Documentacion Arduino

1

Generado por Doxygen 1.12.0

1. Biometría y Medio Ambiente - Arduino

Este proyecto combina tecnologías biométricas y monitoreo ambiental utilizando dispositivos basados en Arduino. El objetivo es recopilar datos biométricos y ambientales (como la calidad del aire, temperatura, humedad, etc.)

y emplearlos para análisis y seguimiento. Este sistema puede ser útil para monitorear entornos en tiempo real y hacer un análisis profundo de las condiciones ambientales.

1.1. Tecnologías utilizadas

- **Plataforma:** Arduino
- **Lenguaje de programación:** C++
- **Componentes de hardware:**
  - Placa Arduino: `SparkFun Pro nRF52340 Mini`
  - Batería externa: `Lithium Ion Battery - 850mAh`
  - Sensor de Gas: `ULPSM-O3 968-046` (sensor de ozono)
  - Otros sensores opcionales para medición biométrica o ambiental.

1.2. Instalación y configuración

1.2.1. Requisitos previos:

1. **Arduino IDE**: Descarga e instala el `Arduino IDE` si aún no lo tienes.
2. **Librerías necesarias**:
   - **Adafruit nRF52**: Esta librería es necesaria para que la placa SparkFun Pro nRF52340 Mini sea reconocida y funcione correctamente.
   - Sigue la guía de instalación de SparkFun `aquí` para agregar el soporte en Arduino IDE.

1.2.2. Instrucciones:

1. Clona el repositorio del proyecto:
   ```
   git clone https://github.com/SentoMarcos/Biometr-a-y-Medio-Ambiente-Arduino.git
   ```
2. Abre el archivo `.ino` principal en el Arduino IDE.
3. Conecta tu placa **SparkFun Pro nRF52340 Mini** a tu ordenador mediante un cable USB.
4. En el **Arduino IDE**, selecciona la placa y el puerto correspondientes. Esto se hace en el menú `Herramientas > Placa` y `Herramientas > Puerto`.
5. Asegúrate de que todas las librerías necesarias estén instaladas y sincronizadas, luego carga el código a la placa.
6. Conecta los sensores (como el **ULPSM-O3 968-046**) a los pines adecuados según el diagrama de conexión proporcionado en la documentación del proyecto.
7. Conecta la batería **Lithium Ion Battery - 850mAh** para asegurar la alimentación cuando el sistema no esté conectado por USB.

1.3. Uso

Una vez que el sistema esté configurado y cargado con el código, la placa comenzará a recopilar datos ambientales (como niveles de ozono) a través del sensor de gas **ULPSM-O3 968-046**. Los datos se pueden visualizar en tiempo real a través del Monitor Serie del Arduino IDE, o se pueden transmitir a una plataforma externa para su análisis.

Es posible adaptar el código para agregar otros sensores biométricos o ambientales, según las necesidades del proyecto.

1.3.1. Ejemplo de salida:

```
Niveles de Ozono: 0.04 ppm
Temperatura: 25.5 °C
```

1.4. Autores

- SentoMarcos

1.5. Proyectos Relacionados

- Biometría y Medio Ambiente - Android
- Biometría y Medio Ambiente - Web

2. Índice de clases

2.1. Lista de clases

Lista de clases, estructuras, uniones e interfaces con breves descripciones:

3. Índice de archivos

3.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

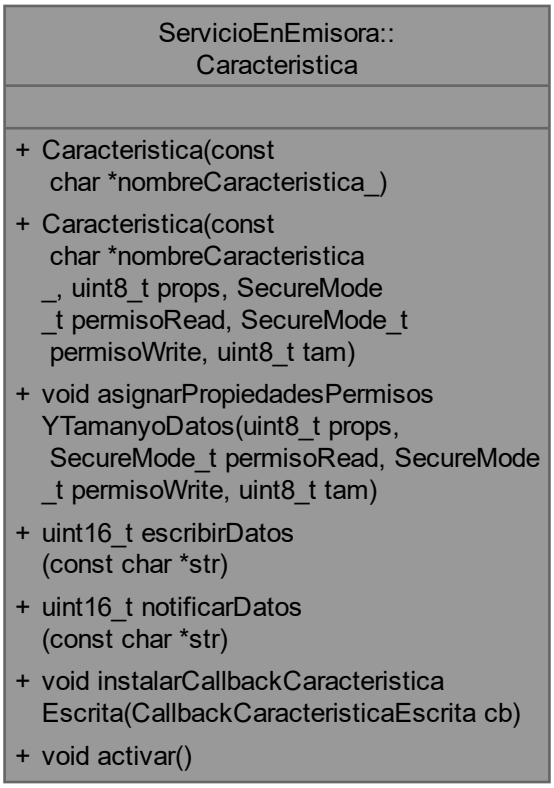
4. Documentación de clases

4.1. Referencia de la clase ServicioEnEmisora::Caracteristica

Clase para añadir características a un servicio BLE.

`#include <ServicioEnEmisora.h>`

Diagrama de colaboración de ServicioEnEmisora::Caracteristica:



Métodos públicos

- Caracteristica (const char *nombreCaracteristica_)

  Constructor de la clase Caracteristica.
- Caracteristica (const char *nombreCaracteristica_, uint8_t props, SecureMode_t permisoRead, SecureMode_t permisoWrite, uint8_t tam)

  Constructor de la clase Caracteristica.
- void asignarPropiedadesPermisosYTamanyoDatos (uint8_t props, SecureMode_t permisoRead, SecureMode_t permisoWrite, uint8_t tam)

  @function asignarPropiedadesPermisosYTamanyoDatos
- uint16_t escribirDatos (const char *str)

  Escribe datos en la característica.
- uint16_t notificarDatos (const char *str)

  Notifica datos en la característica.
- void instalarCallbackCaracteristicaEscrita (CallbackCaracteristicaEscrita cb)

  @function instalarCallbackCaracteristicaEscrita
- void activar ()

  @function activar

4.1.1. Descripción detallada

Clase para añadir características a un servicio BLE.

Definición en la línea 71 del archivo ServicioEnEmisora.h.

4.1.2. Documentación de constructores y destructores

Caracteristica() [1/2]

```
ServicioEnEmisora::Caracteristica::Caracteristica (
            const char * nombreCaracteristica_)  [inline]
```

Constructor de la clase Caracteristica.

Parámetros

*nombreCaracteristica_*	Nombre de la característica.
props	Propiedades de la característica.
permisoRead	Permisos de lectura.
permisoWrite	Permisos de escritura.
tam	Tamaño de los datos.

Nota

Este constructor inicializa una característica BLE con los valores dados.

Definición en la línea 104 del archivo ServicioEnEmisora.h.

Caracteristica() [2/2]

```
ServicioEnEmisora::Caracteristica::Caracteristica (
            const char * nombreCaracteristica_,
            uint8_t props,
            SecureMode_t permisoRead,
            SecureMode_t permisoWrite,
            uint8_t tam)  [inline]
```

Constructor de la clase Caracteristica.

Parámetros

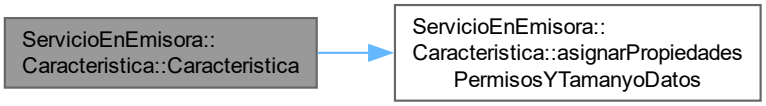
*nombre↩ Caracteristica_*	Nombre de la característica.
props	Propiedades de la característica.
permisoRead	Permisos de lectura.
permisoWrite	Permisos de escritura.
tam	Tamaño de los datos.

Nota

Este constructor inicializa una característica BLE con los valores dados.

Definición en la línea 118 del archivo ServicioEnEmisora.h.

Gráfico de llamadas de esta función:



4.1.3. Documentación de funciones miembro

activar()

```
void ServicioEnEmisora::Caracteristica::activar ()  [inline]
```

@function activar

Activa la característica.

Definición en la línea 223 del archivo ServicioEnEmisora.h.

asignarPropiedadesPermisosYTamanyoDatos()

```
void ServicioEnEmisora::Caracteristica::asignarPropiedadesPermisosYTamanyoDatos (
            uint8_t props,
            SecureMode_t permisoRead,
            SecureMode_t permisoWrite,
            uint8_t tam)  [inline]
```

@function asignarPropiedadesPermisosYTamanyoDatos

Asigna propiedades, permisos y tamaño de datos a la característica.

Parámetros

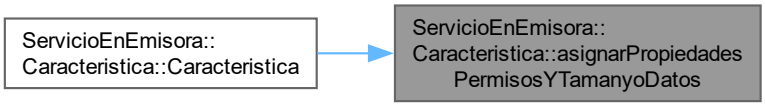
props	Propiedades de la característica.
permisoRead	Permisos de lectura.
permisoWrite	Permisos de escritura.
tam	Tamaño de los datos.

Nota

Este método asigna propiedades, permisos y tamaño de datos a la característica.

Definición en la línea 172 del archivo ServicioEnEmisora.h.

Gráfico de llamadas a esta función:



escribirDatos()

```
uint16_t ServicioEnEmisora::Caracteristica::escribirDatos (
            const char * str)  [inline]
```

Escribe datos en la característica.

Parámetros

str	Datos a escribir.

Devuelve

Número de bytes escritos.

