
iot-bus Documentation

Release latest

oddWires

Mar 05, 2019

1 IoT-Bus Overview	3
2 IoT-Bus Pinout	13
3 Choosing a Platform and Framework	15
4 Getting Started with PlatformIO	17
5 Getting Started with Arduino	19
6 Getting Started with esp-idf	21
7 Getting Started with Mozilla IoT	23
8 Getting Started with Micro-Python	29
9 Getting Started with Moddable	31
10 Getting Started with MicroBlocks	33
11 Io	35
12 Proteus	39
13 JTAG	43
14 2.4" QVGA Touch Display	47
15 Motor	51
16 Relay	53
17 CAN Bus	55
18 LoRa	57
19 IoT-Bus Examples Index	61
20 IoT-Bus Blink Example	63

21 IoT-Bus Hello World Example	65
22 IoT-Bus Touch Draw Example	67
23 IoT-Bus Relay Example	73
24 IoT-Bus CAN Bus Example	75
25 IoT-Bus LoRa Example	79
26 IoT-Bus Motor Example	83
27 IoT-Bus SD_MMC Card Example	85
28 IoT-Bus Mozilla IoT Examples	91
29 IoT-Bus LED Touch Thing	95
30 IoT-Bus LED Thing	97
31 IoT-Bus LED Lamp Thing	99
32 IoT-Bus Relay Thing	103
33 IoT-Bus Relay Display-Touch Thing	105
34 IoT-Bus Window and Door Sensor Thing	111
35 IoT-Bus DHT11 Thing	113
36 IoT-Bus HC-SR04 Thing	119
37 IoT-Bus HC-SR501 PIR Thing	123
38 IoT-Bus Calculator Thing	127
39 IoT-Bus Mozilla IoT Tutorials	135
40 LED Thing Tutorial	137
41 Touch Switch Thing Tutorial	143
42 Mozilla Rules Engine	149
43 Frameworks	155
44 Platforms	157
45 Espressif32 Key Features	159

CHAPTER 1

IoT-Bus Overview

The oddWires IoT-Bus system is based on a low-cost, open design that includes multiple main boards including a minimalist, breadboard-friendly form-factor, the IoT-Bus Io and a version with a large prototyping area which enables a single-board IoT solution, the Proteus.

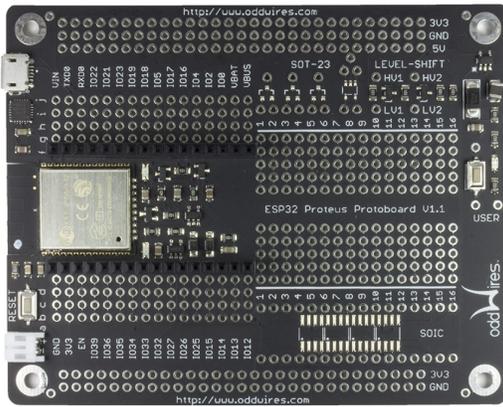
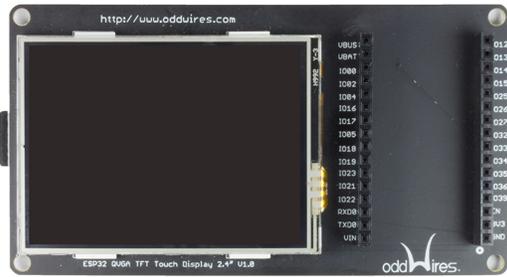
The IoT-Bus is designed to be “plug and play” - the new range of oddWires open IoT-Bus boards use the Espressif ESP32 microprocessor (240Mhz, 32-bit, 4MB) for rapid, low-cost IoT development and deployment.

Our aim is to provide an open platform, easy to adopt and build on, adaptable, tested, with many solutions already available. There are no lock-in costs, and overall it provide a low-cost solution for faster development of professional, educational and hobbyist applications.

A variety of programming languages, environments and frameworks that work right out of the box with IoT-Bus and PlatformIO supports most of them in a professional or serious educational or hobby environment. You can use either the esp-idf framework for a professional multi-tasking system based on freertos or the popular Arduino environment for simplicity.

So take your choice! Other platforms include javascript by [Moddable](#), the [Mongoose](#) server or python by [MicroPython](#). There’s even a graphical way of programming IoT-Bus - [microBlocks](#).





Some IoT-Bus Boards

1.1 IoT-Bus System

- Open Design
- Low-cost
- Plug and Play, Expandable
- Powerful 240MHz, 32-bit Processor with 4Mb of Flash Memory
- Multiple Form Factor Main Boards (Io, Proteus)
- Connected in Many Ways (Wi-Fi, Bluetooth, BLE 4.0. LoRa and CAN Bus are available options)
- Integrated 2.4" Touch TFT QVGA Display Available
- Solder-able Prototype Board with Controller
- IOT-Ready, Relay and Motor Controller
- Multiple open platforms
- Supports C++, MicroPython and javascript

At the heart of the system is an ESP32 processor providing two SPI and an I2C interface with plenty of general purpose I/O. The Espressif ESP-WROOM-32 has been selected as the micro-controller enabling very low-cost deployment of production IoT devices. It offers 240Mhz, 32-bit processing with 4MB of flash as standard.

The first controller boards drive relays and motors and there are a wide range of connectivity options including Wi-Fi, Bluetooth, CAN Bus, and LoRa.

Developing open IoT applications means being able to see the schematics for the hardware, using open tools, frameworks and platforms and very importantly the cloud you use has to be open.

1.2 Mozilla Project Things - An Open Internet of Things

Internet of Things (IoT) devices have become more popular over the last few years, but there is no single standard for how these devices should talk to each other. Each vendor typically creates a custom application that only works with their own brand. If the future of connected IoT devices continues to involve proprietary solutions, then costs will stay high, while the market remains fragmented and slow to grow. Consumers should not be locked into a specific product, brand, or platform. This will only lead to paying premium prices for something as simple as a “smart light bulb”.

We are aligned with mozilla and believe the future of connected devices should be more like the open web. The future should be decentralized, and should put the power and control into the hands of the people who use those devices. This is why we are committed to supporting open standards and frameworks.

We are partnering with mozilla to offer kits that can be used to quickly integrate with mozilla-iot. Watch this space for more details. We have also created many examples using iot-bus with mozilla-iot. [See our examples on github.](#)

1.3 Two main-board form-factors

Io Very small and breadboard-friendly with option of male, female or both (stackable headers). Includes a dual-core 240 MHz ESP32 with WiFi and Bluetooth. You can use the WiFi both in station (device) mode and access point mode. It includes traditional Bluetooth as well as BLE 4.0.

On-board is a 3.3V regulator and a battery charging device that enables you to switch between using USB or battery power. The battery is automatically charged in the USB is plugged in. A status light shows if it is charging or fully charged. All ESP32 pins bar the flash pins are exposed and available for your use.

Proteus This board is larger and designed to make it possible to add your own circuitry to make a complete IoT solution on one board. It includes a dual-core 240 MHz ESP32 with WiFi and Bluetooth. You can use the WiFi both in station (device) mode and access point mode. It includes traditional Bluetooth as well as BLE 4.0. On-board is a 3.3V regulator and a battery charging device that enables you to switch between using USB or battery power.

The battery is automatically charged in the USB is plugged in. A status light shows if it is charging or fully charged. All ESP32 pins bar the flash pins are exposed and available for your use.

The board includes a large prototyping area that includes room for traditional DIP and through-hole components as well as SMD parts such as SOIC and SOT-23. A user LED and switch is included but not connected to any pins so you can use them how you wish. Two level shifters are included so you can interface with 5V devices.

The Proteus includes both 3.3V and 5V rails. Both these rails are available whether powered by the USB or the battery as the 5V is derived from the lower voltage.

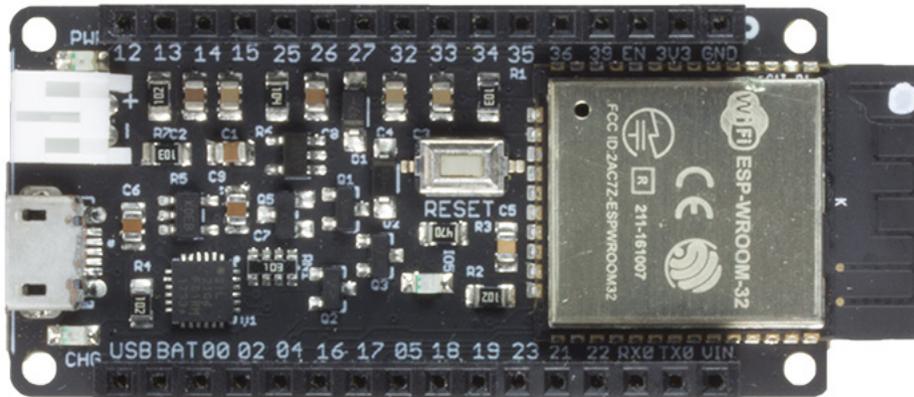


Fig. 1: Io

1.4 JTAG

Both the Io and Proteus processor boards can accept a specially designed JTAG board offering hardware debugging. Our JTAG board is based on the FT232H and it enables comprehensive JTAG debugging support. You can use OpenOCD and GDB in combination to use it but our recommendation is to use PlatformIO. PlatformIO has taken away all the hard work of configuring OpenOCD and GDB. You simply select it is your debugging choice as described [here](#). Take a look at how easy it is to use with [PlatformIO's Unified Debugger](#). Just plug it in and start debugging! No more printing to the terminal!

