ThermoGuard

1.0

Gerado por Doxygen 1.13.2

Capítulo 1

Índice da hierarquia

1.1 Hierarquia de classes

Esta lista de heranças está organizada, dentro do possível, por ordem alfabética:

Capítulo 2

Índice das estruturas de dados

2.1 Estruturas de dados

Lista das estruturas de dados com uma breve descrição:

Capítulo 3

Índice dos ficheiros

3.1 Lista de ficheiros

Lista de todos os ficheiros com uma breve descrição:

Capítulo 4

Documentação da estruturas de dados

4.1 Referência à classe CharCallback

Callback para escrita em características BLE. Atualiza variáveis globais.

Diagrama de heranças da classe CharCallback

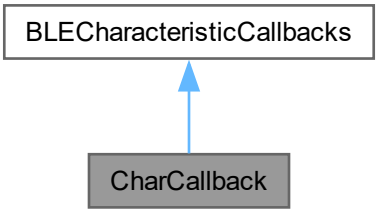
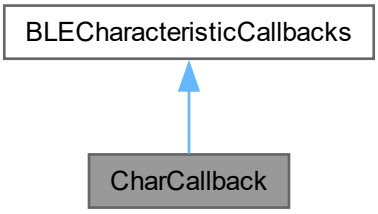


Diagrama de colaboração para CharCallback:



Membros privados

- void onWrite (BLECharacteristic *pCharacteristic) override

4.1.1 Descrição detalhada

Callback para escrita em características BLE. Atualiza variáveis globais.

- Interpreta strings recebidas do App via BLE.
- Atualiza variáveis globais (nome e senha do Wi-Fi, telefone, setpoint, intervalo).
- Chama a função SalvaConfig() para persistência imediata na NVS.

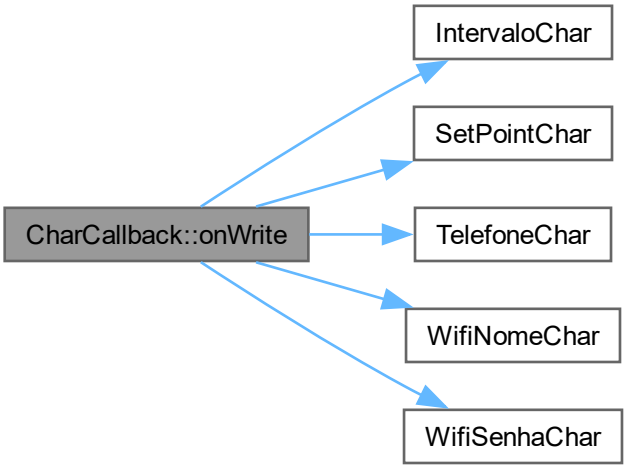
4.1.2 Documentação das funções

4.1.2.1 onWrite()

```
void CharCallback::onWrite (
            BLECharacteristic * pCharacteristic)  [inline], [override], [private]
```

```
00083                                                                    {
00084     std::string value = pCharacteristic->getValue();
00085     if (pCharacteristic == &WifiNomeChar) {
00086       nomeWifi = String(value.c_str());
00087       Serial.print("Novo nome do WiFi recebido: ");
00088       Serial.println(nomeWifi);
00089     } else if (pCharacteristic == &WifiSenhaChar) {
00090       senhaWifi = String(value.c_str());
00091       Serial.print("Nova senha do WiFi recebida: ");
00092       Serial.println(senhaWifi);
00093     } else if (pCharacteristic == &SetPointChar) {
00094       setPointTemp = String(value.c_str()).toFloat();
00095       Serial.print("Novo setpoint recebido: ");
00096       Serial.println(setPointTemp);
00097     } else if (pCharacteristic == &IntervaloChar) {
00098       intervaloTemp = String(value.c_str()).toInt();
00099       Serial.print("Novo intervalo recebido: ");
00100       Serial.println(intervaloTemp);
00101     } else if (pCharacteristic == &TelefoneChar) {
00102       numCelular = String(value.c_str());
00103       Serial.print("Número de celular recebido: ");
00104       Serial.println(numCelular);
00105     }
00106   }
```

Grafo de chamadas desta função:



A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- bleconfig.cpp

4.2 Referência à classe MyServerCallbacks

Callback de eventos do servidor BLE. Atualiza estado da conexão.

Diagrama de heranças da classe MyServerCallbacks

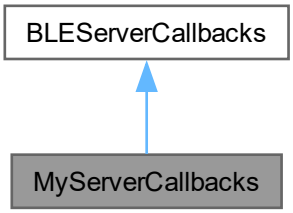
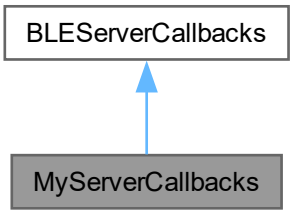


Diagrama de colaboração para MyServerCallbacks:



Membros privados

- void onConnect (BLEServer ∗pServer)
- void onDisconnect (BLEServer ∗pServer)

4.2.1 Descrição detalhada

Callback de eventos do servidor BLE. Atualiza estado da conexão.

- onConnect(): Define 'deviceConnected = true'.
- onDisconnect(): Define 'deviceConnected = false'.

4.2.2 Documentação das funções

4.2.2.1 onConnect()

```
void MyServerCallbacks::onConnect (
            BLEServer * pServer)  [inline], [private]
00063                                    {
00064       Serial.println("Conectando...");
00065       deviceConnected = true;
00066     };
```

4.2.2.2 onDisconnect()

```
void MyServerCallbacks::onDisconnect (
            BLEServer * pServer)  [inline], [private]
00067                                       {
00068       Serial.println("Desconectando...");
00069       deviceConnected = false;
00070     }
```

A documentação para esta classe foi gerada a partir do seguinte ficheiro:

- bleconfig.cpp

Capítulo 5

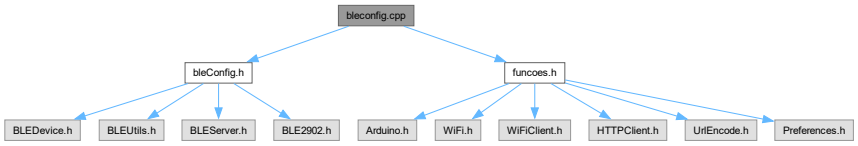
Documentação do ficheiro

5.1 Referência ao ficheiro bleconfig.cpp

Implementação da configuração BLE do ESP32.

```
#include "bleConfig.h"
#include "funcoes.h"
```

Diagrama de dependências de inclusão para bleconfig.cpp:



Estruturas de Dados

- class MyServerCallbacks

  Callback de eventos do servidor BLE. Atualiza estado da conexão.
- class CharCallback

  Callback para escrita em características BLE. Atualiza variáveis globais.