Definición en la línea 187 del archivo ServicioEnEmisora.h.

instalarCallbackCaracteristicaEscrita()

```
void ServicioEnEmisora::Caracteristica::instalarCallbackCaracteristicaEscrita (
            CallbackCaracteristicaEscrita cb)  [inline]
```

@function instalarCallbackCaracteristicaEscrita

Instala un callback para manejar escrituras en la característica.

Parámetros

cb	Función callback para manejar escrituras.

Definición en la línea 215 del archivo ServicioEnEmisora.h.

notificarDatos()

```
uint16_t ServicioEnEmisora::Caracteristica::notificarDatos (
            const char * str)  [inline]
```

Notifica datos en la característica.

Parámetros

str	Datos a notificar.

Devuelve

Número de bytes notificados.

Definición en la línea 203 del archivo ServicioEnEmisora.h.

La documentación de esta clase está generada del siguiente archivo:

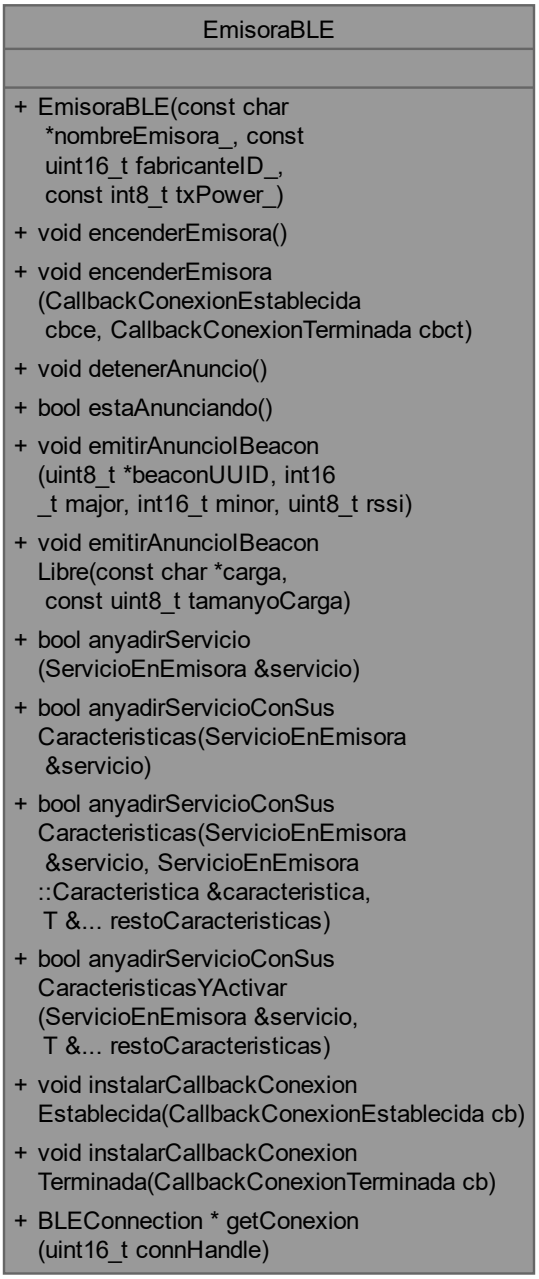
- ServicioEnEmisora.h

4.2. Referencia de la clase EmisoraBLE

Clase para manejar una emisora Bluetooth Low Energy (BLE).

```
#include <EmisoraBLE.h>
```

Diagrama de colaboración de EmisoraBLE:



Tipos públicos

- using CallbackConexionEstablecida = void(uint16_t connHandle)

  Definición de un tipo de función callback para manejar conexiones BLE establecidas.
- using CallbackConexionTerminada = void(uint16_t connHandle, uint8_t reason)

  Definición de un tipo de función callback para manejar desconexiones BLE.

Métodos públicos

- EmisoraBLE (const char *nombreEmisora_, const uint16_t fabricanteID_, const int8_t txPower_)

  Constructor de la clase EmisoraBLE.
- void encenderEmisora ()

  Enciende la emisora BLE.
- void encenderEmisora (CallbackConexionEstablecida cbce, CallbackConexionTerminada cbct)

  Enciende la emisora BLE y configura callbacks para eventos de conexión.
- void detenerAnuncio ()

  Detiene cualquier anuncio BLE activo.
- bool estaAnunciando ()

  Verifica si se está emitiendo un anuncio.
- void emitirAnuncioIBeacon (uint8_t *beaconUUID, int16_t major, int16_t minor, uint8_t rssi)

  Emite un anuncio iBeacon con los parámetros dados.
- void emitirAnuncioIBeaconLibre (const char *carga, const uint8_t tamanyoCarga)

  Emite un anuncio iBeacon con una carga personalizada.
- bool anyadirServicio (ServicioEnEmisora &servicio)

  Añade un servicio a la emisora BLE.
- bool anyadirServicioConSusCaracteristicas (ServicioEnEmisora &servicio)

  Añade un servicio BLE con sus características a la emisora.
- template<typename... T>
  bool anyadirServicioConSusCaracteristicas (ServicioEnEmisora &servicio, ServicioEnEmisora::Caracteristica &caracteristica, T &... restoCaracteristicas)

  Añade un servicio y varias características a la emisora.
- template<typename... T>
  bool anyadirServicioConSusCaracteristicasYActivar (ServicioEnEmisora &servicio, T &... resto↩
  Caracteristicas)

  Añade un servicio y sus características y lo activa.
- void instalarCallbackConexionEstablecida (CallbackConexionEstablecida cb)

  Instala un callback para cuando se establezca una conexión BLE.
- void instalarCallbackConexionTerminada (CallbackConexionTerminada cb)

  Instala un callback para cuando se termine una conexión BLE.
- BLEConnection * getConexion (uint16_t connHandle)

  Obtiene el objeto de conexión BLE dado un identificador de conexión.

4.2.1. Descripción detallada

Clase para manejar una emisora Bluetooth Low Energy (BLE).

Definición en la línea 32 del archivo EmisoraBLE.h.

4.2.2. Documentación de los «Typedef» miembros de la clase

CallbackConexionEstablecida

```
using EmisoraBLE::CallbackConexionEstablecida = void(uint16_t connHandle)
```

Definición de un tipo de función callback para manejar conexiones BLE establecidas.

Parámetros

connHandle	Identificador de la conexión BLE.

Definición en la línea 47 del archivo EmisoraBLE.h.

CallbackConexionTerminada

```
using EmisoraBLE::CallbackConexionTerminada = void(uint16_t connHandle, uint8_t reason)
```

Definición de un tipo de función callback para manejar desconexiones BLE.

Parámetros

connHandle	Identificador de la conexión BLE.
reason	Razón por la cual la conexión BLE se ha terminado.

Definición en la línea 54 del archivo EmisoraBLE.h.

4.2.3. Documentación de constructores y destructores

EmisoraBLE()

```
EmisoraBLE::EmisoraBLE (
            const char * nombreEmisora_,
            const uint16_t fabricanteID_,
            const int8_t txPower_)  [inline]
```

Constructor de la clase EmisoraBLE.

Este constructor inicializa una emisora BLE con los valores dados.

Parámetros

*nombre↩ Emisora_*	Nombre de la emisora.
*fabricanteID_*	ID del fabricante del beacon.
*txPower_*	Potencia de transmisión del beacon.

Definición en la línea 65 del archivo EmisoraBLE.h.

4.2.4. Documentación de funciones miembro

anyadirServicio()

```
bool EmisoraBLE::anyadirServicio (
            ServicioEnEmisora & servicio)  [inline]
```

Añade un servicio a la emisora BLE.

Parámetros

servicio	Servicio BLE que se va a añadir.

Devuelve

`true` si el servicio fue añadido con éxito, de lo contrario `false`.

Definición en la línea 317 del archivo EmisoraBLE.h.

anyadirServicioConSusCaracteristicas() [1/2]

```
bool EmisoraBLE::anyadirServicioConSusCaracteristicas (
            ServicioEnEmisora & servicio)  [inline]
```

Añade un servicio BLE con sus características a la emisora.

Parámetros

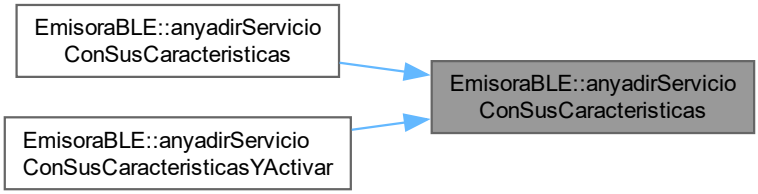
servicio	Servicio BLE que se va a añadir.

Devuelve

`true` si el servicio fue añadido con éxito, de lo contrario `false`.

Definición en la línea 341 del archivo EmisoraBLE.h.

Gráfico de llamadas a esta función:



anyadirServicioConSusCaracteristicas() [2/2]

```
template<typename... T>
bool EmisoraBLE::anyadirServicioConSusCaracteristicas (
            ServicioEnEmisora & servicio,
            ServicioEnEmisora::Caracteristica & caracteristica,
            T &... restoCaracteristicas)  [inline]
```

Añade un servicio y varias características a la emisora.

Este método permite añadir un servicio junto con un número variable de características.

Parámetros de plantilla

T	Tipos de las características.

