

## Proyecto3A\_Arduino

Generado por Doxygen 1.12.0



# Capítulo 1

## Proyecto3A\_Arduino

Este proyecto implementa una emisora Bluetooth Low Energy (BLE) que permite la publicación de datos de sensores, como CO2 y temperatura, utilizando la arquitectura de clases en C++. Utiliza un servicio BLE para transmitir datos a dispositivos conectados.

### 1.1. Tabla de Contenidos

- Descripción
- Características
- Instalación
- Uso
- Estructura de Archivos
- Contribuciones
- Licencia

### 1.2. Descripción

La emisora BLE está diseñada para transmitir datos de medición a través de un servicio BLE. Este programa utiliza clases para encapsular la funcionalidad de la emisora, las características y el servicio BLE, facilitando su reutilización y mantenimiento.

### 1.3. Características

- Publicación de datos de mediciones (CO2, temperatura, ruido).
- Interfaz de comunicación por puerto serie.
- Soporte para múltiples características en un servicio BLE.
- Posibilidad de añadir características dinámicamente.

## 1.4. Instalación

### 1. Requisitos:

- Arduino IDE instalado en tu máquina.
- Biblioteca BLE para Arduino (puede ser instalada a través del gestor de bibliotecas del IDE).

### 2. Clonar el repositorio:

```
git clone https://github.com/tu_usuario/tu_repositorio.git
```

### 3. Importar el proyecto:

- Abre el Arduino IDE y selecciona Archivo > Abrir... para importar los archivos del proyecto.

### 4. Cargar el programa:

- Conecta tu dispositivo Arduino y selecciona la placa adecuada en Herramientas > Placa.
- Carga el programa en el dispositivo.

## 1.5. Uso

1. **Inicializar la Emisora:** Llama al método `encenderEmisora()` en el `setup()` de tu programa para activar la emisora BLE.

### 2. Publicar Mediciones:

- Utiliza `publicarCO2(valorCO2, contador, tiempoEspera)` para publicar el nivel de CO2.
- Utiliza `publicarTemperatura(valorTemperatura, contador, tiempoEspera)` para publicar la temperatura.

3. **Verificar Datos:** Conecta un dispositivo BLE y utiliza una aplicación compatible para verificar los datos publicados.

## 1.6. Estructura de Archivos

```
/proyecto-emisora-ble
Publicador.h           # Clase que gestiona la emisora BLE y la publicación de datos.
PuertoSerie.h         # Clase para gestionar la comunicación por puerto serie.
ServicioEmisora.h     # Clase que define el servicio BLE y sus características.
README.md              # Este archivo.
```

## 1.7. Contribuciones

Las contribuciones son bienvenidas. Si deseas contribuir, por favor sigue estos pasos:

1. Haz un fork del repositorio.
2. Crea una nueva rama (`git checkout -b feature/nueva-funcionalidad`).
3. Realiza tus cambios y haz commit (`git commit -m 'Añadir nueva funcionalidad'`).
4. Haz push a la rama (`git push origin feature/nueva-funcionalidad`).
5. Crea un nuevo Pull Request.

## 1.8. Licencia

Este proyecto está licenciado bajo la MIT License - consulta el archivo LICENSE para más detalles.

## Capítulo 2

# Índice de espacios de nombres

### 2.1. Lista de espacios de nombres

Lista de los espacios de nombres documentados, con breves descripciones:

#### Globales

Contiene objetos globales para manejar el [LED](#), la comunicación por puerto serie y otros módulos . . . . . ??

#### Loop

Espacio de nombres para variables del bucle principal . . . . . ??



## Capítulo 3

# Índice de clases

### 3.1. Lista de clases

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

<a href="#">ServicioEnEmisora::Caracteristica</a>	Clase que representa una característica dentro de un servicio BLE . . . . .	??
<a href="#">EmisoraBLE</a>	Clase para manejar la emisora Bluetooth Low Energy (BLE) . . . . .	??
<a href="#">LED</a>	Clase para controlar un <a href="#">LED</a> conectado a un pin específico . . . . .	??
<a href="#">Medidor</a>	Clase para simular la medición de CO2 y temperatura . . . . .	??
<a href="#">Publicador</a>	Clase para publicar mediciones de CO2 y temperatura mediante BLE . . . . .	??
<a href="#">PuertoSerie</a>	Clase para manejar la comunicación por puerto serie . . . . .	??
<a href="#">ServicioEnEmisora</a>	Clase que gestiona un servicio BLE con características . . . . .	??





## Capítulo 4

# Índice de archivos

### 4.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

HolaMundoIBeacon/ <a href="#">EmisoraBLE.h</a>	
Clase para manejar la emisora BLE y la emisión de beacons . . . . .	??
HolaMundoIBeacon/ <a href="#">HolaMundoIBeacon.ino</a>	
Implementación principal del programa que utiliza BLE para publicar datos de sensores . . . .	??
HolaMundoIBeacon/ <a href="#">LED.h</a>	
Definición de la clase <a href="#">LED</a> para controlar un <a href="#">LED</a> en un pin específico . . . . .	??
HolaMundoIBeacon/ <a href="#">Medidor.h</a>	
Definición de la clase <a href="#">Medidor</a> para medir CO2 y temperatura . . . . .	??
HolaMundoIBeacon/ <a href="#">Publicador.h</a>	
Definición de la clase <a href="#">Publicador</a> para emitir anuncios BLE de CO2 y temperatura . . . . .	??
HolaMundoIBeacon/ <a href="#">PuertoSerie.h</a>	
Definición de la clase <a href="#">PuertoSerie</a> para la comunicación por puerto serie . . . . .	??
HolaMundoIBeacon/ <a href="#">ServicioEnEmisora.h</a>	
Definición de la clase <a href="#">ServicioEnEmisora</a> y Clase que representa una característica dentro de un servicio BLE que gestionan un servicio BLE con características . . . . .	??



## Capítulo 5

# Documentación de espacios de nombres

### 5.1. Referencia del espacio de nombres Globales

Contiene objetos globales para manejar el [LED](#), la comunicación por puerto serie y otros módulos.

#### Variables

- [LED](#) `elLED` (7)  
*Objeto para controlar el [LED](#) conectado al pin 7.*
- [PuertoSerie](#) `elPuerto` (115200)  
*Objeto para gestionar la comunicación por puerto serie a 115200 baudios.*
- [Publicador](#) `elPublicador`  
*Objeto para manejar la publicación de datos vía BLE.*
- [Medidor](#) `elMedidor`  
*Objeto para gestionar las mediciones de CO2 y temperatura.*

#### 5.1.1. Descripción detallada

Contiene objetos globales para manejar el [LED](#), la comunicación por puerto serie y otros módulos.

#### 5.1.2. Documentación de variables

##### 5.1.2.1. `elLED`

```
LED Globales::elLED( 7) (
    7 )
```

Objeto para controlar el [LED](#) conectado al pin 7.

##### 5.1.2.2. `elMedidor`

```
Medidor Globales::elMedidor
```

Objeto para gestionar las mediciones de CO2 y temperatura.

### 5.1.2.3. `elPublicador`

```
Publicador Globales::elPublicador
```

Objeto para manejar la publicación de datos vía BLE.

### 5.1.2.4. `elPuerto`

```
PuertoSerie Globales::elPuerto( 115200) (  
    115200 )
```

Objeto para gestionar la comunicación por puerto serie a 115200 baudios.

## 5.2. Referencia del espacio de nombres Loop

Espacio de nombres para variables del bucle principal.

### Variables

- `uint8_t cont` = 0

### 5.2.1. Descripción detallada

Espacio de nombres para variables del bucle principal.

### 5.2.2. Documentación de variables

#### 5.2.2.1. `cont`

```
uint8_t Loop::cont = 0
```

Contador de iteraciones del loop.

## Capítulo 6

# Documentación de clases

### 6.1. Referencia de la clase ServicioEnEmisora::Caracteristica

Clase que representa una característica dentro de un servicio BLE.

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

Diagrama de colaboración de ServicioEnEmisora::Caracteristica:

ServicioEnEmisora:: Caracteristica
- uuidCaracteristica
- laCaracteristica
+ Caracteristica()
+ Caracteristica()
+ asignarPropiedadesPermisos YTamanyoDatos()
+ escribirDatos()
+ notificarDatos()
+ instalarCallbackCaracteristica Escrita()
+ activar()
- asignarPropiedades()
- asignarPermisos()
- asignarTamanyoDatos()

## Métodos públicos

- [Caracteristica](#) (const char \*nombreCaracteristica\_)  
*Constructor que inicializa la característica con su UUID.*
- [Caracteristica](#) (const char \*nombreCaracteristica\_, uint8\_t props, SecureMode\_t permisoRead, SecureMode\_t permisoWrite, uint8\_t tam)  
*Constructor que permite definir propiedades y permisos de la característica.*
- void [asignarPropiedadesPermisosYTamanyoDatos](#) (uint8\_t props, SecureMode\_t permisoRead, SecureMode\_t permisoWrite, uint8\_t tam)  
*Asigna propiedades, permisos y tamaño de datos de la característica.*
- uint16\_t [escribirDatos](#) (const char \*str)  
*Escribe datos en la característica.*
- uint16\_t [notificarDatos](#) (const char \*str)  
*Notifica datos a los suscriptores de la característica.*
- void [instalarCallbackCaracteristicaEscrita](#) ([CallbackCaracteristicaEscrita](#) cb)  
*Instala un callback para cuando se escriben datos en la característica.*
- void [activar](#) ()  
*Activa la característica para que esté lista para operar.*

## Métodos privados

- void [asignarPropiedades](#) (uint8\_t props)
- void [asignarPermisos](#) (SecureMode\_t permisoRead, SecureMode\_t permisoWrite)
- void [asignarTamanyoDatos](#) (uint8\_t tam)

## Atributos privados

- uint8\_t [uuidCaracteristica](#) [16]
- BLECharacteristic [laCaracteristica](#)

### 6.1.1. Descripción detallada

Clase que representa una característica dentro de un servicio BLE.

### 6.1.2. Documentación de constructores y destructores

#### 6.1.2.1. [Caracteristica\(\)](#) [1/2]

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

Constructor que inicializa la característica con su UUID.

#### Parámetros

<a href="#">nombreCaracteristica_</a>	Nombre que será convertido a UUID.
---------------------------------------	------------------------------------

### 6.1.2.2. Caracteristica() [2/2]

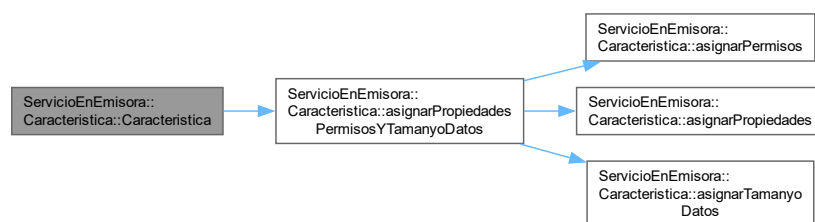
```
ServicioEnEmisora::Caracteristica::Caracteristica (
    const char * nombreCaracteristica_,
    uint8_t props,
    SecureMode_t permisoRead,
    SecureMode_t permisoWrite,
    uint8_t tam) [inline]
```

Constructor que permite definir propiedades y permisos de la característica.