Funções

- BLECharacteristic TemperaturaChar (BLEUUID((uint16_t) 0x2A6E), BLECharacteristic::PROPERTY_↩ READ|BLECharacteristic::PROPERTY_NOTIFY)
- BLEDescriptor TemperaturaDesc (BLEUUID((uint16_t) 0x2912))
- BLECharacteristic IntervaloChar (BLEUUID((uint16_t) 0x2B8F), BLECharacteristic::PROPERTY_↩ READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic::PROPERTY_WRITE|BLECharacteristic↩ ::PROPERTY_INDICATE)
- BLEDescriptor IntervaloDesc (BLEUUID((uint16_t) 0x290E))
- BLECharacteristic SetPointChar ("22399fc7-2eef-48c8-924c-bba26ca25376", BLECharacteristic::↩ PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic::PROPERTY_WRITE|BLECharacteristic↩ ::PROPERTY_INDICATE)

- BLEDescriptor SetPointDesc (BLEUUID((uint16_t) 0x2903))
- BLECharacteristic WifiNomeChar ("09caff73-bfcf-4066-90a1-b142a385d7f6", BLECharacteristic::↩ PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic::PROPERTY_WRITE|BLECharacteristic↩ ::PROPERTY_INDICATE)
- BLEDescriptor WifiNomeDesc (BLEUUID((uint16_t) 0x2903))
- BLECharacteristic WifiSenhaChar ("74ddaf0d-dd25-4c8f-bb76-bee0df68c786", BLECharacteristic::↩ PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic::PROPERTY_WRITE|BLECharacteristic↩ ::PROPERTY_INDICATE)
- BLEDescriptor WifiSenhaDesc (BLEUUID((uint16_t) 0x2903))
- BLECharacteristic TelefoneChar ("bd96b429-e88d-4702-9450-f023c08e509c", BLECharacteristic::↩ PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic::PROPERTY_WRITE|BLECharacteristic↩ ::PROPERTY_INDICATE)
- BLEDescriptor TelefoneDesc (BLEUUID((uint16_t) 0x2903))
- void IniciarBLE ()

  Inicializa o Bluetooth Low Energy (BLE).
- void EncerrarBLE ()

  Encerra o Bluetooth Low Energy (BLE).

5.1.1 Descrição detalhada

Implementação da configuração BLE do ESP32.

Define as características BLE (temperatura, setpoint, intervalo, Wi-Fi e telefone). Controla os eventos de conexão e escrita nas características, atualizando as variáveis globais e salvando-as na NVS quando necessário.

5.1.2 Documentação das funções

5.1.2.1 EncerrarBLE()

```
void EncerrarBLE ()
```

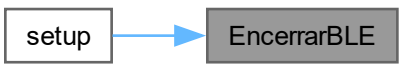
Encerra o Bluetooth Low Energy (BLE).

Encerra a comunicação BLE e libera recursos.

Finaliza o serviço BLE e libera os recursos alocados.

```
00165                   {
00166     BLEDevice::deinit(true);
00167 }
```

Este é o diagrama das funções que utilizam esta função:



5.1.2.2 IniciarBLE()

```
void IniciarBLE ()
```

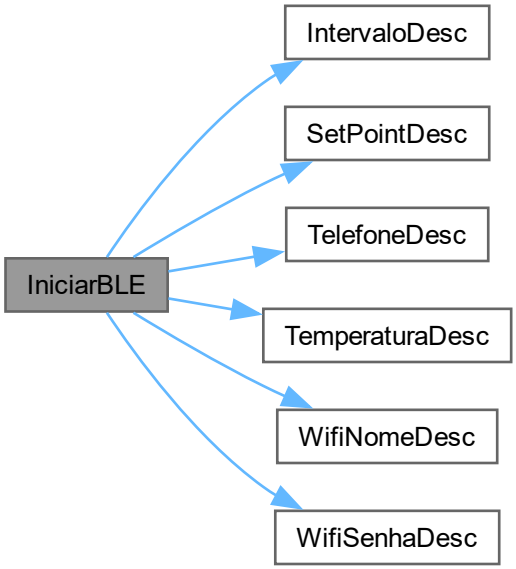
Inicializa o Bluetooth Low Energy (BLE).

Inicializa o serviço BLE com todas as características e inicia o advertising.

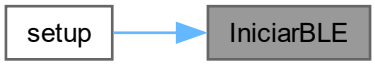
Configura o servidor BLE, cria as características e descritores para temperatura, intervalo de leitura e dados de Wi-Fi, e inicia o advertising BLE.

```
00116                 {
00117     BLEDevice::init("ThermoGuard");
00118     BLEServer *pServer = BLEDevice::createServer();
00119     pServer->setCallbacks(new MyServerCallbacks());
00120     BLEService *pService = pServer->createService(SERVICE_UUID);
00121     BLEService *pServiceWifi = pServer->createService(SERVICE_WIFI_UUID);
00122 
00123     // Temperatura
00124     TemperaturaDesc.setValue("Temperatura lida do sensor em celsius");
00125     TemperaturaChar.addDescriptor(&TemperaturaDesc);
00126     pService->addCharacteristic(&TemperaturaChar);
00127     SetPointDesc.setValue("Setpoint para alerta");
00128     SetPointChar.addDescriptor(&SetPointDesc);
00129     SetPointChar.setCallbacks(new CharCallback());
00130     pService->addCharacteristic(&SetPointChar);
00131     // Intervalo
00132     IntervaloDesc.setValue("Tempo entre leitura em minutos");
00133     IntervaloChar.addDescriptor(&IntervaloDesc);
00134     IntervaloChar.setCallbacks(new CharCallback());
00135     pService->addCharacteristic(&IntervaloChar);
00136 
00137     // Wifi
00138     WifiNomeDesc.setValue("Nome WiFi");
00139     WifiNomeChar.addDescriptor(&WifiNomeDesc);
00140     WifiNomeChar.setCallbacks(new CharCallback());
00141     pServiceWifi->addCharacteristic(&WifiNomeChar);
00142     WifiSenhaDesc.setValue("Senha WiFi");
00143     WifiSenhaChar.addDescriptor(&WifiSenhaDesc);
00144     WifiSenhaChar.setCallbacks(new CharCallback());
00145     pServiceWifi->addCharacteristic(&WifiSenhaChar);
00146     // Telefone
00147     TelefoneDesc.setValue("Numero de telefone");
00148     TelefoneChar.addDescriptor(&TelefoneDesc);
00149     TelefoneChar.setCallbacks(new CharCallback());
00150     pServiceWifi->addCharacteristic(&TelefoneChar);
00151 
00152     pService->start();
00153     pServiceWifi->start();
00154     BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
00155     pAdvertising->addServiceUUID(SERVICE_UUID);
00156     pAdvertising->addServiceUUID(SERVICE_WIFI_UUID);
00157     pServer->getAdvertising()->start();
00158 }
```