Parámetros

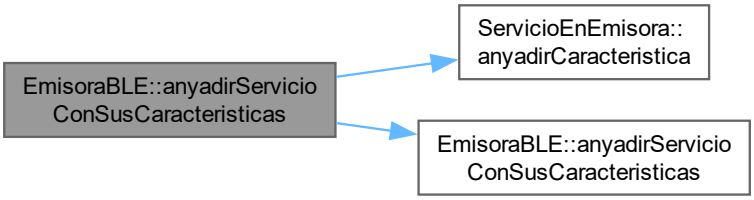
servicio	Servicio BLE que se va a añadir.
caracteristica	Primera característica a añadir.
restoCaracteristicas	Resto de características.

Devuelve

`true` si el servicio y las características fueron añadidos con éxito.

Definición en la línea 357 del archivo EmisoraBLE.h.

Gráfico de llamadas de esta función:



anyadirServicioConSusCaracteristicasYActivar()

```
template<typename... T>
bool EmisoraBLE::anyadirServicioConSusCaracteristicasYActivar (
            ServicioEnEmisora & servicio,
            T &... restoCaracteristicas)  [inline]
```

Añade un servicio y sus características y lo activa.

Parámetros de plantilla

T	Tipos de las características.

Parámetros

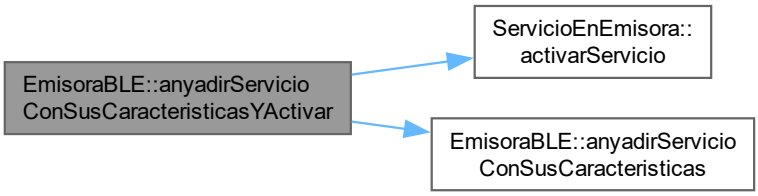
servicio	Servicio BLE que se va a añadir.
restoCaracteristicas	Características a añadir.

Devuelve

`true` si el servicio fue añadido y activado con éxito.

Definición en la línea 376 del archivo EmisoraBLE.h.

Gráfico de llamadas de esta función:



detenerAnuncio()

```
void EmisoraBLE::detenerAnuncio ()  [inline]
```

Detiene cualquier anuncio BLE activo.

Definición en la línea 127 del archivo EmisoraBLE.h.

emitirAnuncioIBeacon()

```
void EmisoraBLE::emitirAnuncioIBeacon (
            uint8_t * beaconUUID,
            int16_t major,
            int16_t minor,
            uint8_t rssi)  [inline]
```

Emite un anuncio iBeacon con los parámetros dados.

Parámetros

beaconUUID	UUID del beacon.
major	Valor mayor del beacon.
minor	Valor menor del beacon.
rssi	Valor RSSI (Received Signal Strength Indicator).

Definición en la línea 156 del archivo EmisoraBLE.h.

emitirAnuncioIBeaconLibre()

```
void EmisoraBLE::emitirAnuncioIBeaconLibre (
            const char * carga,
            const uint8_t tamanyoCarga)  [inline]
```

Emite un anuncio iBeacon con una carga personalizada.

Parámetros

carga	Datos a emitir en la carga del beacon.
tamanyoCarga	Tamaño de la carga a emitir.

Definición en la línea 251 del archivo EmisoraBLE.h.

encenderEmisora() [1/2]

```
void EmisoraBLE::encenderEmisora ()  [inline]
```

Enciende la emisora BLE.

Este método inicializa la emisora BLE y detiene cualquier anuncio existente.

Definición en la línea 100 del archivo EmisoraBLE.h.

Gráfico de llamadas a esta función:

encenderEmisora() [2/2]

```
void EmisoraBLE::encenderEmisora (
            CallbackConexionEstablecida cbce,
            CallbackConexionTerminada cbct)  [inline]
```

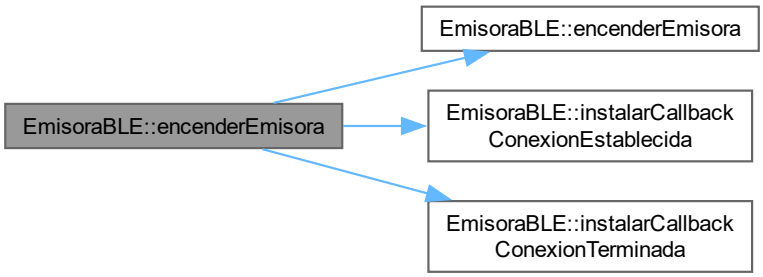
Enciende la emisora BLE y configura callbacks para eventos de conexión.

Parámetros

cbce	Callback que se ejecuta cuando se establece una conexión.
cbct	Callback que se ejecuta cuando se termina una conexión.

Definición en la línea 114 del archivo EmisoraBLE.h.

Gráfico de llamadas de esta función:



estaAnunciando()

```
bool EmisoraBLE::estaAnunciando ()  [inline]
```

Verifica si se está emitiendo un anuncio.

Devuelve

`true` si la emisora está anunciando, de lo contrario `false`.

Definición en la línea 144 del archivo EmisoraBLE.h.

getConexion()

```
BLEConnection * EmisoraBLE::getConexion (
            uint16_t connHandle)  [inline]
```

Obtiene el objeto de conexión BLE dado un identificador de conexión.

Parámetros

connHandle	Identificador de la conexión BLE.

Devuelve

Puntero a la conexión BLE correspondiente.

Definición en la línea 413 del archivo EmisoraBLE.h.

instalarCallbackConexionEstablecida()

```
void EmisoraBLE::instalarCallbackConexionEstablecida (
            CallbackConexionEstablecida cb)  [inline]
```

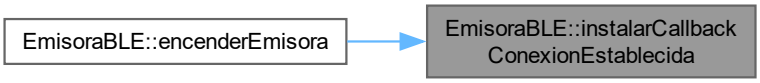
Instala un callback para cuando se establezca una conexión BLE.

Parámetros

cb	Función callback para manejar la conexión.

Definición en la línea 394 del archivo EmisoraBLE.h.

Gráfico de llamadas a esta función:



instalarCallbackConexionTerminada()

```
void EmisoraBLE::instalarCallbackConexionTerminada (
            CallbackConexionTerminada cb)  [inline]
```

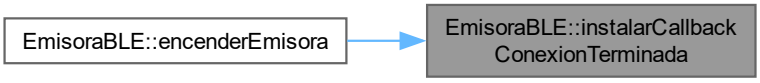
Instala un callback para cuando se termine una conexión BLE.

Parámetros

cb	Función callback para manejar la desconexión.

Definición en la línea 403 del archivo EmisoraBLE.h.

Gráfico de llamadas a esta función:



La documentación de esta clase está generada del siguiente archivo:

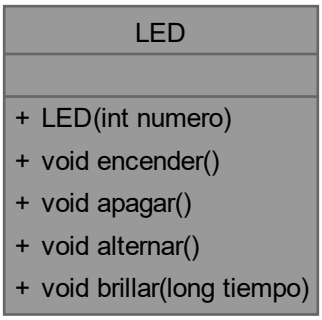
- EmisoraBLE.h

4.3. Referencia de la clase LED

Clase para manejar LEDs.

`#include <LED.h>`

Diagrama de colaboración de LED:



Métodos públicos

- LED (int numero)
  Constructor de la clase LED.
- void encender ()
  @function encender
- void apagar ()
  @function apagar
- void alternar ()
  @function alternar
- void brillar (long tiempo)
  @function brillar

4.3.1. Descripción detallada

Clase para manejar LEDs.

Definición en la línea 25 del archivo LED.h.

4.3.2. Documentación de constructores y destructores

LED()

```
LED::LED (
            int numero)  [inline]
```

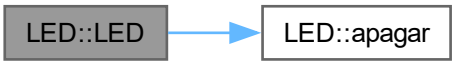
Constructor de la clase LED.

Parámetros

numero	Número del pin del LED.

Definición en la línea 43 del archivo LED.h.

Gráfico de llamadas de esta función:



4.3.3. Documentación de funciones miembro

alternar()

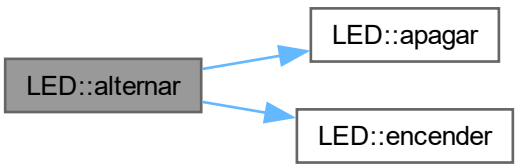
```
void LED::alternar ()  [inline]
```

@function alternar

Alterna el estado del LED.

Definición en la línea 71 del archivo LED.h.

Gráfico de llamadas de esta función:



apagar()

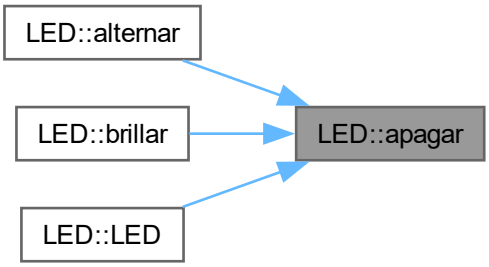
```
void LED::apagar ()  [inline]
```

@function apagar

Apaga el LED.

Definición en la línea 62 del archivo LED.h.

