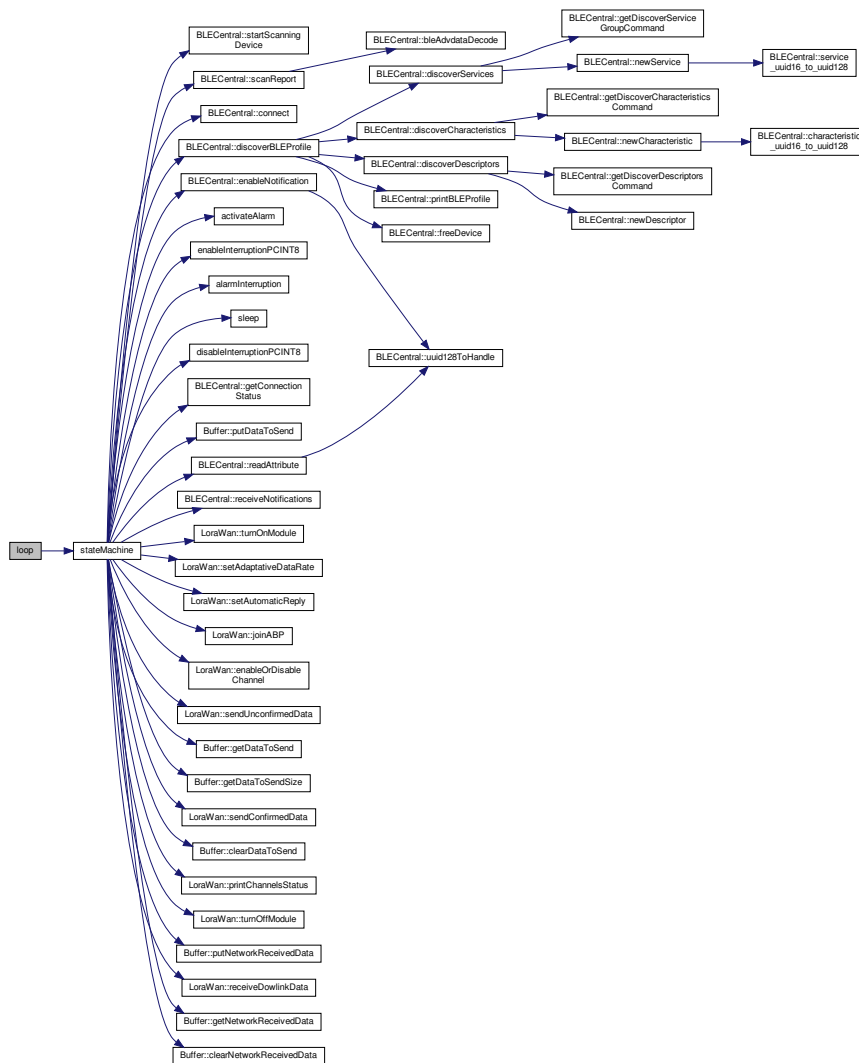
```

```

357     {
358
359     stateMachine();
360 }

```

Here is the call graph for this function:



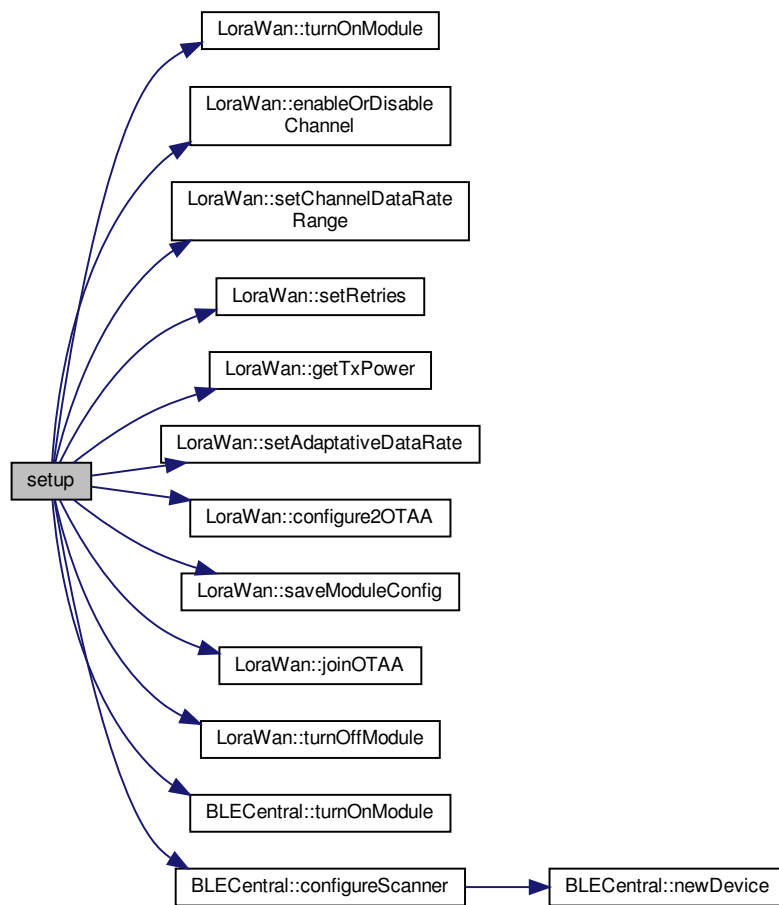
4.8.2.7 setup()

```
void setup ( )
```

```

324     {
325         USB.println(F("____Starting setup"));
326         USB.println(F(""));
327         USB.println(F("____Starting LoRaWAN module configuration"));
328         state = BLE_SCANNING; //Initial state of the State Machine (BLE-LoRaWAN Node)
329         hours = 0; //Initial hour and minute, 00:02, to receive the sensors data
330         minutes = 2;
331         memset(sensorsBitMap, 0x01, sizeof(sensorsBitMap)); //By default all sensors
values to send
332         lorawan.turnOnModule(SOCKET1);
333         //Obliged because the nanoGateway, Lopy4, has a single channel (Waspmote has 0..15 channels, but only
0,1,2, are enabled by default)
334         lorawan.enableOrDisableChannel(1, "off");
335         lorawan.enableOrDisableChannel(2, "off");
336         lorawan.setChannelDataRateRange(0, 5, 5); //We use the channel 0
--->frec=868100000 and data rate=5-->sf=7 (Lopy4, has a single data rate)
337         lorawan.setRetries(2); //Number of retries for the send with confirmation (Used to send
hall sensor events)
338         //~ lorawan.setTxPower(4);
339         lorawan.getTxPower();
340         lorawan.setAdaptativeDataRate("off"); //This parameter cannot be stored in
the module's EEPROM using the saveConfig() function
341         lorawan.configure2OTAA(DEVICE_EUI, APP_EUI,
APP_KEY);
342         lorawan.saveModuleConfig();
343         lorawan.joinOTAA();
344         lorawan.turnOffModule();
345         USB.println(F("____LoRaWAN module configuration completed"));
346         USB.println(F(""));
347         USB.println(F("____BLE module configuration"));
348         bleCentral.turnOnModule(SOCKET0);
349         bleCentral.configureScanner(
BLE_GAP_DISCOVER_OBSERVATION, TX_POWER,
SCAN_INTERVAL, SCAN_WINDOW, BLE_PASSIVE_SCANNING);
350         USB.println(F("____Finished setup"));
351         USB.println(F(""));
352     }
```

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

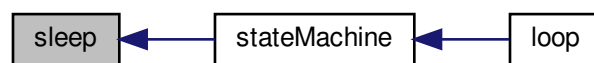
void	This function turn waspmote in sleep mode(SLEEP_MODE_PWR_DOWN)
------	--

```

137     {
138     /* There are five different sleep modes in order of power saving:
139     SLEEP_MODE_IDLE - the lowest power saving mode
140     SLEEP_MODE_ADC
141     SLEEP_MODE_PWR_SAVE
142     SLEEP_MODE_STANDBY
143     SLEEP_MODE_EXT_STANDBY
144     SLEEP_MODE_PWR_DOWN - the highest power saving mode
145     */
146     set_sleep_mode(SLEEP_MODE_PWR_DOWN);
147     sleep_mode(); //Put the device into sleep mode, taking care of setting the SE bit before, and clearing
148     it afterwards