#### Parámetros

<i>nombreCaracteristica_</i>	Nombre de la característica.
<i>props</i>	Propiedades (lectura, escritura, notificación).
<i>permisoRead</i>	Permiso de lectura.
<i>permisoWrite</i>	Permiso de escritura.
<i>tam</i>	Tamaño máximo de los datos.

Gráfico de llamadas de esta función:



### 6.1.3. Documentación de funciones miembro

#### 6.1.3.1. activar()

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

Activa la característica para que esté lista para operar.

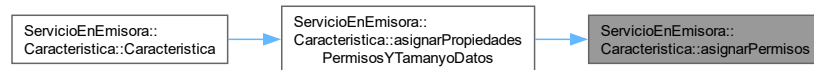
Gráfico de llamadas de esta función:



### 6.1.3.2. asignarPermisos()

```
void ServicioEnEmisora::Caracteristica::asignarPermisos (
    SecureMode_t permisoRead,
    SecureMode_t permisoWrite) [inline], [private]
```

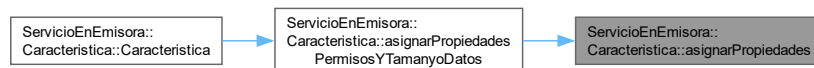
Gráfico de llamadas a esta función:



### 6.1.3.3. asignarPropiedades()

```
void ServicioEnEmisora::Caracteristica::asignarPropiedades (
    uint8_t props) [inline], [private]
```

Gráfico de llamadas a esta función:



### 6.1.3.4. asignarPropiedadesPermisosYTamanyoDatos()

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

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

Gráfico de llamadas de esta función:

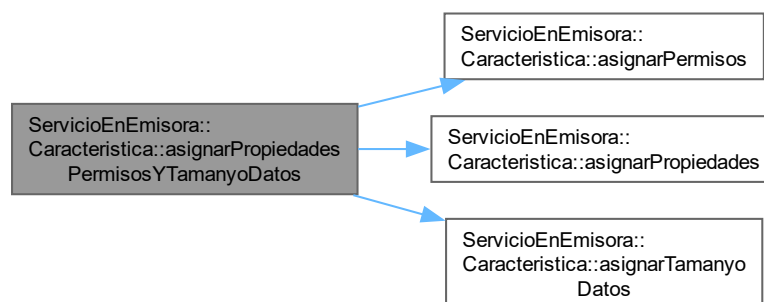
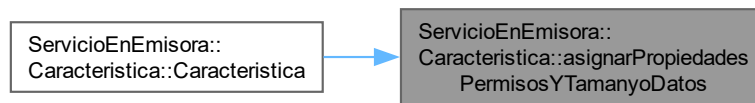




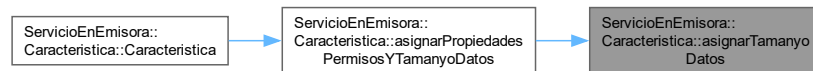
Gráfico de llamadas a esta función:



#### 6.1.3.5. asignarTamanyoDatos()

```
void ServicioEnEmisora::Caracteristica::asignarTamanyoDatos (
    uint8_t tam) [inline], [private]
```

Gráfico de llamadas a esta función:



#### 6.1.3.6. escribirDatos()

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

Escribe datos en la característica.

##### Parámetros

<i>str</i>	Cadena de caracteres que se escribirá.
------------	--

##### Devuelve

uint16\_t Número de bytes escritos.

#### 6.1.3.7. instalarCallbackCaracteristicaEscrita()

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

Instala un callback para cuando se escriben datos en la característica.

#### 6.1.3.8. notificarDatos()

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

Notifica datos a los suscriptores de la característica.

**Parámetros**

<i>str</i>	Cadena de caracteres que será notificada.
------------	---

**Devuelve**

uint16\_t Número de bytes notificados.

## 6.1.4. Documentación de datos miembro

### 6.1.4.1. laCaracteristica

```
BLECharacteristic ServicioEnEmisora::Caracteristica::laCaracteristica [private]
```

### 6.1.4.2. uuidCaracteristica

```
uint8_t ServicioEnEmisora::Caracteristica::uuidCaracteristica[16] [private]
```

**Valor inicial:**

```
= {  
    '0', '1', '2', '3', '4', '5', '6', '7',  
    '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'  
}
```

La documentación de esta clase está generada del siguiente archivo:

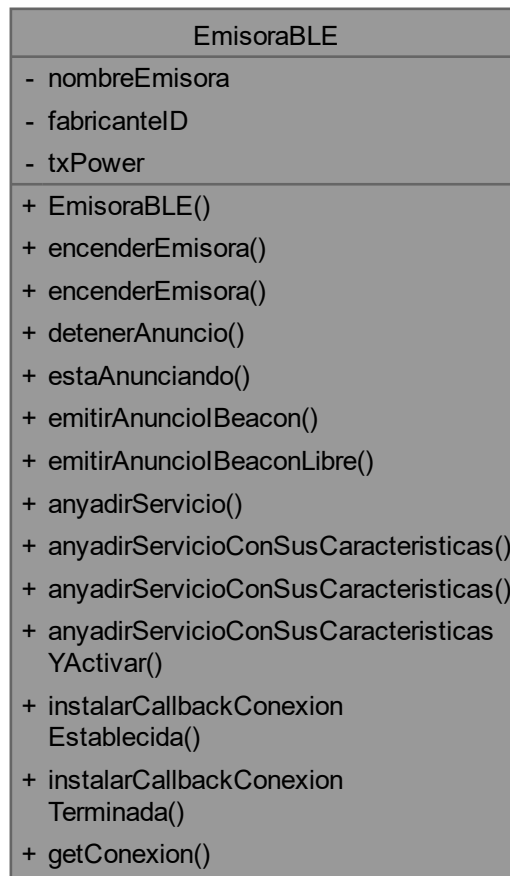
- HolaMundoIBeacon/[ServicioEnEmisora.h](#)

## 6.2. Referencia de la clase EmisoraBLE

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

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

Diagrama de colaboración de EmisoraBLE:



### Tipos públicos

- using [CallbackConexionEstablecida](#) = void ( uint16\_t connHandle )  
*Tipo de callback para manejar conexiones establecidas.*
- using [CallbackConexionTerminada](#) = void ( uint16\_t connHandle, uint8\_t reason)  
*Tipo de callback para manejar conexiones terminadas.*

### 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 con callbacks de conexión.*
- void [detenerAnuncio](#) ()

- Detiene el anuncio BLE actual.*
  - bool `estaAnunciando` ()
- Verifica si la emisora está anunciando.*
  - void `emitirAnuncioIBeacon` (uint8\_t \*beaconUUID, int16\_t major, int16\_t minor, uint8\_t rssi)
- Emite un iBeacon con los valores especificados.*
  - void `emitirAnuncioIBeaconLibre` (const char \*carga, const uint8\_t tamanyoCarga)
- Emite un anuncio iBeacon con una carga libre.*
  - bool `anyadirServicio` (`ServicioEnEmisora` &servicio)
- Añade un servicio a la emisora BLE.*
  - bool `anyadirServicioConSusCaracteristicas` (`ServicioEnEmisora` &servicio)
- Añade un servicio junto con sus características a la emisora BLE.*
  - template<typename ... T>  
bool `anyadirServicioConSusCaracteristicas` (`ServicioEnEmisora` &servicio, `ServicioEnEmisora::Caracteristica` &caracteristica, T &... restoCaracteristicas)
- Añade un servicio con múltiples características a la emisora BLE.*
  - template<typename ... T>  
bool `anyadirServicioConSusCaracteristicasYActivar` (`ServicioEnEmisora` &servicio, T &... restoCaracteristicas)
- Añade un servicio y sus características y luego lo activa.*
  - void `instalarCallbackConexionEstablecida` (`CallbackConexionEstablecida` cb)
- Instala el callback para manejar la conexión establecida.*
  - void `instalarCallbackConexionTerminada` (`CallbackConexionTerminada` cb)
- Instala el callback para manejar la desconexión.*
  - BLEConnection \* `getConexion` (uint16\_t connHandle)
- Obtiene la conexión BLE a través del manejador.*

## Atributos privados

- const char \* `nombreEmisora`
- const uint16\_t `fabricantID`
- const int8\_t `txPower`

## 6.2.1. Descripción detallada

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

Esta clase permite gestionar la activación de la emisora, la configuración de iBeacons, y la adición de servicios BLE.

## 6.2.2. Documentación de los «Typedef» miembros de la clase

### 6.2.2.1. CallbackConexionEstablecida

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

Tipo de callback para manejar conexiones establecidas.

## Parámetros

<i>connHandle</i>	Identificador del manejador de la conexión.
-------------------	---

## 6.2.2.2. CallbackConexionTerminada

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

Tipo de callback para manejar conexiones terminadas.

## Parámetros

<i>connHandle</i>	Identificador del manejador de la conexión.
<i>reason</i>	Razón por la cual se terminó la conexión.

## 6.2.3. Documentación de constructores y destructores

## 6.2.3.1. EmisoraBLE()

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

Constructor de la clase [EmisoraBLE](#).

## Parámetros

<i>nombreEmisora_</i>	Nombre de la emisora BLE.
<i>fabricanteID_</i>	ID del fabricante para los beacons.
<i>txPower_</i>	Potencia de transmisión de la emisora.

## 6.2.4. Documentación de funciones miembro

## 6.2.4.1. anyadirServicio()

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

Añade un servicio a la emisora BLE.

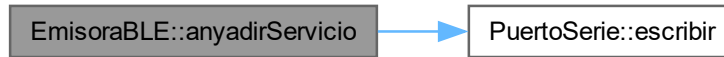
## Parámetros

<i>servicio</i>	Servicio BLE a añadir.
-----------------	------------------------

**Devuelve**

true si el servicio fue añadido exitosamente, false en caso contrario.

Gráfico de llamadas de esta función:

**6.2.4.2. anyadirServicioConSusCaracteristicas() [1/2]**

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

Añade un servicio junto con sus características a la emisora BLE.

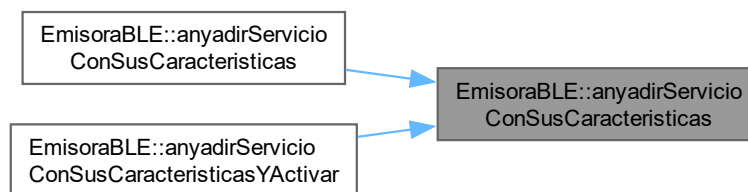
**Parámetros**

<i>servicio</i>	Servicio BLE a añadir.
-----------------	------------------------

**Devuelve**

true si el servicio y las características fueron añadidos exitosamente.