Gráfico de llamadas a esta función:



brillar()

```
void LED::brillar (
            long tiempo)  [inline]
```

@function brillar

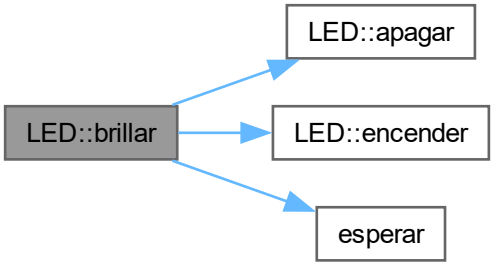
Enciende el LED durante un tiempo dado.

Parámetros

tiempo	Tiempo en milisegundos.

Definición en la línea 84 del archivo LED.h.

Gráfico de llamadas de esta función:



encender()

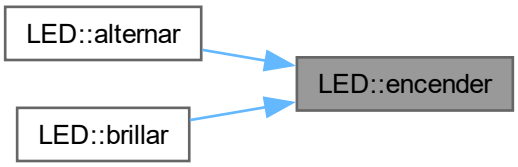
```
void LED::encender ()  [inline]
```

@function encender

Enciende el LED.

Definición en la línea 53 del archivo LED.h.

Gráfico de llamadas a esta función:



La documentación de esta clase está generada del siguiente archivo:

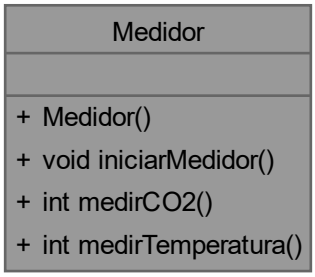
- LED.h

4.4. Referencia de la clase Medidor

Clase para medir la concentración de CO2 y la temperatura.

```
#include <Medidor.h>
```

Diagrama de colaboración de Medidor:



Métodos públicos

- Medidor ()
  Constructor de la clase Medidor.
- void iniciarMedidor ()
  @function iniciarMedidor
- int medirCO2 ()
  @function medirCO2
- int medirTemperatura ()
  @function medirTemperatura

4.4.1. Descripción detallada

Clase para medir la concentración de CO2 y la temperatura.

Definición en la línea 16 del archivo Medidor.h.

4.4.2. Documentación de constructores y destructores

Medidor()

```
Medidor::Medidor ()  [inline]
```

Constructor de la clase Medidor.

Definición en la línea 27 del archivo Medidor.h.

4.4.3. Documentación de funciones miembro

iniciarMedidor()

```
void Medidor::iniciarMedidor ()  [inline]
```

@function iniciarMedidor

Inicializa el medidor.

Definición en la línea 34 del archivo Medidor.h.

medirCO2()

```
int Medidor::medirCO2 ()  [inline]
```

@function medirCO2

Mide la concentración de CO2.

Devuelve

Concentración de CO2 en ppm.

Nota

Este método devuelve un valor fijo para pruebas.

Definición en la línea 44 del archivo Medidor.h.

medirTemperatura()

```
int Medidor::medirTemperatura ()  [inline]
```

@function medirTemperatura

Mide la temperatura.

Devuelve

Temperatura en grados Celsius.

Nota

Este método devuelve un valor fijo para pruebas.

Definición en la línea 54 del archivo Medidor.h.

La documentación de esta clase está generada del siguiente archivo:

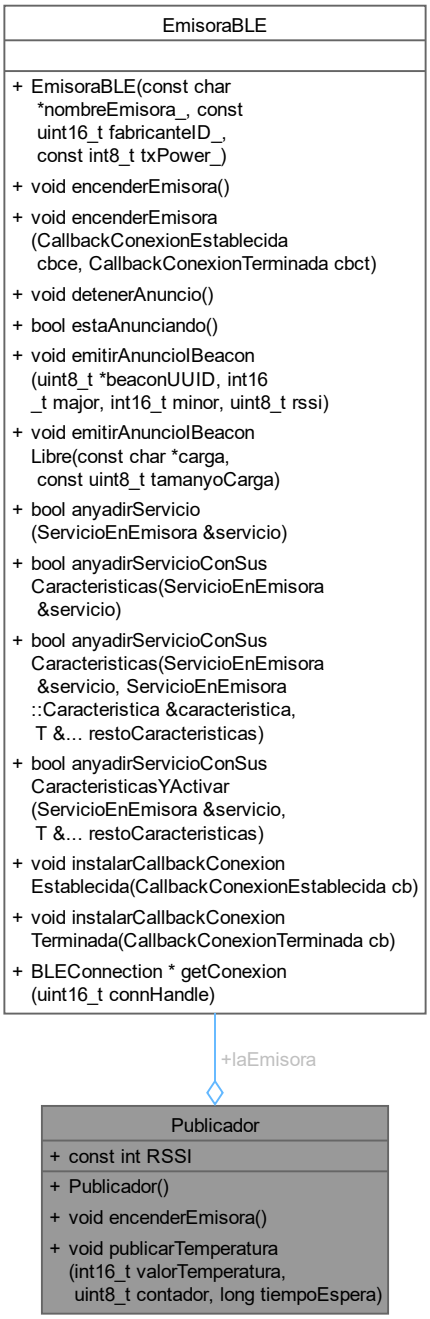
- Medidor.h

4.5. Referencia de la clase Publicador

Clase para publicar mediciones de CO2, temperatura y ruido a través de BLE.

```
#include <Publicador.h>
```

Diagrama de colaboración de Publicador:



Tipos públicos

- enum MedicionesID { CO2 = 11 , TEMPERATURA = 12 , RUIDO = 13 }

  Enumeración para identificar las mediciones.

Métodos públicos

- Publicador ()

Constructor de la clase Publicador.

- void encenderEmisora ()

  @function encenderEmisora
- void publicarTemperatura (int16_t valorTemperatura, uint8_t contador, long tiempoEspera)

  @function publicarTemperatura

Atributos públicos

- EmisoraBLE laEmisora

  Emisora BLE.
- const int RSSI = -53

  Valor RSSI (Received Signal Strength Indicator).

4.5.1. Descripción detallada

Clase para publicar mediciones de CO2, temperatura y ruido a través de BLE.

Definición en la línea 15 del archivo Publicador.h.

4.5.2. Documentación de las enumeraciones miembro de la clase

MedicionesID

```
enum Publicador::MedicionesID
```

Enumeración para identificar las mediciones.

Parámetros

CO2	Identificador de la medición de CO2.
TEMPERATURA	Identificador de la medición de temperatura.
RUIDO	Identificador de la medición de ruido.

Valores de enumeraciones

CO2	
TEMPERATURA	
RUIDO	

Definición en la línea 56 del archivo Publicador.h.

4.5.3. Documentación de constructores y destructores

Publicador()

```
Publicador::Publicador ()  [inline]
```

Constructor de la clase Publicador.

Definición en la línea 65 del archivo Publicador.h.

4.5.4. Documentación de funciones miembro

encenderEmisora()

```
void Publicador::encenderEmisora ()  [inline]
```

@function encenderEmisora

Inicializa el publicador.

Definición en la línea 74 del archivo Publicador.h.

publicarTemperatura()

```
void Publicador::publicarTemperatura (
            int16_t valorTemperatura,
            uint8_t contador,
            long tiempoEspera)  [inline]
```

@function publicarTemperatura

Publica una medición de temperatura.

Parámetros

valorTemperatura	Valor de temperatura en grados Celsius.
contador	Contador de la medición.

Definición en la línea 127 del archivo Publicador.h.

Gráfico de llamadas de esta función:

4.5.5. Documentación de datos miembro

laEmisora

```
EmisoraBLE Publicador::laEmisora
```

Valor inicial:

```
{
    "GTI-3A",
    0x004c,
    4
  }
```

Emisora BLE.

Definición en la línea 37 del archivo Publicador.h.

RSSI

```
const int Publicador::RSSI = -53
```

Valor RSSI (Received Signal Strength Indicator).

Definición en la línea 44 del archivo Publicador.h.

La documentación de esta clase está generada del siguiente archivo:

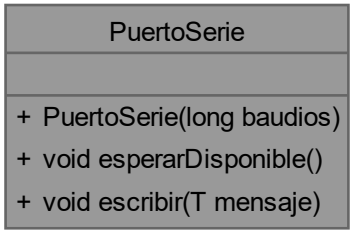
- Publicador.h

4.6. Referencia de la clase PuertoSerie

Clase para manejar un puerto serie.

`#include <PuertoSerie.h>`

Diagrama de colaboración de PuertoSerie:



Métodos públicos

- PuertoSerie (long baudios)

  Constructor de la clase PuertoSerie.
- void esperarDisponible ()

  Espera a que el puerto serie esté disponible.
- template<typename T >
  void escribir (T mensaje)

  Escribe un mensaje en el puerto serie.

4.6.1. Descripción detallada

Clase para manejar un puerto serie.

Definición en la línea 16 del archivo PuertoSerie.h.

4.6.2. Documentación de constructores y destructores

PuertoSerie()

```
PuertoSerie::PuertoSerie (
            long baudios)  [inline]
```

Constructor de la clase PuertoSerie.

Parámetros

baudios	Velocidad de transmisión en baudios.

Definición en la línea 23 del archivo PuertoSerie.h.

4.6.3. Documentación de funciones miembro

escribir()

```
template<typename T >
void PuertoSerie::escribir (
             T mensaje)  [inline]
```

Escribe un mensaje en el puerto serie.