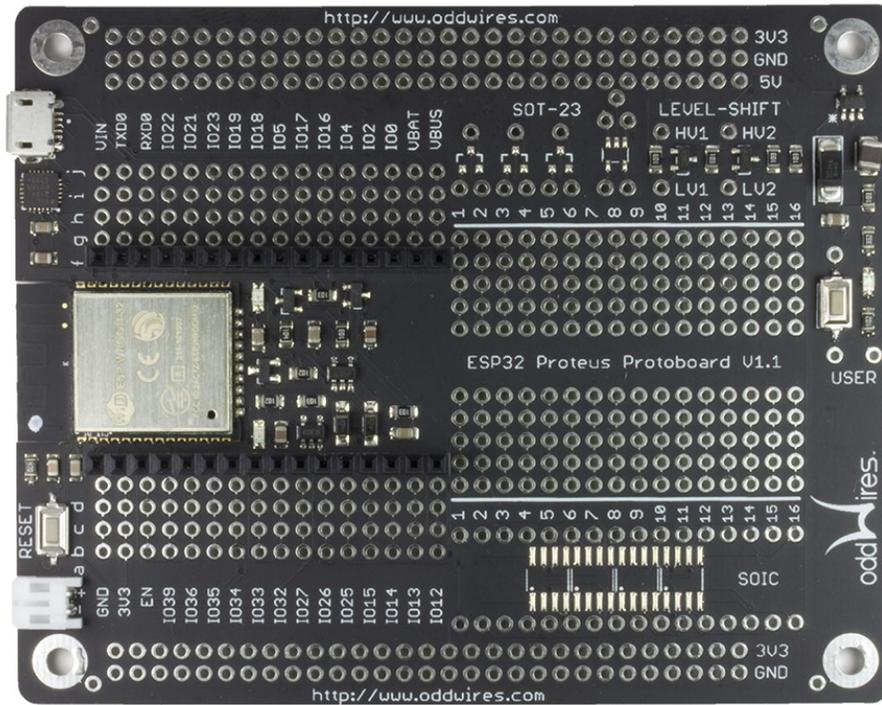


Fig. 2: Proteus

1.5 2.4" QVGA TFT Touch Display

This is a nice 2.4" 320x240 QVGA TFT Touch Display offering plug and play display output and touch sensing together with a 4-bit SD-MMC SD Card. We picked 2.4" over 2.8" as it has a crisper display at 320 x 240 resolution and its slightly smaller size helps in IoT applications. Designed primarily for development use it has an IoT-Bus socket at the side.

1.6 Two Additional Connectivity Options

CAN Bus

The IoT-Bus CAN Bus module offers a transceiver that enables you to use

the onboard ESP32 CAN controller. You can connect the terminals to any required connection.

LoRa

This IoT-Bus module utilizes the Hope RFM95 to offer low-cost, LoRa ra-

dio transmission and a Wi-Fi/LoRa gateway. It uses the correct 915 MHz rather than the 433 MHz european standard

often found. The RFM95W transceivers feature the LoRa long range modem that provides ultra-long range spread spectrum communication and high interference immunity whilst minimizing current consumption.

Using
Hope
RF's
patented
LoRa
mod-
u-
la-
tion
tech-
nique
RFM95W
can
achieve
a
sen-
si-
tiv-

ity of over -148dBm using a low cost crystal and bill of materials. The high sensitivity combined with the integrated +20 dBm power amplifier yields industry-leading link budget making it optimal for any application requiring range or robustness.

LoRa™
also
pro-
vides
sig-
nif-
i-
cant
ad-
van-
tages
in
both
block-
ing
and
se-
lec-

tivity over conventional modulation techniques, solving the traditional design compromise between range, interference immunity and energy consumption. These devices also support high performance (G)FSK modes for systems including WMBus, IEEE802.15.4g. The RFM95W deliver exceptional phase noise, selectivity, receiver linearity and IIP3 for significantly lower current consumption than competing devices.

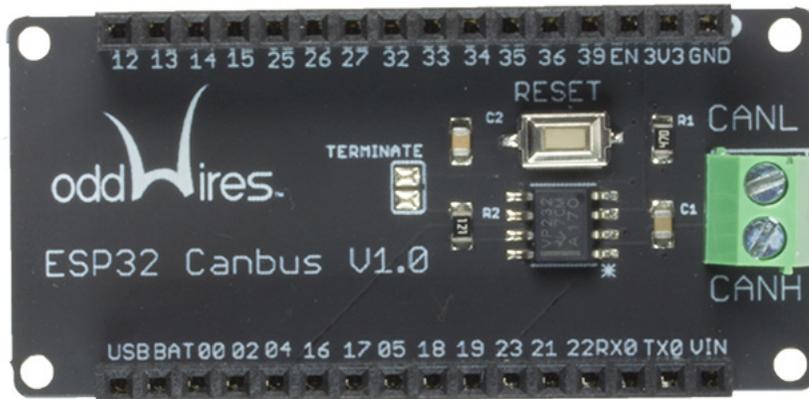


Fig. 5: CAN Bus

1.7 Two Controller Boards

Relay This is an opto-isolated relay board driven by a single digital pin. It is a 110V, 10A maximum AC relay board in the IoT-Bus form factor.

Motor This IoT-Bus module provides a motor controller. It uses two TB6612FNG motor drivers controlled by a PCA9685 on the I2C bus. It supports two stepper motors or four DC Motors.



Fig. 7: Relay

1.8 Platforms

Name	Description
<i>Espressif32</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

1.9 Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

CHAPTER 2

IoT-Bus Pinout

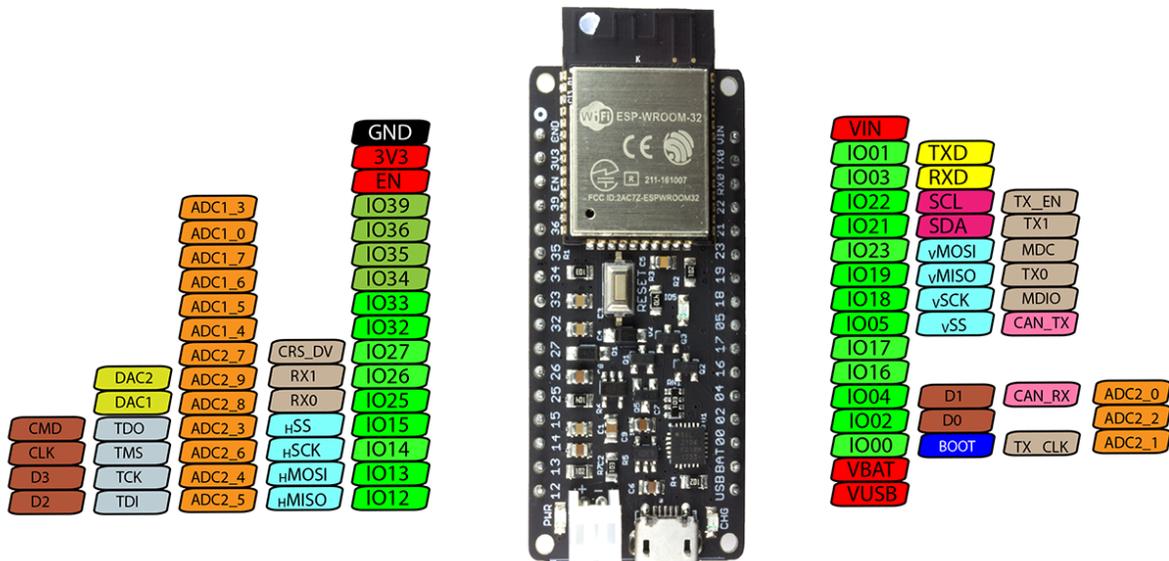
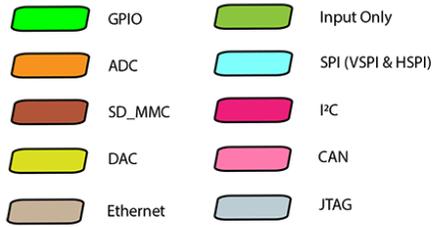


Fig. 1: IoT-Bus Pinout

Choosing a Platform and Framework

One of the great things about IoT-Bus is that you're free to choose from a large number of platforms and frameworks. You know that you want to create an IoT project but where do you start? In our view it very depends on what you know now.

3.1 Arduino

If you have never programmed or know nothing about embedded hardware then you may be best to start with Arduino as a framework and as an editor and programmer. Similarly, if you are in education and do not want to be explaining things to your students about the IDE and just want to create very simple applications then again, you may pick Arduino as a Framework and as your development environment. Many, many users have chosen Arduino over the years and many still do. It's a great place to start. You'll find more projects on the web that Arduino based so if you want to find somebody that's done something similar to what you want to do. You'll probably find it as an Arduino project somewhere.

3.2 PlatformIO

On the other hand if you are a professional programmer or a serious hobbyist then you will probably pick PlatformIO. There are other environments that are just fine if you already use them like Eclipse or NetBeans. But if you are investing your time and money in the future then PlatformIO is the way to go. It's also not hard to take that same project and import it into PlatformIO and take it from there.

Similarly, if maintaining source code integrity, versioning and testing and debugging are becoming more important you as a Arduino user then take a look at PlatformIO. The advantage that PlatformIO has taken of leveraging an excellent open IDE from Microsoft and extending it, again, in open fashion to the embedded world If you're a seasoned C++ programmer then PlatformIO and the esp-idf framework is probably the way to go.

Now in terms of framework you again have a choice. Arduino or esp-idf are the c++ development choices. Arduino is simpler and more "black-box" - you can get going very quickly and without necessarily fully understanding everything you can get an IoT project up and running. However, if things like a multi-tasking environment and the availability of professional-class libraries and support are more important then esp-idf is more likely to be the framework for you. It

is lower-level and you may have to do more work but you are also more likely to get a professional application as a result.