Grafo de chamadas desta função:



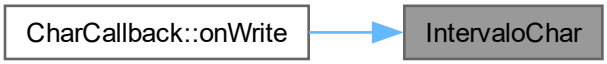
Este é o diagrama das funções que utilizam esta função:



5.1.2.3 IntervaloChar()

```
BLECharacteristic IntervaloChar (
            BLEUUID((uint16_t) 0x2B8F) ,
            BLECharacteristic::PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic↩
::PROPERTY_WRITE|BLECharacteristic::PROPERTY_INDICATE )
```

Este é o diagrama das funções que utilizam esta função:



5.1.2.4 IntervaloDesc()

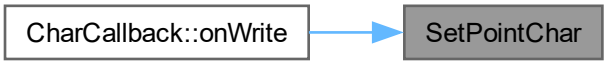
```
BLEDescriptor IntervaloDesc (
            BLEUUID((uint16_t) 0x290E) )
```

Este é o diagrama das funções que utilizam esta função:

5.1.2.5 SetPointChar()

```
BLECharacteristic SetPointChar (
            "22399fc7-2eef-48c8-924c-bba26ca25376" ,
            BLECharacteristic::PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic↩
::PROPERTY_WRITE|BLECharacteristic::PROPERTY_INDICATE )
```

Este é o diagrama das funções que utilizam esta função:



5.1.2.6 SetPointDesc()

```
BLEDescriptor SetPointDesc (
            BLEUUID((uint16_t) 0x2903) )
```

Este é o diagrama das funções que utilizam esta função:

5.1.2.7 TelefoneChar()

```
BLECharacteristic TelefoneChar (
            "bd96b429-e88d-4702-9450-f023c08e509c" ,
            BLECharacteristic::PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic↩
::PROPERTY_WRITE|BLECharacteristic::PROPERTY_INDICATE )
```

Este é o diagrama das funções que utilizam esta função:

5.1.2.8 TelefoneDesc()

```
BLEDescriptor TelefoneDesc (
            BLEUUID((uint16_t) 0x2903) )
```

Este é o diagrama das funções que utilizam esta função:

5.1.2.9 TemperaturaChar()

```
BLECharacteristic TemperaturaChar (
            BLEUUID((uint16_t) 0x2A6E) ,
            BLECharacteristic::PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY )
```

5.1.2.10 TemperaturaDesc()

```
BLEDescriptor TemperaturaDesc (
            BLEUUID((uint16_t) 0x2912) )
```

Este é o diagrama das funções que utilizam esta função:

5.1.2.11 WifiNomeChar()

```
BLECharacteristic WifiNomeChar (
            "09caff73-bfcf-4066-90a1-b142a385d7f6" ,
            BLECharacteristic::PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic↩
::PROPERTY_WRITE|BLECharacteristic::PROPERTY_INDICATE )
```

Este é o diagrama das funções que utilizam esta função:

5.1.2.12 WifiNomeDesc()

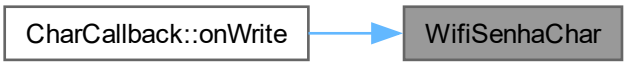
```
BLEDescriptor WifiNomeDesc (
            BLEUUID((uint16_t) 0x2903) )
```

Este é o diagrama das funções que utilizam esta função:

5.1.2.13 WifiSenhaChar()

```
BLECharacteristic WifiSenhaChar (
            "74ddaf0d-dd25-4c8f-bb76-bee0df68c786" ,
            BLECharacteristic::PROPERTY_READ|BLECharacteristic::PROPERTY_NOTIFY|BLECharacteristic↩
::PROPERTY_WRITE|BLECharacteristic::PROPERTY_INDICATE )
```

Este é o diagrama das funções que utilizam esta função:



5.1.2.14 WifiSenhaDesc()

```
BLEDescriptor WifiSenhaDesc (
            BLEUUID((uint16_t) 0x2903) )
```

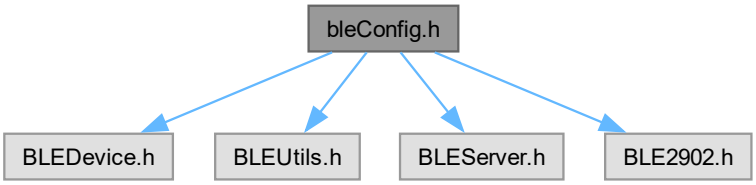
Este é o diagrama das funções que utilizam esta função:

5.2 Referência ao ficheiro bleConfig.h

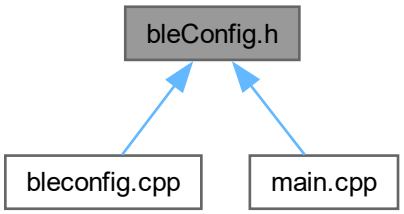
Declarações relacionadas à configuração do BLE no ESP32.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <BLE2902.h>
```

Diagrama de dependências de inclusão para bleConfig.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Macros

- #define BLECONFIGS_H
- #define SERVICE_UUID BLEUUID((uint16_t)0x1809)
- #define SERVICE_WIFI_UUID "7be95774-8734-4f46-a326-573d290c1e9f"

Funções

- void IniciarBLE ()

  Inicializa o serviço BLE com todas as características e inicia o advertising.
- void EncerrarBLE ()

  Encerra a comunicação BLE e libera recursos.

Variáveis

- volatile bool deviceConnected

5.2.1 Descrição detalhada

Declarações relacionadas à configuração do BLE no ESP32.

Define UUIDs, variáveis globais e funções públicas para controle BLE. Contém os protótipos das funções IniciarBLE() e EncerrarBLE().

5.2.2 Documentação das macros

5.2.2.1 BLECONFIGS_H

```
#define BLECONFIGS_H
```

5.2.2.2 SERVICE_UUID

```
#define SERVICE_UUID BLEUUID((uint16_t)0x1809)
```

5.2.2.3 SERVICE_WIFI_UUID

```
#define SERVICE_WIFI_UUID "7be95774-8734-4f46-a326-573d290c1e9f"
```

5.2.3 Documentação das funções

5.2.3.1 EncerrarBLE()

```
void EncerrarBLE ()
```

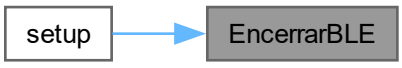
Encerra a comunicação BLE e libera recursos.

Encerra a comunicação BLE e libera recursos.

Finaliza o serviço BLE e libera os recursos alocados.