Parámetros

mensaje	Mensaje a escribir.

Definición en la línea 44 del archivo PuertoSerie.h.

esperarDisponible()

```
void PuertoSerie::esperarDisponible ()  [inline]
```

Espera a que el puerto serie esté disponible.

Definición en la línea 31 del archivo PuertoSerie.h.

La documentación de esta clase está generada del siguiente archivo:

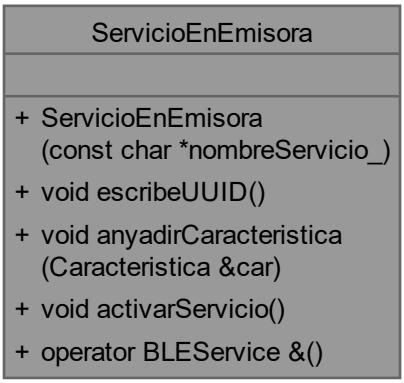
- PuertoSerie.h

4.7. Referencia de la clase ServicioEnEmisora

Clase para añadir servicios y características a una emisora BLE.

```
#include <ServicioEnEmisora.h>
```

Diagrama de colaboración de ServicioEnEmisora:



Clases

- class Caracteristica

  Clase para añadir características a un servicio BLE.

Tipos públicos

- using CallbackCaracteristicaEscrita

  @function CallbackCaracteristicaEscrita

Métodos públicos

- ServicioEnEmisora (const char ∗nombreServicio_)

  Constructor de la clase ServicioEnEmisora.
- void escribeUUID ()

  Constructor de la clase ServicioEnEmisora.
- void anyadirCaracteristica (Caracteristica &car)

  @function anyadirCaracteristica]
- void activarServicio ()

  @function activarServicio
- operator BLEService & ()

4.7.1. Descripción detallada

Clase para añadir servicios y características a una emisora BLE.

Definición en la línea 50 del archivo ServicioEnEmisora.h.

4.7.2. Documentación de los «Typedef» miembros de la clase

CallbackCaracteristicaEscrita

```
using ServicioEnEmisora::CallbackCaracteristicaEscrita
```

Valor inicial:

```
 void(uint16_t conn_handle,
                                          BLECharacteristic* chr,
                                          uint8_t* data, uint16_t len)
```

@function CallbackCaracteristicaEscrita

Definición de un tipo de función callback para manejar escrituras en una característica BLE.

Parámetros

conn_handle	Identificador de la conexión BLE.
chr	Característica BLE.
data	Datos escritos.
len	Longitud de los datos escritos.

Definición en la línea 63 del archivo ServicioEnEmisora.h.

4.7.3. Documentación de constructores y destructores

ServicioEnEmisora()

```
ServicioEnEmisora::ServicioEnEmisora (
             const char * nombreServicio_)  [inline]
```

Constructor de la clase ServicioEnEmisora.

Parámetros

*nombre↩ Servicio_*	Nombre del servicio.

Nota

Este constructor inicializa un servicio BLE con el nombre dado.

Definición en la línea 267 del archivo ServicioEnEmisora.h.

4.7.4. Documentación de funciones miembro

activarServicio()

```
void ServicioEnEmisora::activarServicio ()  [inline]
```

@function activarServicio

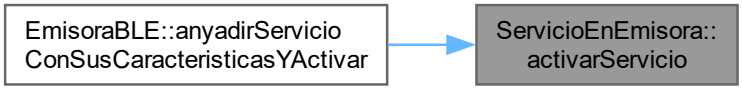
Activa el servicio.

Nota

Este método activa el servicio.

Definición en la línea 301 del archivo ServicioEnEmisora.h.

Gráfico de llamadas a esta función:



anyadirCaracteristica()

```
void ServicioEnEmisora::anyadirCaracteristica (
            Caracteristica & car)  [inline]
```

@function anyadirCaracteristica]

Añade una característica al servicio.

Parámetros

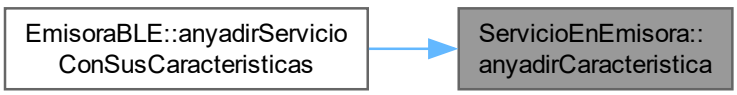
car	Característica a añadir.

Nota

Este método añade una característica al servicio.

Definición en la línea 292 del archivo ServicioEnEmisora.h.

Gráfico de llamadas a esta función:



escribeUUID()

```
void ServicioEnEmisora::escribeUUID ()  [inline]
```

Constructor de la clase ServicioEnEmisora.

Parámetros

*nombre↩ Servicio_*	Nombre del servicio.
*uuidServicio_*	UUID del servicio.

Nota

Este constructor inicializa un servicio BLE con el nombre y UUID dados.

Definición en la línea 278 del archivo ServicioEnEmisora.h.

operator BLEService &()

```
ServicioEnEmisora::operator BLEService & ()  [inline]
```

Definición en la línea 314 del archivo ServicioEnEmisora.h.

La documentación de esta clase está generada del siguiente archivo:

- ServicioEnEmisora.h

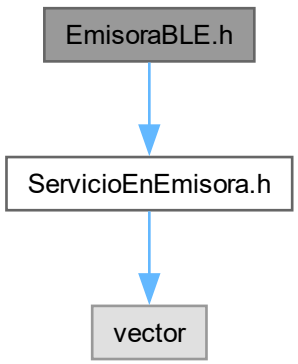
5. Documentación de archivos

5.1. Referencia del archivo EmisoraBLE.h

Controlador para emitir y gestionar señales Bluetooth Low Energy (BLE) a través de Bluefruit.

```
#include "ServicioEnEmisora.h"
```

Gráfico de dependencias incluidas en EmisoraBLE.h:



Clases

- class EmisoraBLE

  Clase para manejar una emisora Bluetooth Low Energy (BLE).

5.1.1. Descripción detallada

Controlador para emitir y gestionar señales Bluetooth Low Energy (BLE) a través de Bluefruit.

Autor

Sento Marcos Ibarra

Esta clase maneja la configuración de emisoras BLE, incluyendo el encendido, la configuración de los beacons y la gestión de servicios y características BLE.

Ver también

ServicioEnEmisora.h

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/

https://github.com/nkolban/ESP32_BLE_Arduino/blob/master/src/BLEBeacon.h

https://www.instructables.com/id/Beaconeddystone-and-Adafruit-NRF52-Advertise-Your-/

https://learn.adafruit.com/bluefruit-nrf52-feather-learning-guide/bleadvertising

Definición en el archivo EmisoraBLE.h.

5.2. EmisoraBLE.h

Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00019 #ifndef EMISORA_H_INCLUIDO
00020 #define EMISORA_H_INCLUIDO
00021
00022
00023 #include "ServicioEnEmisora.h"
00024
00025 // ---------------------------------------------------------
00026 // ---------------------------------------------------------
00027
00032 class EmisoraBLE {
00033 private:
00034
00035   const char* nombreEmisora;
00036   const uint16_t fabricanteID;
00037   const int8_t txPower;
00038
00039 public:
00040
00041
00047   using CallbackConexionEstablecida = void(uint16_t connHandle);
00054   using CallbackConexionTerminada = void(uint16_t connHandle, uint8_t reason);
00055
00065   EmisoraBLE(const char* nombreEmisora_, const uint16_t fabricanteID_,
00066              const int8_t txPower_)
00067     : nombreEmisora(nombreEmisora_),
00068       fabricanteID(fabricanteID_),
00069       txPower(txPower_) {
```

```
00070     // no encender ahora la emisora, tal vez sea por el println()
00071     // que hace que todo falle si lo llamo en el contructor
00072     // ( = antes que configuremos Serial )
00073     // No parece que sea por el println,
00074     // por tanto NO_encenderEmisora();
00075   }  // ()
00076
00077   // .....................................................
00078   // .....................................................
00079   /* creo que no me sirve esta versión porque parece
00080      que no se instalen los callbacks si la emisora no está encendida,
00081      pero no la puedo encender en el constructor
00082   EmisoraBLE( const char * nombreEmisora_, const uint16_t fabricanteID_,
00083               const int8_t txPower_,
00084               CallbackConexionEstablecida cbce,
00085               CallbackConexionTerminada cbct
00086               )
00087     :
00088     EmisoraBLE ( nombreEmisora_, fabricanteID_, txPower_ )
00089   {
00090     instalarCallbackConexionEstablecida( cbce );
00091     instalarCallbackConexionTerminada( cbct );
00092   } // ()
00093   */
00094
00100   void encenderEmisora() {
00101     // Serial.println ( "Bluefruit.begin() " );
00102     Bluefruit.begin();
00103
00104     // por si acaso:
00105     (*this).detenerAnuncio();
00106   }  // ()
00107
00114   void encenderEmisora(CallbackConexionEstablecida cbce,
00115                        CallbackConexionTerminada cbct) {
00116
00117     encenderEmisora();
00118
00119     instalarCallbackConexionEstablecida(cbce);
00120     instalarCallbackConexionTerminada(cbct);
00121
00122   }  // ()
00123
00127   void detenerAnuncio() {
00128
00129     if ((*this).estaAnunciando()) {
00130       // Serial.println ( "Bluefruit.Advertising.stop() " );
00131       Bluefruit.Advertising.stop();
00132     }
00133
00134   }  // ()
00135
00136   // .....................................................
00137   // estaAnunciando() -> Boleano
00138   // .....................................................
00144   bool estaAnunciando() {
00145     return Bluefruit.Advertising.isRunning();
00146   }  // ()
00147
00156   void emitirAnuncioIBeacon(uint8_t* beaconUUID, int16_t major, int16_t minor, uint8_t rssi) {
00157
00158     //
00159     //
00160     //
00161     (*this).detenerAnuncio();
00162
00163     //
00164     // creo el beacon
00165     //
00166     BLEBeacon elBeacon(beaconUUID, major, minor, rssi);
00167     elBeacon.setManufacturer((*this).fabricanteID);
00168
00169     //
00170     // parece que esto debe ponerse todo aquí
00171     //
00172
00173     Bluefruit.setTxPower((*this).txPower);
00174     Bluefruit.setName((*this).nombreEmisora);
00175     Bluefruit.ScanResponse.addName();  // para que envíe el nombre de emisora (?!)
00176
00177     //
00178     // pongo el beacon
00179     //
00180     Bluefruit.Advertising.setBeacon(elBeacon);
00181
00182     //
00183     // ? qué valorers poner aquí
```