Gráfico de llamadas a esta función:

**6.2.4.3. anyadirServicioConSusCaracteristicas() [2/2]**

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

Añade un servicio con múltiples características a la emisora BLE.

## Parámetros de plantilla

<i>T</i>	Tipos de características adicionales.
----------	---------------------------------------

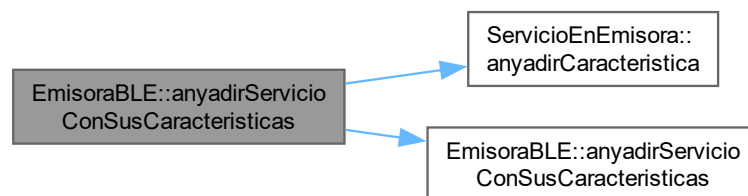
## Parámetros

<i>servicio</i>	Servicio BLE a añadir.
<i>caracteristica</i>	Primera característica.
<i>restoCaracteristicas</i>	Otras características adicionales.

## Devuelve

true si todo fue añadido correctamente.

Gráfico de llamadas de esta función:



## 6.2.4.4. anyadirServicioConSusCaracteristicasYActivar()

```

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

Añade un servicio y sus características y luego lo activa.

## Parámetros de plantilla

<i>T</i>	Tipos de características adicionales.
----------	---------------------------------------

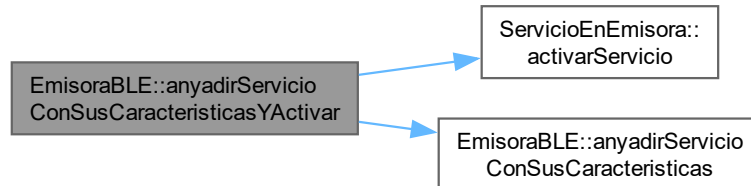
## Parámetros

<i>servicio</i>	Servicio BLE a añadir y activar.
-----------------	----------------------------------

**Devuelve**

true si todo fue añadido y activado correctamente.

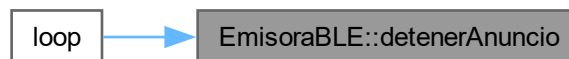
Gráfico de llamadas de esta función:

**6.2.4.5. detenerAnuncio()**

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

Detiene el anuncio BLE actual.

Gráfico de llamadas a esta función:

**6.2.4.6. emitirAnuncioIBeacon()**

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

Emita un iBeacon con los valores especificados.

**Parámetros**

<i>beaconUUID</i>	UUID del iBeacon.
<i>major</i>	Valor mayor del iBeacon.
<i>minor</i>	Valor menor del iBeacon.
<i>rssi</i>	Valor de la señal de transmisión (RSSI).



#### 6.2.4.7. emitirAnuncioIBeaconLibre()

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

Emite un anuncio iBeacon con una carga libre.

##### Parámetros

<i>carga</i>	Datos que se desean transmitir.
<i>tamanyoCarga</i>	Tamaño de los datos.

Gráfico de llamadas de esta función:

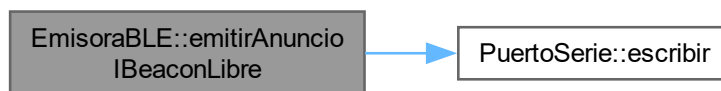
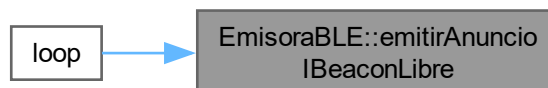


Gráfico de llamadas a esta función:



#### 6.2.4.8. encenderEmisora() [1/2]

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

Enciende la emisora BLE.

Gráfico de llamadas a esta función:



#### 6.2.4.9. encenderEmisora() [2/2]

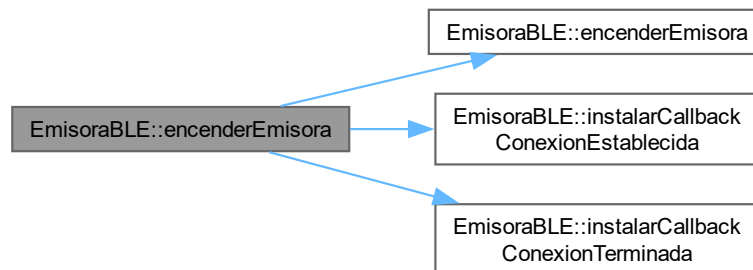
```
void EmisoraBLE::encenderEmisora (
    CallbackConexionEstablecida cbce,
    CallbackConexionTerminada cbct) [inline]
```

Enciende la emisora BLE con callbacks de conexión.

##### Parámetros

<i>cbce</i>	Callback para conexión establecida.
<i>cbct</i>	Callback para conexión terminada.

Gráfico de llamadas de esta función:



#### 6.2.4.10. estaAnunciando()

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

Verifica si la emisora está anunciando.

##### Devuelve

true si la emisora está anunciando, false en caso contrario.

#### 6.2.4.11. getConexion()

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

Obtiene la conexión BLE a través del manejador.

##### Parámetros

<i>connHandle</i>	Manejador de la conexión.
-------------------	---------------------------

##### Devuelve

Puntero a la conexión BLE.

#### 6.2.4.12. instalarCallbackConexionEstablecida()

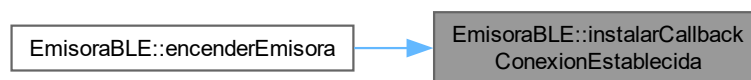
```
void EmisoraBLE::instalarCallbackConexionEstablecida (  
    CallbackConexionEstablecida cb) [inline]
```

Instala el callback para manejar la conexión establecida.

##### Parámetros

<i>cb</i>	Callback de conexión establecida.
-----------	-----------------------------------

Gráfico de llamadas a esta función:



#### 6.2.4.13. instalarCallbackConexionTerminada()

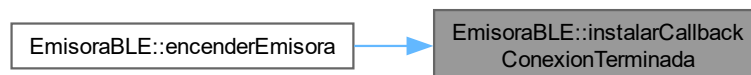
```
void EmisoraBLE::instalarCallbackConexionTerminada (  
    CallbackConexionTerminada cb) [inline]
```

Instala el callback para manejar la desconexión.

##### Parámetros

<i>cb</i>	Callback de desconexión.
-----------	--------------------------

Gráfico de llamadas a esta función:



### 6.2.5. Documentación de datos miembro

#### 6.2.5.1. fabricanteID

```
const uint16_t EmisoraBLE::fabricanteID [private]
```

### 6.2.5.2. nombreEmisora

```
const char* EmisoraBLE::nombreEmisora [private]
```

### 6.2.5.3. txPower

```
const int8_t EmisoraBLE::txPower [private]
```

La documentación de esta clase está generada del siguiente archivo:

- HolaMundoIBeacon/[EmisoraBLE.h](#)

## 6.3. Referencia de la clase LED

Clase para controlar un [LED](#) conectado a un pin específico.

```
#include <LED.h>
```

Diagrama de colaboración de LED:

LED
- numeroLED
- encendido
+ LED()
+ encender()
+ apagar()
+ alternar()
+ brillar()

### Métodos públicos

- [LED](#) (int numero)  
*Constructor de la clase [LED](#).*
- void [encender](#) ()  
*Enciende el [LED](#).*
- void [apagar](#) ()  
*Apaga el [LED](#).*
- void [alternar](#) ()  
*Alterna el estado del [LED](#).*
- void [brillar](#) (long tiempo)  
*Hace que el [LED](#) brille por un tiempo determinado.*

### Atributos privados

- int `numeroLED`
- bool `encendido`

### 6.3.1. Descripción detallada

Clase para controlar un LED conectado a un pin específico.

Esta clase permite encender, apagar, alternar y hacer brillar un LED que está conectado a un pin digital del micro-controlador.

### 6.3.2. Documentación de constructores y destructores

#### 6.3.2.1. LED()

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

Constructor de la clase LED.

Inicializa el pin del LED y lo apaga por defecto.

#### Parámetros

<code>numero</code>	Número del pin donde está conectado el LED.
---------------------	---

< Apaga el LED al inicializar. Gráfico de llamadas de esta función:



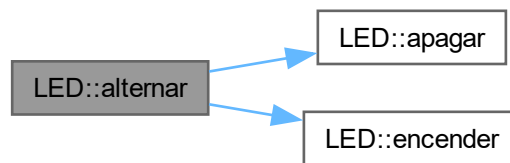
### 6.3.3. Documentación de funciones miembro

#### 6.3.3.1. alternar()

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

Alterna el estado del LED.

Si el **LED** está encendido, lo apaga; si está apagado, lo enciende. Gráfico de llamadas de esta función:

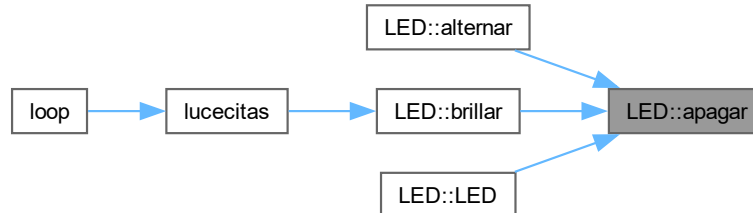


### 6.3.3.2. apagar()

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

Apaga el **LED**.

Gráfico de llamadas a esta función:



### 6.3.3.3. brillar()

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

Hace que el **LED** brille por un tiempo determinado.

Enciende el **LED**, espera el tiempo especificado, y luego lo apaga.

#### Parámetros

<i>tiempo</i>	Tiempo en milisegundos durante el cual el <b>LED</b> estará encendido.
---------------	--

Gráfico de llamadas de esta función:

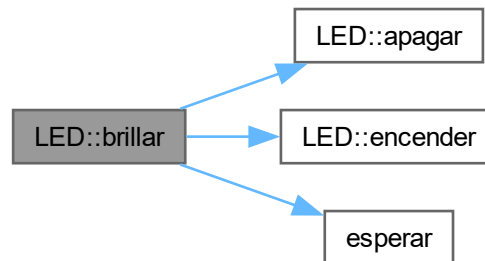
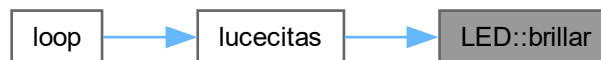


Gráfico de llamadas a esta función:

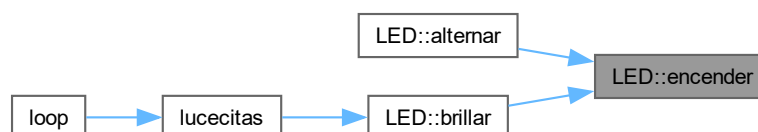


#### 6.3.3.4. encender()

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

Enciende el [LED](#).

Gráfico de llamadas a esta función:



### 6.3.4. Documentación de datos miembro

#### 6.3.4.1. encendido

```
bool LED::encendido [private]
```

Estado actual del [LED](#) (encendido = true, apagado = false).

#### 6.3.4.2. numeroLED