```
00165                   {
00166     BLEDevice::deinit(true);
00167 }
```

Este é o diagrama das funções que utilizam esta função:



5.2.3.2 IniciarBLE()

```
void IniciarBLE ()
```

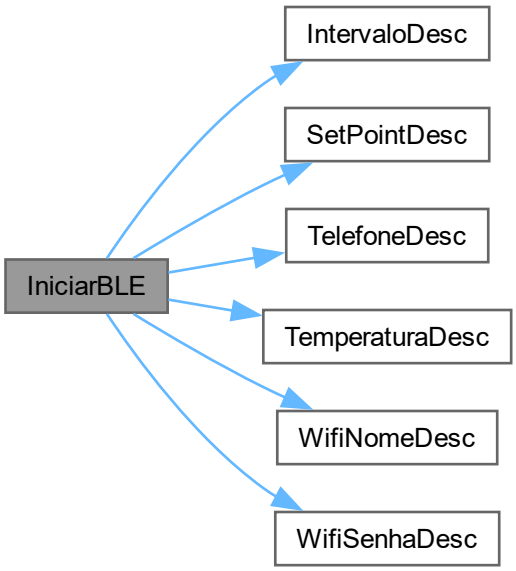
Inicializa o serviço BLE com todas as características e inicia o advertising.

Inicializa o serviço BLE com todas as características e inicia o advertising.

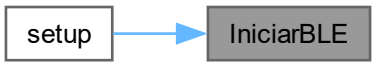
Configura o servidor BLE, cria as características e descritores para temperatura, intervalo de leitura e dados de Wi-Fi, e inicia o advertising BLE.

```
00116                 {
00117     BLEDevice::init("ThermoGuard");
00118     BLEServer *pServer = BLEDevice::createServer();
00119     pServer->setCallbacks(new MyServerCallbacks());
00120     BLEService *pService = pServer->createService(SERVICE_UUID);
00121     BLEService *pServiceWifi = pServer->createService(SERVICE_WIFI_UUID);
00122 
00123     // Temperatura
00124     TemperaturaDesc.setValue("Temperatura lida do sensor em celsius");
00125     TemperaturaChar.addDescriptor(&TemperaturaDesc);
00126     pService->addCharacteristic(&TemperaturaChar);
00127     SetPointDesc.setValue("Setpoint para alerta");
00128     SetPointChar.addDescriptor(&SetPointDesc);
00129     SetPointChar.setCallbacks(new CharCallback());
00130     pService->addCharacteristic(&SetPointChar);
00131     // Intervalo
00132     IntervaloDesc.setValue("Tempo entre leitura em minutos");
00133     IntervaloChar.addDescriptor(&IntervaloDesc);
00134     IntervaloChar.setCallbacks(new CharCallback());
00135     pService->addCharacteristic(&IntervaloChar);
00136 
00137     // Wifi
00138     WifiNomeDesc.setValue("Nome WiFi");
00139     WifiNomeChar.addDescriptor(&WifiNomeDesc);
00140     WifiNomeChar.setCallbacks(new CharCallback());
00141     pServiceWifi->addCharacteristic(&WifiNomeChar);
00142     WifiSenhaDesc.setValue("Senha WiFi");
00143     WifiSenhaChar.addDescriptor(&WifiSenhaDesc);
00144     WifiSenhaChar.setCallbacks(new CharCallback());
00145     pServiceWifi->addCharacteristic(&WifiSenhaChar);
00146     // Telefone
00147     TelefoneDesc.setValue("Numero de telefone");
00148     TelefoneChar.addDescriptor(&TelefoneDesc);
00149     TelefoneChar.setCallbacks(new CharCallback());
00150     pServiceWifi->addCharacteristic(&TelefoneChar);
00151 
00152     pService->start();
00153     pServiceWifi->start();
00154     BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
00155     pAdvertising->addServiceUUID(SERVICE_UUID);
00156     pAdvertising->addServiceUUID(SERVICE_WIFI_UUID);
00157     pServer->getAdvertising()->start();
00158 }
```

Grafo de chamadas desta função:



Este é o diagrama das funções que utilizam esta função:



5.2.4 Documentação das variáveis

5.2.4.1 deviceConnected

```
volatile bool deviceConnected  [extern]
```

5.3 bleConfig.h

Ir para a documentação deste ficheiro.

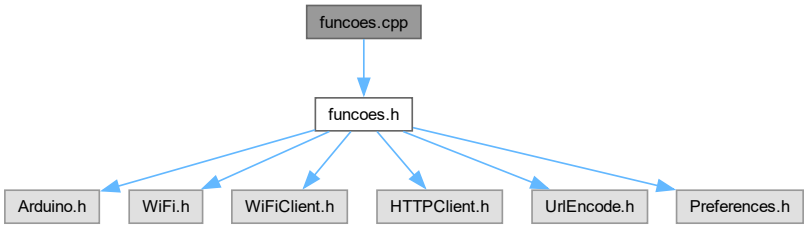
```
00001
00009
00010 #ifndef BLECONFIG_H
00011 #define BLECONFIGS_H
```

```
00012
00013 #define SERVICE_UUID BLEUUID((uint16_t)0x1809)
00014 #define SERVICE_WIFI_UUID "7be95774-8734-4f46-a326-573d290c1e9f"
00015
00016 #include <BLEDevice.h>
00017 #include <BLEUtils.h>
00018 #include <BLEServer.h>
00019 #include <BLE2902.h>
00020
00021 extern volatile bool deviceConnected;
00022
00026 void IniciarBLE();
00027
00031 void EncerrarBLE();
00032
00033 #endif
```

5.4 Referência ao ficheiro funcoes.cpp

Implementação das funções auxiliares para envio de dados e armazenamento.

`#include "funcoes.h"`
Diagrama de dependências de inclusão para funcoes.cpp:



Funções

- uint8_t EnviarWhats (String mensagem)

  Envia uma mensagem via WhatsApp usando CallMeBot API.
- uint8_t EnviarWeb (float valor)

  Envia um valor numérico para ThingSpeak via HTTP GET.
- uint8_t ConectaWifi (String nome, String senha)

  Conecta o ESP32 à rede Wi-Fi especificada.
- void DesconectaWifi ()

  Desconecta o ESP32 da rede Wi-Fi e desativa o Wi-Fi.
- void SalvaConfig (String wifi, String senha, float setPoint, uint32_t intervalo, String numero)

  Salva as configurações fornecidas na memória não volátil (NVS).
- void CarregarConfig ()

  Carrega as configurações armazenadas na NVS.

5.4.1 Descrição detalhada

Implementação das funções auxiliares para envio de dados e armazenamento.

As funções incluem:

- Enviar mensagem via CallMeBot (WhatsApp)
- Enviar valor para ThingSpeak
- Salvar e carregar configurações com Preferences
- Conectar/desconectar Wi-Fi

5.4.2 Documentação das funções

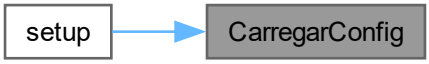
5.4.2.1 CarregarConfig()

```
void CarregarConfig ()
```

Carrega as configurações armazenadas na NVS.