There is a “third-way” option of using Platformio as your development environment and using Arduino as your framework if you are a new user or already familiar with Arduino libraries. This option perhaps gives you the best of both worlds. You’re able to get up and running quickly and it would be easier to move to esp-idf you should choose to.

3.3 MicroPython

Python is becoming ubiquitous (this documentation was produced using it!). And MicroPython is a great implementation of python for the embedded environment. So if you are already familiar with python that’s a great way to leverage the IoT-Bus system. We’ll show you later in this section how to get started with it.

3.4 Moddable

Javascript is another great alternative. It’s not only used in front-end code, there are dedicated servers for it and a whole development infrastructure that has sprung up around it. And Moddable is the company that just “does javascript right”. We’ll show you later in this section how to get started with it.

3.5 MicroBlocks

Lastly there are other choices that may be even better when teaching students graphically such as MicroBlocks which runs wonderfully on IoT-Bus and does not require traditional line-by-line programming.

3.6 Mozilla-IoT

Now Mozilla IoT is not a platform in the same sense of others here. But it certainly is a platform for IoT and irrespective of the framework or the language you can take advantage of Mozilla IoT to discover, inspect and control your IoT devices. So if you’re thinking about a smart-home, a home security project or simply want the ability to control a device at home from your phone or PC then do look at Mozilla IoT

3.7 Summary

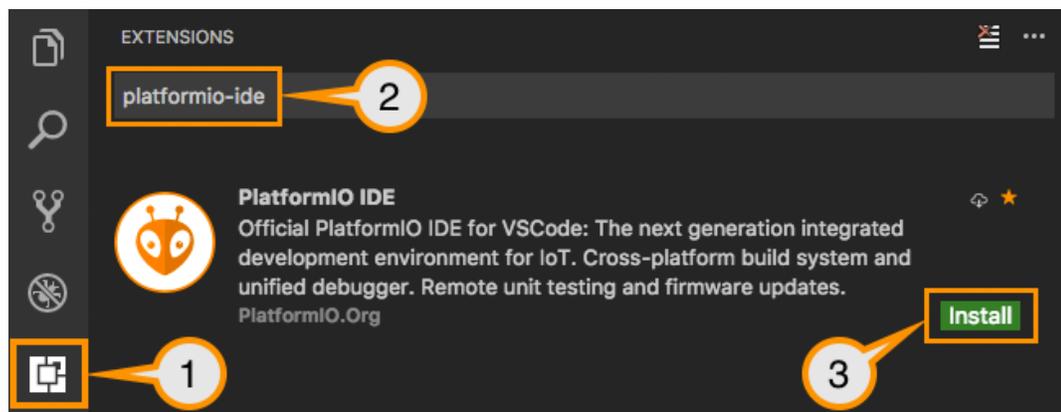
So what would I do today?

1. Buy a kit from oddWires.
2. Choose a platform/framework. You can’t really go wrong. Your investment in IoT-Bus will work in every case and you can change your mind!
3. Get started with IoT-Bus and Mozilla IoT!

Getting Started with PlatformIO

Thank you for choosing PlatformIO IDE for VSCode

1. **Download** and install official Microsoft's Visual Studio Code, PlatformIO IDE is built on top of it.
2. Open VSCode Extension Manager and search for official platformio-ide extension:



3. Install PlatformIO IDE. You can find details on using PlatformIO with the IoT-Bus Io board [here](#).
4. Git clone or download the IoT-Bus examples from *Github* <<https://github.com/iot-bus/iot-bus-examples-platformio>>.
5. Plug in IoT-Bus Io. Open the iot-bus-blink example and run. Onboard LED should blink once a second. When you create a new PlatformIO application simply select the oddWires IoT-Bus Io as the board. This will also ensure you have the correct default debugger when using the IoT-Bus JTAG board.
6. To debug, simply plug in a JTAG board connect up the USB cable to the JTAG board and start debugging. No configuration of OpenOCD or GDB required.

Getting Started with Arduino

To get started with Arduino we are first going to install Arduino. Then we install the platform package for Espressif32. Finally, we plug in an IoT-Bus processor and run the Blink example.

1. Installation

On this page [Arduino Software](#) you will find links for each major operating system. Download it and follow the instructions.

2. Install the Espressif32 platform package

Start Arduino and open Preferences window. Enter https://dl.espressif.com/dl/package_esp32_index.json into Additional Board Manager URLs field. You can add multiple URLs, separating them with commas. Open Boards Manager from Tools > Board menu and install esp32 platform (select ESP32 Dev Module from Tools > Board menu after installation).

3. Run the Blink example

Download the IoT-Bus examples from [here](#). Then open the `iot-bus-blink` example and run. You'll see the on-board LED blink once a second.

s

CHAPTER 6

Getting Started with esp-idf

The hard way to setup esp-idf is [here](#) . It's much easier using platformIO because it does all the hard work behind the scenes. However it maybe useful to see the steps involved. See *Getting Started with Platformio* for the easy way to use esp-idf. You can download or clone the IoT-Bus esp-idf examples from [Github](#).

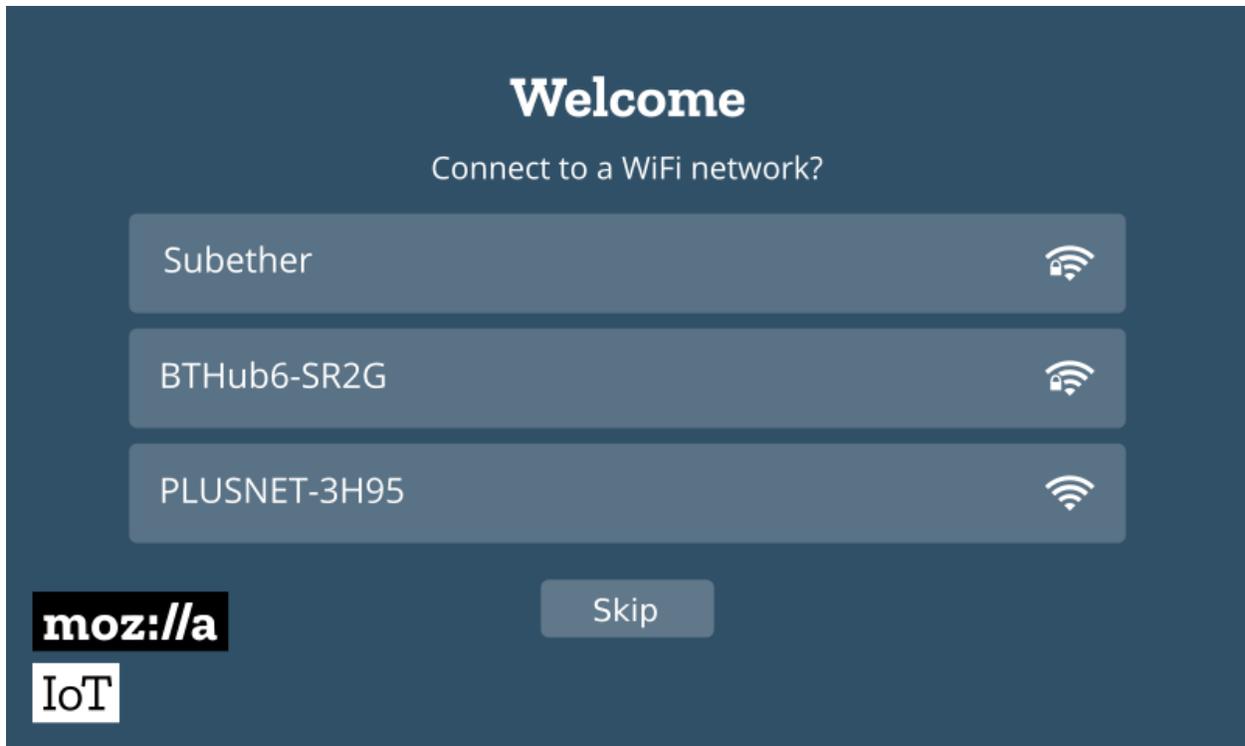
Getting Started with Mozilla IoT

This section of the documentation describes how to setup a gateway on a Raspberry Pi for the Mozilla-IoT platform. You will need a Raspberry Pi an SD Card and an IoT-Bus Io board. If you buy a Mozilla IoT Starter Kit from oddWires you will have everything you need.