```
int LED::numeroLED [private]
```

Número del pin al que está conectado el [LED](#).

La documentación de esta clase está generada del siguiente archivo:

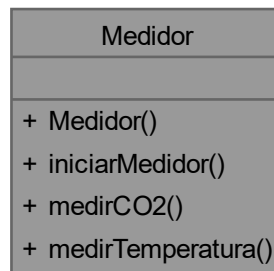
- HolaMundoIBeacon/[LED.h](#)

## 6.4. Referencia de la clase Medidor

Clase para simular la medición de CO2 y temperatura.

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

Diagrama de colaboración de Medidor:



### Métodos públicos

- [Medidor](#) ()  
*Constructor de la clase [Medidor](#).*
- void [iniciarMedidor](#) ()  
*Método para iniciar el medidor.*
- int [medirCO2](#) ()  
*Medir el nivel de CO2.*
- int [medirTemperatura](#) ()  
*Medir la temperatura.*



### 6.4.1. Descripción detallada

Clase para simular la medición de CO2 y temperatura.

Esta clase simula la medición de niveles de CO2 y temperatura. En un sistema real, esta clase podría interactuar con sensores reales para obtener lecturas.

### 6.4.2. Documentación de constructores y destructores

#### 6.4.2.1. Medidor()

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

Constructor de la clase [Medidor](#).

Inicializa el medidor. Actualmente, no realiza ninguna acción en el constructor.

### 6.4.3. Documentación de funciones miembro

#### 6.4.3.1. iniciarMedidor()

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

Método para iniciar el medidor.

Se utiliza para realizar cualquier configuración que no pueda hacerse en el constructor. Actualmente no realiza ninguna acción. Gráfico de llamadas a esta función:



#### 6.4.3.2. medirCO2()

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

Medir el nivel de CO2.

Devuelve un valor simulado de la medición de CO2.

**Devuelve**

int Valor del nivel de CO2 (simulado).

< Valor simulado de CO2. Gráfico de llamadas a esta función:

**6.4.3.3. medirTemperatura()**

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

Medir la temperatura.

Devuelve un valor simulado de la medición de temperatura.

**Devuelve**

int Valor de la temperatura (simulado).

< Valor simulado de temperatura. Gráfico de llamadas a esta función:



La documentación de esta clase está generada del siguiente archivo:

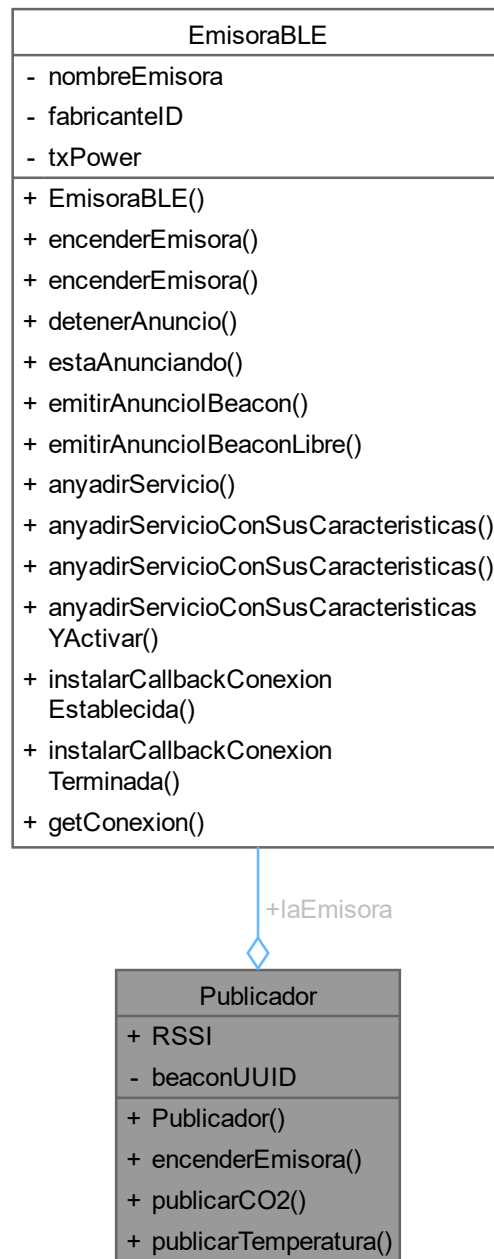
- HolaMundoIBeacon/[Medidor.h](#)

## 6.5. Referencia de la clase Publicador

Clase para publicar mediciones de CO2 y temperatura mediante BLE.

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

Diagrama de colaboración de Publicador:



## Tipos públicos

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

*Identificadores de tipo de medición para el campo mayor de iBeacon.*

## Métodos públicos

- `Publicador` ()  
*Constructor de la clase `Publicador`.*
- void `encenderEmisora` ()  
*Enciende la emisora BLE.*
- void `publicarCO2` (int16\_t valorCO2, uint8\_t contador, long tiempoEspera)  
*Publica un valor de CO2 mediante un anuncio iBeacon.*
- void `publicarTemperatura` (int16\_t valorTemperatura, uint8\_t contador, long tiempoEspera)  
*Publica un valor de temperatura mediante un anuncio iBeacon.*

## Atributos públicos

- `EmisoraBLE laEmisora`  
*Emisora BLE que se encarga de realizar los anuncios.*
- const int `RSSI` = -53  
*Nivel RSSI utilizado para los anuncios. Valor predeterminado.*

## Atributos privados

- uint8\_t `beaconUUID` [16]  
*UUID del beacon utilizado para identificar el origen de los anuncios.*

### 6.5.1. Descripción detallada

Clase para publicar mediciones de CO2 y temperatura mediante BLE.

Esta clase permite emitir anuncios BLE con los valores medidos de CO2 y temperatura. Se utiliza la clase `EmisoraBLE` para manejar los anuncios de tipo iBeacon.

### 6.5.2. Documentación de las enumeraciones miembro de la clase

#### 6.5.2.1. `MedicionesID`

```
enum Publicador::MedicionesID
```

Identificadores de tipo de medición para el campo mayor de iBeacon.

Estos valores se utilizan para distinguir entre distintos tipos de mediciones en los anuncios.

## Valores de enumeraciones

CO2	Identificador de la medición de CO2.
TEMPERATURA	Identificador de la medición de temperatura.
RUIDO	Identificador de la medición de ruido (no implementado).

**6.5.3. Documentación de constructores y destructores****6.5.3.1. Publicador()**

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

Constructor de la clase [Publicador](#).

Inicializa el publicador, pero no enciende la emisora BLE. La emisora debe encenderse explícitamente llamando a [encenderEmisora\(\)](#).

**6.5.4. Documentación de funciones miembro****6.5.4.1. encenderEmisora()**

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

Enciende la emisora BLE.

Este método debe ser llamado desde el setup del sistema para iniciar la emisora BLE. Gráfico de llamadas a esta función:

**6.5.4.2. publicarCO2()**

```
void Publicador::publicarCO2 (
    int16_t valorCO2,
    uint8_t contador,
    long tiempoEspera) [inline]
```

Publica un valor de CO2 mediante un anuncio iBeacon.

Emite un anuncio BLE con el valor de CO2 especificado, incluyendo un contador para diferenciar anuncios.

**Parámetros**

<i>valorCO2</i>	Valor de la medición de CO2 (minor).
<i>contador</i>	Contador para identificar la secuencia de anuncios (parte del mayor).
<i>tiempoEspera</i>	Tiempo de espera antes de detener el anuncio.

Gráfico de llamadas de esta función:

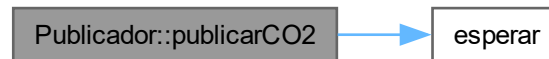


Gráfico de llamadas a esta función:

**6.5.4.3. publicarTemperatura()**

```

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

Publica un valor de temperatura mediante un anuncio iBeacon.

Emite un anuncio BLE con el valor de temperatura especificado, incluyendo un contador para diferenciar anuncios.

**Parámetros**

<i>valorTemperatura</i>	Valor de la medición de temperatura (minor).
<i>contador</i>	Contador para identificar la secuencia de anuncios (parte del mayor).
<i>tiempoEspera</i>	Tiempo de espera antes de detener el anuncio.

Gráfico de llamadas de esta función:

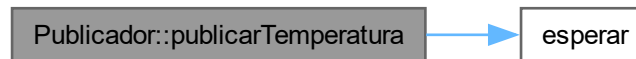
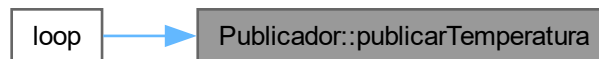


Gráfico de llamadas a esta función:



## 6.5.5. Documentación de datos miembro

### 6.5.5.1. beaconUUID

```
uint8_t Publicador::beaconUUID[16] [private]
```

**Valor inicial:**

```
= {
    'E', 'P', 'S', 'G', '-', 'G', 'T', 'I',
    '-', 'P', 'R', 'O', 'Y', '-', '3', 'A'
}
```

UUID del beacon utilizado para identificar el origen de los anuncios.

### 6.5.5.2. laEmisora

```
EmisoraBLE Publicador::laEmisora
```

**Valor inicial:**

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

Emisora BLE que se encarga de realizar los anuncios.

Nombre de la emisora BLE. ID del fabricante (Apple). Potencia de transmisión (txPower).

### 6.5.5.3. RSSI

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

Nivel RSSI utilizado para los anuncios. Valor predeterminado.

La documentación de esta clase está generada del siguiente archivo:

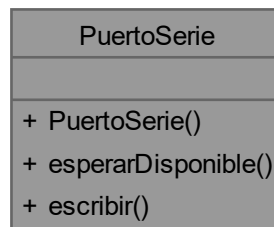
- HolaMundoIBeacon/[Publicador.h](#)

## 6.6. Referencia de la clase PuertoSerie

Clase para manejar la comunicación por puerto serie.

```
#include <PuertoSerie.h>
```

Diagrama de colaboración de PuertoSerie:



### Métodos públicos

- [PuertoSerie](#) (long baudios)  
*Constructor de la clase [PuertoSerie](#).*
- void [esperarDisponible](#) ()  
*Espera un pequeño tiempo antes de que el puerto serie esté disponible.*
- template<typename T >  
void [escribir](#) (T mensaje)  
*Escribe un mensaje a través del puerto serie.*

### 6.6.1. Descripción detallada

Clase para manejar la comunicación por puerto serie.

Esta clase permite inicializar la comunicación serie y escribir mensajes a través del puerto serie de manera sencilla.

### 6.6.2. Documentación de constructores y destructores

#### 6.6.2.1. PuertoSerie()

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

Constructor de la clase [PuertoSerie](#).

Inicializa el puerto serie con la velocidad especificada en baudios.



## Parámetros

<i>baudios</i>	Velocidad en baudios para la comunicación serie.
----------------	--

### 6.6.3. Documentación de funciones miembro

#### 6.6.3.1. escribir()

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

Escribe un mensaje a través del puerto serie.

Este método utiliza plantillas para escribir cualquier tipo de dato compatible con `Serial.print()`.