149 }

```

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
------	--

```

157     {
158
159     uint8_t response = 0;
160
161     switch(state) {
162
163     case BLE_SCANNING:
164         #if DEBUG >= 1
165         USB.println(F("State: BLE_SCANNING"));
166         #endif
167         response = bleCentral.startScanningDevice(
168         MAC);
169         if(response){
170             response = bleCentral.scanReport("Thunder Sense #02735");
171             if (response){
172                 state = BLE_CONNECT;
173             }
174             break;
175

```

```

176     case BLE_CONNECT:
177         #if DEBUG >= 1
178             USB.println(F("State: BLE_CONNECT"));
179         #endif
180         response = bleCentral.connect(MAC);
181         if (response){
182             state = DISCOVER_BLE_PROFILE;
183             delay(1000);
184         }else{
185             state = BLE_SCANNING;
186         }
187         break;
188
189     case DISCOVER_BLE_PROFILE:
190         #if DEBUG >= 1
191             USB.println(F("State: DISCOVER_BLE_PROFILE"));
192         #endif
193         response = bleCentral.discoverBLEProfile();
194         if (response){
195             state = ENABLE_BLE_NOTIFICATIONS;
196         }else{
197             state = BLE_SCANNING;
198         }
199         break;
200
201     case ENABLE_BLE_NOTIFICATIONS:
202         #if DEBUG >= 1
203             USB.println(F("State: ENABLE_BLE_NOTIFICATIONS"));
204         #endif
205         bleCentral.enableNotification(
ServiceA_Characteristic0_State_uuid);
206         state = ENABLE_INTERRUPTIONS;
207         break;
208
209     case ENABLE_INTERRUPTIONS:
210         #if DEBUG >= 1
211             USB.println(F("State: ENABLE_INTERRUPTIONS"));
212         #endif
213         activateAlarm(hours, minutes);
214         enableInterruptPCINT8();
215         attachInterrupt(RTC_INT, alarmInterrupt, 1);
216         state = SLEEP;
217         break;
218
219     case SLEEP:
220         #if DEBUG >= 1
221             USB.println(F("State: SLEEP"));
222         #endif
223         alarmFlag = 0;
224         sleep();
225         USB.println(F("Waspmote wake up"));
226         disableInterruptPCINT8();
227         state = WAKE_UP_AND_CKECK;
228         break;
229
230     case WAKE_UP_AND_CKECK:
231         #if DEBUG >= 1
232             USB.println(F("State: WAKE_UP_AND_CKECK"));
233         #endif
234         if(bleCentral.getConnectionStatus() != 1){//The BLE connection has
been disconnected
235             buffer.putDataToSend(BLE_Disconnected,
BLE_DISCONNECT_TYPE);
236         }else if( alarmFlag == 1){//Attend the Alarm, the established time has been met
237             if(sensorsBitMap[1]==1)
238                 buffer.putDataToSend(bleCentral.