```
00184     //
00185     Bluefruit.Advertising.restartOnDisconnect(true);  // no hace falta, pero lo pongo
00186     Bluefruit.Advertising.setInterval(100, 100);      // in unit of 0.625 ms
00187
00188     //
00189     // empieza el anuncio, 0 = tiempo indefinido (ya lo pararán)
00190     //
00191     Bluefruit.Advertising.start(0);
00192
00193   }  // ()
00194
00195   // .....................................................
00196   //
00197   // Ejemplo de Beacon (31 bytes)
00198   //
00199   // https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/
00200   //
00201   // The iBeacon Prefix contains the hex data : 0x0201061AFF004C0215. This breaks down as follows:
00202   //
00203   // 0x020106 defines the advertising packet as BLE General Discoverable and BR/EDR high-speed
      incompatible.
00204   // Effectively it says this is only broadcasting, not connecting.
00205   //
00206   // 0x1AFF says the following data is 26 bytes long and is Manufacturer Specific Data.
00207   //
00208   // 0x004C is Apple's Bluetooth Sig ID and is the part of this spec that makes it Apple-dependent.
00209   //
00210   // 0x02 is a secondary ID that denotes a proximity beacon, which is used by all iBeacons.
00211   //
00212   // 0x15 defines the remaining length to be 21 bytes (16+2+2+1).
00213   //
00214   // Por ejemmplo:
00215   //
00216   // 1. prefijo: 9bytes
00217   //       0x02, 0x01, 0x06,       // advFlags 3bytes
00218   //       0x1a, 0xff,             // advHeader 2 (0x1a = 26 = 25(lenght de 0x4c a 0xca)+1)   0xFF ->
      BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA
00219   //       0x4c, 0x00,             // companyID 2bytes
00220   //       0x02,                   // ibeacon type 1 byte
00221   //       0x15,                   // ibeacon length 1 byte (dec=21 lo que va a continuación: desde
      la 'f' hasta 0x01)
00222   //
00223   // 2. uuid: 16bytes
00224   // 'f', 'i', 's', 't', 'r', 'o', 'f', 'i', 's', 't', 'r', 'o', 0xa7, 0x10, 0x96, 0xe0
00225   //
00226   // 2 major: 2bytes
00227   // 0x04, 0xd2,
00228   //
00229   // minor: 2bytes
00230   // 0x10, 0xe1,
00231   //
00232   // 0xca, // tx power : 1bytes
00233   //
00234   // 0x01, // este es el byte 31 = BLE_GAP_ADV_SET_DATA_SIZE_MAX, parece que sobra
00235   //
00236   // .....................................................
00237   // Para enviar como carga libre los últimos 21 bytes de un iBeacon (lo que normalmente sería uuid-16
      major-2 minor-2 txPower-1)
00238   // .....................................................
00239   /*
00240   void emitirAnuncioIBeaconLibre( const char * carga ) {
00241
00242     const uint8_t tamanyoCarga = strlen( carga );
00243   */
00244
00251   void emitirAnuncioIBeaconLibre(const char* carga, const uint8_t tamanyoCarga) {
00252
00253     (*this).detenerAnuncio();
00254
00255     Bluefruit.Advertising.clearData();
00256     Bluefruit.ScanResponse.clearData();  // hace falta?
00257
00258     // Bluefruit.setTxPower( (*this).txPower ); creo que no lo pongo porque es uno de los bytes de la
      parte de carga que utilizo
00259     Bluefruit.setName((*this).nombreEmisora);
00260     Bluefruit.ScanResponse.addName();
00261
00262     Bluefruit.Advertising.addFlags(BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE);
00263
00264     // con este parece que no va  !
00265     // Bluefruit.Advertising.addFlags(BLE_GAP_ADV_FLAG_LE_GENERAL_DISC_MODE);
00266
00267     //
00268     // hasta ahora habrá, supongo, ya puestos los 5 primeros bytes. Efectivamente.
00269     // Falta poner 4 bytes fijos (company ID, beacon type, longitud) y 21 de carga
00270     //
00271     uint8_t restoPrefijoYCarga[4 + 21] = {
```

```
00272       0x4c, 0x00,  // companyID 2
00273       0x02,        // ibeacon type 1byte
00274       21,          // ibeacon length 1byte (dec=21)  longitud del resto // 0x15 // ibeacon length
      1byte (dec=21)  longitud del resto
00275       '-', '-', '-', '-',
00276       '-', '-', '-', '-',
00277       '-', '-', '-', '-',
00278       '-', '-', '-', '-',
00279       '-', '-', '-', '-',
00280       '-'
00281     };
00282
00283     //
00284     // addData() hay que usarlo sólo una vez. Por eso copio la carga
00285     // en el anterior array, donde he dejado 21 sitios libres
00286     //
00287     memcpy(&restoPrefijoYCarga[4], &carga[0], (tamanyoCarga > 21 ? 21 : tamanyoCarga));
00288
00289     //
00290     // copio la carga para emitir
00291     //
00292     Bluefruit.Advertising.addData(BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA,
00293                                   &restoPrefijoYCarga[0],
00294                                   4 + 21);
00295
00296     //
00297     // ? qué valores poner aquí ?
00298     //
00299     Bluefruit.Advertising.restartOnDisconnect(true);
00300     Bluefruit.Advertising.setInterval(100, 100);  // in unit of 0.625 ms
00301
00302     Bluefruit.Advertising.setFastTimeout(1);  // number of seconds in fast mode
00303     //
00304     // empieza el anuncio, 0 = tiempo indefinido (ya lo pararán)
00305     //
00306     Bluefruit.Advertising.start(0);
00307
00308     Globales::elPuerto.escribir("emitiriBeacon libre  Bluefruit.Advertising.start( 0 );  \n");
00309   }  // ()
00310
00317   bool anyadirServicio(ServicioEnEmisora& servicio) {
00318
00319     Globales::elPuerto.escribir(" Bluefruit.Advertising.addService( servicio ); \n");
00320
00321     bool r = Bluefruit.Advertising.addService(servicio);
00322
00323     if (!r) {
00324       Serial.println(" SERVICION NO AÑADIDO \n");
00325     }
00326
00327
00328     return r;
00329     // nota: uso conversión de tipo de servicio (ServicioEnEmisora) a BLEService
00330     // para addService()
00331   }  // ()
00332
00333
00334
00341   bool anyadirServicioConSusCaracteristicas(ServicioEnEmisora& servicio) {
00342     return (*this).anyadirServicio(servicio);
00343   }  //
00344
00356   template<typename... T>
00357   bool anyadirServicioConSusCaracteristicas(ServicioEnEmisora& servicio,
00358                                             ServicioEnEmisora::Caracteristica& caracteristica,
00359                                             T&... restoCaracteristicas) {
00360
00361     servicio.anyadirCaracteristica(caracteristica);
00362
00363     return anyadirServicioConSusCaracteristicas(servicio, restoCaracteristicas...);
00364
00365   }  // ()
00366
00375   template<typename... T>
00376   bool anyadirServicioConSusCaracteristicasYActivar(ServicioEnEmisora& servicio,
00377                                                     // ServicioEnEmisora::Caracteristica &
      caracteristica,
00378                                                     T&... restoCaracteristicas) {
00379
00380     bool r = anyadirServicioConSusCaracteristicas(servicio, restoCaracteristicas...);
00381
00382     servicio.activarServicio();
00383
00384     return r;
00385
00386   }  // ()
00387
```