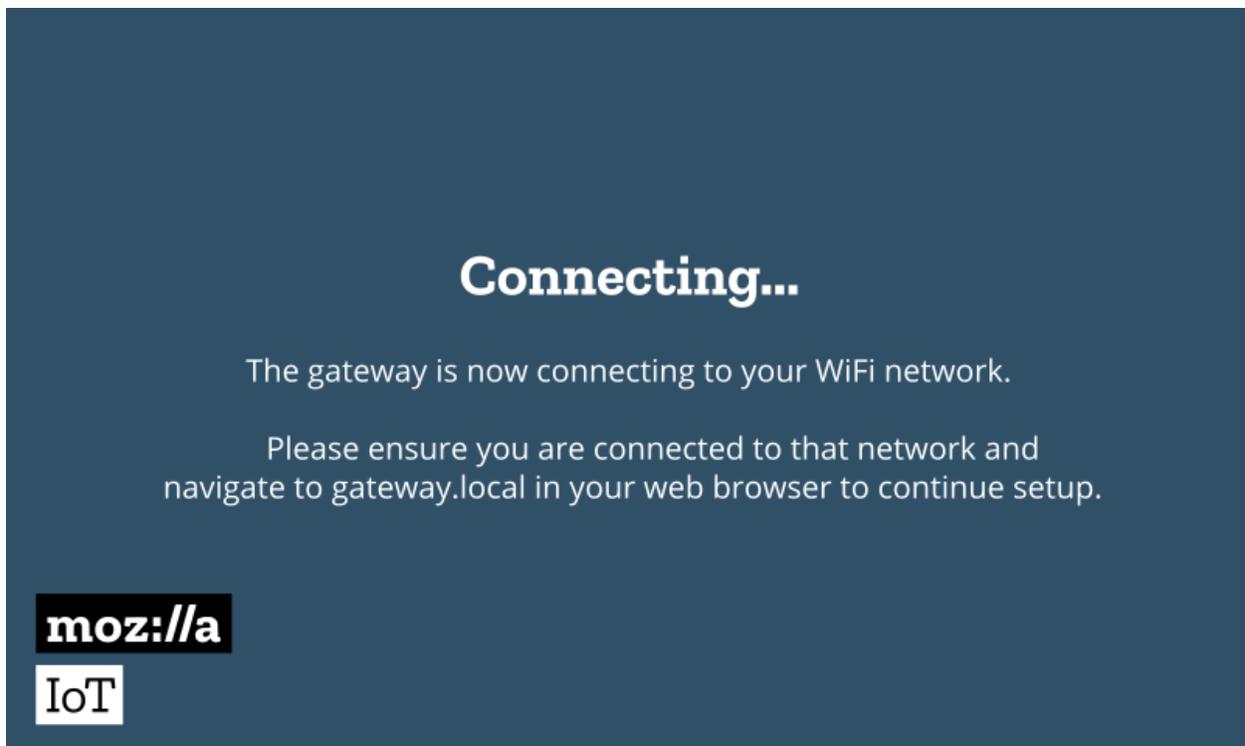
7.1 Gateway First Time Setup

If you didn't buy a Mozilla IoT get from oddWires which includes an SD Card flashed with the Mozilla IoT Gateway you will need to download gateway image as a zip file from [here](#). Expand it and flash it onto an SD Card using a product like [etcher](#).

Place it into your Raspberry Pi and power up. The image is enabled for WiFi and will create an access point "Mozilla IoT Gateway". You can connect to that WiFi hot-spot with your laptop or smartphone which should automatically direct you to a setup page which looks like this:



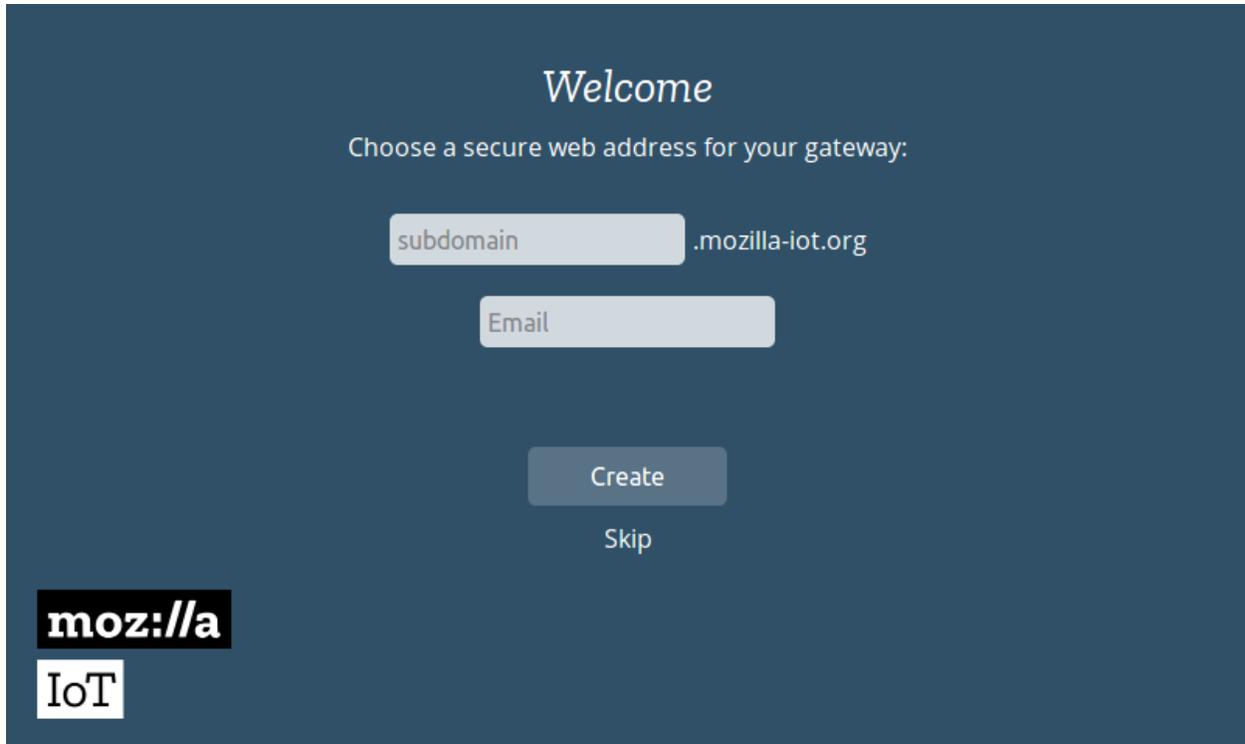
Choose to connect to a WiFi network and you'll be prompted for the WiFi password. The gateway will connect to that network and then you'll need to make sure you're connected to that same network in order to continue setup. If you're directly connected via ethernet you do not need to do this.



Next, you'll be asked to choose a unique subdomain for your gateway, which will automatically generate an SSL certificate for you using LetsEncrypt and set up a secure tunnel to the Internet so you can access the gateway remotely.

You'll be asked for an email address so you can reclaim your subdomain in future if necessary. You can also choose your own domain name if you don't want to use the tunneling service, but you'll need to generate your own SSL certificate and configure DNS yourself.

This is the screen you will see:



Welcome

Choose a secure web address for your gateway:

subdomain .mozilla-iot.org

Email

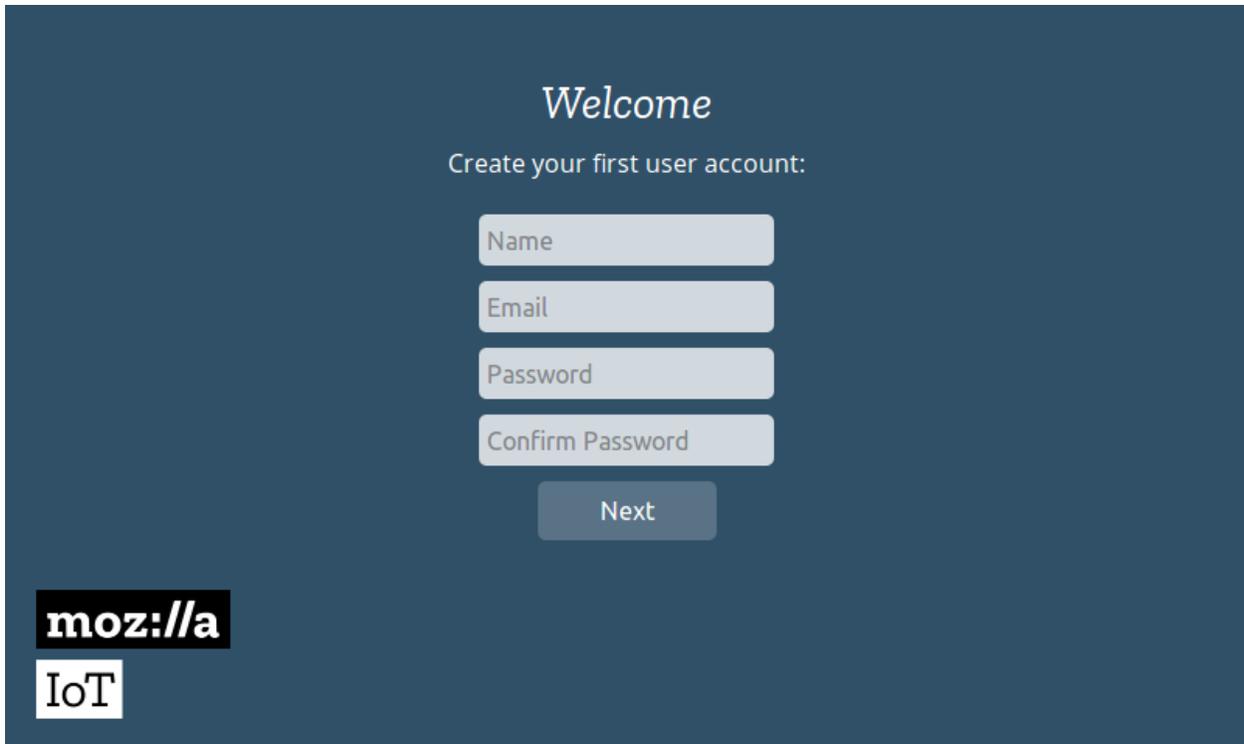
Create

Skip

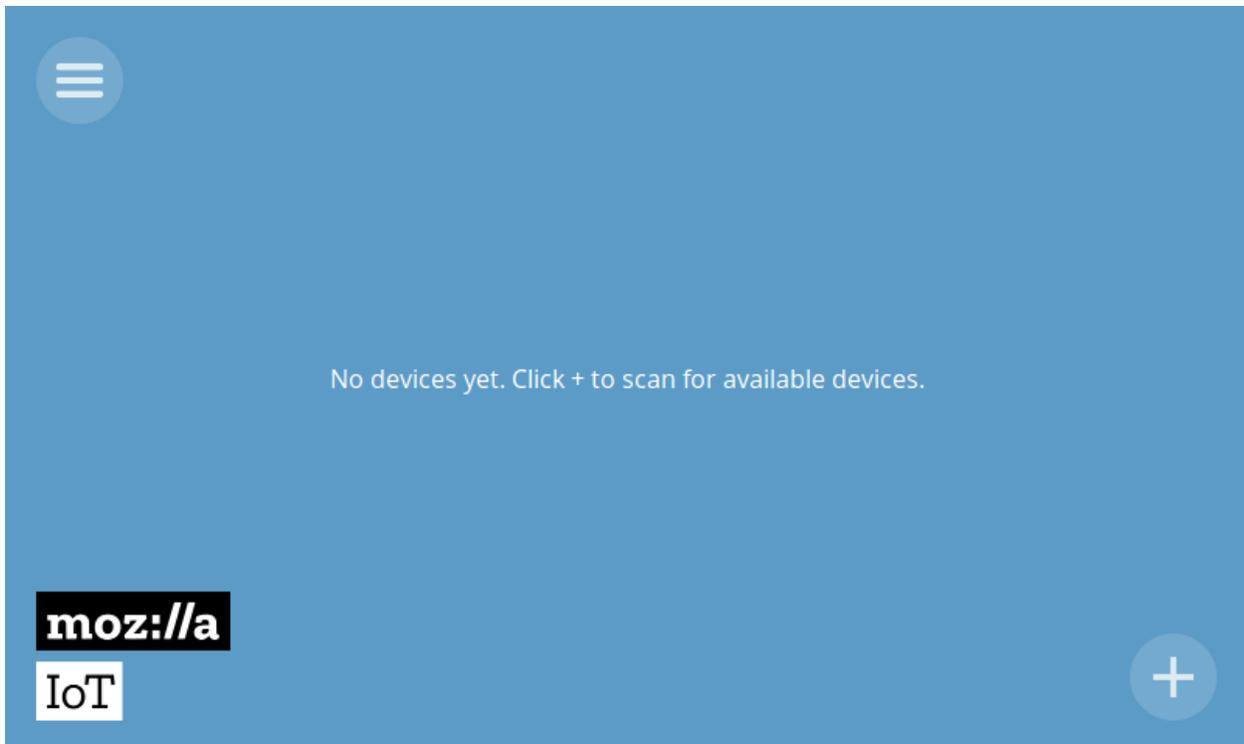
moz://a

IoT

You will then be securely redirected to your new subdomain and you'll be prompted to create your user account on the gateway.



You'll then automatically be logged into the gateway and will be ready to start adding things. Note that the gateway's web interface is a Progressive Web App that you can add to home-screen on your smartphone with Firefox. Now you should see this screen and the gateway is ready to add Things.



7.2 Adding Things

You are now ready to add Things. To add devices to your gateway, click on the “+” icon at the bottom right of the screen. This will put all the attached adapters into pairing mode. Follow the instructions for your individual device to pair it with the gateway (this often involves pressing a button on the device while the gateway is in pairing mode).

Devices that have been successfully paired with the gateway will appear in the add device screen and you can give them a name of your choice before saving them on the gateway.

Getting Started with Micro-Python

MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimized to run on micro-controllers and in constrained environments. You can find out more [here](#). You can download the firmware from [this location](#).

For convenience, this is a link to an ESP32 binary: [esp32-20180924-v1.9.4-575-g6ea6c7cc9.bin](#)

You will need `esptool.py` (available [right here](#)). If you are putting MicroPython on for the first time then you should first erase the entire flash using:

```
esptool.py --chip esp32 erase_flash
```

Program your board using the `esptool.py` program, and put the firmware starting at address `0x1000`. For example:

```
esptool.py --chip esp32 --port /dev/ttyUSB1 write_flash -z 0x1000 esp32-20180511-v1.9.  
↪4.bin
```


CHAPTER 9

Getting Started with Moddable

You can find details on Getting Started with Moddable [here](#).

Getting Started with MicroBlocks

MicroBlocks is a new programming language inspired by Scratch that runs right inside micro-controller boards such as the micro:bit, the NodeMCU and many Arduino boards. The MicroBlocks system allows for dynamic, parallel and interactive programming, just like in Scratch, but with the twist of letting your projects run autonomously inside the board without being tethered to a computer. Thus, MicroBlocks provides the immediacy and liveness of tethered blocks programming, while supporting real-world applications that require precision timing, autonomous operation, or physically embedding the processor into projects. For example, one might write a program to record acceleration data, then embed the microcontroller and a small battery in a model rocket to explore G-forces at launch time.

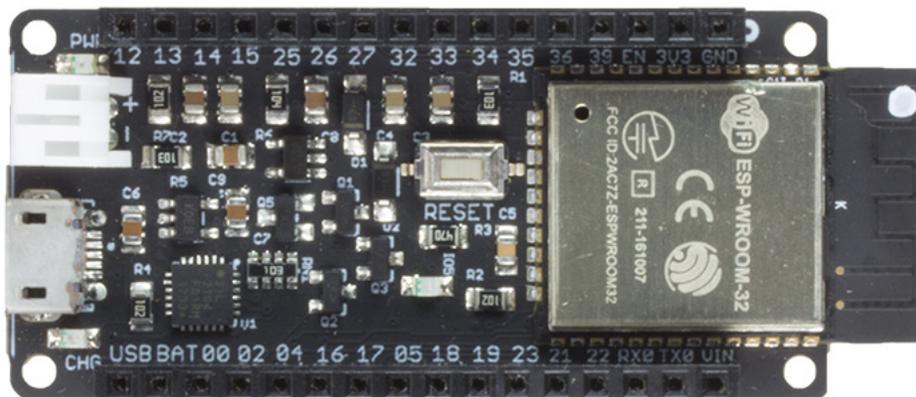
In other words, MicroBlocks lets you program your boards exactly like you would in Snap4Arduino, S4A or the Arduino extension for Scratch, and when you are happy with your program you can just unplug the board from your computer and everything will keep running as if by magic!

With MicroBlocks, you can build your own “real world” digital instruments, interactive jewelry, electronic games and measuring devices, all by using blocks.

You can download MicroBlocks from [here](#).

CHAPTER 11

lo



Very small and breadboard-friendly with option of male, female or both (stackable headers). Includes a dual-core 240 MHz ESP32 with WiFi and Bluetooth. You can use the WiFi both in station (device) mode and access point mode. It includes traditional Bluetooth as well as BLE 4.0.

On-board is a 3.3V regulator and a battery charging device that enables you to switch between using USB or battery power. The battery is automatically charged in the USB is plugged in. A status light shows if it is charging or fully charged. All ESP32 pins bar the flash pins are exposed and available for your use.

[Buy it in the oddWires store...](#)

11.1 Pins Used

IOT-Bus Pin	Description
5	On-board LED

Note: Pin 5 is the default SS for the vSPI interface and provided you are not using them at the same time it will work just fine for either purpose.

11.2 Libraries

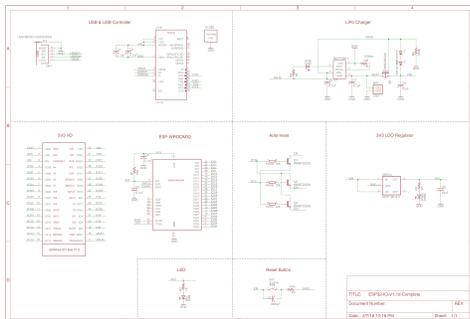
This is not an exhaustive list of libraries available for IoT-Bus but it is a useful list of some of the libraries we have used and especially those we have used for examples. The license for each of the libraries can be found on each library's GitHub page.

Name	Board	Framework	Description
webthing-arduino	Io/Proteus	Arduino	Simple server for WiFi101, ESP8266, or ESP32 boards compliant with Mozilla's proposed WoT API
ESPAsyncWebServer	Io/Proteus	Arduino	Asynchronous HTTP and WebSocket Server for ESP32
ArduinoJson	Io/Proteus	Arduino	C++ JSON library for IoT. Simple and efficient.
arduino-Lora	LoRa	Arduino	Sandeep Mistry's arduino-LoRa library
Arduino-CAN	CAN Bus	Arduino	Sandeep Mistry's Arduino-CAN library
Arduino-OBD2	CAN Bus	Arduino	Sandeep Mistry's Arduino-OBD2 library requires Arduino-CAN
Adafruit_ILI9341	Display	Arduino	Adafruit's ILI9341 library - use with Adafruit-GFX-Library.
Adafruit-GFX-Library	Display	Arduino	Adafruit's general purpose graphics library.
TFTeSPI	Display	Arduino	Bodmer's TFT library has been forked to setup defaults.
XPT2046_Touchscreen	Display	Arduino	Fork of Paul Stoffgren's XPT2046_Touchscreen library.
ESP32_TFT_library	Display	esp-idf	loboris TFT library for ESP32
Adafruit_Motor_Shield	Motor Library	Arduino	Adafruit V2 Motor Shield library
esp-mqtt	Io/Proteus	esp-idf	Espressif MQTT library
esp-idf-lib	IO/Proteus	esp-idf	UncleRus components for esp-idf framework. Mostly ports from esp-open-rtos