readAttribute(Service4_Characteristic0_UV_Index_uuid),
UV_INDEX_TYPE);
239             if(sensorsBitMap[2]==1)
240                 buffer.putDataToSend(bleCentral.
readAttribute(Service4_Characteristic1_Pressure_uuid),
PRESSURE_TYPE);
241             if(sensorsBitMap[3]==1)
242                 buffer.putDataToSend(bleCentral.
readAttribute(Service4_Characteristic2_Temperature_uuid
), TEMPERATURE_TYPE);
243             if(sensorsBitMap[4]==1)
244                 buffer.putDataToSend(bleCentral.
readAttribute(Service4_Characteristic4_Ambient_Light_uuid
), AMBIENT_LIGHT_TYPE);
245             if(sensorsBitMap[5]==1)
246                 buffer.putDataToSend(bleCentral.
readAttribute(Service4_Characteristic5_Sound_Level_uuid
), SOUND_LEVEL_TYPE);
247             if(sensorsBitMap[6]==1)
248                 buffer.putDataToSend(bleCentral.
readAttribute(Service4_Characteristic3_Humidity_uuid),

```

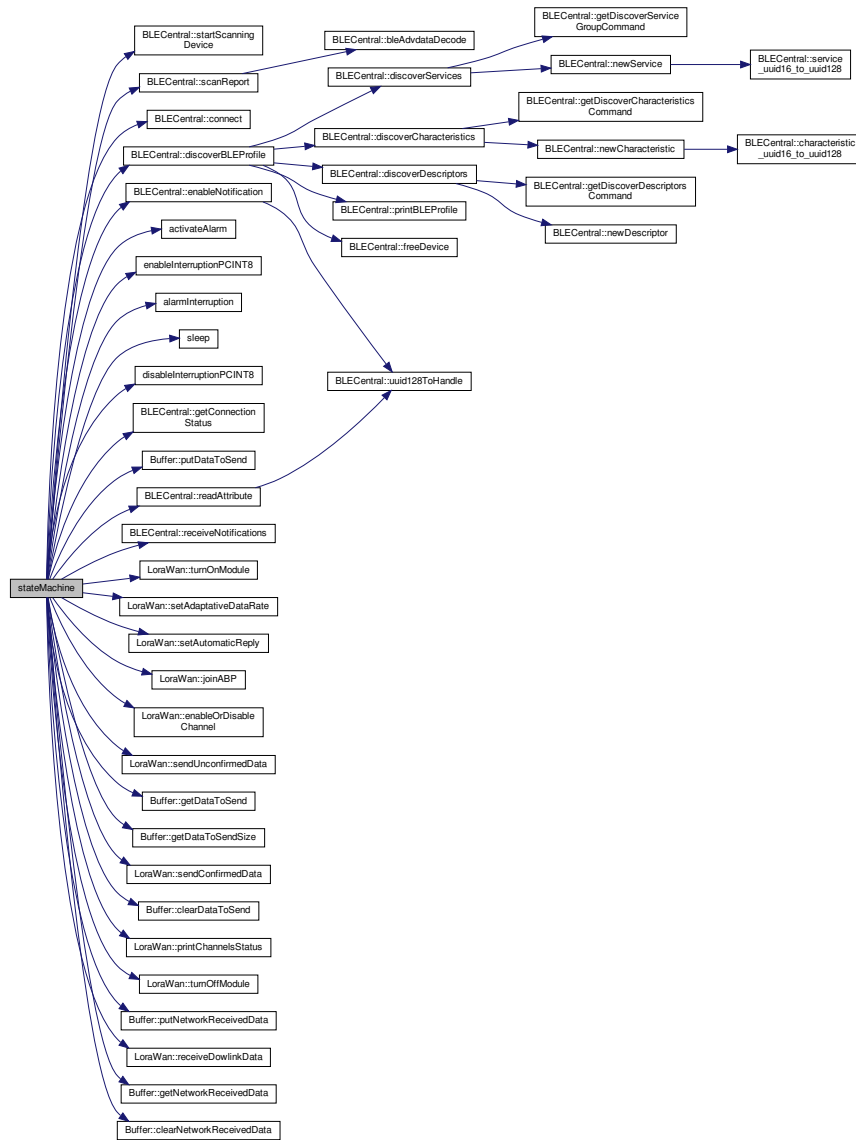
```

        HUMIDITY_TYPE);
249         if(sensorsBitMap[7]==1)
250             buffer.putDataToSend(bleCentral.
