Proyecto_BioM_Arduino

Generado por Doxygen 1.12.0

Capítulo 1

Índice de espacios de nombres

1.1. Lista de espacios de nombres

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

Capítulo 2

Índice de clases

2.1. Lista de clases

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

Capítulo 3

Índice de archivos

3.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

Capítulo 4

Documentación de espacios de nombres

4.1. Referencia del espacio de nombres Globales

Variables

- LED elLED (7)

  Inicializa el LED en el pin 7.
- PuertoSerie elPuerto (115200)

  Inicializa el puerto serie a 115200 baudios.
- Publicador elPublicador

  Inicializa el objeto Publicador.
- Medidor elMedidor

  Inicializa el objeto Medidor.

4.1.1. Documentación de variables

4.1.1.1. elLED

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

Inicializa el LED en el pin 7.

4.1.1.2. elMedidor

```
Medidor Globales::elMedidor
```

Inicializa el objeto Medidor.

4.1.1.3. elPublicador

```
Publicador Globales::elPublicador
```

Inicializa el objeto Publicador.

4.1.1.4. elPuerto

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

Inicializa el puerto serie a 115200 baudios.

4.2. Referencia del espacio de nombres Loop

Variables

- uint8_t cont = 0

4.2.1. Documentación de variables

4.2.1.1. cont

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

Capítulo 5

Documentación de clases

5.1. Referencia de la clase EmisoraBLE

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

`#include <EmisoraBLE.h>`

Tipos públicos

- using CallbackConexionEstablecida = void (uint16_t connHandle)

  Tipo de callback para la conexión establecida.
- using CallbackConexionTerminada = void (uint16_t connHandle, uint8_t reason)

  Tipo de callback para la conexión terminada.

Métodos públicos

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

  Constructor de la clase EmisoraBLE.
- void encenderEmisora ()

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

  Enciende la emisora y establece callbacks de conexión.
- void detenerAnuncio ()

  Detiene el anuncio si está activo.
- bool estaAnunciando ()

  Verifica si la emisora está anunciando.
- void emitirAnuncioIBeacon (uint8_t ∗beaconUUID, int16_t major, int16_t minor, uint8_t rssi)

  Emite un anuncio iBeacon.
- void emitirAnuncioIBeaconLibre (const char ∗carga, const uint8_t tamanyoCarga)

  Emite un anuncio iBeacon con carga libre.
- void instalarCallbackConexionEstablecida (CallbackConexionEstablecida cbce)

  Instala el callback para la conexión establecida.
- void instalarCallbackConexionTerminada (CallbackConexionTerminada cbct)

  Instala el callback para la conexión terminada.

Atributos privados

- const char * nombreEmisora

  Nombre de la emisora.
- const uint16_t fabricanteID

  ID del fabricante.
- const int8_t txPower

  Potencia de transmisión.

5.1.1. Descripción detallada

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

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

5.1.2.1. CallbackConexionEstablecida

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

Tipo de callback para la conexión establecida.

Parámetros

connHandle	Manejador de conexión.

5.1.2.2. CallbackConexionTerminada

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

Tipo de callback para la conexión terminada.

Parámetros

connHandle	Manejador de conexión.
reason	Razón de la desconexión.

5.1.3. Documentación de constructores y destructores

5.1.3.1. EmisoraBLE()

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

Constructor de la clase EmisoraBLE.

Parámetros

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

5.1.4. Documentación de funciones miembro

5.1.4.1. detenerAnuncio()

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

Detiene el anuncio si está activo.

Gráfico de llamadas a esta función:

5.1.4.2. emitirAnuncioIBeacon()

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

Emite un anuncio iBeacon.

Parámetros

beaconUUID	UUID del beacon.
major	Número mayor del beacon.
minor	Número menor del beacon.
rssi	RSSI del beacon.

5.1.4.3. emitirAnuncioIBeaconLibre()

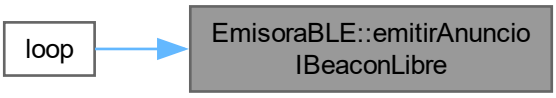
```
void EmisoraBLE::emitirAnuncioIBeaconLibre (
            const char * carga,
            const uint8_t tamanyoCarga)  [inline]
```

Emite un anuncio iBeacon con carga libre.

Parámetros

carga	Carga a enviar.
tamanyoCarga	Tamaño de la carga.

Gráfico de llamadas a esta función:



5.1.4.4. encenderEmisora() [1/2]

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

Enciende la emisora.

Gráfico de llamadas a esta función:

5.1.4.5. encenderEmisora() [2/2]

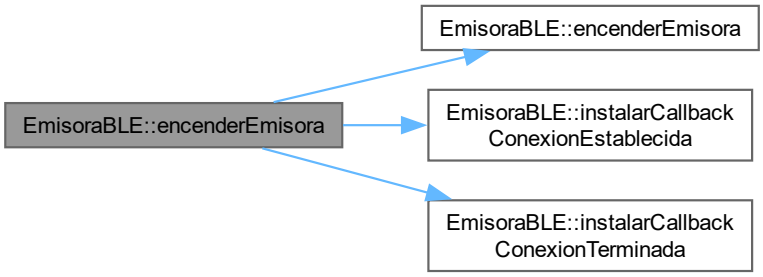
```
void EmisoraBLE::encenderEmisora (
            CallbackConexionEstablecida cbce,
            CallbackConexionTerminada cbct)  [inline]
```

Enciende la emisora y establece callbacks de conexión.

Parámetros

cbce	Callback para conexión establecida.
cbct	Callback para conexión terminada.

Gráfico de llamadas de esta función:



5.1.4.6. estaAnunciando()

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

Verifica si la emisora está anunciando.

Devuelve

true si está anunciando, false en caso contrario.

5.1.4.7. instalarCallbackConexionEstablecida()

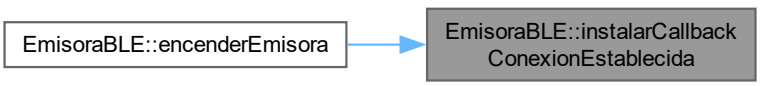
```
void EmisoraBLE::instalarCallbackConexionEstablecida (
            CallbackConexionEstablecida cbce)  [inline]
```

Instala el callback para la conexión establecida.

Parámetros

cbce	Callback de conexión establecida.

Gráfico de llamadas a esta función:



5.1.4.8. instalarCallbackConexionTerminada()

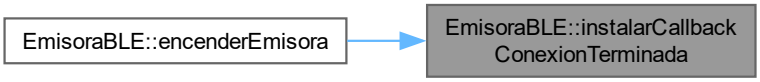
```
void EmisoraBLE::instalarCallbackConexionTerminada (
            CallbackConexionTerminada cbct)  [inline]
```

Instala el callback para la conexión terminada.