11.3 esp-idf-lib Components

Component	Description	License	Thread safety
i2cdev	I2C utilities	MIT	Yes
ds1307	Driver for DS1307 RTC module	BSD	Yes
ds3231	Driver for DS3231 high precision RTC module	MIT	Yes
hmc5883l	Driver for HMC5883L 3-axis digital compass	BSD	Yes
onewire	Bit-banging one wire driver	MIT*	No
ds18x20	Driver for DS18B20/DS18S20 families of one-wire temperature sensor ICs	BSD	No
dht	Driver for DHT11/DHT22 temperature and humidity sensors	BSD	No
bmp180	Driver for BMP180 digital pressure sensor	MIT	Yes
bmp280	Driver for BMP280/BME280 digital pressure sensor	MIT	Yes
bh1750	Driver for BH1750 light sensor	BSD	Yes
ultrasonic	Driver for ultrasonic range meters, e.g. HC-SR04, HY-SRF05	BSD	No
pcf8574	Driver for PCF8574 remote 8-bit I/O expander for I2C-bus	MIT	Yes
hd44780	Universal driver for HD44780 LCD display	BSD	No
pca9685	Driver for 16-channel, 12-bit PWM PCA9685	BSD	Yes
ms5611	Driver for barometric pressure sensor MS5611-01BA03	BSD	Yes
ads111x	Driver for ADS1113/ADS1114/ADS1115 I2C ADC	BSD	Yes
pcf8591	Driver for 8-bit ADC and an 8-bit DAC PCF8591	BSD	Yes
tsl2561	Driver for light-to-digital converter TSL2561	BSD	Yes
max7219	Driver for 8-Digit LED display drivers, MAX7219/MAX7221	BSD	Yes
mcp23017	Driver for 16-bit I2C GPIO expander	BSD	Yes
tda74xx	Driver for TDA7439/TDA7439DS/TDA7440D audio-processors	MIT	Yes

11.4 Schematic

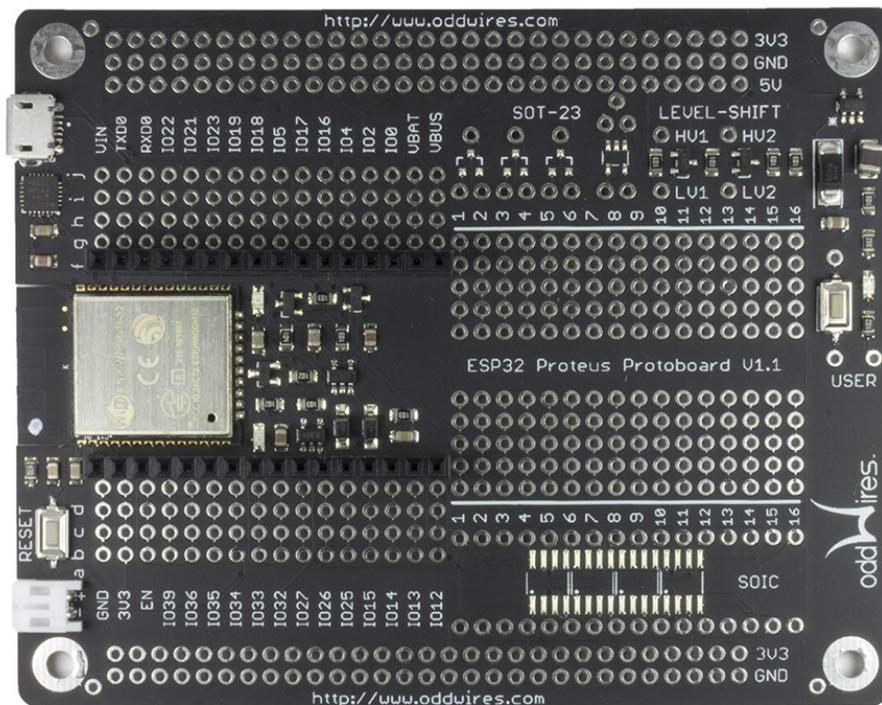


11.5 Platforms

Name	Description
<i>Espressif32</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

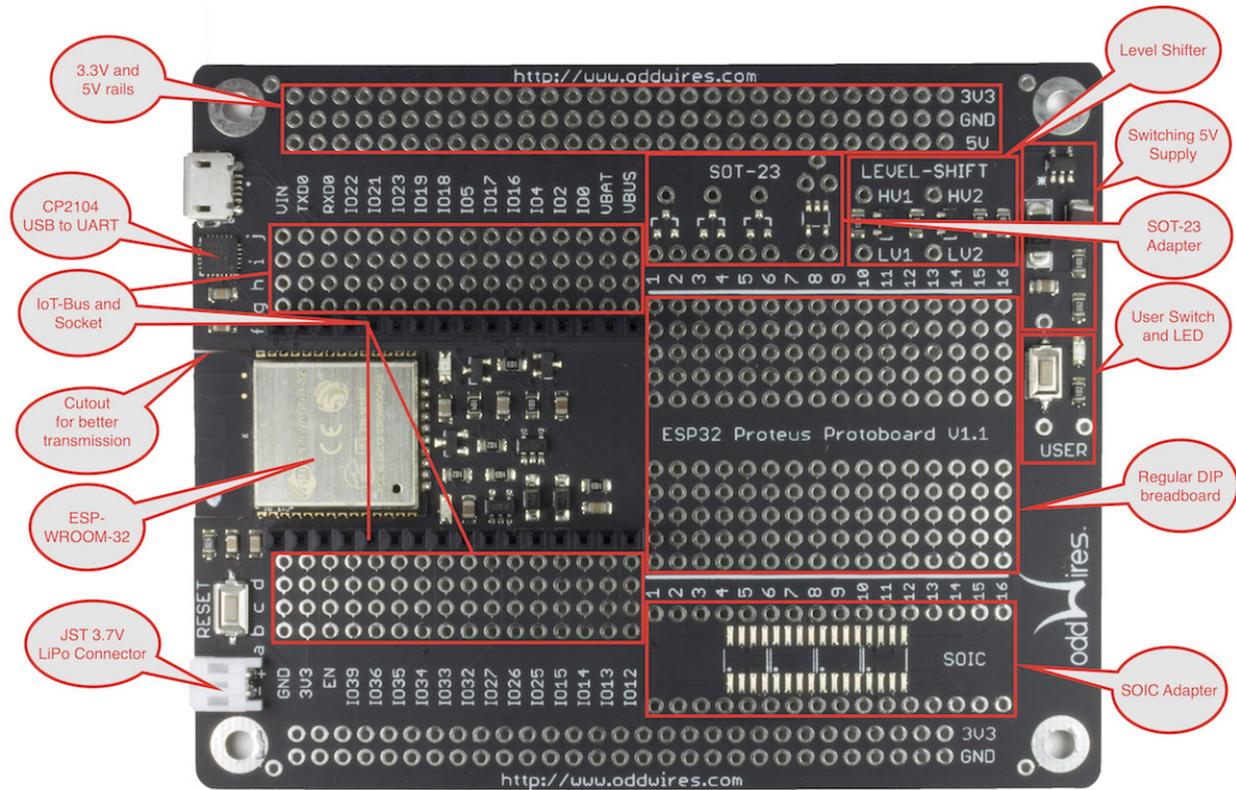
11.6 Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.



This board is larger and designed to make it possible to add your own circuitry to make a complete IoT solution on one board. It includes a dual-core 240 MHz ESP32 with WiFi and Bluetooth. You can use the WiFi both in station (device) mode and access point mode. It includes traditional Bluetooth as well as BLE 4.0. On-board is a 3.3V regulator and a battery charging device that enables you to switch between using USB or battery power.

The battery is automatically charged in the USB is plugged in. A status light shows if it is charging or fully charged. All ESP32 pins bar the flash pins are exposed and available for your use.



The board includes a large prototyping area that includes room for traditional DIP and through-hole components as well as SMD parts such as SOIC and SOT-23. A user LED and switch is included but not connected to any pins so you can use them how you wish. Two level shifters are included so you can interface with 5V devices.

The Proteus includes both 3.3V and 5V rails. Both these rails are available whether powered by the USB or the battery as the 5V is derived from the lower voltage.

Buy it in the oddWires store...

12.1 Pins Used

Pin	Description
None	No pins are utilized by the Proteus board

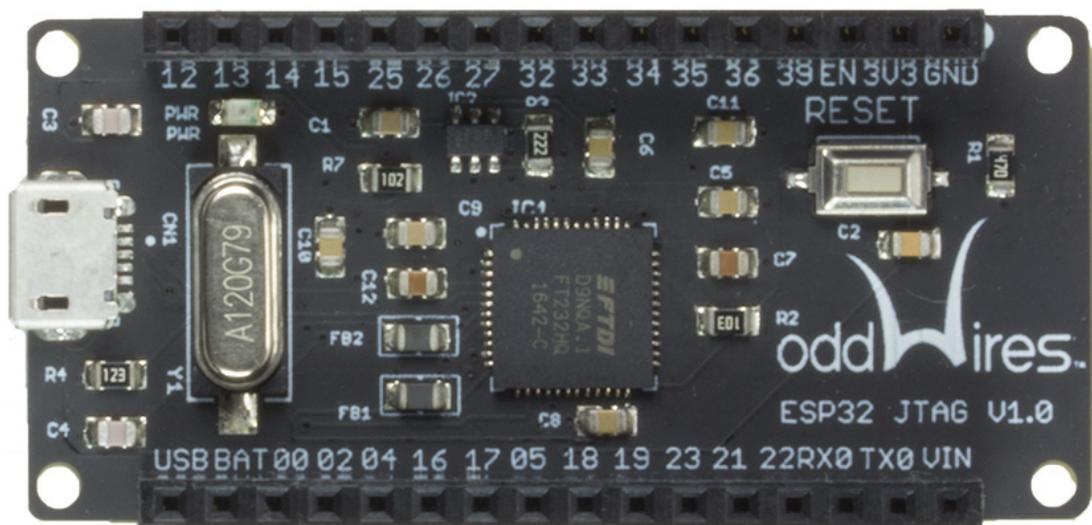
12.2 Libraries

This is not an exhaustive list of libraries available for IoT-Bus but it is a useful list of some of the libraries we have used and especially those we have used for examples. The license for each of the libraries can be found on each library's GitHub page.