readAttribute(Service3_Characrteristic0_Battery_Level_uuid
), BATTERY_LEVEL_TYPE);
251         if(sensorsBitMap[8]==1)
252             buffer.putDataToSend(bleCentral.
readAttribute(Service6_Characrteristic0_ECO2_uuid),
ECO2_TYPE);
253         if(sensorsBitMap[9]==1)
254             buffer.putDataToSend(bleCentral.
readAttribute(Service6_Characrteristic1_TVOC_uuid),
TVOC_TYPE);
255         if(sensorsBitMap[10]==1)
256             buffer.putDataToSend(bleCentral.
readAttribute(ServiceA_Characrteristic0_State_uuid),
HALL_STATE_TYPE);
257         if(sensorsBitMap[11]==1)
258             buffer.putDataToSend(bleCentral.
readAttribute(ServiceA_Characrteristic1_Field_Strength_uuid
), FIELD_STRENGHT_TYPE);
259     }else{//Attend the Hall sensor notification
260         buffer.putDataToSend(bleCentral.
receiveNotifications(), HALL_STATE_TYPE);
261     }
262     state = LORAWAN_SEND_UPLINK;
263     break;
264
265     case LORAWAN_SEND_UPLINK:
266         #if DEBUG >= 1
267             USB.println(F("State: LORAWAN_SEND_UPLINK"));
268         #endif
269         lorawan.turnOnModule(SOCKET1);
270         lorawan.setAdaptativeDataRate("off");
271         lorawan.setAutomaticReply("on");
272         lorawan.joinABP();
273         lorawan.enableOrDisableChannel(1, "off");
274         lorawan.enableOrDisableChannel(2, "off");
275         if(alarmFlag == 1) {
276             response = lorawan.sendUnconfirmedData(
DATA_PORT, buffer.getDataToSend(), buffer.
getDataToSendSize());
277         }else{
278             response = lorawan.sendConfirmedData(
EVENT_PORT, buffer.getDataToSend(), buffer.
getDataToSendSize());
279         }
280         if(response == 1){
281             state = LORAWAN_RECEIVE_DOWNLINK;
282         }else{
283             state = ENABLE_INTERRUPTIONS;
284         }
285         buffer.clearDataToSend();
286         lorawan.printChannelsStatus();
287         lorawan.turnOffModule();
288         USB.println(F(""));
289         break;
290
291     case LORAWAN_RECEIVE_DOWNLINK:
292         #if DEBUG >= 1
293             USB.println(F("State: LORAWAN_RECEIVE_DOWNLINK"));
294         #endif
295         buffer.putNetworkReceivedData( lorawan.
receiveDowlinkData());
296         if(buffer.getNetworkReceivedData(0) ==
CONFIGURE_TIME_TYPE){
297             hours = buffer.getNetworkReceivedData(1);
298             minutes = buffer.getNetworkReceivedData(2);
299             USB.print(F("Hours received: "));
300             USB.println(hours,DEC);
301             USB.print(F("Minutes received: "));
302             USB.println(minutes,DEC);
303         }else if(buffer.getNetworkReceivedData(0) ==
CONFIGURE_SELECTED_SENSORS_TYPE){
304             USB.println(F("Selected sensors = "));
305             for(uint8_t i = 0; i < 12; i++){
306                 sensorsBitMap[i] = buffer.
