# Documentación de usuario

## Introducción

Los usuarios objetivos de este software son en realidad los encargados de la programacion. El siguiente manual compone una guía para asistir en ese proceso. Se asumen unos conocimientos básicos de programación como por ejemplo, uso de variables, funciones, bucles y compilación.

## Instalación

Es necesario haber seguido los pasos detallados en el apéndice D. Partimos de un nodo central funcionando y un ordenador con el entorno de desarrollo debidamente configurado para compilar el código.

Todo el código puede descargarse de https://github.com/turbolargoo/Free-Connect

## Funciones y parámetros

Resumen de funciones y parámetros de configuración.

### Credenciales wifi y OTA

Introducimos las credenciales para el acceso a la red wifi y creamos el nombre y contraseña que utilizará el nodo para recibir actualizaciones. Esta manera de programar el módulo es mucho mas cómoda y nos permitirá hacer cualquier cambio en el código del nodo sin necesidad de apagarlo y conectarlo mediante cable. El ordenador utilizado para subir el sketch debe estar en la misma red que el nodo.

```
//wifi credentials. They sould match your network's
const char *ssid = "name";
const char *password = "pssword";

//OTA credentials. we define those here
const char nombreOTA[]= "awesomeModule";
const char passOTA[]= "admin";
```

Figura 1: Credenciales wifi y OTA

### Ajustes webthings

En el siguiente fragmento de código configuramos los parámetros para la comunicación con el nodo central.

- deviceName. Nombre que toma el nodo. Sera el nombre que veremos en la interfaz al añadir el nodo.
- Capacidades. Funcionalidades que asignamos al nodo. Podemos poner tantas como queramos y necesario que la lista acabe en "nullptr". La lista completa se encuentra aquí. https://iot.mozilla.org/schemas/#capabilities/ Hay que prestar atención a las propiedades marcadas como obligatorias.
- Sensor. Dentro del mismo nodo podemos tener varios dispositivos. Serán reconocidos como nodos diferentes en la interfaz. Este constructor toma tres parametros: `Sensor("nombre","descripcion","lista_capacidades")`
- Propiedad1. Cada uno de los parametros que tiene el nodo. Suelen corresponder con las magnitudes medidas. El constructor toma 4 parámetros: `Propiedad("nombre","descripcion",unidad,tipo_magnitud)` Lista completa tipos de magnitud: https://iot.mozilla.org/schemas/#properties La unidad define el tipo de variable que se usará como: NUMBER, INTEGER, BOOLEAN...

```
//parameters for the connection with the main node. Please check the manual for a complete reference on these
const char deviceName[] ="deviceName";
WebThingAdapter *adapter;//main objet we will use
const char *funcionalities[] = {"MultiLevelSensor", nullptr};//list of functionality categories
ThingDevice Sensor("name", "description", funcionalities);//one device on our project
ThingProperty Property1("name", "description", NUMBER, "LevelProperty");//define each property of the device
```

Figura 2: Ajustes webthings

De manera similar podemos crear eventos:

```
ThingEvent Evento("Something happenes","What hapenes",BOOLEAN, "AlarmEvent");
```

Figura 3: Definición de un nuevo evento

### Ajustes Telegram

La comunicación con Telegram se lleva a cabo mediante el sistema de bots.

Cada bot representa un usuario capaz de enviar y recibir menjes. El bot será implementado por nuestro nodo, eso permitirá comunicarnos con él por este canal.