Parámetros

cbct	Callback de conexión terminada.

Gráfico de llamadas a esta función:



5.1.5. Documentación de datos miembro

5.1.5.1. fabricanteID

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

ID del fabricante.

5.1.5.2. nombreEmisora

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

Nombre de la emisora.

5.1.5.3. txPower

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

Potencia de transmisión.

La documentación de esta clase está generada del siguiente archivo:

- HolaMundoIBeacon/EmisoraBLE.h

5.2. Referencia de la clase LED

Clase para controlar un LED.

`#include <LED.h>`

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 (encendido a apagado o viceversa).
- void brillar (long tiempo)
  Hace brillar el LED durante un tiempo específico.

Atributos privados

- int numeroLED
  Número del pin al que está conectado el LED.
- bool encendido
  Estado del LED (encendido o apagado).

5.2.1. Descripción detallada

Clase para controlar un LED.

5.2.2. Documentación de constructores y destructores

5.2.2.1. LED()

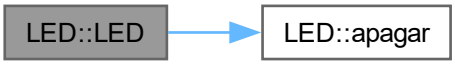
```
LED::LED (
            int numero)  [inline]
```

Constructor de la clase LED.

Parámetros

numero	Número del pin donde se conecta el LED.

Gráfico de llamadas de esta función:



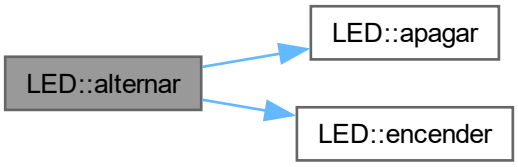
5.2.3. Documentación de funciones miembro

5.2.3.1. alternar()

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

Alterna el estado del LED (encendido a apagado o viceversa).

Gráfico de llamadas de esta función:

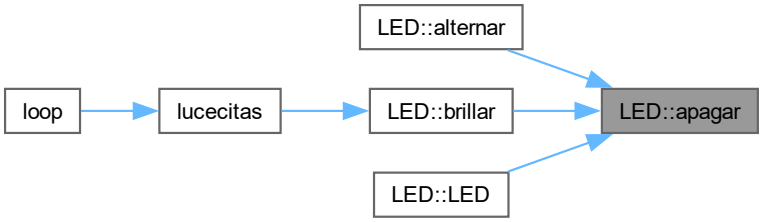


5.2.3.2. apagar()

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

Apaga el LED.

Gráfico de llamadas a esta función:



5.2.3.3. brillar()

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

Hace brillar el LED durante un tiempo específico.

Parámetros

tiempo	Tiempo en milisegundos para el cual el LED estará encendido.

Gráfico de llamadas de esta función:

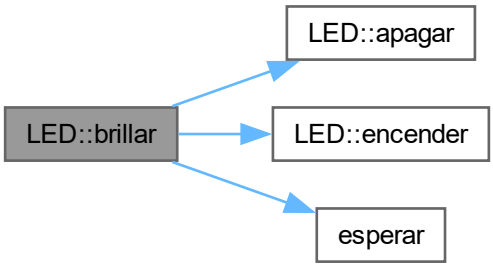


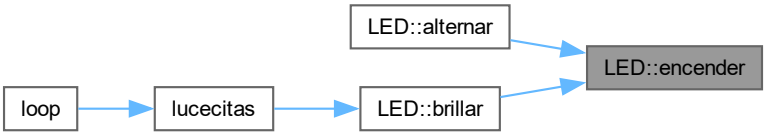
Gráfico de llamadas a esta función:

5.2.3.4. encender()

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

Enciende el LED.

Gráfico de llamadas a esta función:



5.2.4. Documentación de datos miembro

5.2.4.1. encendido

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

Estado del LED (encendido o apagado).

5.2.4.2. 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

5.3. Referencia de la clase Medidor

Clase para medir CO2 y temperatura.

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

Métodos públicos

- Medidor ()
  Constructor de la clase Medidor.
- void iniciarMedidor ()
  Inicializa el medidor.
- int medirCO2 ()
  Mide la concentración de CO2.
- int medirTemperatura ()
  Mide la temperatura.

5.3.1. Descripción detallada

Clase para medir CO2 y temperatura.

5.3.2. Documentación de constructores y destructores

5.3.2.1. Medidor()

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

Constructor de la clase Medidor.

Inicializa un objeto de la clase Medidor. Se pueden agregar inicializaciones adicionales si es necesario.

5.3.3. Documentación de funciones miembro

5.3.3.1. iniciarMedidor()

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

Inicializa el medidor.

Realiza las configuraciones necesarias que no se puedan hacer en el constructor. Gráfico de llamadas a esta función:

5.3.3.2. medirCO2()

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

Mide la concentración de CO2.

Devuelve

Devuelve un valor entero que representa la concentración de CO2 medida (en partes por millón, ppm).

Gráfico de llamadas a esta función:

5.3.3.3. medirTemperatura()

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

Mide la temperatura.

Devuelve

Devuelve un valor entero que representa la temperatura medida (en grados Celsius).

Gráfico de llamadas a esta función:

La documentación de esta clase está generada del siguiente archivo:

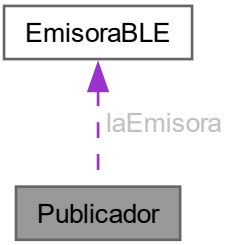
- HolaMundoIBeacon/Medidor.h

5.4. Referencia de la clase Publicador

Clase para publicar datos de sensores a través de BLE.

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

Diagrama de colaboración de Publicador:



Tipos públicos

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

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 el valor de CO2 mediante un anuncio IBeacon.
- void publicarTemperatura (int16_t valorTemperatura, uint8_t contador, long tiempoEspera)

  Publica el valor de temperatura mediante un anuncio IBeacon.

Atributos públicos

- EmisoraBLE laEmisora
- const int RSSI = -53

Atributos privados

- uint8_t beaconUUID [16]

5.4.1. Descripción detallada

Clase para publicar datos de sensores a través de BLE.

5.4.2. Documentación de las enumeraciones miembro de la clase

5.4.2.1. MedicionesID

```
enum Publicador::MedicionesID
```

Identificadores de las mediciones.

Valores de enumeraciones

CO2	ID para medir CO2.
TEMPERATURA	ID para medir temperatura.
RUIDO	ID para medir ruido.

5.4.3. Documentación de constructores y destructores

5.4.3.1. Publicador()

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

Constructor de la clase Publicador.

Inicializa un objeto de la clase Publicador. Se sugiere encender la emisora desde el método setup() en lugar de hacerlo aquí.