Name	Board	Framework	Description
webthing-arduino	Io/Proteus	Arduino	Simple server for WiFi101, ESP8266, or ESP32 boards compliant with Mozilla's proposed WoT API
ESPAsyncWebServer	Io/Proteus	Arduino	Asynchronous HTTP and WebSocket Server for ESP32
ArduinoJson	Io/Proteus	Arduino	C++ JSON library for IoT. Simple and efficient.
arduino-Lora	LoRa	Arduino	Sandeep Mistry's arduino-LoRa library
Arduino-CAN	CAN Bus	Arduino	Sandeep Mistry's Arduino-CAN library
Arduino-OB2	CAN Bus	Arduino	Sandeep Mistry's Arduino-OB2 library requires Arduino-CAN
Adafruit_ILI9341	Display	Arduino	Adafruit's ILI9341 library - use with Adafruit-GFX-Library.
Adafruit-GFX-Library	Display	Arduino	Adafruit's general purpose graphics library.
TFTeSPI	Display	Arduino	Bodmer's TFT library has been forked to setup defaults.
XPT2046_Touchscreen	Display	Arduino	Fork of Paul Stoffgren's XPT2046_Touchscreen library.
ESP32_TFT_library	Display	esp-idf	loboris TFT library for ESP32
Adafruit_Motor_Shield	Motor Library	Arduino	Adafruit V2 Motor Shield library
esp-mqtt	Io/Proteus	esp-idf	Espressif MQTT library
esp-idf-lib	IO/Proteus	esp-idf	UncleRus components for esp-idf framework. Mostly ports from esp-open-rtos

12.3 esp-idf-lib Components

Component	Description	License	Thread safety
i2cdev	I2C utilities	MIT	Yes
ds1307	Driver for DS1307 RTC module	BSD	Yes
ds3231	Driver for DS3231 high precision RTC module	MIT	Yes
hmc5883l	Driver for HMC5883L 3-axis digital compass	BSD	Yes
onewire	Bit-banging one wire driver	MIT*	No
ds18x20	Driver for DS18B20/DS18S20 families of one-wire temperature sensor ICs	BSD	No
dht	Driver for DHT11/DHT22 temperature and humidity sensors	BSD	No
bmp180	Driver for BMP180 digital pressure sensor	MIT	Yes
bmp280	Driver for BMP280/BME280 digital pressure sensor	MIT	Yes
bh1750	Driver for BH1750 light sensor	BSD	Yes
ultrasonic	Driver for ultrasonic range meters, e.g. HC-SR04, HY-SRF05	BSD	No
pcf8574	Driver for PCF8574 remote 8-bit I/O expander for I2C-bus	MIT	Yes
hd44780	Universal driver for HD44780 LCD display	BSD	No
pca9685	Driver for 16-channel, 12-bit PWM PCA9685	BSD	Yes
ms5611	Driver for barometric pressure sensor MS5611-01BA03	BSD	Yes
ads111x	Driver for ADS1113/ADS1114/ADS1115 I2C ADC	BSD	Yes
pcf8591	Driver for 8-bit ADC and an 8-bit DAC PCF8591	BSD	Yes
tsl2561	Driver for light-to-digital converter TSL2561	BSD	Yes
max7219	Driver for 8-Digit LED display drivers, MAX7219/MAX7221	BSD	Yes
mcp23017	Driver for 16-bit I2C GPIO expander	BSD	Yes
tda74xx	Driver for TDA7439/TDA7439DS/TDA7440D audio-processors	MIT	Yes



This IoT-Bus module provides JTAG debugging for the *Io* and *Proteus* boards (can be used with other boards too, see wiring connections below).

Both the *Io* and *Proteus* processor boards can accept a specially designed JTAG board offering hardware debugging. Our JTAG board is based on the FT232H and it enables comprehensive JTAG debugging support. You can use OpenOCD and GDB in combination to use it but our recommendation is to use PlatformIO. PlatformIO has taken away all the hard work of configuring OpenOCD and GDB. You simply select it is your debugging choice as described [here](#). Take a look at how easy it is to use with PlatformIO's [Unified Debugger](#). Just plug it in and start debugging! No more printing to the terminal!

Buy it in the oddWires store...

13.1 PlatformIO Configuration

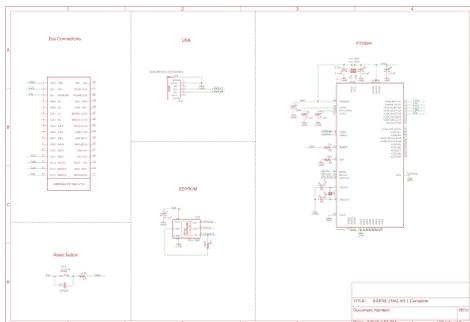
You can configure the debugging tool using the `debug_tool` option in `platformio.ini`. If you are using an Io board (`iot-busio` in Platformio) or a Proteus board (`iotbusproteus` in PlatformIO) then you do not need to explicitly set `debug_tool` as it will be set implicitly:

```
[env:myenv]
platform = ...
board = ...
debug_tool = iot-bus-jtag
```

13.2 Pins Used

IOT-Bus JTAG Pin	Board JTAG Pin
3V3	Positive Supply Voltage — Power supply for JTAG interface drivers
GND	GND - Digital Ground
12	TDI - Test Data In pin
14	TMS - Test Mode State pin
15	TCK - JTAG Return Test Clock
13	TDO - Test Data Out pin
EN	RESET

13.3 Schematic



Click image to enlarge.

13.4 Platforms

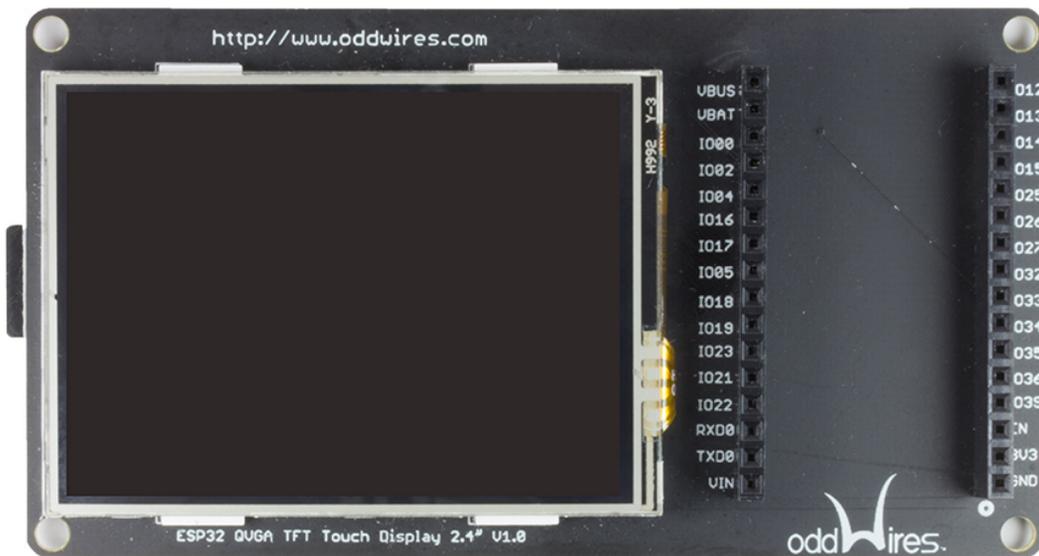
Name	Description
<i>Espressif32</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

13.5 Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

CHAPTER 14

2.4" QVGA Touch Display



This is a nice 2.4" 320x240 QVGA TFT Touch Display offering plug and play display output and touch sensing together with a 4-bit SDMMC SD Card. We picked 2.4" over 2.8" as it has a crisper display at 320 x 240 resolution and its slightly smaller size helps in IoT applications. Designed primarily for development use it has an IoT-Bus socket at the side for easy access. Here's the back view:



Buy it in the oddWires store...

14.1 Pins Used

IOT-Bus Pin	Description
2	DAT0 (SD Card)
4	DAT1 (SD Card)
5	SS (TFT)
12	DAT2 (SD Card)
13	DAT2 (SD Card)
14	CLK (SD Card)
15	CMD (SD)
16	SS (Touch Screen)
17	IRQ (Touch Screen)
18	SCK (TFT)
19	MISO (TFT)
23	MOSI (TFT)
27	DC (TFT)
33	Backlight (TFT) - you will not see anything if you do not turn on the backlight!
EN	RESET (TFT)
3V3	Power
GND	Ground

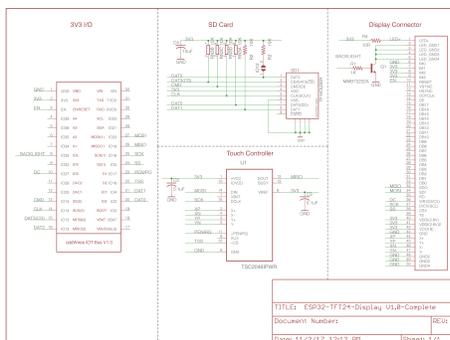
Note: This board uses a lot of pins if you are using everything. If you are not using the SDCard then those pins

may be freely used. You can also use the SCard in one pin mode freeing up DAT1, DAT2 and DAT3. If you are not using the touch capabilities of the module then you can utilize those pins. IRQ is not used by the forked version of the XPT2046 library freeing up pin 17. The current versions of the IoT-Bus CAN Bus and LoRa modules cannot be used with this display.

14.2 Libraries

Name	Description	Framework
Adafruit_ILI9341	Arduino	Adafruit's ILI9341 library - use with Adafruit-GFX-Library.
Adafruit-GFX-Library	Arduino	Adafruit's general purpose graphics library.
TFTeSPI	Arduino	Bodmer's TFT library has been forked to setup defaults.
XPT2046_Touchscreen	Arduino	Fork of Paul Stoffgren's XPT2046_Touchscreen library.
ESP32_TFT_library	esp-idf	loboris TFT library for ESP32

14.3 Schematic



Click image to enlarge.