getNetworkReceivedData(i);
307                 USB.print(sensorsBitMap[i],DEC);
308                 USB.print(F(" "));
309             }
310             USB.println(F(""));
311         }
312         buffer.clearNetworkReceivedData();
313         state = ENABLE_INTERRUPTIONS;
314         break;
315     }

```

316 }

Here is the call graph for this function:



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 alarmInterrupt

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

- ~BLECentral
 - BLECentral, 8
- ~Buffer
 - Buffer, 33
- ~LoraWan
 - LoraWan, 48
- APP_EUI
 - main.pde, 99
- APP_KEY
 - main.pde, 99
- acceleration_orientation_service_uuid
 - defines.h, 79
- activateAlarm
 - main.pde, 88
- alarmFlag
 - main.pde, 99
- alarmInterruption
 - main.pde, 89
- automation_io_service_uuid
 - defines.h, 79
- BLE_Disconnected
 - main.pde, 99
- BLE_GAP_DISCOVER_OBSERVATION
 - defines.h, 75
- BLE_PASSIVE_SCANNING
 - defines.h, 76
- BLECentral, 5
 - ~BLECentral, 8
 - BLECentral, 7
 - bleAdvdataDecode, 8
 - characteristic_uuid16_to_uuid128, 9
 - configureScanner, 10
 - connect, 11
 - connectWithSelectedParameters, 11
 - device, 32
 - disconnect, 12
 - discoverBLEProfile, 12
 - discoverCharacteristics, 13
 - discoverDescriptors, 14
 - discoverServices, 15
 - enableNotification, 16
 - freeDevice, 17
 - getConnectionHandler, 18
 - getConnectionStatus, 18
 - getDiscoverCharacteristicsCommand, 18
 - getDiscoverDescriptorsCommand, 19
 - getDiscoverServiceGroupCommand, 19
 - newCharacteristic, 20
 - newDescriptor, 21
 - newDevice, 22
 - newService, 23
 - printBLEProfile, 24
 - readAttribute, 25
 - receiveNotifications, 26
 - scanReport, 27
 - service_uuid16_to_uuid128, 28
 - startScanning, 28
 - startScanningDevice, 29
 - turnOffModule, 29
 - turnOnModule, 29
 - uuid128ToHandle, 30
 - uuid16ToHandle, 31
 - writeAttribute, 31
- BLECentral.cpp, 69
- BLECentral.h, 70
- battery_service_uuid
 - defines.h, 79
- bleAdvdataDecode
 - BLECentral, 8
- bleCentral
 - main.pde, 99
- Buffer, 32
 - ~Buffer, 33
 - Buffer, 33
 - clearDataToSend, 34
 - clearNetworkReceivedData, 34
 - dataToSend, 37
 - dataToSend_Index, 37
 - getDataToSend, 34
 - getDataToSendSize, 35
 - getNetworkReceivedData, 35
 - networkReceivedData, 37
 - putDataToSend, 36
 - putNetworkReceivedData, 36
- buffer
 - main.pde, 99
- Buffer.cpp, 71
- Buffer.h, 72
 - dataToSend_Size, 72
 - ReceivedData_Size, 73
- charac
 - characteristic_t, 39
- characteristic
 - service_t, 67
- characteristic_t, 38
 - charac, 39
 - descriptor, 39

- numberOfDescriptors, 39
- characteristic_uuid16_to_uuid128
 - BLECentral, 9
- classID
 - findInformationCommand_t, 42
 - readByGroupCommand_t, 64
- clearDataToSend
 - Buffer, 34
- clearNetworkReceivedData
 - Buffer, 34
- commandID
 - findInformationCommand_t, 42
 - readByGroupCommand_t, 64
- configure2ABP
 - LoraWan, 49
- configure2OTAA
 - LoraWan, 49
- configureScanner
 - BLECentral, 10
- connect
 - BLECentral, 11
- connectWithSelectedParameters
 - BLECentral, 11
- connected_handle
 - Device_t, 41
- Connectionhandle
 - findInformationCommand_t, 43
 - readByGroupCommand_t, 65
- DATA_PORT
 - defines.h, 76
- DEBUG
 - defines.h, 76
- DEVICE_EUI
 - main.pde, 99
- dataToSend
 - Buffer, 37
- dataToSend_Index
 - Buffer, 37
- dataToSend_Size
 - Buffer.h, 72
- defines.h, 73
 - acceleration_orientation_service_uuid, 79
 - automation_io_service_uuid, 79
 - BLE_GAP_DISCOVER_OBSERVATION, 75
 - BLE_PASSIVE_SCANNING, 76
 - battery_service_uuid, 79
 - DATA_PORT, 76
 - DEBUG, 76
 - device_information_service_uuid, 79
 - downlinktypes, 77
 - downlinktypes_t, 77
 - EVENT_PORT, 76
 - environmental_sensing_service_uuid, 79
 - generic_access_service_uuid, 79
 - generic_attribute_service_uuid, 80
 - hall_effect_service_uuid, 80
 - iaq_service_uuid, 80
 - power_management_service_uuid, 80
 - SCAN_INTERVAL, 76
 - SCAN_WINDOW, 76
 - SOCKET0, 76
 - SOCKET1, 76