5.4.4. Documentación de funciones miembro

5.4.4.1. encenderEmisora()

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

Enciende la emisora BLE.

Llama al método de la emisora para encenderla. Gráfico de llamadas a esta función:

5.4.4.2. publicarCO2()

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

Publica el valor de CO2 mediante un anuncio IBeacon.

Parámetros

valorCO2	Valor de CO2 a publicar (en ppm).
contador	Contador para el ID mayor del IBeacon.
tiempoEspera	Tiempo en milisegundos a esperar antes de detener el anuncio.

Gráfico de llamadas de esta función:

Gráfico de llamadas a esta función:

5.4.4.3. publicarTemperatura()

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

Publica el valor de temperatura mediante un anuncio IBeacon.

Parámetros

valorTemperatura	Valor de temperatura a publicar (en grados Celsius).
contador	Contador para el ID mayor del IBeacon.
tiempoEspera	Tiempo en milisegundos a esperar antes de detener el anuncio.

Gráfico de llamadas de esta función:

Gráfico de llamadas a esta función:

5.4.5. Documentación de datos miembro

5.4.5.1. 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.

5.4.5.2. laEmisora

```
EmisoraBLE Publicador::laEmisora
```

Valor inicial:

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

Instancia de EmisoraBLE para la emisión de anuncios.

5.4.5.3. RSSI

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

La documentación de esta clase está generada del siguiente archivo:

- HolaMundoIBeacon/Publicador.h

5.5. Referencia de la clase PuertoSerie

Clase para manejar la comunicación serie.

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

Métodos públicos

- PuertoSerie (long baudios)

  Constructor de la clase PuertoSerie.
- void esperarDisponible ()

  Espera un breve momento para asegurar que el puerto serie esté disponible.
- template<typename T >
  void escribir (T mensaje)

  Escribe un mensaje en el puerto serie.

5.5.1. Descripción detallada

Clase para manejar la comunicación serie.

5.5.2. Documentación de constructores y destructores

5.5.2.1. PuertoSerie()

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

Constructor de la clase PuertoSerie.

Parámetros

baudios	Velocidad de comunicación en baudios.

Inicializa la comunicación serie con la velocidad especificada.

5.5.3. Documentación de funciones miembro

5.5.3.1. escribir()

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

Escribe un mensaje en el puerto serie.

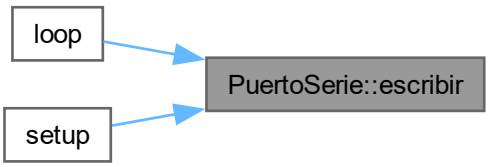
Parámetros de plantilla

T	Tipo del mensaje a enviar.

Parámetros

mensaje	El mensaje que se desea enviar al puerto serie.

Utiliza la función Serial.print para enviar el mensaje. Gráfico de llamadas a esta función:



5.5.3.2. esperarDisponible()

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

Espera un breve momento para asegurar que el puerto serie esté disponible.

Esta función introduce un retraso de 10 milisegundos. Gráfico de llamadas a esta función:

La documentación de esta clase está generada del siguiente archivo:

- HolaMundoIBeacon/PuertoSerie.h

5.6. Referencia de la clase ServicioEnEmisora

Clase que representa un servicio en una emisora BLE.

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

5.6.1. Descripción detallada

Clase que representa un servicio en una emisora BLE.

La documentación de esta clase está generada del siguiente archivo:

- HolaMundoIBeacon/ServicioEnEmisora.h

Capítulo 6

Documentación de archivos

6.1. Referencia del archivo HolaMundoIBeacon/EmisoraBLE.h

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

Gráfico de dependencias incluidas en EmisoraBLE.h:

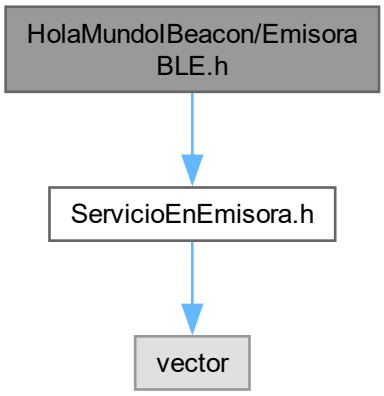
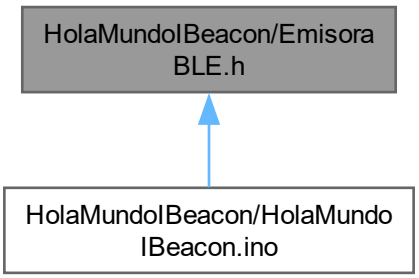