Lo primero que debemos hacer es crear nuestro propio bot. Para ello acudimos al botfather, un bot que nos guiará en el proceso. Podemos encontrar mas información en https://core.telegram.org/bots

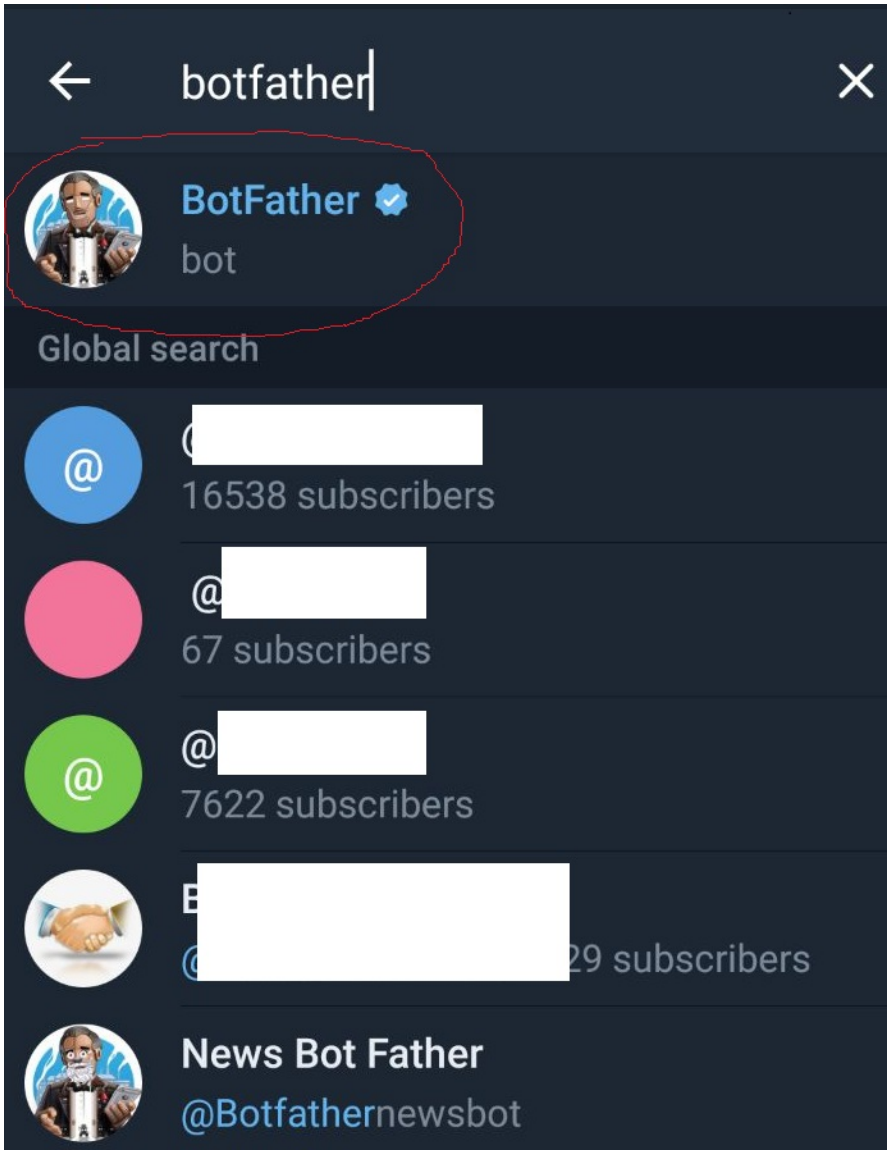


Figura 4: Botfather Telegram

Las características de un usuario bot son similares a las de las personas pero tienen algunas características propias que debemos tener en cuenta. La primera de ellas es que por razones de seguridad, un bot no puede comenzar una conversación. Para que nuestro nodo pueda hablarnos, primero debemos hablarle nosotros a él. Otra cosa que debemos tener en cuenta es que los bots tienen un espacio de almacenamiento en los servidores de Telegram limitado(aunque es muy grande). Los mensajes más antiguos pueden ser eliminados de manera automática.