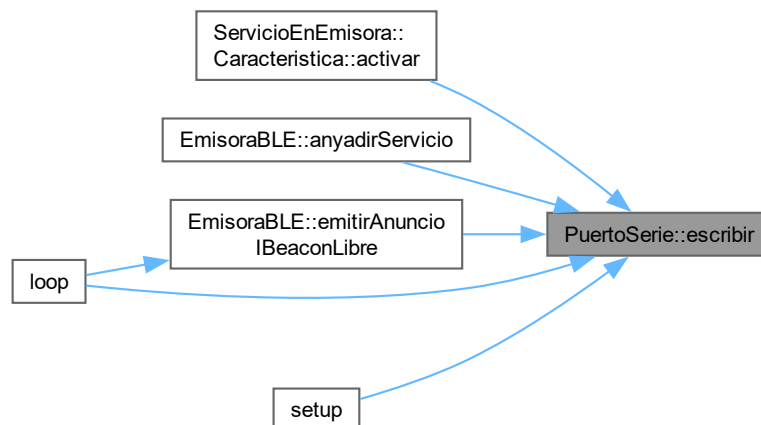
## Parámetros de plantilla

<i>T</i>	Tipo del mensaje a escribir.
----------	------------------------------

## Parámetros

<i>mensaje</i>	El mensaje a escribir por el puerto serie.
----------------	--

Gráfico de llamadas a esta función:

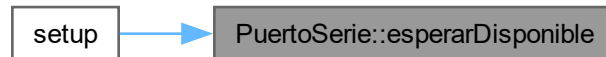


### 6.6.3.2. esperarDisponible()

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

Espera un pequeño tiempo antes de que el puerto serie esté disponible.

Este método puede ser útil para esperar brevemente antes de operar con el puerto serie. Gráfico de llamadas a esta función:



La documentación de esta clase está generada del siguiente archivo:

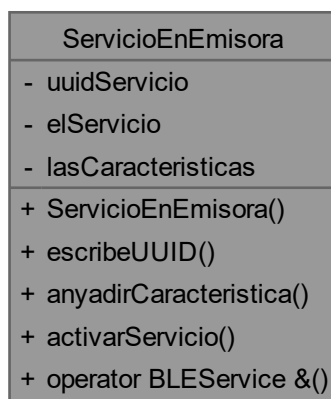
- HolaMundoIBeacon/[PuertoSerie.h](#)

## 6.7. Referencia de la clase ServicioEnEmisora

Clase que gestiona un servicio BLE con características.

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

Diagrama de colaboración de ServicioEnEmisora:



## Clases

- class [Caracteristica](#)

*Clase que representa una característica dentro de un servicio BLE.*

## Tipos públicos

- using [CallbackCaracteristicaEscrita](#) = void (uint16\_t conn\_handle, BLECharacteristic \* chr, uint8\_t \* data, uint16\_t len)

*Definición del tipo de callback que se ejecuta al escribir en una característica.*

## Métodos públicos

- [ServicioEnEmisora](#) (const char \*nombreServicio\_)  
*Constructor que inicializa el servicio con su UUID.*
- void [escribeUUID](#) ()  
*Escribe el UUID del servicio en el puerto serie.*
- void [anyadirCaracteristica](#) ([Caracteristica](#) &car)  
*Añade una característica al servicio.*
- void [activarServicio](#) ()  
*Activa el servicio y todas sus características.*
- operator [BLEService](#) & ()  
*Operador de conversión a BLEService.*

## Atributos privados

- uint8\_t [uuidServicio](#) [16]
- BLEService [elServicio](#)
- std::vector< [Caracteristica](#) \* > [lasCaracteristicas](#)

### 6.7.1. Descripción detallada

Clase que gestiona un servicio BLE con características.

Esta clase representa un servicio en una emisora BLE con soporte para múltiples características.

### 6.7.2. Documentación de los «Typedef» miembros de la clase

#### 6.7.2.1. CallbackCaracteristicaEscrita

```
using ServicioEnEmisora::CallbackCaracteristicaEscrita = void (uint16_t conn_handle, BLECharacteristic
* chr, uint8_t * data, uint16_t len)
```

Definición del tipo de callback que se ejecuta al escribir en una característica.

### 6.7.3. Documentación de constructores y destructores

#### 6.7.3.1. ServicioEnEmisora()

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

Constructor que inicializa el servicio con su UUID.

## Parámetros

<i>nombre</i> ↔	Nombre del servicio que será convertido a UUID.
<i>Servicio_</i>	

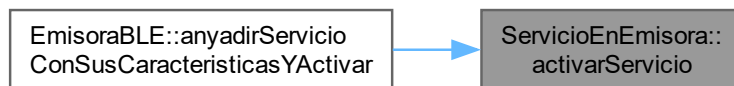
## 6.7.4. Documentación de funciones miembro

### 6.7.4.1. activarServicio()

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

Activa el servicio y todas sus características.

Gráfico de llamadas a esta función:



### 6.7.4.2. anyadirCaracteristica()

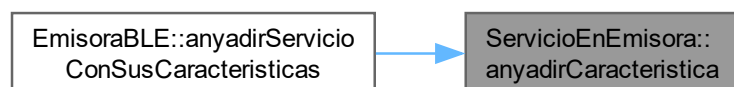
```
void ServicioEnEmisora::anyadirCaracteristica (
    Caracteristica & car) [inline]
```

Añade una característica al servicio.

## Parámetros

<i>car</i>	Referencia a la característica que se añadirá.
------------	--

Gráfico de llamadas a esta función:



#### 6.7.4.3. escribeUUID()

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

Escribe el UUID del servicio en el puerto serie.

#### 6.7.4.4. operator BLEService &()

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

Operador de conversión a BLEService.

Permite usar un objeto de esta clase donde se necesita un BLEService.

### 6.7.5. Documentación de datos miembro

#### 6.7.5.1. elServicio

```
BLEService ServicioEnEmisora::elServicio [private]
```

#### 6.7.5.2. lasCaracteristicas

```
std::vector<Caracteristica *> ServicioEnEmisora::lasCaracteristicas [private]
```

#### 6.7.5.3. uuidServicio

```
uint8_t ServicioEnEmisora::uuidServicio[16] [private]
```

##### Valor inicial:

```
= {  
    '0', '1', '2', '3', '4', '5', '6', '7',  
    '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'  
}
```

La documentación de esta clase está generada del siguiente archivo:

- HolaMundoIBeacon/[ServicioEnEmisora.h](#)



## Capítulo 7

# Documentación de archivos

### 7.1. Referencia del archivo HolaMundoBeacon/EmisoraBLE.h

Clase para manejar la emisora BLE y la emisión de beacons.

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

Gráfico de dependencias incluidas en EmisoraBLE.h:

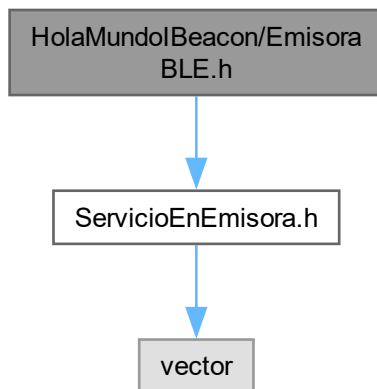
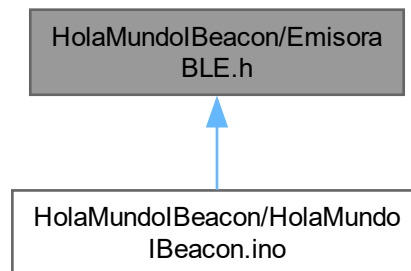


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Clases

- class [EmisoraBLE](#)

*Clase para manejar la emisora Bluetooth Low Energy (BLE).*

### 7.1.1. Descripción detallada

Clase para manejar la emisora BLE y la emisión de beacons.

Este archivo contiene la definición de la clase [EmisoraBLE](#) que permite gestionar la emisora BLE, emitir iBeacons y manejar servicios publicitarios a través del protocolo BLE.

## 7.2. EmisoraBLE.h

[Ir a la documentación de este archivo.](#)

```

00001 // -*- mode: c++ -*-
00002
00003 #ifndef EMISORA_H_INCLUIDO
00004 #define EMISORA_H_INCLUIDO
00005
00015 // -----
00016 // -----
00017 #include "ServicioEnEmisora.h"
00018
00027 // -----
00028 // -----
00029 class EmisoraBLE {
00030 private:
00031
00032     const char * nombreEmisora;
00033     const uint16_t fabricanteID;
00034     const int8_t txPower;
00035
00036 public:
00037
00044     using CallbackConexionEstablecida = void ( uint16_t connHandle );
00045
00053     using CallbackConexionTerminada = void ( uint16_t connHandle, uint8_t reason);
00054
00062     EmisoraBLE( const char * nombreEmisora_, const uint16_t fabricanteID_,
00063               const int8_t txPower_ )
00064     :
00065         nombreEmisora( nombreEmisora_ ) ,
  
```



```

00066     fabricanteID( fabricanteID_ ) ,
00067     txPower( txPower_ ){} // ()
00068
00073 void encenderEmisora() {
00074     // Serial.println ( "Bluefruit.begin() " );
00075     Bluefruit.begin();
00076
00077     // por si acaso:
00078     (*this).detenerAnuncio();
00079 } // ()
00080
00087 void encenderEmisora( CallbackConexionEstablecida cbce,
00088                     CallbackConexionTerminada cbct ) {
00089
00090     encenderEmisora();
00091
00092     instalarCallbackConexionEstablecida( cbce );
00093     instalarCallbackConexionTerminada( cbct );
00094
00095 } // ()
00096
00100 void detenerAnuncio() {
00101
00102     if ( (*this).estaAnunciando() ) {
00103         // Serial.println ( "Bluefruit.Advertising.stop() " );
00104         Bluefruit.Advertising.stop();
00105     }
00106
00107 } // ()
00108
00114 bool estaAnunciando() {
00115     return Bluefruit.Advertising.isRunning();
00116 } // ()
00117
00126 void emitirAnuncioIBeacon( uint8_t * beaconUUID, int16_t major, int16_t minor, uint8_t rssi ) {
00127
00128     //
00129     //
00130     //
00131     (*this).detenerAnuncio();
00132
00133     //
00134     // creo el beacon
00135     //
00136     BLEBeacon elBeacon( beaconUUID, major, minor, rssi );
00137     elBeacon.setManufacturer( (*this).fabricanteID );
00138
00139     //
00140     // parece que esto debe ponerse todo aquí
00141     //
00142
00143     Bluefruit.setTxPower( (*this).txPower );
00144     Bluefruit.setName( (*this).nombreEmisora );
00145     Bluefruit.ScanResponse.addName(); // para que envíe el nombre de emisora (!)
00146
00147     //
00148     // pongo el beacon
00149     //
00150     Bluefruit.Advertising.setBeacon( elBeacon );
00151
00152     //
00153     // ? qué valores poner aquí
00154     //
00155     Bluefruit.Advertising.restartOnDisconnect(true); // no hace falta, pero lo pongo
00156     Bluefruit.Advertising.setInterval(100, 100); // in unit of 0.625 ms
00157
00158     //
00159     // empieza el anuncio, 0 = tiempo indefinido (ya lo pararán)
00160     //
00161     Bluefruit.Advertising.start( 0 );
00162
00163 } // ()
00164
00165 // .....
00166 //
00167 // Ejemplo de Beacon (31 bytes)
00168 //
00169 // https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/
00170 //
00171 // The iBeacon Prefix contains the hex data : 0x0201061AFF004C0215. This breaks down as follows:
00172 //
00173 // 0x020106 defines the advertising packet as BLE General Discoverable and BR/EDR high-speed
00174 // incompatible.
00175 // Effectively it says this is only broadcasting, not connecting.
00176 // 0x1AFF says the following data is 26 bytes long and is Manufacturer Specific Data.
00177 //