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class EmisoraBLE

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

6.2. EmisoraBLE.h

Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002 //
00003 // ---------------------------------------------------------
00004 // Jordi Bataller i Mascarell
00005 // 2019-07-07
00006 // ---------------------------------------------------------
00007 #ifndef EMISORA_H_INCLUIDO
00008 #define EMISORA_H_INCLUIDO
00009
00010 // Buena introducción: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap
00011 // https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/
00012
00013 // fuente: https://www.instructables.com/id/Beaconeddystone-and-Adafruit-NRF52-Advertise-Your-/
00014 // https://github.com/nkolban/ESP32_BLE_Arduino/blob/master/src/BLEBeacon.h
00015
00016 // https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/
00017 // https://learn.adafruit.com/bluefruit-nrf52-feather-learning-guide/bleadvertising
00018
00019 // ---------------------------------------------------------
00020 // ---------------------------------------------------------
00021 #include "ServicioEnEmisora.h"
00022
00023 // ---------------------------------------------------------
00024 // ---------------------------------------------------------
00029 class EmisoraBLE {
00030 private:
00031   const char * nombreEmisora;
00032   const uint16_t fabricanteID;
00033   const int8_t txPower;
00034
00035 public:
00036
00037   // .....................................................
00038   // .....................................................
00043   using CallbackConexionEstablecida = void ( uint16_t connHandle );
00044
00050   using CallbackConexionTerminada = void ( uint16_t connHandle, uint8_t reason);
00051
00052   // .....................................................
00053   // .....................................................
00060   EmisoraBLE( const char * nombreEmisora_, const uint16_t fabricanteID_,
```

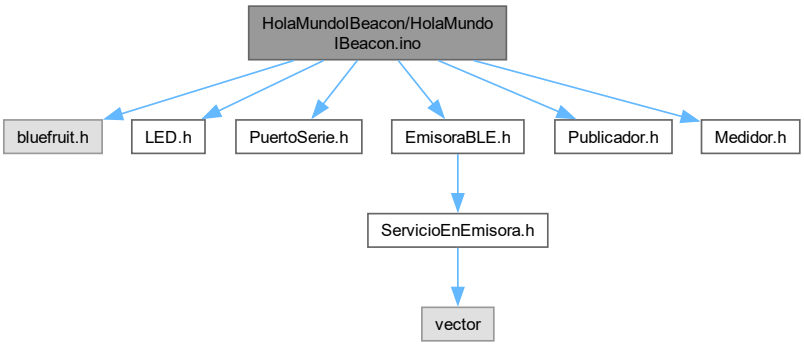
```
00061               const int8_t txPower_ )
00062     :
00063     nombreEmisora( nombreEmisora_ ) ,
00064     fabricanteID( fabricanteID_ ) ,
00065     txPower( txPower_ )
00066   {
00067     // no encender ahora la emisora, tal vez sea por el println()
00068     // que hace que todo falle si lo llamo en el contructor
00069     // ( = antes que configuremos Serial )
00070     // No parece que sea por el println,
00071     // por tanto NO_encenderEmisora();
00072   } // ()
00073
00074   // .....................................................
00075   // .....................................................
00079   void encenderEmisora() {
00080     // Serial.println ( "Bluefruit.begin() " );
00081     Bluefruit.begin();
00082
00083     // por si acaso:
00084     (*this).detenerAnuncio();
00085   } // ()
00086
00087   // .....................................................
00088   // .....................................................
00094   void encenderEmisora( CallbackConexionEstablecida cbce,
00095                         CallbackConexionTerminada cbct ) {
00096     encenderEmisora();
00097     instalarCallbackConexionEstablecida( cbce );
00098     instalarCallbackConexionTerminada( cbct );
00099   } // ()
00100
00101   // .....................................................
00102   // .....................................................
00106   void detenerAnuncio() {
00107     if ( (*this).estaAnunciando() ) {
00108       // Serial.println ( "Bluefruit.Advertising.stop() " );
00109       Bluefruit.Advertising.stop();
00110     }
00111   }  // ()
00112
00113   // .....................................................
00114   // estaAnunciando() -> Boleano
00115   // .....................................................
00120   bool estaAnunciando() {
00121     return Bluefruit.Advertising.isRunning();
00122   } // ()
00123
00124   // .....................................................
00125   // .....................................................
00133   void emitirAnuncioIBeacon( uint8_t * beaconUUID, int16_t major, int16_t minor, uint8_t rssi ) {
00134     (*this).detenerAnuncio();
00135
00136     // creo el beacon
00137     BLEBeacon elBeacon( beaconUUID, major, minor, rssi );
00138     elBeacon.setManufacturer( (*this).fabricanteID );
00139
00140     // parece que esto debe ponerse todo aquí
00141     Bluefruit.setTxPower( (*this).txPower );
00142     Bluefruit.setName( (*this).nombreEmisora );
00143     Bluefruit.ScanResponse.addName(); // para que envíe el nombre de emisora (?!)
00144
00145     // pongo el beacon
00146     Bluefruit.Advertising.setBeacon( elBeacon );
00147
00148     // ? qué valorers poner aquí
00149     Bluefruit.Advertising.restartOnDisconnect(true); // no hace falta, pero lo pongo
00150     Bluefruit.Advertising.setInterval(100, 100);    // en unidad de 0.625 ms
00151
00152     // empieza el anuncio, 0 = tiempo indefinido (ya lo pararán)
00153     Bluefruit.Advertising.start( 0 );
00154   } // ()
00155
00156   // .....................................................
00157   //
00158   // Ejemplo de Beacon (31 bytes)
00159   //
00160   // https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/
00161   //
00162   // El prefijo de iBeacon contiene los datos hexadecimales: 0x0201061AFF004C0215.
00163   // Se desglosa como sigue:
00164   //
00165   // 0x020106 define el paquete de publicidad como BLE General Discoverable
00166   // y BR/EDR incompatible de alta velocidad.
00167   // Efectivamente dice que esto solo está transmitiendo, no conectando.
00168   //
00169   // 0x1AFF dice que los siguientes datos son de longitud 26 bytes y son datos específicos del
```

```
      fabricante.
00170   //
00171   // 0x004C es el ID de Bluetooth Sig de Apple y es parte de esta especificación que lo hace
      dependiente de Apple.
00172   //
00173   // 0x02 es un ID secundario que denota un beacon de proximidad, que es utilizado por todos los
      iBeacons.
00174   //
00175   // 0x15 define la longitud restante como 21 bytes (16+2+2+1).
00176   //
00177   // Ejemplo:
00178   //
00179   // 1. prefijo: 9bytes
00180   //       0x02, 0x01, 0x06,       // advFlags 3bytes
00181   //       0x1a, 0xff,             // advHeader 2 (0x1a = 26 = 25(longitud de 0x4c a 0xca)+1) 0xFF ->
      BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA
00182   //       0x4c, 0x00,             // companyID 2bytes
00183   //       0x02,                   // ibeacon type 1 byte
00184   //       0x15,                   // ibeacon length 1 byte (dec=21 lo que va a continuación: desde la
      'f' hasta 0x01)
00185   //
00186   // 2. uuid: 16bytes
00187   // 'f', 'i', 's', 't', 'r', 'o', 'f', 'i', 's', 't', 'r', 'o', 0xa7, 0x10, 0x96, 0xe0
00188   //
00189   // 2 major: 2bytes
00190   // 0x04, 0xd2,
00191   //
00192   // minor: 2bytes
00193   // 0x10, 0xe1,
00194   //
00195   // 0xca, // tx power : 1bytes
00196   //
00197   // 0x01, // este es el byte 31 = BLE_GAP_ADV_SET_DATA_SIZE_MAX, parece que sobra
00198   //
00199   // .....................................................
00200   // Para enviar como carga libre los últimos 21 bytes de un iBeacon (lo que normalmente sería uuid-16
      major-2 minor-2 txPower-1)
00201   // .....................................................
00202   /*
00203   void emitirAnuncioIBeaconLibre( const char * carga ) {
00204
00205     const uint8_t tamanyoCarga = strlen( carga );
00206   */
00212   void emitirAnuncioIBeaconLibre( const char * carga, const uint8_t tamanyoCarga ) {
00213     (*this).detenerAnuncio();
00214
00215     Bluefruit.Advertising.clearData();
00216     Bluefruit.ScanResponse.clearData(); // Elimina datos de respuesta.
00217
00218     // Establece el tamaño de carga en bytes (debe ser menor o igual a 21).
00219     if ( tamanyoCarga <= 21 ) {
00220       // Serial.print ( "carga: " );
00221       // Serial.println ( carga );
00222       Bluefruit.Advertising.addData( carga, tamanyoCarga ); // Carga
00223     }
00224
00225     Bluefruit.Advertising.restartOnDisconnect(true); // Si hay desconexión, se reinicia.
00226     Bluefruit.Advertising.setInterval(100, 100); // Establece intervalo de anuncio.
00227     Bluefruit.Advertising.start( 0 ); // Inicia el anuncio indefinidamente.
00228   } // ()
00229
00230   // .....................................................
00231   // Instalador de callbacks de conexión establecida
00232   // .....................................................
00237   void instalarCallbackConexionEstablecida( CallbackConexionEstablecida cbce ) {
00238     // callback para conexiones:
00239     Bluefruit.Connection.setConnCallback( cbce );
00240   } // ()
00241
00242   // .....................................................
00243   // Instalador de callbacks de conexión terminada
00244   // .....................................................
00249   void instalarCallbackConexionTerminada( CallbackConexionTerminada cbct ) {
00250     Bluefruit.Connection.setDisconnCallback( cbct );
00251   } // ()
00252 }; // class EmisoraBLE
00253
00254 #endif // EMISORA_H_INCLUIDO
```

6.3. Referencia del archivo HolaMundoIBeacon/HolaMundoIBeacon.ino