```
00388
00394   void instalarCallbackConexionEstablecida(CallbackConexionEstablecida cb) {
00395     Bluefruit.Periph.setConnectCallback(cb);
00396   }  // ()
00397
00403   void instalarCallbackConexionTerminada(CallbackConexionTerminada cb) {
00404     Bluefruit.Periph.setDisconnectCallback(cb);
00405   }  // ()
00406
00413   BLEConnection* getConexion(uint16_t connHandle) {
00414     return Bluefruit.Connection(connHandle);
00415   }  // ()
00416
00417 };  // class
00418
00419 #endif
00420
00421 // --------------------------------------------------------
00422 // --------------------------------------------------------
00423 // --------------------------------------------------------
00424 // --------------------------------------------------------
```

5.3. Referencia del archivo LED.h

Controlador para manejar LEDs.

Clases

- class LED

  Clase para manejar LEDs.

Funciones

- void esperar (long tiempo)

  @function esperar

5.3.1. Descripción detallada

Controlador para manejar LEDs.

Marcos Ibarra

Definición en el archivo LED.h.

5.3.2. Documentación de funciones

esperar()

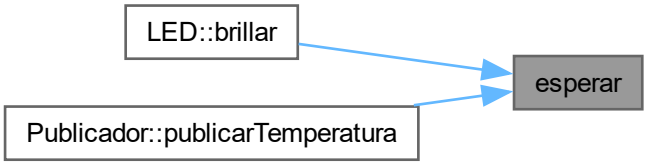
```
void esperar (
             long tiempo)
```

@function esperar

Espera un tiempo dado en milisegundos.

Definición en la línea 17 del archivo LED.h.

Gráfico de llamadas a esta función:



5.4. LED.h

Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00009 #ifndef LED_H_INCLUIDO
00010 #define LED_H_INCLUIDO
00011
00017 void esperar(long tiempo) {
00018   delay(tiempo);
00019 }
00020
00025 class LED {
00034 private:
00035   int numeroLED;
00036   bool encendido;
00037 public:
00038
00043   LED(int numero)
00044     : numeroLED(numero), encendido(false) {
00045     pinMode(numeroLED, OUTPUT);
00046     apagar();
00047   }
00048
00053   void encender() {
00054     digitalWrite(numeroLED, HIGH);
00055     encendido = true;
00056   }
00057
00062   void apagar() {
00063     digitalWrite(numeroLED, LOW);
00064     encendido = false;
00065   }
00066
00071   void alternar() {
00072     if (encendido) {
00073       apagar();
00074     } else {
00075       encender();
00076     }
00077   }  // ()
00078
00084   void brillar(long tiempo) {
00085     encender();
00086     esperar(tiempo);
00087     apagar();
00088   }
00089 };  // class
00090
00091 // --------------------------------------------------------
00092 // --------------------------------------------------------
00093 // --------------------------------------------------------
00094 // --------------------------------------------------------
00095 #endif
```

5.5. Referencia del archivo Medidor.h

Controlador para medir la concentración de CO2 y la temperatura.

Clases

- class Medidor

  Clase para medir la concentración de CO2 y la temperatura.

5.5.1. Descripción detallada

Controlador para medir la concentración de CO2 y la temperatura.

Autor

Sento Marcos Ibarra

Definición en el archivo Medidor.h.

5.6. Medidor.h

Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00009 #ifndef MEDIDOR_H_INCLUIDO
00010 #define MEDIDOR_H_INCLUIDO
00011
00016 class Medidor {
00017
00018   // ....................................................
00019   // ....................................................
00020 private:
00021
00022 public:
00023
00027   Medidor() {
00028   }  // ()
00029
00034   void iniciarMedidor() {
00035     // las cosas que no se puedan hacer en el constructor, if any
00036   }  // ()
00037
00044   int medirCO2() {
00045     return 235;
00046   }  // ()
00047
00054   int medirTemperatura() {
00055     return -12;  // qué frío !
00056   }              // ()
00057
00058 };  // class
00059
00060 // ----------------------------------------------------------
00061 // ----------------------------------------------------------
00062 // ----------------------------------------------------------
00063 // ----------------------------------------------------------
00064 #endif
```

5.7. Referencia del archivo Publicador.h

Controlador para publicar mediciones de CO2, temperatura y ruido a través de BLE.

Clases

- class Publicador

  Clase para publicar mediciones de CO2, temperatura y ruido a través de BLE.

5.7.1. Descripción detallada

Controlador para publicar mediciones de CO2, temperatura y ruido a través de BLE.

Autor

Sento Marcos Ibarra

Definición en el archivo Publicador.h.

5.8. Publicador.h

Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00009 #ifndef PUBLICADOR_H_INCLUIDO
00010 #define PUBLICADOR_H_INCLUIDO
00011
00015 class Publicador {
00016
00022 private:
00023
00024   uint8_t beaconUUID[16] = {
00025     'E', 'P', 'S', 'G', '-', 'G', 'T', 'I',
00026     '-', 'P', 'R', 'O', 'Y', '-', '3', 'D'
00027   };
00028
00029   // ------------------------------------------------------------
00030   // ------------------------------------------------------------
00031 public:
00032
00037   EmisoraBLE laEmisora{
00038     "GTI-3A",  //  nombre emisora
00039     0x004c,    // fabricanteID (Apple)
00040     4          // txPower
00041   };
00042
00043
00044   const int RSSI = -53;
00045
00046   // ------------------------------------------------------------
00047   // ------------------------------------------------------------
00048 public:
00049
00056   enum MedicionesID {
00057     CO2 = 11,
00058     TEMPERATURA = 12,
00059     RUIDO = 13
00060   };
00061
00065   Publicador() {
00066     // ATENCION: no hacerlo aquí. (*this).laEmisora.encenderEmisora();
00067     // Pondremos un método para llamarlo desde el setup() más tarde
00068   }  // ()
00069
00074   void encenderEmisora() {
00075     (*this).laEmisora.encenderEmisora();
00076   }  // ()
00077
00084   void publicarCO2(int16_t valorCO2, uint8_t contador,
00085                    long tiempoEspera) {
00086
00092     uint16_t major = (MedicionesID::CO2 << 8) + contador;
00093     (*this).laEmisora.emitirAnuncioIBeacon((*this).beaconUUID,
00094                                            major,
00095                                            valorCO2,     // minor
00096                                            (*this).RSSI  // rssi
00097     );
00098
00099     /*
00100     Globales::elPuerto.escribir( "   publicarCO2(): valor=" );
00101     Globales::elPuerto.escribir( valorCO2 );
00102     Globales::elPuerto.escribir( "   contador=" );
00103     Globales::elPuerto.escribir( contador );
00104     Globales::elPuerto.escribir( "   todo="  );
00105     Globales::elPuerto.escribir( major );
```

```
00106     Globales::elPuerto.escribir( "\n" );
00107     */
00108
00109     //
00110     // 2. esperamos el tiempo que nos digan
00111     //
00112     esperar(tiempoEspera);
00113
00114     //
00115     // 3. paramos anuncio
00116     //
00117
00118     (*this).laEmisora.detenerAnuncio();
00119   }  // ()
00120
00127   void publicarTemperatura(int16_t valorTemperatura,
00128                            uint8_t contador, long tiempoEspera) {
00129
00130     uint16_t major = (MedicionesID::TEMPERATURA « 8) + contador;
00131     (*this).laEmisora.emitirAnuncioIBeacon((*this).beaconUUID,
00132                                            major,
00133                                            valorTemperatura,  // minor
00134                                            (*this).RSSI       // rssi
00135     );
00136     esperar(tiempoEspera);
00137
00138     (*this).laEmisora.detenerAnuncio();
00139   }  // ()
00140
00141 };  // class
00142
00143 // -----------------------------------------------------------------
00144 // -----------------------------------------------------------------
00145 // -----------------------------------------------------------------
00146 // -----------------------------------------------------------------
00147 #endif
```

5.9. Referencia del archivo PuertoSerie.h

Controlador para manejar un puerto serie.

Clases

- class PuertoSerie

  Clase para manejar un puerto serie.

5.9.1. Descripción detallada

Controlador para manejar un puerto serie.

Autor

Sento Marcos Ibarra

Definición en el archivo PuertoSerie.h.

5.10. PuertoSerie.h

Ir a la documentación de este archivo.

```
00001
00002 // -*- mode: c++ -*-
00003
00009 #ifndef PUERTO_SERIE_H_INCLUIDO
00010 #define PUERTO_SERIE_H_INCLUIDO
00011
00016 class PuertoSerie {
00017
00018 public:
00023   PuertoSerie(long baudios) {
00024     Serial.begin(baudios);
00025     // mejor no poner esto aquí: while ( !Serial ) delay(10);
00026   }  // ()
00027
00031   void esperarDisponible() {
00032
00033     while (!Serial) {
00034       delay(10);
00035     }
00036
00037   }  // ()
00038
00043   template<typename T>
00044   void escribir(T mensaje) {
00045     Serial.print(mensaje);
00046   }  // ()
00047
00048 };  // class PuertoSerie
00049
00050 // --------------------------------------------------------------
00051 // --------------------------------------------------------------
00052 // --------------------------------------------------------------
00053 // --------------------------------------------------------------
00054 #endif
```

5.11. Referencia del archivo README.md

5.12. Referencia del archivo ServicioEnEmisora.h

Controlador para añadir servicios y características a una emisora BLE.

`#include <vector>`

Gráfico de dependencias incluidas en ServicioEnEmisora.h:

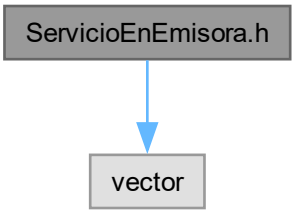
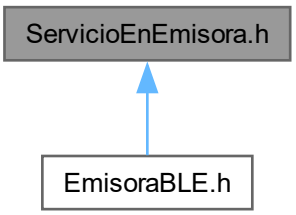


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class ServicioEnEmisora
  Clase para añadir servicios y características a una emisora BLE.
- class ServicioEnEmisora::Caracteristica
  Clase para añadir características a un servicio BLE.