```
00080                       {
00081 prefs.begin("config", true);
00082 nomeWifi = prefs.getString("wifi", "");
00083 senhaWifi = prefs.getString("senha", "");
00084 setPointTemp = prefs.getFloat("setPoint", 38.0);
00085 intervaloTemp = prefs.getUInt("intervalo", 15); // 15 min padrão
00086 numCelular = prefs.getString("cel", "");
00087 prefs.end();
00088 }
```

Este é o diagrama das funções que utilizam esta função:



5.4.2.2 ConectaWifi()

```
uint8_t ConectaWifi (
            String nome,
            String senha)
```

Conecta o ESP32 à rede Wi-Fi especificada.

Parâmetros

nome	SSID da rede.
senha	Senha da rede.

Retorna

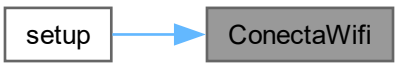
uint8_t

Valores retornados

1	Conectado com sucesso
0	Falha na conexão

```
00052                                                {
00053 WiFi.begin(nome, senha);
00054 unsigned long startTime = millis();
00055 while(millis() - startTime < 10000)
00056     if (WiFi.status() == WL_CONNECTED)
00057         break;
00058 if (WiFi.status() == WL_CONNECTED) {
00059     return 1;
00060 } else {
00061     return 0;
00062 }
00063 }
```

Este é o diagrama das funções que utilizam esta função:



5.4.2.3 DesconectaWifi()

```
void DesconectaWifi ()
```

Desconecta o ESP32 da rede Wi-Fi e desativa o Wi-Fi.

```
00065                       {
00066 WiFi.disconnect(true);
00067 WiFi.mode(WIFI_OFF);
00068 }
```

Este é o diagrama das funções que utilizam esta função:

5.4.2.4 EnviarWeb()

```
uint8_t EnviarWeb (
             float valor)
```

Envia um valor numérico para ThingSpeak via HTTP GET.

Parâmetros

valor	Valor float a ser enviado.

Retorna

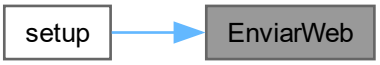
uint8_t

Valores retornados

1	Sucesso (HTTP 200)
0	Falha na requisição
2	Wi-Fi não conectado

```
00034                                {
00035 if (WiFi.status() == WL_CONNECTED) {
00036     HTTPClient http;
00037     String url = server + "?api_key=" + apiKey + "&field1=" + String(valor);
00038     http.begin(url);
00039     int httpResponseCode = http.GET();
00040
00041     if (httpResponseCode == 200) {
00042     http.end();
00043     return 1;
00044     } else {
00045     http.end();
00046     return 0;
00047     }
00048 }
00049 return 2;
00050 }
```

Este é o diagrama das funções que utilizam esta função:



5.4.2.5 EnviarWhats()

```
uint8_t EnviarWhats (
            String mensagem)
```

Envia uma mensagem via WhatsApp usando CallMeBot API.

Parâmetros

mensagem	Texto da mensagem a ser enviada.

Retorna

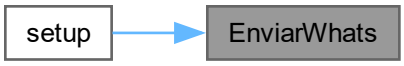
uint8_t

Valores retornados

1	Sucesso (HTTP 200)
0	Falha na requisição
2	Wi-Fi não conectado

```
00015                                     {
00016 if (WiFi.status() == WL_CONNECTED) {
00017     String url = "https://api.callmebot.com/whatsapp.php?phone=" + numCelular + "&apikey=" + apiWhats
      + "&text=" + urlEncode(mensagem);
00018     HTTPClient http;
00019     http.begin(url);
00020     http.addHeader("Content-Type", "application/x-www-form-urlencoded");
00021     int httpResponseCode = http.POST(url);
00022
00023     if (httpResponseCode == 200) {
00024     http.end();
00025     return 1;
00026     } else {
00027     http.end();
00028     return 0;
00029     }
00030 }
00031 return 2;
00032 }
```

Este é o diagrama das funções que utilizam esta função:



5.4.2.6 SalvaConfig()

```
void SalvaConfig (
             String wifi,
             String senha,
             float setPoint,
             uint32_t intervalo,
             String numero)
```

Salva as configurações fornecidas na memória não volátil (NVS).

Parâmetros

wifi	Nome da rede Wi-Fi.
senha	Senha da rede.
setPoint	Temperatura alvo de referência.
intervalo	Intervalo entre medições (em minutos).
numero	Número de telefone para alertas.

```
00070                                                                                   {
00071 prefs.begin("config", false);
00072 prefs.putString("wifi", wifi);
00073 prefs.putString("senha", senha);
```

```
00074 prefs.putFloat("setPoint", setPoint);
00075 prefs.putUInt("intervalo", intervalo);
00076 prefs.putString("cel", numero);
00077 prefs.end();
00078 }
```

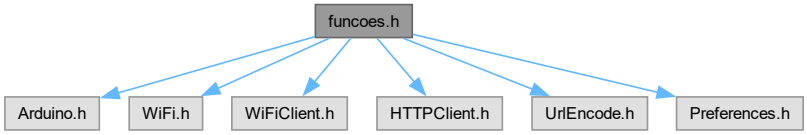
Este é o diagrama das funções que utilizam esta função:

5.5 Referência ao ficheiro funcoes.h

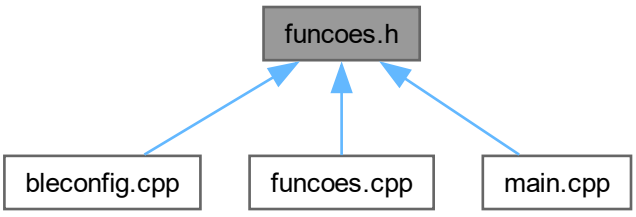
Declaração das funções auxiliares para comunicação Wi-Fi e armazenamento.

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <HTTPClient.h>
#include <UrlEncode.h>
#include <Preferences.h>
```