En este punto, nuestro nuevo bot es funcional. Pasamos a continuación a revisar la sección de ajustes dedicada a ello.

```
//telegram bot settings
#define bOTtoken "yourToken"
uTLGBot Bot(bOTtoken);
const int debugLevelBot = 0;
const int numChats=2; //we can specify to send the messages to more than one account
//we specify here the chat ids we want the messages sent to
const char chatID[]={"chatId1"};
const char chatID2[]={"chatId2"};
const char *chatIDs[numChats]={chatID,chatID2};
```

Figura 5: Ajustes de Telegram

bOTtoken es la variable correspondiente con el bot que acabamos de crear. Podemos ver nuestro token desde el chat con botfather.

*debugLevel* permite configurar mensajes de depuración para el bot.

*numChats* especifica el número de usuarios a los que hablará nuestro nodo.

A continuación almacenaremos la id de usuario de cada uno de esos usuarios. Es importante que todos los usuarios hayan inicado la conversación con el bot. Podemos comprobar que la comunicación funciona y averiguar esta id cargando en nuesto esp32 el código suministrado para ello llamado *echo bot*.

### Funciones propias de Webthings

Dentro del setup encontramos un fragmento de código marcado como configuración para webthings.

```
//webthings setup
adapter = new WebThingAdapter(deviceName, WiFi.localIP());
Sensor.addProperty(&Propiedad1);
adapter->addDevice(&Sensor);
adapter->begin();
```

Figura 6: Setup Webthings

Con la primera instrucción creamos el objeto que contendrá todas las caraterísticas que hemos definido. Los parámetros que toma son el nombre escogido y la dirección ip local del nodo.

Lo siguiente que hacemos es añadir cada propiedad y evento a algún sensor. Finalmente, añadimos todos los sensores al adaptador que acabamos de instanciar y llamamos a la función begin();

Adicionalmente podemos modificar desde aquí alguna de las propiedades:

- **Property.unit()** Nos permite cambiar la unidad que veremos junto a la lectura en la interfaz. Valores posibles: "degree celsius", "percent", `"milliseconds"` y otras unidades del sistema internacional.
- **Property.readOnly():** Atributo de solo lectura. Property.readOnly()=true;
- **Property.title():** Permite cambiar el nombre que se muestra en la interfaz.
- **Property.maximum():** Establece el nivel máximo. Útil en propiedades de nivel para ajustar el rango de valores en el indicador visual.
- **Property.minimun():** Establece el nivel mínimo.

**Uso de la intefaz:**

A continuación explico el proceso para envío y recepción de datos.

- **Envío de un dato:** Para enviar un dato a la interfaz seguiremos los siguientes pasos:
  1. Crear un objeto del tipo ThingPropertyValue. Ejemplo: ThingPropertyValue Valor;
  2. Guardar en ese objeto el valor que queremos enviar. Debemos especificar el mismo tipo que utilizamos durante la configuración.
     Ejemplo para entero: Valor.number = 7;
     Ejemplo para booleano: Valor.boolean = true;
  3. Pasar el objeto con la propiedad a una de las propiedades definidas. Ejemplo: Propiedad.setValue(Valor)
- **Recepción de un dato:** Para leer un dato de la interfaz utilizaremos el método getValue() de la propiedad que queramos consultar. Obtenderemos un objeto del tipo ThingPropertyValue del cual podremos consultar directamente el valor.
  Ejemplo: int valorLeido = Propiedad.getValue().number;
- **Envío de un evento:** Los eventos son muy útiles cuando queremos desencadenar una cierta acción en la interfaz. Para enviar un evento seguiremos los siguientes pasos:
  1. Creamos un objeto del tipo ThingProperyValue y asignamos el valor deseado de la misma manera que en el caso de un envío de dato Ejemplo:
     ThingProperyValue Valor ; Valor.boolean=true;

2. Creamos un puntero para un objeto del tipo ThingEventObject. Ejemplo: ThingEventObject *Event;
3. Asignamos en ese puntero un evento utilizando el constructor ThingEventObject(`"EventoDefinido"`, tipoValor, valor).
   Ejemplo: Event= new ThingEventObject(`"Evento"`, BOOLEAN, Valor):
4. Ponemos el evento en la cola del sensor con la función queueEventObject(Evento). Ejemplo: Sensor.queueEventObject(Evento);

Nota importante: Los eventos deben ser atendidos por la interfaz como si de una interrupción se tratase. Enviar demasiados eventos seguidos puede ocasionar errores. El sistema de eventos está diseñado para situaciones temporales, no para envío constante de datos como por ejemplo en la lectura de un sensor.