```
#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
- namespace Loop

Funciones

- void inicializarPlaquita ()

  Inicializa los componentes de la placa.
- void setup ()

  Configuración inicial del programa.
- void lucecitas ()

  Parpadeo del LED.
- void loop ()

  Ciclo principal del programa.

Variables

- LED Globales::elLED (7)

  Inicializa el LED en el pin 7.
- PuertoSerie Globales::elPuerto (115200)

  Inicializa el puerto serie a 115200 baudios.
- Publicador Globales::elPublicador

  Inicializa el objeto Publicador.
- Medidor Globales::elMedidor

  Inicializa el objeto Medidor.
- uint8_t Loop::cont = 0

6.3.1. Documentación de funciones

6.3.1.1. inicializarPlaquita()

```
void inicializarPlaquita ()
```

Inicializa los componentes de la placa.

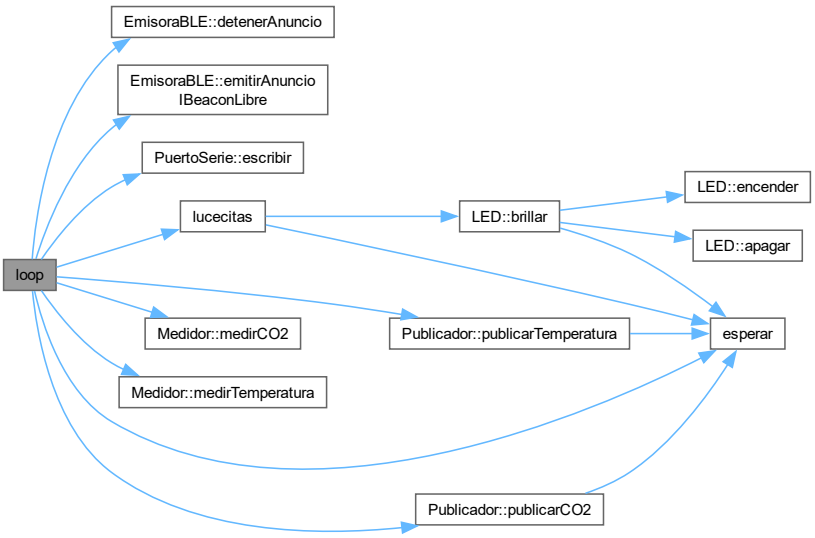
Esta función se utiliza para preparar los componentes que se usarán en el programa, aunque actualmente no realiza ninguna acción. Gráfico de llamadas a esta función:

6.3.1.2. loop()

```
void loop ()
```

Ciclo principal del programa.

Esta función se ejecuta repetidamente mientras el programa está en marcha. Se encarga de medir los valores de CO2 y temperatura, publicarlos, y manejar la emisión de iBeacon. Gráfico de llamadas de esta función:



6.3.1.3. lucecitas()

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

Parpadeo del LED.

Esta función hace que el LED parpadee en un patrón específico. Gráfico de llamadas de esta función:

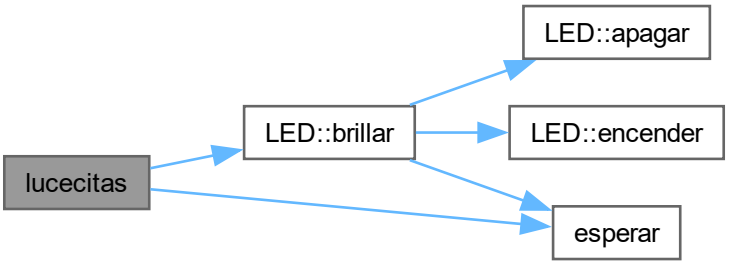
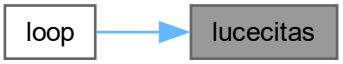


Gráfico de llamadas a esta función:



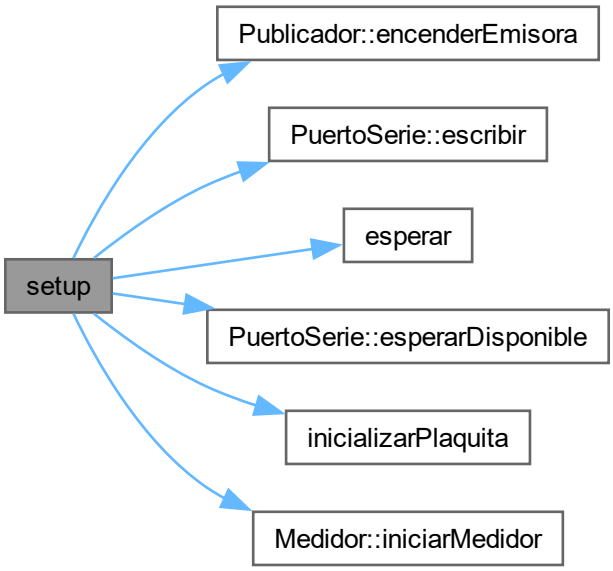
6.3.1.4. setup()

```
void setup ()
```

Configuración inicial del programa.

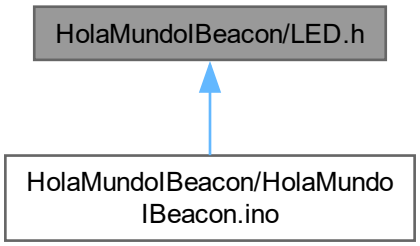
Esta función se ejecuta una vez al inicio del programa. Se encarga de esperar a que el puerto serie esté disponible, inicializa la placa, activa la emisora BLE y comienza la medición de los valores. Gráfico de llamadas de esta

función:



6.4. Referencia del archivo HolaMundoIBeacon/LED.h

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



Clases

- class LED

  Clase para controlar un LED.