Diagrama de dependências de inclusão para funcoes.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Funções

- uint8_t EnviarWhats (String mensagem)

  Envia uma mensagem via WhatsApp usando CallMeBot API.
- uint8_t EnviarWeb (float valor)

  Envia um valor numérico para ThingSpeak via HTTP GET.
- uint8_t ConectaWifi (String nome, String senha)

  Conecta o ESP32 à rede Wi-Fi especificada.
- void DesconectaWifi ()

  Desconecta o ESP32 da rede Wi-Fi e desativa o Wi-Fi.
- void SalvaConfig (String wifi, String senha, float setPoint, uint32_t intervalo, String numero)

  Salva as configurações fornecidas na memória não volátil (NVS).
- void CarregarConfig ()

  Carrega as configurações armazenadas na NVS.

Variáveis

- String numCelular
- String apiWhats
- String nomeWifi

  Nome da rede Wi-Fi para conexão.
- String senhaWifi

  Senha da rede Wi-Fi.
- String apiKey

  API key de acesso ao ThingSpeak.
- String server

  URL do servidor ThingSpeak para envio de dados.
- float setPointTemp
- uint32_t intervaloTemp
- Preferences prefs

5.5.1 Descrição detalhada

Declaração das funções auxiliares para comunicação Wi-Fi e armazenamento.

Contém funções para:

- Conexão e desconexão do Wi-Fi
- Envio de dados via HTTP
- Armazenamento e carregamento de configurações na NVS

5.5.2 Documentação das funções

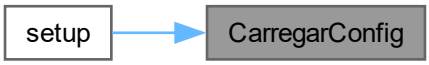
5.5.2.1 CarregarConfig()

```
void CarregarConfig ()
```

Carrega as configurações armazenadas na NVS.

```
00080                       {
00081 prefs.begin("config", true);
00082 nomeWifi = prefs.getString("wifi", "");
00083 senhaWifi = prefs.getString("senha", "");
00084 setPointTemp = prefs.getFloat("setPoint", 38.0);
00085 intervaloTemp = prefs.getUInt("intervalo", 15); // 15 min padrão
00086 numCelular = prefs.getString("cel", "");
00087 prefs.end();
00088 }
```

Este é o diagrama das funções que utilizam esta função:



5.5.2.2 ConectaWifi()

```
uint8_t ConectaWifi (
            String nome,
            String senha)
```

Conecta o ESP32 à rede Wi-Fi especificada.

Parâmetros

nome	SSID da rede.
senha	Senha da rede.

Retorna

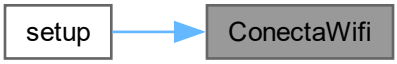
uint8_t

Valores retornados

1	Conectado com sucesso
0	Falha na conexão

```
00052                                                {
00053 WiFi.begin(nome, senha);
00054 unsigned long startTime = millis();
00055 while(millis() - startTime < 10000)
00056     if (WiFi.status() == WL_CONNECTED)
00057         break;
00058 if (WiFi.status() == WL_CONNECTED) {
00059     return 1;
00060 } else {
00061     return 0;
00062 }
00063 }
```

Este é o diagrama das funções que utilizam esta função:



5.5.2.3 DesconectaWifi()

```
void DesconectaWifi ()
```

Desconecta o ESP32 da rede Wi-Fi e desativa o Wi-Fi.

```
00065                       {
00066 WiFi.disconnect(true);
00067 WiFi.mode(WIFI_OFF);
00068 }
```

Este é o diagrama das funções que utilizam esta função:

5.5.2.4 EnviarWeb()

```
uint8_t EnviarWeb (
             float valor)
```

Envia um valor numérico para ThingSpeak via HTTP GET.

Parâmetros

valor	Valor float a ser enviado.

Retorna

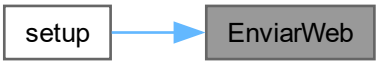
uint8_t

Valores retornados

1	Sucesso (HTTP 200)
0	Falha na requisição
2	Wi-Fi não conectado

```
00034                                {
00035 if (WiFi.status() == WL_CONNECTED) {
00036     HTTPClient http;
00037     String url = server + "?api_key=" + apiKey + "&field1=" + String(valor);
00038     http.begin(url);
00039     int httpResponseCode = http.GET();
00040
00041     if (httpResponseCode == 200) {
00042     http.end();
00043     return 1;
00044     } else {
00045     http.end();
00046     return 0;
00047     }
00048 }
00049 return 2;
00050 }
```

Este é o diagrama das funções que utilizam esta função:



5.5.2.5 EnviarWhats()

```
uint8_t EnviarWhats (
            String mensagem)
```

Envia uma mensagem via WhatsApp usando CallMeBot API.

Parâmetros

mensagem	Texto da mensagem a ser enviada.

Retorna

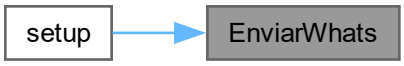
uint8_t

Valores retornados

1	Sucesso (HTTP 200)
0	Falha na requisição
2	Wi-Fi não conectado

```
00015                                     {
00016 if (WiFi.status() == WL_CONNECTED) {
00017     String url = "https://api.callmebot.com/whatsapp.php?phone=" + numCelular + "&apikey=" + apiWhats
      + "&text=" + urlEncode(mensagem);
00018     HTTPClient http;
00019     http.begin(url);
00020     http.addHeader("Content-Type", "application/x-www-form-urlencoded");
00021     int httpResponseCode = http.POST(url);
00022
00023     if (httpResponseCode == 200) {
00024     http.end();
00025     return 1;
00026     } else {
00027     http.end();
00028     return 0;
00029     }
00030 }
00031 return 2;
00032 }
```

Este é o diagrama das funções que utilizam esta função:



5.5.2.6 SalvaConfig()

```
void SalvaConfig (
             String wifi,
             String senha,
             float setPoint,
             uint32_t intervalo,
             String numero)
```

Salva as configurações fornecidas na memória não volátil (NVS).