### Funciones propias de Telegram

Una vez terminada la función *setup* el objeto *Bot* estará listo para ser utilizado. Estas son algunas de las funciones que podemos utilizar:

- **Bot.getUpdates():** Nos devuelve el número de nuevos mensajes.
- **Bot.sendMessage():** Comando utilizado para enviar mensajes. El primer parámetro es el chat de destino y el segundo el texto a enviar.
- **Bot.received_msg:** Nos devuelve el último mensaje. Con cada llamada a getUpdates() iteramos hacia el siguiente mensaje.

Contenido del objeto *message*:

- **update_id:** Número entero que identifica el mensaje.
- **message:** Contenido del mensaje. Esto a su vez es un objeto con el siguiente contenido útil:
  - **message_id:** Número identificador del mensaje propio del chat con esa persona.
  - **date:** Fecha del mensaje.
  - **text:** String con el texto del mensaje (en mensajes de texto)
  - **from:** Objeto User con la siguiente información interesante:
    - **id:** Id del usuario que envía el mensaje. Entero
    - **username:** nombre del usuario (@ en telegram)

## Funciones y parámetros adicionales

Funciones y parámetros utilizados en los ejemplos.

- **map(valor,min,max,nuevoMin,nuevoMax)** Convierte el valor pasado de el rango especificado como inicial al nuevo rango especificado
- **dtostrf(origen,numEnteros,numDecimales,destino)**Convierte números decimales en cadenas de texto. Origen es el integer que queremos convertir, numEnteros y numDecimales las posiciones enteras y decimales que queremos tener respectivamente y destino la variable donde almacenar la cadena resultante.

### Funciones propias del módulo con pantalla OLED

La librería mas usada para utilizar este tipo de pantallas probablemente sea la oficial de *Adafruit*. Esta librería no tiene apenas opciones de personalización y causa problemas con la placa elegida para el desarrollo. La librería de *Dragonsmith* es mucho más completa y no causa problemas de dependencias.

Estas son algunas de las funciones útiles que incluye:

- **Constructor:** Nos permite inicializar un objeto display con los siguientes parámetros: SSD1306Wire display(dirección i2c, pin SDA, pin SCL); En la mayoria de pantallas la dirección i2c es 0x3c
- **init():** Inicia la comunicación.
- **setTextAlignment(opcion):** Alinea el texto con una de estas opciones: `TEXT_ALIGN_LEFT, TEXT_ALIGN_CENTER, TEXT_ALIGN_RIGHT, TEXT_ALIGN_CENTER_BOTH`
- **displayOn():** Enciende el display.
- **displayOff():** Apaga el display.
- **clear():** Limpia el contenido de la pantalla.
- **drawString(posX,posY,texto):**Envía a la pantalla el texto en la posicion. especificada. El nuevo texto no es visible hasta ejecutar la función display().
- **display():** Actualiza el contenido.

### Funciones propias del sensor DHT11

El constructor no toma parámetros. Lo creamos de la siguiente manera:

*DHTesp dht;*

Funciones útiles de esta biblioteca:

- **setup(pin,tipoSensor):** Especificamos el pin donde está conectado y DHTesp::DHT22 o DHTesp::DHT11 dependiendo de nuestro modelo.
- **getTemperature():** Devuelve la temperatura en grados centígrados.
- **getHumidity():** Devuelve la humedad relativa en el ambiente en porcentaje.
- **toFahrenheit(float celcius):** Convierte de grados Celsius a grados Farenheit.
- **computeDewPoint(temperatura, humedad, farenheit(boolean)):**. Nos devuelve el punto de condensación. Muy útil para entornos de trabajo con equipos electrónicos por ejemplo.