Funciones

- template<typename T >
  T * alReves (T *p, int n)
- uint8_t * stringAUint8AlReves (const char *pString, uint8_t *pUint, int tamMax)

5.12.1. Descripción detallada

Controlador para añadir servicios y características a una emisora BLE.

Autor

Sento Marcos Ibarra

Definición en el archivo ServicioEnEmisora.h.

5.12.2. Documentación de funciones

alReves()

```
template<typename T >
T * alReves (
            T * p,
            int n)
```

Definición en la línea 21 del archivo ServicioEnEmisora.h.

stringAUint8AlReves()

```
uint8_t * stringAUint8AlReves (
              const char * pString,
              uint8_t * pUint,
              int tamMax)
```

Definición en la línea 34 del archivo ServicioEnEmisora.h.

5.13. ServicioEnEmisora.h

Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00009 #ifndef SERVICIO_EMISORA_H_INCLUIDO
00010 #define SERVICIO_EMISORA_H_INCLUIDO
00011
00012 // ----------------------------------------------------
00013 // ----------------------------------------------------
00014 #include <vector>
00015
00016 // ----------------------------------------------------
00017 // alReves() utilidad
00018 // pone al revés el contenido de una array en el mismo array
00019 // ----------------------------------------------------
00020 template< typename T >
00021 T* alReves(T* p, int n) {
00022   T aux;
00023
00024   for (int i = 0; i < n / 2; i++) {
00025     aux = p[i];
00026     p[i] = p[n - i - 1];
00027     p[n - i - 1] = aux;
00028   }
00029   return p;
00030 }  // ()
00031
00032 // ----------------------------------------------------
00033 // ----------------------------------------------------
00034 uint8_t* stringAUint8AlReves(const char* pString, uint8_t* pUint, int tamMax) {
00035
00036   int longitudString = strlen(pString);
00037   int longitudCopiar = (longitudString > tamMax ? tamMax : longitudString);
00038   // copio nombreServicio -> uuidServicio pero al revés
00039   for (int i = 0; i <= longitudCopiar - 1; i++) {
00040     pUint[tamMax - i - 1] = pString[i];
00041   }  // for
00042
00043   return pUint;
00044 }  // ()
00045
00050 class ServicioEnEmisora {
00051
00052 public:
00053
00054
00063   using CallbackCaracteristicaEscrita = void(uint16_t conn_handle,
00064                                              BLECharacteristic* chr,
00065                                              uint8_t* data, uint16_t len);
00066
00071   class Caracteristica {
00079   private:
00080     uint8_t uuidCaracteristica[16] = {  // el uuid se copia aquí (al revés) a partir de un string-c
00081       // least signficant byte, el primero
00082       '0', '1', '2', '3',
00083       '4', '5', '6', '7',
00084       '8', '9', 'A', 'B',
00085       'C', 'D', 'E', 'F'
00086     };
00087
00088     //
00089     //
00090     //
00091     BLECharacteristic laCaracteristica;
00092
00093   public:
00094
00104     Caracteristica(const char* nombreCaracteristica_)
```

```
00105       : laCaracteristica(stringAUint8AlReves(nombreCaracteristica_, &uuidCaracteristica[0], 16)) {
00106
00107     }  // ()
00108
00118     Caracteristica(const char* nombreCaracteristica_,
00119                    uint8_t props,
00120                    SecureMode_t permisoRead,
00121                    SecureMode_t permisoWrite,
00122                    uint8_t tam)
00123       : Caracteristica(nombreCaracteristica_)  // llamada al otro constructor
00124     {
00125       (*this).asignarPropiedadesPermisosYTamanyoDatos(props, permisoRead, permisoWrite, tam);
00126     }  // ()
00127
00128   private:
00129     // .....................................................
00130     // CHR_PROPS_WRITE , CHR_PROPS_READ ,  CHR_PROPS_NOTIFY
00131     // .....................................................
00132
00138     void asignarPropiedades(uint8_t props) {
00139       // no puedo escribir AUN si el constructor llama a esto: Serial.println( "
      laCaracteristica.setProperties( props ); ");
00140       (*this).laCaracteristica.setProperties(props);
00141     }  // ()
00142
00143     // .....................................................
00144     // BleSecurityMode::SECMODE_OPEN  , BleSecurityMode::SECMODE_NO_ACCESS
00145     // .....................................................
00146
00147     void asignarPermisos(SecureMode_t permisoRead, SecureMode_t permisoWrite) {
00148       // no puedo escribir AUN si el constructor llama a esto: Serial.println(
      "laCaracteristica.setPermission( permisoRead, permisoWrite ); " );
00149       (*this).laCaracteristica.setPermission(permisoRead, permisoWrite);
00150     }  // ()
00151
00152     // .....................................................
00153     // .....................................................
00154     void asignarTamanyoDatos(uint8_t tam) {
00155       // no puedo escribir AUN si el constructor llama a esto: Serial.print( "
      (*this).laCaracteristica.setFixedLen( tam = " );
00156       // no puedo escribir AUN si el constructor llama a esto: Serial.println( tam );
00157       // (*this).laCaracteristica.setFixedLen( tam );
00158       (*this).laCaracteristica.setMaxLen(tam);
00159     }  // ()
00160
00161   public:
00162
00172     void asignarPropiedadesPermisosYTamanyoDatos(uint8_t props,
00173                                                  SecureMode_t permisoRead,
00174                                                  SecureMode_t permisoWrite,
00175                                                  uint8_t tam) {
00176       asignarPropiedades(props);
00177       asignarPermisos(permisoRead, permisoWrite);
00178       asignarTamanyoDatos(tam);
00179     }  // ()
00180
00181
00187     uint16_t escribirDatos(const char* str) {
00188       // Serial.print( " return (*this).laCaracteristica.write( str  = " );
00189       // Serial.println( str );
00190       uint16_t r = (*this).laCaracteristica.write(str);
00191
00192       // Serial.print( ">>Escritos " ); Serial.print( r ); Serial.println( " bytes con write() " );
00193
00194       return r;
00195     }  // ()
00196
00203     uint16_t notificarDatos(const char* str) {
00204
00205       uint16_t r = laCaracteristica.notify(&str[0]);
00206
00207       return r;
00208     }  //  ()
00209
00215     void instalarCallbackCaracteristicaEscrita(CallbackCaracteristicaEscrita cb) {
00216       (*this).laCaracteristica.setWriteCallback(cb);
00217     }  // ()
00218
00223     void activar() {
00224       err_t error = (*this).laCaracteristica.begin();
00225       Globales::elPuerto.escribir(" (*this).laCaracteristica.begin(); error = ");
00226       Globales::elPuerto.escribir(error);
00227     }  // ()
00228
00229   };  // class Caracteristica
00230
00240 private:
```

```
00241
00242   uint8_t uuidServicio[16] = {  // el uuid se copia aquí (al revés) a partir de un string-c
00243     // least signficant byte, el primero
00244     '0', '1', '2', '3',
00245     '4', '5', '6', '7',
00246     '8', '9', 'A', 'B',
00247     'C', 'D', 'E', 'F'
00248   };
00249
00250   //
00251   //
00252   //
00253   BLEService elServicio;
00254
00255   //
00256   //
00257   //
00258   std::vector< Caracteristica* > lasCaracteristicas;
00259
00260 public:
00261
00267   ServicioEnEmisora(const char* nombreServicio_)
00268     : elServicio(stringAUint8AlReves(nombreServicio_, &uuidServicio[0], 16)) {
00269
00270   }  // ()
00271
00278   void escribeUUID() {
00279     Serial.println("**********");
00280     for (int i = 0; i <= 15; i++) {
00281       Serial.print((char)uuidServicio[i]);
00282     }
00283     Serial.println("\n**********");
00284   }  // ()
00285
00292   void anyadirCaracteristica(Caracteristica& car) {
00293     (*this).lasCaracteristicas.push_back(&car);
00294   }  // ()
00295
00301   void activarServicio() {
00302     // entiendo que al llegar aquí ya ha sido configurado
00303     // todo: características y servicio
00304
00305     err_t error = (*this).elServicio.begin();
00306     Serial.print(" (*this).elServicio.begin(); error = ");
00307     Serial.println(error);
00308
00309     for (auto pCar : (*this).lasCaracteristicas) {
00310       (*pCar).activar();
00311     }  // for
00312
00313   }  // ()
00314   operator BLEService&() {
00315     // "conversión de tipo": si pongo esta clase en un sitio donde necesitan un BLEService
00316     return elServicio;
00317   }  // ()
00318
00319 };  // class
00320
00321 #endif
00322
00323 // ---------------------------------------------------------
00324 // ---------------------------------------------------------
00325 // ---------------------------------------------------------
00326 // ---------------------------------------------------------
```

6. Ejemplos

6.1. 13

6.2. false

6.3. EPSG-GTI-PROY-3D

6.4. 0x0B01

6.5. 6E400002-B5A3-F393-E0A9-E50E24DCCA9E

6.6. 6E400001-B5A3-F393-E0A9-E50E24DCCA9E

Índice alfabético