```

```

00178 // 0x004C is Apple's Bluetooth Sig ID and is the part of this spec that makes it Apple-dependent.
00179 //
00180 // 0x02 is a secondary ID that denotes a proximity beacon, which is used by all iBeacons.
00181 //
00182 // 0x15 defines the remaining length to be 21 bytes (16+2+2+1).
00183 //
00184 // Por ejemplo:
00185 //
00186 // 1. prefijo: 9bytes
00187 //      0x02, 0x01, 0x06, // advFlags 3bytes
00188 //      0x1a, 0xff, // advHeader 2 (0x1a = 26 = 25(lenght de 0x4c a 0xca)+1) 0xFF ->
BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA
00189 //      0x4c, 0x00, // companyID 2bytes
00190 //      0x02, // ibeacon type 1 byte
00191 //      0x15, // ibeacon length 1 byte (dec=21 lo que va a continuación: desde
la 'f' hasta 0x01)
00192 //
00193 // 2. uuid: 16bytes
00194 // 'f', 'i', 's', 't', 'r', 'o', 'f', 'i', 's', 't', 'r', 'o', 0xa7, 0x10, 0x96, 0xe0
00195 //
00196 // 2 major: 2bytes
00197 // 0x04, 0xd2,
00198 //
00199 // minor: 2bytes
00200 // 0x10, 0xe1,
00201 //
00202 // 0xca, // tx power : 1bytes
00203 //
00204 // 0x01, // este es el byte 31 = BLE_GAP_ADV_SET_DATA_SIZE_MAX, parece que sobra
00205 //
00206 // .....
00207 // Para enviar como carga libre los últimos 21 bytes de un iBeacon (lo que normalmente sería uuid-16
major-2 minor-2 txPower-1)
00208 // .....
00209 /*
00210 void emitirAnuncioIBeaconLibre( const char * carga ) {
00211
00212     const uint8_t tamanyoCarga = strlen( carga );
00213     */
00214
00221 void emitirAnuncioIBeaconLibre( const char * carga, const uint8_t tamanyoCarga ) {
00222
00223     (*this).detenerAnuncio();
00224
00225     Bluefruit.Advertising.clearData();
00226     Bluefruit.ScanResponse.clearData(); // hace falta?
00227
00228     // Bluefruit.setTxPower( (*this).txPower ); creo que no lo pongo porque es uno de los bytes de la
parte de carga que utilizo
00229     Bluefruit.setName( (*this).nombreEmisora );
00230     Bluefruit.ScanResponse.addName();
00231
00232     Bluefruit.Advertising.addFlags(BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE);
00233
00234     // con este parece que no va !
00235     // Bluefruit.Advertising.addFlags(BLE_GAP_ADV_FLAG_LE_GENERAL_DISC_MODE);
00236
00237     //
00238     // hasta ahora habrá, supongo, ya puestos los 5 primeros bytes. Efectivamente.
00239     // Falta poner 4 bytes fijos (company ID, beacon type, longitud) y 21 de carga
00240     //
00241     uint8_t restoPrefijoYCarga[4+21] = {
00242         0x4c, 0x00, // companyID 2
00243         0x02, // ibeacon type 1byte
00244         21, // ibeacon length 1byte (dec=21) longitud del resto // 0x15 // ibeacon length 1byte
(dec=21) longitud del resto
00245         '-', '-', '-', '-',
00246         '-', '-', '-', '-',
00247         '-', '-', '-', '-',
00248         '-', '-', '-', '-',
00249         '-', '-', '-', '-',
00250         '-',
00251     };
00252
00253     //
00254     // addData() hay que usarlo sólo una vez. Por eso copio la carga
00255     // en el anterior array, donde he dejado 21 sitios libres
00256     //
00257     memcpy( &restoPrefijoYCarga[4], &carga[0], ( tamanyoCarga > 21 ? 21 : tamanyoCarga ) );
00258
00259     //
00260     // copio la carga para emitir
00261     //
00262     Bluefruit.Advertising.addData( BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA,
00263                                     &restoPrefijoYCarga[0],
00264                                     4+21 );
00265

```

```

00266 //
00267 // ? qué valores poner aquí ?
00268 //
00269 Bluefruit.Advertising.restartOnDisconnect(true);
00270 Bluefruit.Advertising.setInterval(100, 100); // in unit of 0.625 ms
00271
00272 Bluefruit.Advertising.setFastTimeout( 1 ); // number of seconds in fast mode
00273 //
00274 // empieza el anuncio, 0 = tiempo indefinido (ya lo pararán)
00275 //
00276 Bluefruit.Advertising.start( 0 );
00277
00278 Globales::elPuerto.escribir( "emitirIBeacon libre Bluefruit.Advertising.start( 0 ); \n");
00279 } // ()
00280
00281 bool anyadirServicio( ServicioEnEmisora & servicio ) {
00282
00283     Globales::elPuerto.escribir( " Bluefruit.Advertising.addService( servicio ); \n");
00284
00285     bool r = Bluefruit.Advertising.addService( servicio );
00286
00287     if ( ! r ) {
00288         Serial.println( " SERVICION NO AÑADIDO \n");
00289     }
00290
00291     return r;
00292     // nota: uso conversión de tipo de servicio (ServicioEnEmisora) a BLEService
00293     // para addService()
00294 } // ()
00295
00296 bool anyadirServicioConSusCaracteristicas( ServicioEnEmisora & servicio ) {
00297     return (*this).anyadirServicio( servicio );
00298 } // ()
00299
00300 template <typename ... T>
00301 bool anyadirServicioConSusCaracteristicas( ServicioEnEmisora & servicio,
00302     ServicioEnEmisora::Caracteristica & caracteristica,
00303     T& ... restoCaracteristicas ) {
00304     servicio.anyadirCaracteristica( caracteristica );
00305
00306     return anyadirServicioConSusCaracteristicas( servicio, restoCaracteristicas... );
00307 } // ()
00308
00309 template <typename ... T>
00310 bool anyadirServicioConSusCaracteristicasYActivar( ServicioEnEmisora & servicio,
00311     ServicioEnEmisora::Caracteristica &
00312     caracteristica,
00313     T& ... restoCaracteristicas ) {
00314     bool r = anyadirServicioConSusCaracteristicas( servicio, restoCaracteristicas... );
00315
00316     servicio.activarServicio();
00317
00318     return r;
00319 } // ()
00320
00321 void instalarCallbackConexionEstablecida( CallbackConexionEstablecida cb ) {
00322     Bluefruit.Periph.setConnectCallback( cb );
00323 } // ()
00324
00325 void instalarCallbackConexionTerminada( CallbackConexionTerminada cb ) {
00326     Bluefruit.Periph.setDisconnectCallback( cb );
00327 } // ()
00328
00329 BLEConnection * getConnection( uint16_t connHandle ) {
00330     return Bluefruit.Connection( connHandle );
00331 } // ()
00332
00333 }; // class
00334 #endif // EMISORA_H_INCLUIDO

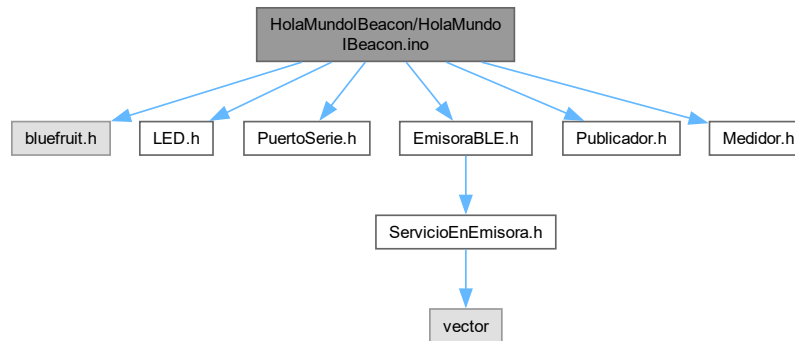
```

## 7.3. Referencia del archivo HolaMundoIBeacon/HolaMundoIBeacon.ino

Implementación principal del programa que utiliza BLE para publicar datos de sensores.

```
#include <bluefruit.h>
#include "LED.h"
#include "PuertoSerie.h"
#include "EmisoraBLE.h"
#include "Publicador.h"
#include "Medidor.h"
```

Gráfico de dependencias incluidas en HolaMundoIBeacon.ino:



## Espacios de nombres

- namespace [Globales](#)  
*Contiene objetos globales para manejar el [LED](#), la comunicación por puerto serie y otros módulos.*
- namespace [Loop](#)  
*Espacio de nombres para variables del bucle principal.*

## Funciones

- void [inicializarPlaquita](#) ()  
*Inicializa la placa y sus módulos. Actualmente no hace nada.*
- void [setup](#) ()  
*Función de configuración (setup). Inicializa los módulos de comunicación, medición y publicación.*
- void [lucecitas](#) ()  
*Realiza una secuencia de encendido y apagado del [LED](#). Utiliza tiempos de espera definidos entre cambios de estado.*
- void [loop](#) ()  
*Función principal (loop) del programa. Ejecuta mediciones de CO2 y temperatura, publica los datos vía BLE y maneja la emisión de un anuncio iBeacon.*

## Variables

- [LED Globales::elLED](#) (7)  
*Objeto para controlar el [LED](#) conectado al pin 7.*
- [PuertoSerie Globales::elPuerto](#) (115200)  
*Objeto para gestionar la comunicación por puerto serie a 115200 baudios.*
- [Publicador Globales::elPublicador](#)  
*Objeto para manejar la publicación de datos vía BLE.*
- [Medidor Globales::elMedidor](#)  
*Objeto para gestionar las mediciones de CO2 y temperatura.*
- uint8\_t [Loop::cont](#) = 0

### 7.3.1. Descripción detallada

Implementación principal del programa que utiliza BLE para publicar datos de sensores.

### 7.3.2. Documentación de funciones

#### 7.3.2.1. inicializarPlaquita()

```
void inicializarPlaquita ()
```

Inicializa la placa y sus módulos. Actualmente no hace nada.