Funciones

- void esperar (long tiempo)

  Pausa la ejecución durante un tiempo especificado.

6.4.1. Documentación de funciones

6.4.1.1. esperar()

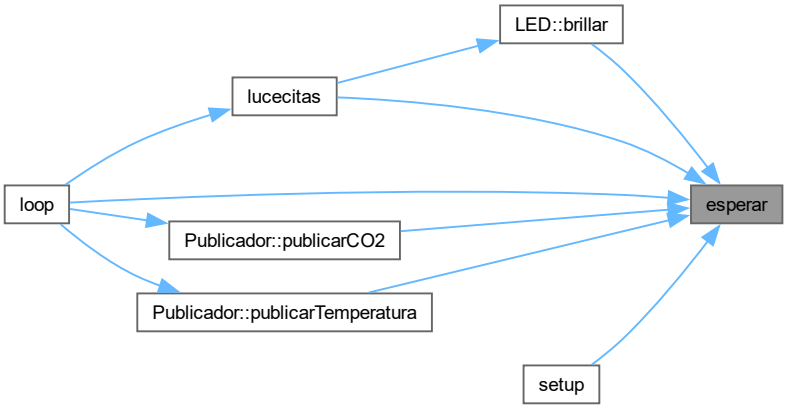
```
void esperar (
            long tiempo)
```

Pausa la ejecución durante un tiempo especificado.

Parámetros

tiempo	Tiempo en milisegundos para esperar.

Gráfico de llamadas a esta función:



6.5. LED.h

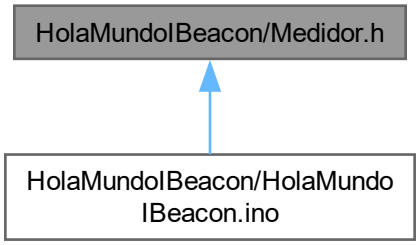
Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00003 #ifndef LED_H_INCLUIDO
00004 #define LED_H_INCLUIDO
00005
00006 // ---------------------------------------------------------
00007 // Jordi Bataller i Mascarell
00008 // 2019-07-07
00009 // ---------------------------------------------------------
00010
00011 // ---------------------------------------------------------
00012 // ---------------------------------------------------------
00017 void esperar(long tiempo) {
00018   delay(tiempo);
00019 }
```

```
00020
00021 // ----------------------------------------------------------
00022 // ----------------------------------------------------------
00027 class LED {
00028 private:
00029   int numeroLED;
00030   bool encendido;
00031
00032 public:
00033
00034   // ....................................................
00035   // ....................................................
00040   LED(int numero)
00041     : numeroLED(numero), encendido(false) {
00042     pinMode(numeroLED, OUTPUT); // Configura el pin como salida.
00043     apagar(); // Asegura que el LED esté apagado al iniciar.
00044   }
00045
00046   // ....................................................
00047   // ....................................................
00051   void encender() {
00052     digitalWrite(numeroLED, HIGH); // Establece el pin en alto.
00053     encendido = true; // Cambia el estado a encendido.
00054   }
00055
00056   // ....................................................
00057   // ....................................................
00061   void apagar() {
00062     digitalWrite(numeroLED, LOW); // Establece el pin en bajo.
00063     encendido = false; // Cambia el estado a apagado.
00064   }
00065
00066   // ....................................................
00067   // ....................................................
00071   void alternar() {
00072     if (encendido) {
00073       apagar(); // Si está encendido, apágalo.
00074     } else {
00075       encender(); // Si está apagado, enciéndelo.
00076     }
00077   } // ()
00078
00079   // ....................................................
00080   // ....................................................
00085   void brillar(long tiempo) {
00086     encender(); // Enciende el LED.
00087     esperar(tiempo); // Espera durante el tiempo especificado.
00088     apagar(); // Apaga el LED después de la espera.
00089   }
00090 }; // class
00091
00092 // ----------------------------------------------------------
00093 // ----------------------------------------------------------
00094 // ----------------------------------------------------------
00095 // ----------------------------------------------------------
00096 #endif // LED_H_INCLUIDO
```

6.6. Referencia del archivo HolaMundoIBeacon/Medidor.h

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



Clases

- class Medidor

  Clase para medir CO2 y temperatura.

6.7. Medidor.h

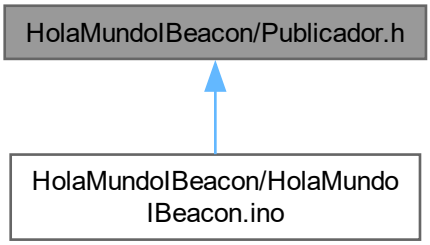
Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00003 #ifndef MEDIDOR_H_INCLUIDO
00004 #define MEDIDOR_H_INCLUIDO
00005
00006 // ------------------------------------------------------
00007 // ------------------------------------------------------
00012 class Medidor {
00013
00014 private:
00015   // Variables privadas pueden declararse aquí si es necesario.
00016
00017 public:
00018
00019   // ....................................................
00020   // constructor
00021   // ....................................................
00028   Medidor() {
00029   } // ()
00030
00031   // ....................................................
00032   // ....................................................
00039   void iniciarMedidor() {
00040     // Las cosas que no se puedan hacer en el constructor, if any
00041   } // ()
00042
00043   // ....................................................
00044   // ....................................................
00050   int medirCO2() {
00051     return 235; // Simulación de un valor de CO2.
00052   } // ()
00053
00054   // ....................................................
00055   // ....................................................
00061   int medirTemperatura() {
00062     return 12; // Qué frío!
00063   } // ()
00064
```

```
00065 }; // class
00066
00067 // -----------------------------------------------------
00068 // -----------------------------------------------------
00069 // -----------------------------------------------------
00070 // -----------------------------------------------------
00071 #endif // MEDIDOR_H_INCLUIDO
```

6.8. Referencia del archivo HolaMundoIBeacon/Publicador.h

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



Clases

- class Publicador

  Clase para publicar datos de sensores a través de BLE.