Parâmetros

wifi	Nome da rede Wi-Fi.
senha	Senha da rede.
setPoint	Temperatura alvo de referência.
intervalo	Intervalo entre medições (em minutos).
numero	Número de telefone para alertas.

```
00070                                                                                                 {
00071 prefs.begin("config", false);
00072 prefs.putString("wifi", wifi);
00073 prefs.putString("senha", senha);
```

```
00074 prefs.putFloat("setPoint", setPoint);
00075 prefs.putUInt("intervalo", intervalo);
00076 prefs.putString("cel", numero);
00077 prefs.end();
00078 }
```

Este é o diagrama das funções que utilizam esta função:

5.5.3 Documentação das variáveis

5.5.3.1 apiKey

```
String apiKey  [extern]
```

API key de acesso ao ThingSpeak.

5.5.3.2 apiWhats

```
String apiWhats  [extern]
```

5.5.3.3 intervaloTemp

```
uint32_t intervaloTemp  [extern]
```

5.5.3.4 nomeWifi

```
String nomeWifi  [extern]
```

Nome da rede Wi-Fi para conexão.

5.5.3.5 numCelular

```
String numCelular  [extern]
```

5.5.3.6 prefs

```
Preferences prefs  [extern]
```

5.5.3.7 senhaWifi

```
String senhaWifi  [extern]
```

Senha da rede Wi-Fi.

5.5.3.8 server

```
String server  [extern]
```

URL do servidor ThingSpeak para envio de dados.

5.5.3.9 setPointTemp

```
float setPointTemp  [extern]
```

5.6 funcoes.h

Ir para a documentação deste ficheiro.

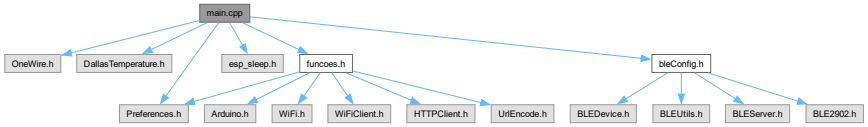
```
00001
00011
00012 #ifndef FUNCOES_H
00013 #define FUNCOES_H
00014
00015 #include <Arduino.h>
00016 #include <WiFi.h>
00017 #include <WiFiClient.h>
00018 #include <HTTPClient.h>
00019 #include <UrlEncode.h>
00020 #include <Preferences.h>
00021
00022
00023 extern String numCelular;
00024 extern String apiWhats;
00025 extern String nomeWifi;
00026 extern String senhaWifi;
00027 extern String apiKey;
00028 extern String server;
00029 extern float setPointTemp;
00030 extern uint32_t intervaloTemp;
00031 extern Preferences prefs;
00032
00042 uint8_t EnviarWhats(String mensagem);
00043
00053 uint8_t EnviarWeb(float valor);
00054
00064 uint8_t ConectaWifi(String nome, String senha);
00065
00069 void DesconectaWifi();
00070
00080 void SalvaConfig(String wifi, String senha, float setPoint, uint32_t intervalo, String numero);
00081
00085 void CarregarConfig();
00086
00087 #endif // FUNCOES_H
```

5.7 Referência ao ficheiro main.cpp

Loop principal do sistema de monitoramento de temperatura com ESP32.

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Preferences.h>
#include <esp_sleep.h>
#include "funcoes.h"
#include "bleConfig.h"
```

Diagrama de dependências de inclusão para main.cpp:



Funções

- OneWire PinOneWire (4)

  Instância do barramento OneWire para comunicação com o sensor DS18B20.
- void setup ()

  Função de configuração inicial do dispositivo.
- void loop ()

Variáveis

- String numCelular = ""
- String apiWhats = "6043662"
- String nomeWifi = ""

  Nome da rede Wi-Fi para conexão.
- String senhaWifi = ""

  Senha da rede Wi-Fi.
- String server = "http://api.thingspeak.com/update"

  URL do servidor ThingSpeak para envio de dados.
- String apiKey = "J7NA0N95H2SGO68W"

  API key de acesso ao ThingSpeak.
- float setPointTemp = 0.0
- uint32_t intervaloTemp = 15
- DallasTemperature TempSensor & PinOneWire

  Instância do sensor de temperatura DallasTemperature.
- RTC_DATA_ATTR float ultimaTemp = 0
- Preferences prefs
- volatile bool deviceConnected = false
- BLECharacteristic TemperaturaChar
- BLECharacteristic SetPointChar
- BLECharacteristic IntervaloChar
- BLECharacteristic WifiNomeChar
- BLECharacteristic WifiSenhaChar
- BLECharacteristic TelefoneChar

5.7.1 Descrição detalhada

Loop principal do sistema de monitoramento de temperatura com ESP32.

- Leitura de temperatura via sensor DS18B20.
- Envio de dados para ThingSpeak.
- Envio de alerta via WhatsApp (CallMeBot).
- Comunicação BLE para leitura e configuração de parâmetros (setpoint, Wi-Fi, telefone).
- Alterna entre modo Wi-Fi (temporizado) e BLE (botão físico).
- Persistência de dados com Preferences e RTC_DATA_ATTR.

Autor

Matheus Orsini

Data

2025-04-26

5.7.2 Documentação das funções

5.7.2.1 loop()

```
void loop ()
00156            {
00157   Serial.println("Nunca Roda");
00158   // put your main code here, to run repeatedly:
00159 }
```

5.7.2.2 PinOneWire()

```
OneWire PinOneWire (
             4 )
```

Instância do barramento OneWire para comunicação com o sensor DS18B20.

5.7.2.3 setup()

```
void setup ()
```

Função de configuração inicial do dispositivo.

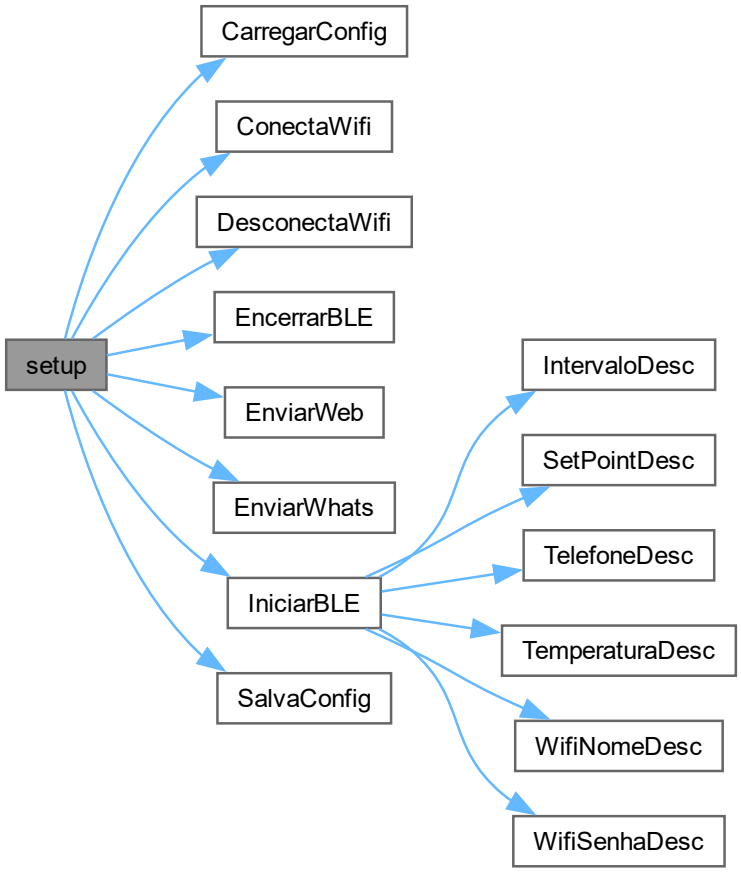
Inicializa o monitor serial, configura o sensor de temperatura, define as causas de despertar do deep sleep (timer e botão físico) e realiza a ação apropriada conforme a causa do despertar:

- Timer: conecta ao Wi-Fi e envia dados ao ThingSpeak.
- Botão: inicia o servidor BLE para envio de dados via Bluetooth.
- Outro: apenas imprime mensagem no console.