Gráfico de llamadas a esta función:

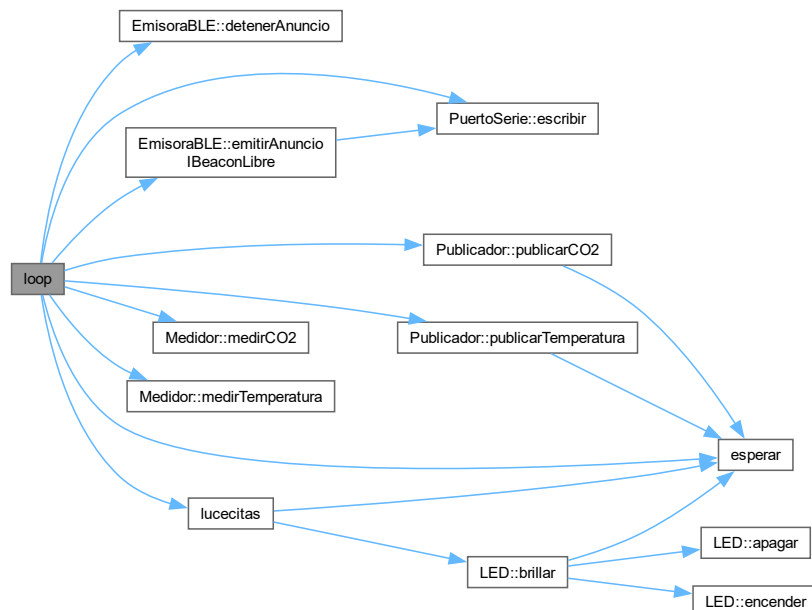


#### 7.3.2.2. loop()

```
void loop ()
```

Función principal (loop) del programa. Ejecuta mediciones de CO2 y temperatura, publica los datos vía BLE y maneja la emisión de un anuncio iBeacon.

Gráfico de llamadas de esta función:



### 7.3.2.3. lucecitas()

```
void lucecitas () [inline]
```

Realiza una secuencia de encendido y apagado del LED. Utiliza tiempos de espera definidos entre cambios de estado.

Gráfico de llamadas de esta función:

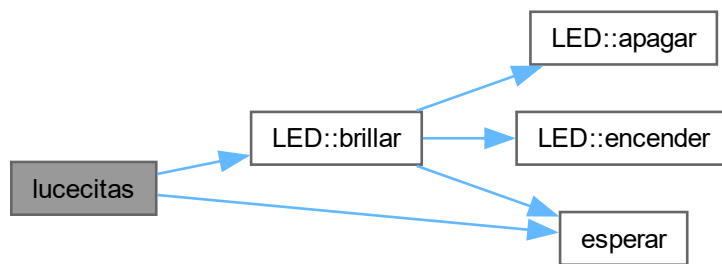


Gráfico de llamadas a esta función:

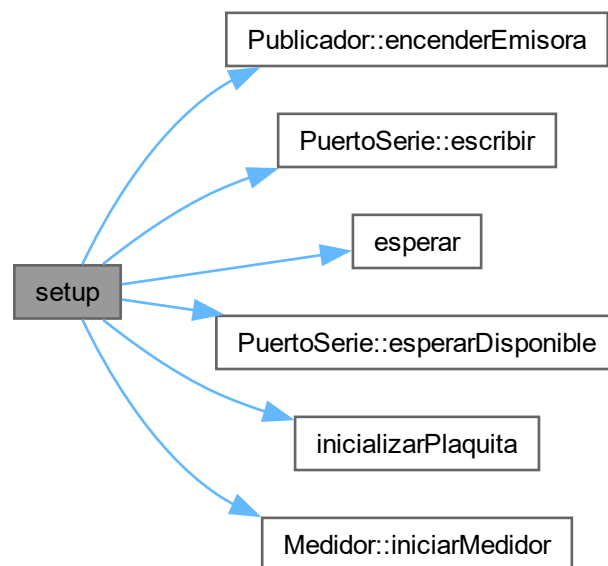


### 7.3.2.4. setup()

```
void setup ()
```

Función de configuración (`setup`). Inicializa los módulos de comunicación, medición y publicación.

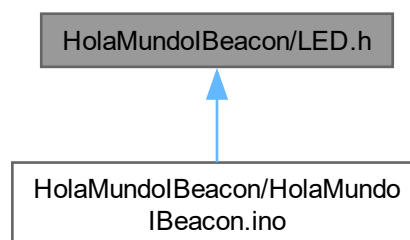
Gráfico de llamadas de esta función:



## 7.4. Referencia del archivo HolaMundoIBeacon/LED.h

Definición de la clase `LED` para controlar un `LED` en un pin específico.

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



### Clases

- class `LED`

*Clase para controlar un `LED` conectado a un pin específico.*

## Funciones

- void `esperar` (long tiempo)

*Función auxiliar para pausar la ejecución por un tiempo determinado.*

### 7.4.1. Descripción detallada

Definición de la clase `LED` para controlar un `LED` en un pin específico.

Este archivo contiene la implementación de la clase `LED` que permite controlar el estado (encendido/apagado) de un `LED` conectado a un pin digital de un microcontrolador. También incluye funciones para alternar el estado y hacer que el `LED` parpadee.

### 7.4.2. Documentación de funciones

#### 7.4.2.1. `esperar()`

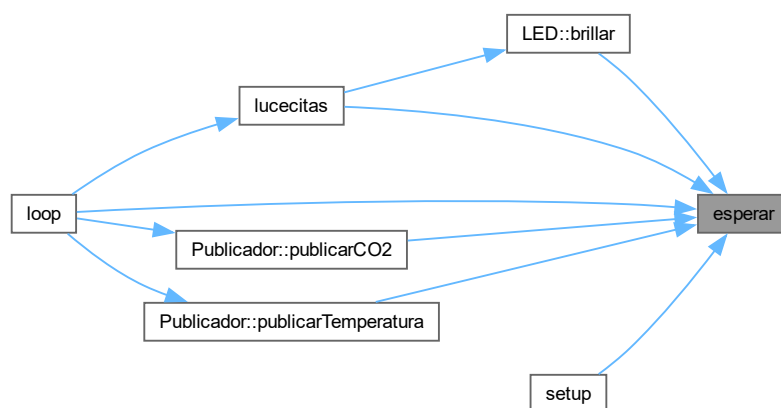
```
void esperar (  
    long tiempo)
```

Función auxiliar para pausar la ejecución por un tiempo determinado.

#### Parámetros

<code>tiempo</code>	Duración de la pausa en milisegundos.
---------------------	---------------------------------------

Gráfico de llamadas a esta función:





## 7.5. LED.h

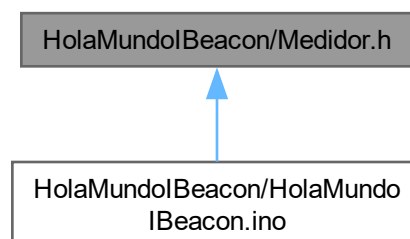
[Ir a la documentación de este archivo.](#)

```
00001 // -*- mode: c++ -*-
00002
00003 #ifndef LED_H_INCLUIDO
00004 #define LED_H_INCLUIDO
00005
00020 void esperar(long tiempo) {
00021     delay(tiempo);
00022 }
00023
00031 class LED {
00032 private:
00033     int numeroLED;
00034     bool encendido;
00036 public:
00037
00045     LED(int numero)
00046         : numeroLED(numero), encendido(false)
00047     {
00048         pinMode(numeroLED, OUTPUT);
00049         apagar();
00050     }
00051
00055     void encender() {
00056         digitalWrite(numeroLED, HIGH);
00057         encendido = true;
00058     }
00059
00063     void apagar() {
00064         digitalWrite(numeroLED, LOW);
00065         encendido = false;
00066     }
00067
00073     void alternar() {
00074         if (encendido) {
00075             apagar();
00076         } else {
00077             encender();
00078         }
00079     }
00080
00088     void brillar(long tiempo) {
00089         encender();
00090         esperar(tiempo);
00091         apagar();
00092     }
00093 }; // class LED
00094
00095 #endif // LED_H_INCLUIDO
```

## 7.6. Referencia del archivo HolaMundoIBeacon/Medidor.h

Definición de la clase [Medidor](#) para medir CO2 y temperatura.

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



## Clases

- class [Medidor](#)

*Clase para simular la medición de CO2 y temperatura.*

### 7.6.1. Descripción detallada

Definición de la clase [Medidor](#) para medir CO2 y temperatura.

Esta clase proporciona una interfaz para simular mediciones de CO2 y temperatura en un sistema. Los valores de medición están actualmente codificados.

## 7.7. Medidor.h

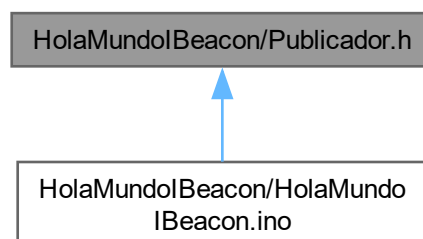
[Ir a la documentación de este archivo.](#)

```
00001 // -*- mode: c++ -*-
00002
00003 #ifndef MEDIDOR_H_INCLUIDO
00004 #define MEDIDOR_H_INCLUIDO
00005
00021 class Medidor {
00022
00023 public:
00024
00030 Medidor() {
00031 }
00032
00039 void iniciarMedidor() {
00040     // Código de inicialización si es necesario en el futuro.
00041 }
00042
00050 int medirCO2() {
00051     return 235;
00052 }
00053
00061 int medirTemperatura() {
00062     return 12;
00063 }
00064
00065 }; // class Medidor
00066
00067 #endif // MEDIDOR_H_INCLUIDO
```

## 7.8. Referencia del archivo HolaMundoIBeacon/Publicador.h

Definición de la clase [Publicador](#) para emitir anuncios BLE de CO2 y temperatura.

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



## Clases

- class `Publicador`

*Clase para publicar mediciones de CO2 y temperatura mediante BLE.*

### 7.8.1. Descripción detallada

Definición de la clase `Publicador` para emitir anuncios BLE de CO2 y temperatura.

Esta clase utiliza una emisora BLE para emitir datos de CO2 y temperatura mediante beacons iBeacon. Los anuncios se envían con un formato específico que incluye un UUID, mayor y menor.