 - Service0_Characrtistic0_Device_Name_uuid, 80
 - Service0_Characrtistic1_Appearance_uuid, 80
 - Service1_Characrtistic0_Service_Changed_↔
uuid, 80
 - Service2_Characrtistic0_Manufacturer_Name_↔
_uuid, 81
 - Service2_Characrtistic1_Model_Number_uuid,
81
 - Service2_Characrtistic2_Serial_Number_uuid,
81
 - Service2_Characrtistic3_Hardware_Revision_↔
uuid, 81
 - Service2_Characrtistic4_Firmware_Revision_↔
uuid, 81
 - Service2_Characrtistic5_System_ID_uuid, 81
 - Service3_Characrtistic0_Battery_Level_uuid, 81
 - Service4_Characrtistic0_UV_Index_uuid, 82
 - Service4_Characrtistic1_Pressure_uuid, 82
 - Service4_Characrtistic2_Temperature_uuid, 82
 - Service4_Characrtistic3_Humidity_uuid, 82
 - Service4_Characrtistic4_Ambient_Light_uuid,
82
 - Service4_Characrtistic5_Sound_Level_uuid, 82
 - Service4_Characrtistic6_Control_Point_uuid, 82
 - Service5_Characrtistic0_Power_Source_uuid,
83
 - Service6_Characrtistic0_ECO2_uuid, 83
 - Service6_Characrtistic1_TVOC_uuid, 83
 - Service6_Characrtistic2_Control_Point_uuid, 83
 - Service7_Characrtistic0_Buttons_uuid, 83
 - Service7_Characrtistic1_Leds_uuid, 83
 - Service7_Characrtistic2_RGB_Leds_uuid, 83
 - Service7_Characrtistic3_Control_Point_uuid, 84
 - Service8_Characrtistic0_Digital_1_uuid, 84
 - Service8_Characrtistic1_Digital_2_uuid, 84
 - Service9_Characrtistic0_Acceleration_uuid, 84
 - Service9_Characrtistic1_Orientation_uuid, 84
 - Service9_Characrtistic2_Control_Point_uuid, 84
 - ServiceA_Characrtistic0_State_uuid, 84
 - ServiceA_Characrtistic1_Field_Strength_uuid,
85
 - ServiceA_Characrtistic2_Control_Point_uuid, 85
 - stateEnum_t, 77
 - states, 78
 - TX_POWER, 77
 - uplinkTypes, 78
 - UplinkTypes_t, 77
 - user_interface_service_uuid, 85
- descriptor
 - characteristic_t, 39
 - descriptor_t, 40
- descriptor_t, 39
 - descriptor, 40
- device

- BLECentral, 32
- device_information_service_uuid
 - defines.h, 79
- Device_t, 40
 - connected_handle, 41
 - mac, 41
 - numberOfServices, 42
 - service, 42
- disableInterruptionPCINT8
 - main.pde, 90
- disconnect
 - BLECentral, 12
- discoverBLEProfile
 - BLECentral, 12
- discoverCharacteristics
 - BLECentral, 13
- discoverDescriptors
 - BLECentral, 14
- discoverServices
 - BLECentral, 15
- downlinktypes
 - defines.h, 77
- downlinktypes_t
 - defines.h, 77
- EVENT_PORT
 - defines.h, 76
- enableInterruptionPCINT8
 - main.pde, 90
- enableNotification
 - BLECentral, 16
- enableOrDisableChannel
 - LoraWan, 49
- end_group_handle
 - gatt_client_service_t, 46
- endLastAttributeHandle
 - findInformationCommand_t, 43
 - readByGroupCommand_t, 65
- environmental_sensing_service_uuid
 - defines.h, 79
- findInformationCommand_t, 42
 - classID, 42
 - commandID, 42
 - Connectionhandle, 43
 - endLastAttributeHandle, 43
 - messageType, 43
 - payloadLenght, 43
 - startFirstAttributeHandle, 43
 - t_length, 43
- freeDevice
 - BLECentral, 17
- gatt_client_characteristic_descriptor_t, 44
 - handle, 44
 - uuid128, 44
 - uuid16, 44
- gatt_client_characteristic_t, 45
 - properties, 45
 - start_handle, 45
 - uuid128, 45
 - uuid16, 45
 - value_handle, 45
- gatt_client_service_t, 46
 - end_group_handle, 46
 - start_group_handle, 46
 - uuid128, 46
 - uuid16, 47
- generic_access_service_uuid
 - defines.h, 79
- generic_attribute_service_uuid
 - defines.h, 80
- getAutomaticReply
 - LoraWan, 50
- getConnectionHandler
 - BLECentral, 18
- getConnectionStatus
 - BLECentral, 18
- getDataToSend
 - Buffer, 34
- getDataToSendSize
 - Buffer, 35
- getDiscoverCharacteristicsCommand
 - BLECentral, 18
- getDiscoverDescriptorsCommand
 - BLECentral, 19
- getDiscoverServiceGroupCommand
 - BLECentral, 19
- getDowlinkRX1Delay
 - LoraWan, 50
- getDowlinkRX2Delay
 - LoraWan, 51
- getDowlinkRX2Parameters
 - LoraWan, 51
- getDownlinkCounter
 - LoraWan, 51
- getGatewayNumber
 - LoraWan, 52
- getNetworkReceivedData
 - Buffer, 35
- getRetries
 - LoraWan, 52
- getTxPower