## Manejo de la interfaz

Estos son los aspectos más relevantes sobre el manejo de la interfaz:

### Añadir nuevos Nodos

En la esquina inferior derecha de la vista general encontramos un botón que nos permite añadir nodos.

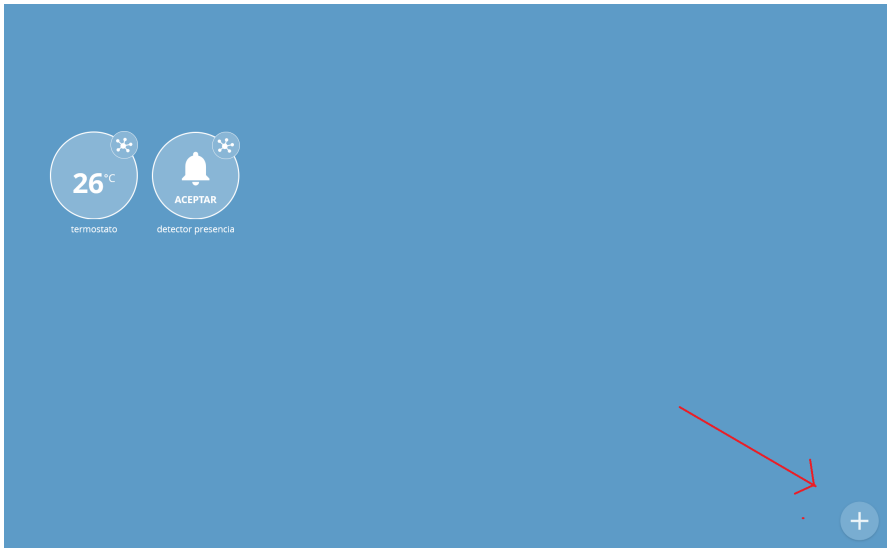


Figura 7: Botón para añadir nodos

Al pulsarlo se desplegará un menú donde podemos seleccionar el nuevo nodo. Veremos el nombre que hemos configurado en el código.

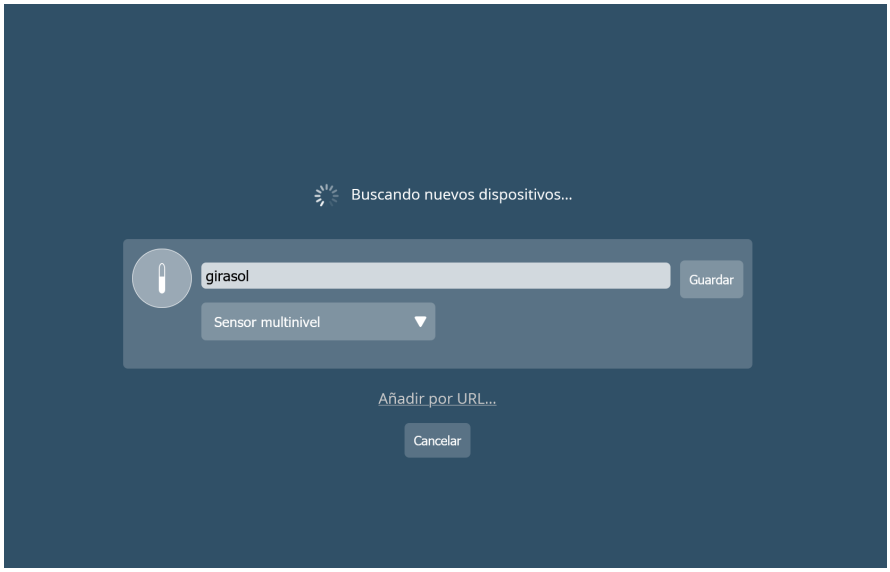


Figura 8: Selección del nodo a añadir

Tras aplicar los cambios podremos ver nuestro nodo en la página principal.