## 7.9. Publicador.h

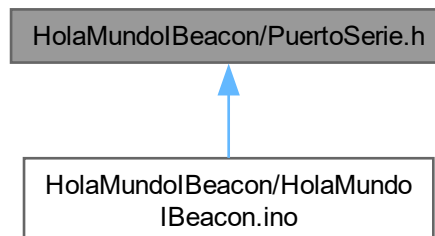
[Ir a la documentación de este archivo.](#)

```
00001 // -*- mode: c++ -*-
00002
00003 #ifndef PUBLICADOR_H_INCLUDEDO
00004 #define PUBLICADOR_H_INCLUDEDO
00005
00021 class Publicador {
00022
00023 private:
00024
00028     uint8_t beaconUUID[16] = {
00029         'E', 'P', 'S', 'G', '-', 'G', 'T', 'I',
00030         '-', 'P', 'R', 'O', 'Y', '-', '3', 'A',
00031     };
00032
00033 public:
00034
00038     EmisoraBLE laEmisora {
00039         "GTI-3A",
00040         0x004c,
00041         4
00042     };
00043
00047     const int RSSI = -53;
00048
00055     enum MedicionesID {
00056         CO2 = 11,
00057         TEMPERATURA = 12,
00058         RUIDO = 13
00059     };
00060
00066     Publicador() {
00067         // No encender la emisora aquí. Debe hacerse en `encenderEmisora()`.
00068     }
00069
00075     void encenderEmisora() {
00076         (*this).laEmisora.encenderEmisora();
00077     }
00078
00088     void publicarCO2( int16_t valorCO2, uint8_t contador, long tiempoEspera ) {
00089         uint16_t major = (MedicionesID::CO2 « 8) + contador;
00090         (*this).laEmisora.emitirAnuncioIBeacon( (*this).beaconUUID,
00091                                                 major,
00092                                                 valorCO2, // minor
00093                                                 (*this).RSSI // rssi
00094                                                 );
00095         esperar( tiempoEspera );
00096         (*this).laEmisora.detenerAnuncio();
00097     }
00098
00108     void publicarTemperatura( int16_t valorTemperatura, uint8_t contador, long tiempoEspera ) {
00109         uint16_t major = (MedicionesID::TEMPERATURA « 8) + contador;
00110         (*this).laEmisora.emitirAnuncioIBeacon( (*this).beaconUUID,
00111                                                 major,
00112                                                 valorTemperatura, // minor
00113                                                 (*this).RSSI // rssi
00114                                                 );
00115         esperar( tiempoEspera );
00116         (*this).laEmisora.detenerAnuncio();
00117     }
00118
00119 }; // class Publicador
00120
00121 #endif // PUBLICADOR_H_INCLUDEDO
```

## 7.10. Referencia del archivo HolaMundoIBeacon/PuertoSerie.h

Definición de la clase [PuertoSerie](#) para la comunicación por puerto serie.

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



### Clases

- class [PuertoSerie](#)

*Clase para manejar la comunicación por puerto serie.*

### 7.10.1. Descripción detallada

Definición de la clase [PuertoSerie](#) para la comunicación por puerto serie.

Esta clase facilita la comunicación a través del puerto serie. Permite inicializar el puerto a una velocidad específica y escribir mensajes.

## 7.11. PuertoSerie.h

[Ir a la documentación de este archivo.](#)

```

00001
00002 // -*- mode: c++ -*-
00003
00004 #ifndef PUERTO_SERIE_H_INCLUDED
00005 #define PUERTO_SERIE_H_INCLUDED
00006
00020 class PuertoSerie {
00021
00022 public:
00023
00031   PuertoSerie (long baudios) {
00032       Serial.begin( baudios );
00033       // Evitar usar el bloqueo de while (!Serial) para permitir un arranque fluido.
00034   }
00035
00041   void esperarDisponible() {
00042       delay(10);
00043   }
00044
00053   template<typename T>
00054   void escribir (T mensaje) {
00055       Serial.print( mensaje );
00056   }
00057
00058 }; // class PuertoSerie
00059
00060 #endif // PUERTO_SERIE_H_INCLUDED
  
```

## 7.12. Referencia del archivo HolaMundoIBeacon/ServicioEnEmisora.h

Definición de la clase [ServicioEnEmisora](#) y Clase que representa una característica dentro de un servicio BLE que gestionan un servicio BLE con características.

```
#include <vector>
```

Gráfico de dependencias incluidas en ServicioEnEmisora.h:

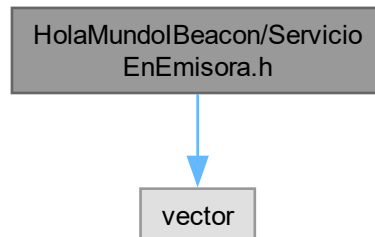
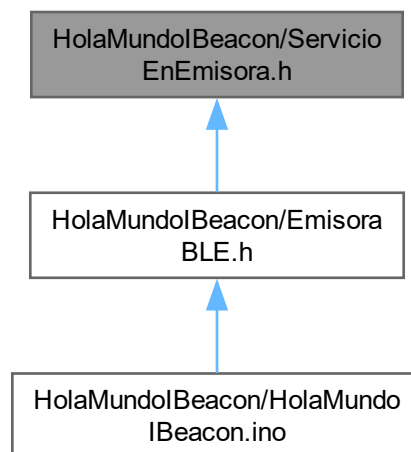


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



### Clases

- class [ServicioEnEmisora](#)  
*Clase que gestiona un servicio BLE con características.*
- class [ServicioEnEmisora::Caracteristica](#)  
*Clase que representa una característica dentro de un servicio BLE.*

## Funciones

- `template<typename T >`  
`T * alReves (T *p, int n)`  
*Función que invierte el contenido de un array.*
- `uint8_t * stringAUint8AlReves (const char *pString, uint8_t *pUint, int tamMax)`  
*Convierte una cadena de caracteres a un array de uint8\_t en orden inverso.*

### 7.12.1. Descripción detallada

Definición de la clase [ServicioEnEmisora](#) y Clase que representa una característica dentro de un servicio BLE que gestionan un servicio BLE con características.

Estas clases representan un servicio en una emisora BLE con soporte para múltiples características.

### 7.12.2. Documentación de funciones

#### 7.12.2.1. alReves()

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

Función que invierte el contenido de un array.

##### Parámetros de plantilla

<i>T</i>	Tipo de los elementos en el array.
----------	------------------------------------

##### Parámetros

<i>p</i>	Puntero al array.
<i>n</i>	Tamaño del array.

##### Devuelve

T\* Puntero al array invertido.

#### 7.12.2.2. stringAUint8AlReves()

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

Convierte una cadena de caracteres a un array de uint8\_t en orden inverso.

## Parámetros

<i>pString</i>	Cadena de entrada.
<i>pUInt</i>	Puntero al array de uint8_t donde se copiará la cadena.
<i>tamMax</i>	Tamaño máximo del array.

## Devuelve

uint8\_t\* Puntero al array resultante.

## 7.13. ServicioEnEmisora.h

[Ir a la documentación de este archivo.](#)

```

00001 // -*- mode: c++ -*-
00002
00003 #ifndef SERVICIO_EMITORA_H_INCLUDEDO
00004 #define SERVICIO_EMITORA_H_INCLUDEDO
00005
00006 #include <vector>
00007
00023 template< typename T >
00024 T * alReves( T * p, int n ) {
00025     T aux;
00026     for( int i = 0; i < n / 2; i++ ) {
00027         aux = p[i];
00028         p[i] = p[n - i - 1];
00029         p[n - i - 1] = aux;
00030     }
00031     return p;
00032 }
00033
00042 uint8_t * stringAUInt8AlReves( const char * pString, uint8_t * pUInt, int tamMax ) {
00043     int longitudString = strlen( pString );
00044     int longitudCopiar = ( longitudString > tamMax ? tamMax : longitudString );
00045     for( int i = 0; i <= longitudCopiar - 1; i++ ) {
00046         pUInt[ tamMax - i - 1 ] = pString[ i ];
00047     }
00048     return pUInt;
00049 }
00050
00057 class ServicioEnEmisora {
00058 public:
00059     using CallbackCaracteristicaEscrita = void (uint16_t conn_handle, BLECharacteristic * chr, uint8_t *
00063     data, uint16_t len);
00064
00069     class Caracteristica {
00070     private:
00071         uint8_t uuidCaracteristica[16] = {
00072             '0', '1', '2', '3', '4', '5', '6', '7',
00073             '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'
00074         };
00075         BLECharacteristic laCaracteristica;
00076     public:
00077         Caracteristica(const char * nombreCaracteristica_)
00083             : laCaracteristica(stringAUInt8AlReves( nombreCaracteristica_, &uuidCaracteristica[0], 16)) {}
00084
00085         Caracteristica(const char * nombreCaracteristica_, uint8_t props, SecureMode_t permisoRead,
00095         SecureMode_t permisoWrite, uint8_t tam)
00096             : Caracteristica(nombreCaracteristica_) {
00097             asignarPropiedadesPermisosYTamanyoDatos(props, permisoRead, permisoWrite, tam);
00098         }
00099     private:
00100         void asignarPropiedades(uint8_t props) {
00101             laCaracteristica.setProperties(props);
00102         }
00103
00104         void asignarPermisos(SecureMode_t permisoRead, SecureMode_t permisoWrite) {
00105             laCaracteristica.setPermission(permisoRead, permisoWrite);
00106         }
00107
00108         void asignarTamanyoDatos(uint8_t tam) {
00109             laCaracteristica.setMaxLen(tam);
00110

```

```

00111     }
00112
00113     public:
00117         void asignarPropiedadesPermisosYTamanyoDatos(uint8_t props, SecureMode_t permisoRead, SecureMode_t
permisoWrite, uint8_t tam) {
00118             asignarPropiedades(props);
00119             asignarPermisos(permisoRead, permisoWrite);
00120             asignarTamanyoDatos(tam);
00121         }
00122
00129         uint16_t escribirDatos(const char * str) {
00130             return laCaracteristica.write(str);
00131         }
00132
00139         uint16_t notificarDatos(const char * str) {
00140             return laCaracteristica.notify(&str[0]);
00141         }
00142
00146         void instalarCallbackCaracteristicaEscrita(CallbackCaracteristicaEscrita cb) {
00147             laCaracteristica.setWriteCallback(cb);
00148         }
00149
00153         void activar() {
00154             err_t error = laCaracteristica.begin();
00155             Globales::elPuerto.escribir("laCaracteristica.begin(); error = ");
00156             Globales::elPuerto.escribir(error);
00157         }
00158     };
00159
00160 private:
00161     uint8_t uuidServicio[16] = {
00162         '0', '1', '2', '3', '4', '5', '6', '7',
00163         '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'
00164     };
00165     BLEService elServicio;
00166     std::vector<Caracteristica *> lasCaracteristicas;
00167
00168 public:
00174     ServicioEnEmisora(const char * nombreServicio_)
00175         : elServicio(stringAUInt8AlReves(nombreServicio_, &uuidServicio[0], 16)) {}
00176
00180     void escribeUUID() {
00181         Serial.println("*****");
00182         for (int i = 0; i <= 15; i++) {
00183             Serial.print((char) uuidServicio[i]);
00184         }
00185         Serial.println("\n*****");
00186     }
00187
00193     void anyadirCaracteristica(Caracteristica & car) {
00194         lasCaracteristicas.push_back(&car);
00195     }
00196
00200     void activarServicio() {
00201         err_t error = elServicio.begin();
00202         Serial.print("elServicio.begin(); error = ");
00203         Serial.println(error);
00204         for (auto pCar : lasCaracteristicas) {
00205             pCar->activar();
00206         }
00207     }
00208
00214     operator BLEService&() {
00215         return elServicio;
00216     }
00217 };
00218
00219 #endif // SERVICIO_EMITORA_H_INCLUIDO

```

## 7.14. Referencia del archivo README.md