6.9. Publicador.h

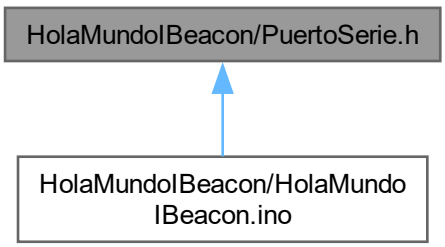
Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00003 // ---------------------------------------------------------------
00004 // Jordi Bataller i Mascarell
00005 // ---------------------------------------------------------------
00006
00007 #ifndef PUBLICADOR_H_INCLUIDO
00008 #define PUBLICADOR_H_INCLUIDO
00009
00010 // ---------------------------------------------------------------
00011 // ---------------------------------------------------------------
00016 class Publicador {
00017
00018 private:
00019
00021   uint8_t beaconUUID[16] = {
00022     'E', 'P', 'S', 'G', '-', 'G', 'T', 'I',
00023     '-', 'P', 'R', 'O', 'Y', '-', '3', 'A'
00024   };
00025
00026 public:
00028   EmisoraBLE laEmisora {
00029     "GTI-3A", // Nombre de la emisora
00030     0x004c,   // ID del fabricante (Apple)
00031     4          // Potencia de transmisión (txPower)
00032   };
```

```
00033
00034   const int RSSI = -53; // Valor de RSSI (Received Signal Strength Indicator)
00035
00036 public:
00037
00039   enum MedicionesID  {
00040     CO2 = 11,
00041     TEMPERATURA = 12,
00042     RUIDO = 13
00043   };
00044
00045   // ---------------------------------------------------------
00046   // constructor
00047   // ---------------------------------------------------------
00055   Publicador() {
00056     // ATENCION: no hacerlo aquí. (*this).laEmisora.encenderEmisora();
00057     // Pondremos un método para llamarlo desde el setup() más tarde
00058   } // ()
00059
00060   // ---------------------------------------------------------
00061   // ---------------------------------------------------------
00067   void encenderEmisora() {
00068     (*this).laEmisora.encenderEmisora();
00069   } // ()
00070
00071   // ---------------------------------------------------------
00072   // ---------------------------------------------------------
00080   void publicarCO2(int16_t valorCO2, uint8_t contador, long tiempoEspera) {
00081     // 1. empezamos anuncio
00082     uint16_t major = (MedicionesID::CO2 « 8) + contador;
00083     (*this).laEmisora.emitirAnuncioIBeacon((*this).beaconUUID,
00084                                            major,
00085                                            valorCO2, // minor
00086                                            (*this).RSSI // rssi
00087     );
00088
00089     /*
00090     Globales::elPuerto.escribir( "   publicarCO2(): valor=" );
00091     Globales::elPuerto.escribir( valorCO2 );
00092     Globales::elPuerto.escribir( "   contador=" );
00093     Globales::elPuerto.escribir( contador );
00094     Globales::elPuerto.escribir( "   todo="  );
00095     Globales::elPuerto.escribir( major );
00096     Globales::elPuerto.escribir( "\n" );
00097     */
00098
00099     // 2. esperamos el tiempo que nos digan
00100     esperar(tiempoEspera);
00101
00102     // 3. paramos anuncio
00103     (*this).laEmisora.detenerAnuncio();
00104   } // ()
00105
00106   // ---------------------------------------------------------
00107   // ---------------------------------------------------------
00115   void publicarTemperatura(int16_t valorTemperatura, uint8_t contador, long tiempoEspera) {
00116     uint16_t major = (MedicionesID::TEMPERATURA « 8) + contador;
00117     (*this).laEmisora.emitirAnuncioIBeacon((*this).beaconUUID,
00118                                            major,
00119                                            valorTemperatura, // minor
00120                                            (*this).RSSI // rssi
00121     );
00122     esperar(tiempoEspera);
00123     (*this).laEmisora.detenerAnuncio();
00124   } // ()
00125
00126 }; // class
00127
00128 // -----------------------------------------------------------------
00129 // -----------------------------------------------------------------
00130 // -----------------------------------------------------------------
00131 // -----------------------------------------------------------------
00132 #endif // PUBLICADOR_H_INCLUIDO
```

6.10. Referencia del archivo HolaMundoIBeacon/PuertoSerie.h

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



Clases

- class PuertoSerie

  Clase para manejar la comunicación serie.

6.11. PuertoSerie.h

Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00003 // ----------------------------------------------------------
00004 // Jordi Bataller i Mascarell
00005 // 2019-07-07
00006 // ----------------------------------------------------------
00007
00008 #ifndef PUERTO_SERIE_H_INCLUIDO
00009 #define PUERTO_SERIE_H_INCLUIDO
00010
00011 // ----------------------------------------------------------
00012 // ----------------------------------------------------------
00017 class PuertoSerie  {
00018
00019 public:
00020   // .....................................................
00021   // .....................................................
00028   PuertoSerie(long baudios) {
00029     Serial.begin(baudios);
00030     // Mejor no poner esto aquí: while (!Serial) delay(10);
00031   } // ()
00032
00033   // .....................................................
00034   // .....................................................
00041   void esperarDisponible() {
00042     delay(10);
00043   } // ()
00044
00045   // .....................................................
00046   // .....................................................
00054   template<typename T>
00055   void escribir(T mensaje) {
00056     Serial.print(mensaje);
00057   } // ()
00058
00059 }; // class PuertoSerie
00060
00061 // ----------------------------------------------------------
00062 // ----------------------------------------------------------
00063 // ----------------------------------------------------------
00064 // ----------------------------------------------------------
00065 #endif // PUERTO_SERIE_H_INCLUIDO
```

6.12. Referencia del archivo HolaMundoIBeacon/ServicioEnEmisora.h

```
#include <vector>
```

Gráfico de dependencias incluidas en ServicioEnEmisora.h:

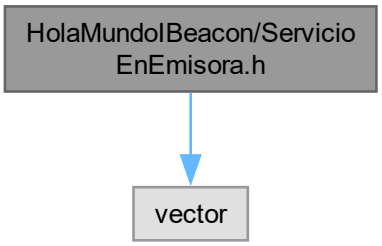
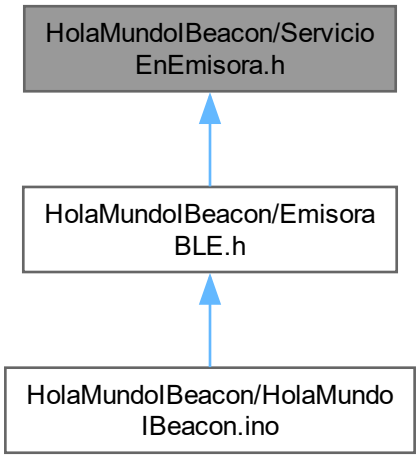


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Funciones

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

  Invierte el contenido de un array en el mismo 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.

6.12.1. Documentación de funciones

6.12.1.1. alReves()

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

Invierte el contenido de un array en el mismo array.

Parámetros de plantilla

T	Tipo de los elementos en el array.

Parámetros

p	Puntero al array.
n	Número de elementos en el array.

Devuelve

Puntero al array invertido.

6.12.1.2. 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

pString	Cadena de entrada.
pUint	Puntero al array de uint8_t donde se almacenará el resultado.
tamMax	Tamaño máximo del array de salida.