 - LoraWan, 52
- getUplinkCounter
 - LoraWan, 53
- hall_effect_service_uuid
 - defines.h, 80
- handle
 - gatt_client_characteristic_descriptor_t, 44
- hours
 - main.pde, 99
- ISR
 - main.pde, 91
- iaq_service_uuid
 - defines.h, 80

- joinABP
 - LoraWan, [53](#)
- joinOTAA
 - LoraWan, [54](#)
- loop
 - main.pde, [92](#)
- LoraWan, [47](#)
 - ~LoraWan, [48](#)
 - configure2ABP, [49](#)
 - configure2OTAA, [49](#)
 - enableOrDisableChannel, [49](#)
 - getAutomaticReply, [50](#)
 - getDowlinkRX1Delay, [50](#)
 - getDowlinkRX2Delay, [51](#)
 - getDowlinkRX2Parameters, [51](#)
 - getDownlinkCounter, [51](#)
 - getGatewayNumber, [52](#)
 - getRetries, [52](#)
 - getTxPower, [52](#)
 - getUplinkCounter, [53](#)
 - joinABP, [53](#)
 - joinOTAA, [54](#)
 - LoraWan, [48](#)
 - printChannelsStatus, [54](#)
 - printDeviceAddr, [55](#)
 - receiveDowlinkData, [55](#)
 - saveModuleConfig, [56](#)
 - sendConfirmedData, [56](#)
 - sendUnconfirmedData, [57](#)
 - setAdaptativeDataRate, [58](#)
 - setAutomaticReply, [58](#)
 - setBatteryLevelStatus, [59](#)
 - setChannelDataRateRange, [59](#)
 - setChannelDutyCycle, [60](#)
 - setChannelFrequency, [60](#)
 - setDataRateNextTransmission, [60](#)
 - setDowlinkRX1Delay, [61](#)
 - setDowlinkRX2Parameters, [61](#)
 - setRetries, [61](#)
 - setTxPower, [62](#)
 - turnOffModule, [62](#)
 - turnOffModule2, [63](#)
 - turnOnModule, [63](#)
- LoraWan.cpp, [85](#)
- LoraWan.h, [86](#)
- lorawan
 - main.pde, [100](#)
- MAC
 - main.pde, [100](#)
- mac
 - Device_t, [41](#)
- main.pde, [87](#)
 - APP_EUI, [99](#)
 - APP_KEY, [99](#)
 - activateAlarm, [88](#)
 - alarmFlag, [99](#)
 - alarmInterruption, [89](#)
 - BLE_Disconnected, [99](#)
 - bleCentral, [99](#)
 - buffer, [99](#)
 - DEVICE_EUI, [99](#)
 - disableInterruptionPCINT8, [90](#)
 - enableInterruptionPCINT8, [90](#)
 - hours, [99](#)
 - ISR, [91](#)
 - loop, [92](#)
 - lorawan, [100](#)
 - MAC, [100](#)
 - minutes, [100](#)
 - pcint8, [100](#)
 - sensorsBitMap, [100](#)
 - setup, [92](#)
 - sleep, [94](#)
 - state, [100](#)
 - stateMachine, [95](#)
- messageType
 - findInformationCommand_t, [43](#)
 - readByGroupCommand_t, [65](#)
- minutes
 - main.pde, [100](#)
- networkReceivedData
 - Buffer, [37](#)
- newCharacteristic
 - BLECentral, [20](#)
- newDescriptor
 - BLECentral, [21](#)
- newDevice
 - BLECentral, [22](#)
- newService
 - BLECentral, [23](#)
- numberOfCharacteristics
 - service_t, [67](#)
- numberOfDescriptors
 - characteristic_t, [39](#)
- numberOfServices
 - Device_t, [42](#)
- payloadLenght
 - findInformationCommand_t, [43](#)
 - readByGroupCommand_t, [65](#)
- pcint8
 - main.pde, [100](#)
- power_management_service_uuid
 - defines.h, [80](#)
- printBLEProfile
 - BLECentral, [24](#)
- printChannelsStatus
 - LoraWan, [54](#)
- printDeviceAddr
 - LoraWan, [55](#)
- properties
 - gatt_client_characteristic_t, [45](#)
- putDataToSend
 - Buffer, [36](#)
- putNetworkReceivedData

- Buffer, 36
- readAttribute
 - BLECentral, 25
- readByGroupCommand_t, 64
 - classID, 64
 - commandID, 64
 - Connectionhandle, 65
 - endLastAttributeHandle, 65
 - messageType, 65
 - payloadLenght, 65
 - startFirstAttributeHandle, 65
 - t_length, 65
 - uuid, 65
 - uuidLenght, 65
- receiveDowlinkData
 - LoraWan, 55
- receiveNotifications
 - BLECentral, 26
- ReceivedData_Size
 - Buffer.h, 73
- SCAN_INTERVAL
 - defines.h, 76
- SCAN_WINDOW
 - defines.h, 76
- SOCKET0
 - defines.h, 76
- SOCKET1
 - defines.h, 76
- saveModuleConfig
 - LoraWan, 56
- scanReport
 - BLECentral, 27
- sendConfirmedData
 - LoraWan, 56
- sendUnconfirmedData
 - LoraWan, 57
- sensorsBitMap
 - main.pde, 100
- service
 - Device_t, 42
 - service_t, 67
- Service0_Characrteristic0_Device_Name_uuid
 - defines.h, 80
- Service0_Characrteristic1_Appearance_uuid
 - defines.h, 80
- Service1_Characrteristic0_Service_Changed_uuid
 - defines.h, 80
- Service2_Characrteristic0_Manufacturer_Name_uuid