```
00072             {
00073   CarregarConfig();
00074
00075   Serial.begin(115200);
00076
00077   esp_sleep_enable_ext0_wakeup(GPIO_NUM_15, 1);
00078
00079   esp_sleep_enable_timer_wakeup(intervaloTemp * 1000000ULL);
00080
00081   Serial.print("ultima temperatura lida:"); Serial.println(ultimaTemp);
00082   Serial.print("set point lidos:"); Serial.println(setPointTemp);
00083   Serial.print("intervalo lidos:"); Serial.println(intervaloTemp);
00084   Serial.print("wifi nome lidos:"); Serial.println(nomeWifi);
00085   Serial.print("wifi senha lidos:"); Serial.println(senhaWifi);
00086   Serial.print("telefone lido:"); Serial.println(numCelular);
00087
00088   TempSensor.begin();
00089   TempSensor.requestTemperatures();
00090   ultimaTemp = TempSensor.getTempCByIndex(0);
00091
00092   switch (esp_sleep_get_wakeup_cause())
00093   {
00094     case ESP_SLEEP_WAKEUP_TIMER:{
00095       if(ConectaWifi(nomeWifi, senhaWifi)){
00096         if (ultimaTemp >= setPointTemp){
00097           EnviarWhats("ALERTA!!\nTemperatura acima da configurada!\nVerifique o paciente!!");
00098         }
00099         if(EnviarWeb(ultimaTemp) == 1){
00100           Serial.println("Enviado com sucesso");
00101         }else{
00102           Serial.println("Falha ao enviar");
00103         }
00104         DesconectaWifi();
00105         Serial.println("Acordei pelo timer!");
00106         Serial.print("Leitura de:");
00107         Serial.println(ultimaTemp);
00108       }else{
00109         Serial.println("Nao foi possível se conectar ao wifi");
00110       }
00111       break;
00112     }
00113     case ESP_SLEEP_WAKEUP_EXT0:{
00114       Serial.println("Acordei pelo botão!");
00115       IniciarBLE();
00116       uint32_t tempo = millis();
00117       while (((millis() - tempo) < 1 * 60 * 1000) || deviceConnected){
00118         if (deviceConnected) {
00119           // TempSensor.requestTemperatures();
00120           // ultimaTemp = TempSensor.getTempCByIndex(0);
00121           // String tempStr = String(ultimaTemp,2);
00122
00123           // TemperaturaChar.setValue(tempStr.c_str());
00124           // TemperaturaChar.notify();
00125
00126           // SetPointChar.setValue(setPointTemp);
00127           // IntervaloChar.notify();
00128
00129           // IntervaloChar.setValue(intervaloTemp);
00130           // IntervaloChar.notify();
00131
00132           // WifiNomeChar.setValue(nomeWifi.c_str());
00133           // WifiNomeChar.notify();
00134
00135           // WifiSenhaChar.setValue(senhaWifi.c_str());
00136           // WifiSenhaChar.notify();
```

```
00137
00138           // TelefoneChar.setValue(numCelular.c_str());
00139           // TelefoneChar.notify();
00140
00141         }
00142       }
00143       EncerrarBLE();
00144       break;
00145     }
00146     default:{
00147       Serial.println("Acordei aleatório!");
00148       break;
00149     }
00150   }
00151   Serial.println("Dormindo...");
00152   SalvaConfig(nomeWifi,senhaWifi,setPointTemp,intervaloTemp,numCelular);
00153   esp_deep_sleep_start();
00154 }
```

Grafo de chamadas desta função:



5.7.3 Documentação das variáveis

5.7.3.1 apiKey

```
String apiKey = "J7NA0N95H2SGO68W"
```

API key de acesso ao ThingSpeak.

5.7.3.2 apiWhats

```
String apiWhats = "6043662"
```

5.7.3.3 deviceConnected

```
volatile bool deviceConnected = false
```

5.7.3.4 IntervaloChar

```
BLECharacteristic IntervaloChar  [extern]
```

5.7.3.5 intervaloTemp

```
uint32_t intervaloTemp = 15
```

5.7.3.6 nomeWifi

```
String nomeWifi = ""
```

Nome da rede Wi-Fi para conexão.

5.7.3.7 numCelular

```
String numCelular = ""
```

5.7.3.8 PinOneWire

```
DallasTemperature TempSensor& PinOneWire
```

Instância do sensor de temperatura DallasTemperature.

5.7.3.9 prefs

```
Preferences prefs
```

5.7.3.10 senhaWifi

```
String senhaWifi = ""
```

Senha da rede Wi-Fi.

5.7.3.11 server

```
String server = "http://api.thingspeak.com/update"
```

URL do servidor ThingSpeak para envio de dados.

5.7.3.12 SetPointChar

```
BLECharacteristic SetPointChar  [extern]
```

5.7.3.13 setPointTemp

```
float setPointTemp = 0.0
```

5.7.3.14 TelefoneChar

```
BLECharacteristic TelefoneChar  [extern]
```

5.7.3.15 TemperaturaChar

```
BLECharacteristic TemperaturaChar  [extern]
```

5.7.3.16 ultimaTemp

```
RTC_DATA_ATTR float ultimaTemp = 0
```

5.7.3.17 WifiNomeChar

```
BLECharacteristic WifiNomeChar  [extern]
```

5.7.3.18 WifiSenhaChar

```
BLECharacteristic WifiSenhaChar  [extern]
```