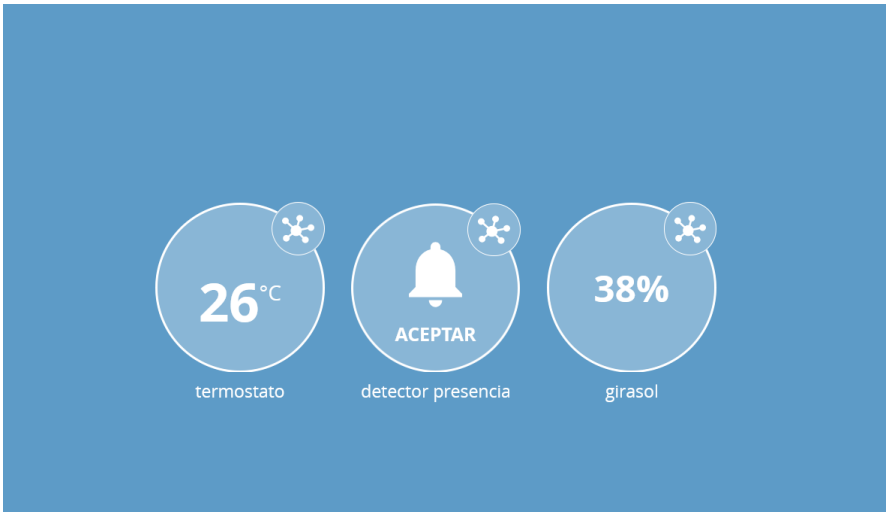


Figura 9: Vista general de los nodos

### Configuración de eventos

Para que el sistema atienda los eventos que mandan los nodos, debemos configurar la interfaz correctamente. Esto lo conseguiremos en la sección llamada `"Reglas"`.

Las reglas están compuestas por dos acciones. La primera es la que desdencadenará la acción (parte izquierda) esta acción será un temporizador o un evento recibido desde un nodo. La segunda acción (parte derecha) será lo que queremos que haga la interfaz, por ejemplo, mostrar una notificación de escritorio.

En esta primera imagen vemos el aspecto de la interfaz cuando nuestros nodos no tienen eventos configurados:

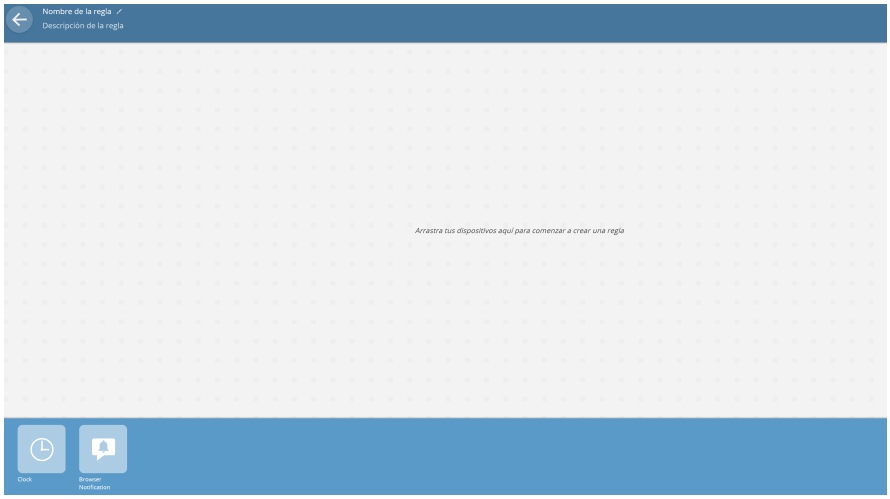


Figura 10: Vista básica de la ventana de configuración de eventos

Cuando configuremos los eventos y conectemos los nodos, podremos ver aparecer aquí las acciones correpondientes:


Figura 11: Acciones personalizadas configuradas en la interfaz