 - defines.h, 81
- Service2_Characrteristic1_Model_Number_uuid
 - defines.h, 81
- Service2_Characrteristic2_Serial_Number_uuid
 - defines.h, 81
- Service2_Characrteristic3_Hardware_Revision_uuid
 - defines.h, 81
- Service2_Characrteristic4_Firmware_Revision_uuid
 - defines.h, 81
- Service2_Characrteristic5_System_ID_uuid
 - defines.h, 81
- Service3_Characrteristic0_Battery_Level_uuid
 - defines.h, 81
- Service4_Characrteristic0_UV_Index_uuid
 - defines.h, 82
- Service4_Characrteristic1_Pressure_uuid
 - defines.h, 82
- Service4_Characrteristic2_Temperature_uuid
 - defines.h, 82
- Service4_Characrteristic3_Humidity_uuid
 - defines.h, 82
- Service4_Characrteristic4_Ambient_Light_uuid
 - defines.h, 82
- Service4_Characrteristic5_Sound_Level_uuid
 - defines.h, 82
- Service4_Characrteristic6_Control_Point_uuid
 - defines.h, 82
- Service5_Characrteristic0_Power_Source_uuid
 - defines.h, 83
- Service6_Characrteristic0_ECO2_uuid
 - defines.h, 83
- Service6_Characrteristic1_TVOC_uuid
 - defines.h, 83
- Service6_Characrteristic2_Control_Point_uuid
 - defines.h, 83
- Service7_Characrteristic0_Buttons_uuid
 - defines.h, 83
- Service7_Characrteristic1_Leds_uuid
 - defines.h, 83
- Service7_Characrteristic2_RGB_Leds_uuid
 - defines.h, 83
- Service7_Characrteristic3_Control_Point_uuid
 - defines.h, 84
- Service8_Characrteristic0_Digital_1_uuid
 - defines.h, 84
- Service8_Characrteristic1_Digital_2_uuid
 - defines.h, 84
- Service9_Characrteristic0_Acceleration_uuid
 - defines.h, 84
- Service9_Characrteristic1_Orientation_uuid
 - defines.h, 84
- Service9_Characrteristic2_Control_Point_uuid
 - defines.h, 84
- service_t, 66
 - characteristic, 67
 - numberOfCharacteristics, 67
 - service, 67
- service_uuid16_to_uuid128
 - BLECentral, 28
- ServiceA_Characrteristic0_State_uuid
 - defines.h, 84
- ServiceA_Characrteristic1_Field_Strength_uuid
 - defines.h, 85
- ServiceA_Characrteristic2_Control_Point_uuid
 - defines.h, 85
- setAdaptativeDataRate
 - LoraWan, 58

- setAutomaticReply
 - LoraWan, [58](#)
- setBatteryLevelStatus
 - LoraWan, [59](#)
- setChannelDataRateRange
 - LoraWan, [59](#)
- setChannelDutyCycle
 - LoraWan, [60](#)
- setChannelFrequency
 - LoraWan, [60](#)
- setDataRateNextTransmission
 - LoraWan, [60](#)
- setDowlinkRX1Delay
 - LoraWan, [61](#)
- setDowlinkRX2Parameters
 - LoraWan, [61](#)
- setRetries
 - LoraWan, [61](#)
- setTxPower
 - LoraWan, [62](#)
- setup
 - main.pde, [92](#)
- sleep
 - main.pde, [94](#)
- start_group_handle
 - gatt_client_service_t, [46](#)
- start_handle
 - gatt_client_characteristic_t, [45](#)
- startFirstAttributeHandle
 - findInformationCommand_t, [43](#)
 - readByGroupCommand_t, [65](#)
- startScanning
 - BLECentral, [28](#)
- startScanningDevice
 - BLECentral, [29](#)
- state
 - main.pde, [100](#)
- stateEnum_t
 - defines.h, [77](#)
- stateMachine
 - main.pde, [95](#)
- states
 - defines.h, [78](#)
- t_length
 - findInformationCommand_t, [43](#)
 - readByGroupCommand_t, [65](#)
- TX_POWER
 - defines.h, [77](#)
- trama_descriptor_t, [67](#)
- trama_grupo_t, [67](#)
- turnOffModule
 - BLECentral, [29](#)
 - LoraWan, [62](#)
- turnOffModule2
 - LoraWan, [63](#)
- turnOnModule
 - BLECentral, [29](#)
 - LoraWan, [63](#)
- uplinkTypes
 - defines.h, [78](#)
- UplinkTypes_t
 - defines.h, [77](#)
- user_interface_service_uuid
 - defines.h, [85](#)
- uuid
 - readByGroupCommand_t, [65](#)
- uuid128
 - gatt_client_characteristic_descriptor_t, [44](#)
 - gatt_client_characteristic_t, [45](#)
 - gatt_client_service_t, [46](#)
- uuid128ToHandle
 - BLECentral, [30](#)
- uuid16
 - gatt_client_characteristic_descriptor_t, [44](#)
 - gatt_client_characteristic_t, [45](#)
 - gatt_client_service_t, [47](#)
- uuid16ToHandle
 - BLECentral, [31](#)
- uuidLenght
 - readByGroupCommand_t, [65](#)
- value_handle
 - gatt_client_characteristic_t, [45](#)
- writeAttribute
 - BLECentral, [31](#)