Devuelve

Puntero al array de uint8_t con la cadena invertida.

6.13. ServicioEnEmisora.h

Ir a la documentación de este archivo.

```
00001 // -*- mode: c++ -*-
00002
00003 // ------------------------------------------------------------
00004 // Jordi Bataller i Mascarell
00005 // 2019-07-17
00006 // ------------------------------------------------------------
00007
00008 #ifndef SERVICIO_EMISORA_H_INCLUIDO
00009 #define SERVICIO_EMISORA_H_INCLUIDO
00010
00011 // ------------------------------------------------------
00012 #include <vector>
00013
00014 // ------------------------------------------------------
00022 template< typename T >
00023 T * alReves(T * p, int n) {
00024   T aux;
00025
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
00034 // ------------------------------------------------------
00043 uint8_t * stringAUint8AlReves(const char * pString, uint8_t * pUint, int tamMax) {
00044   int longitudString = strlen(pString);
00045   int longitudCopiar = (longitudString > tamMax ? tamMax : longitudString);
00046
00047   // Copia el nombre del servicio -> uuidServicio pero al revés
00048   for(int i = 0; i <= longitudCopiar - 1; i++) {
00049     pUint[tamMax - i - 1] = pString[i];
00050   } // for
00051
00052   return pUint;
00053 } // ()
00054
00055 // ------------------------------------------------------------
00060 class ServicioEnEmisora {
00061 public:
00062
00063   // ------------------------------------------------------
00064   // ------------------------------------------------------
00065
00066   // ......................................................
00067   // ......................................................
00068   using CallbackCaracteristicaEscrita = void(uint16_t conn_handle,
00069                                              BLECharacteristic * chr,
00070                                              uint8_t * data, uint16_t len);
00071   // ......................................................
00072   // ......................................................
00077   class Caracteristica {
00078   private:
00079     uint8_t uuidCaracteristica[16] = { // UUID se copia aquí (al revés) a partir de un string-c
00080       // least significant byte, el primero
00081       '0', '1', '2', '3',
00082       '4', '5', '6', '7',
00083       '8', '9', 'A', 'B',
00084       'C', 'D', 'E', 'F'
00085     };
00086
00087     BLECharacteristic laCaracteristica; // Instancia de la característica
00088
00089   public:
00090     // ......................................................
00091     // ......................................................
00098     Caracteristica(const char * nombreCaracteristica_)
00099       : laCaracteristica(stringAUint8AlReves(nombreCaracteristica_, &uuidCaracteristica[0], 16)) {
00100     } // ()
00101
00102     // ......................................................
00103     // ......................................................
00114     Caracteristica(const char * nombreCaracteristica_,
00115                    uint8_t props,
00116                    SecureMode_t permisoRead,
00117                    SecureMode_t permisoWrite,
00118                    uint8_t tam)
00119       : Caracteristica(nombreCaracteristica_) { // llamada al otro constructor
00120       asignarPropiedadesPermisosYTamanyoDatos(props, permisoRead, permisoWrite, tam);
00121     } // ()
```

```
00122
00123   private:
00124     // ------------------------------------------------------
00125     // ------------------------------------------------------
00130     void asignarPropiedades(uint8_t props) {
00131       (*this).laCaracteristica.setProperties(props);
00132     } // ()
00133
00134     // ------------------------------------------------------
00135     // ------------------------------------------------------
00141     void asignarPermisos(SecureMode_t permisoRead, SecureMode_t permisoWrite) {
00142       (*this).laCaracteristica.setPermission(permisoRead, permisoWrite);
00143     } // ()
00144
00145     // ------------------------------------------------------
00146     // ------------------------------------------------------
00151     void asignarTamanyoDatos(uint8_t tam) {
00152       (*this).laCaracteristica.setMaxLen(tam);
00153     } // ()
00154
00155   public:
00156     // ------------------------------------------------------
00157     // ------------------------------------------------------
00165     void asignarPropiedadesPermisosYTamanyoDatos(uint8_t props,
00166                                                  SecureMode_t permisoRead,
00167                                                  SecureMode_t permisoWrite,
00168                                                  uint8_t tam) {
00169       asignarPropiedades(props);
00170       asignarPermisos(permisoRead, permisoWrite);
00171       asignarTamanyoDatos(tam);
00172     } // ()
00173
00174
00175     // ------------------------------------------------------
00176     // ------------------------------------------------------
00182     uint16_t escribirDatos(const char * str) {
00183       uint16_t r = (*this).laCaracteristica.write(str);
00184       return r;
00185     } // ()
00186
00187     // ------------------------------------------------------
00188     // ------------------------------------------------------
00194     uint16_t notificarDatos(const char * str) {
00195       uint16_t r = laCaracteristica.notify(&str[0]);
00196       return r;
00197     } // ()
00198
00199     // ------------------------------------------------------
00200     // ------------------------------------------------------
00205     void instalarCallbackCaracteristicaEscrita(CallbackCaracteristicaEscrita cb) {
00206       (*this).laCaracteristica.setWriteCallback(cb);
00207     } // ()
00208
00209     // ------------------------------------------------------
00210     // ------------------------------------------------------
00214     void activar() {
00215       err_t error = (*this).laCaracteristica.begin();
00216       Globales::elPuerto.escribir(" (*this).laCaracteristica.begin(); error = ");
00217       Globales::elPuerto.escribir(error);
00218     } // ()
00219   }; // class Caracteristica
00220
00221   // -------------------------------------------------------
00222   // -------------------------------------------------------
00223 private:
00224   uint8_t uuidServicio[16] = { // UUID se copia aquí (al revés) a partir de un string-c
00225     // least significant byte, el primero
00226     '0', '1', '2', '3',
00227     '4', '5', '6', '7',
00228     '8', '9', 'A', 'B',
00229     'C', 'D', 'E', 'F'
00230   };
00231
00232   BLEService elServicio; // Instancia del servicio
00233   std::vector<Caracteristica *> lasCaracteristicas; // Vector de características
00234
00235 public:
00236   // ------------------------------------------------------
00237   // ------------------------------------------------------
00244   ServicioEnEmisora(const char * nombreServicio_)
00245     : elServicio(stringAUint8AlReves(nombreServicio_, &uuidServicio[0], 16)) {
00246   } // ()
00247
00248   // ------------------------------------------------------
00249   // ------------------------------------------------------
00253   void escribeUUID() {
00254     Serial.println("**********");
```

```
00255     for(int i = 0; i <= 15; i++) {
00256       Serial.print((char)uuidServicio[i]);
00257     }
00258     Serial.println("\n**********");
00259   } // ()
00260
00261   // .....................................................
00262   // .....................................................
```

Índice alfabético