14.4 Platforms

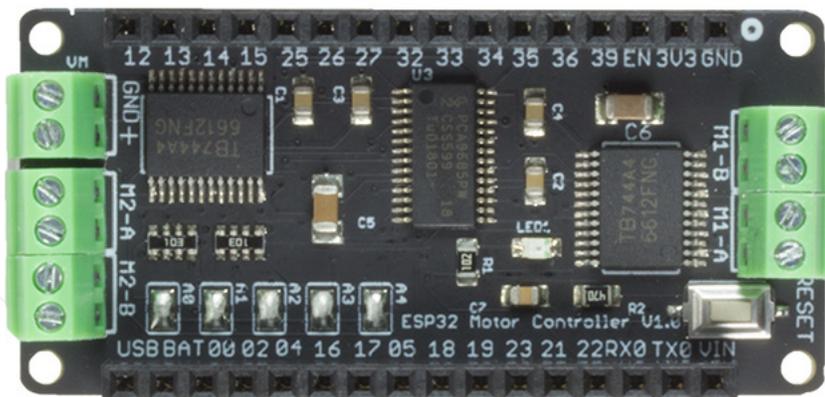
Name	Description
<i>Espressif32</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

14.5 Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

CHAPTER 15

Motor



Buy it in the oddWires store...

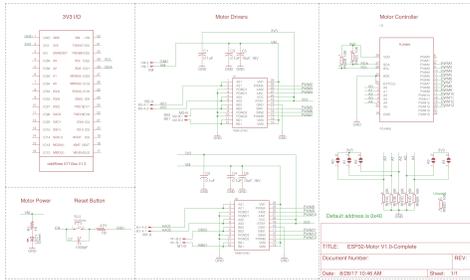
15.1 Pins Used

IOT-Bus Pin	Description
21	SDA
22	SCL
3V3	Power
GND	Ground

15.2 Libraries

Name	Framework	Description
Adafruit_Motor_Shield_V2_Library	Arduino	Adafruit V2 Motor Shield library

15.3 Schematic



Click image to enlarge.

15.4 Platforms

Name	Description
<i>Espressif32</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

15.5 Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.



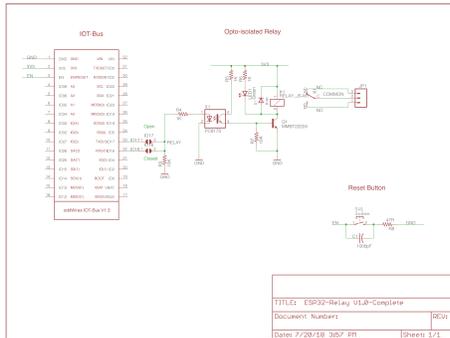
This is an opto-isolated relay board driven by a single digital pin. It is a 110V, 10A maximum AC relay board in the IoT-Bus form factor.

Buy it in the oddWires store...

16.1 Pins Used

IOT-Bus Pin	Description
17	Relay
16	Relay (alternative jumper)

16.2 Schematic



Click image to enlarge.

16.3 Platforms

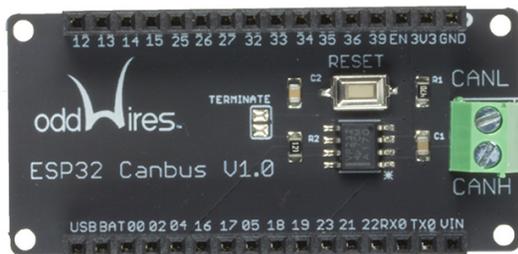
Name	Description
<i>Espressif32</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

16.4 Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

CHAPTER 17

CAN Bus



Buy it in the oddWires store...

17.1 Pins Used

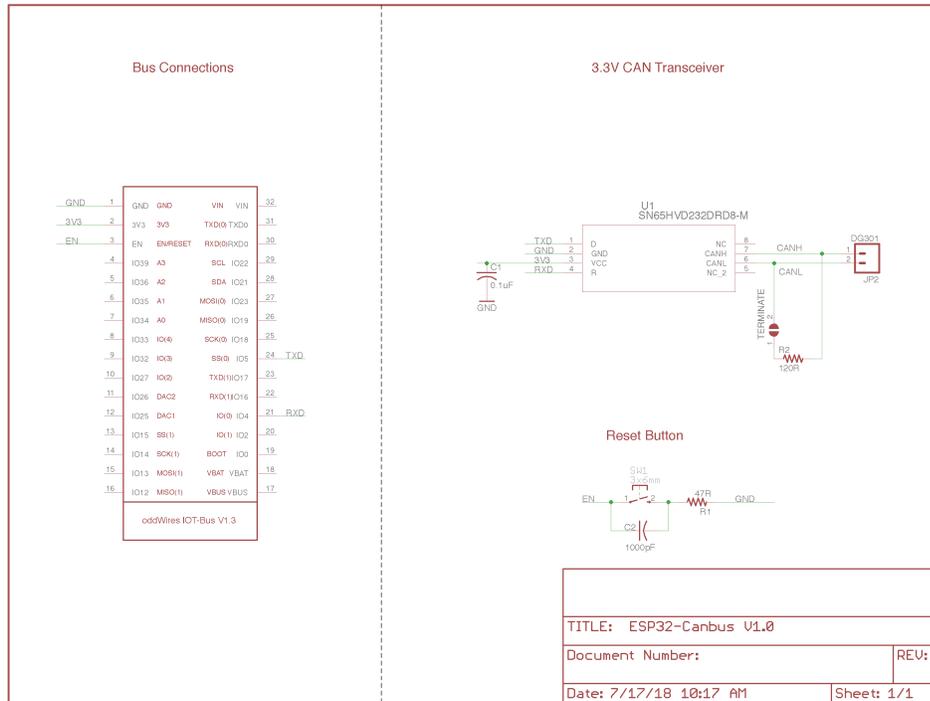
IOT-Bus Pin	Description
4	CAN Bus RXD
5	CAN Bus TXD

Note: CAN Bus cannot be used at the same time as LoRa.

17.2 Libraries

Name	Framework	Description
Arduino-CAN	Arduino	Sandeep Mistry's Arduino-CAN library
Arduino-OBD2	Arduino	Sandeep Mistry's Arduino-OBD2 library requires Arduino-CAN

17.3 Schematic



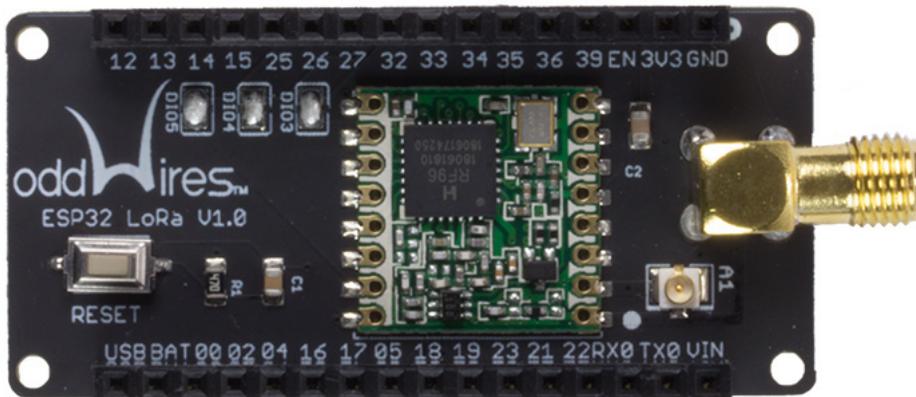
Click image to enlarge.

17.4 Platforms

Name	Description
Espressif32	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

17.5 Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.



This IoT-Bus module utilizes the Hope RFM95 to offer low-cost, LoRa radio transmission and a Wi-Fi/LoRa gateway. It uses the correct 915 MHz rather than the 433 MHz european standard often found. The RFM95W transceivers feature the LoRa long range modem that provides ultra-long range spread spectrum communication and high interference immunity whilst minimizing current consumption.

Using Hope RF's patented LoRa modulation technique RFM95W can achieve a sensitivity of over -148dBm using a low cost crystal and bill of materials. The high sensitivity combined with the integrated +20 dBm power amplifier yields industry-leading link budget making it optimal for any application requiring range or robustness.

LoRa also provides significant advantages in both blocking and selectivity over conventional modulation techniques, solving the traditional design compromise between range, interference immunity and energy consumption. These devices also support high performance (G)FSK modes for systems including WMBus, IEEE802.15.4g. The RFM95W deliver exceptional phase noise, selectivity, receiver linearity and IIP3 for significantly lower current consumption than

competing devices.

Features of RF Transceiver LoRa Module RFM95W:

- LoRa Modem.
- 915 MHz
- 168 dB maximum link budget.
- +20 dBm - 100 mW constant RF output vs. V supply.
- +14 dBm high efficiency PA.
- Programmable bit rate up to 300 Kbps.
- High sensitivity: down to -148 dBm.
- Bullet-proof front end: IIP3 = -12.5 dBm.
- Excellent blocking immunity.
- Low RX current of 10.3 mA, 200 nA register retention.
- Fully integrated synthesizer with a resolution of 61 Hz.
- FSK, GFSK, MSK, GMSK, LoRa and OOK modulation.
- Built-in bit synchronizer for clock recovery.
- Preamble detection.
- 127 dB Dynamic Range RSSI.
- Automatic RF Sense and CAD with ultra-fast AFC.
- Packet engine up to 256 bytes with CRC.
- Built-in temperature sensor and low battery indicator.
- Module Size 16* 16mm

Potential Applications of the RF Transceiver Module RFM95W:

- Automated Meter Reading.
- Home and Building Automation.
- Wireless Alarm and Security Systems.
- Industrial Monitoring and Control
- Long range Irrigation Systems

[Buy it in the oddWires store...](#)

18.1 Pins Used

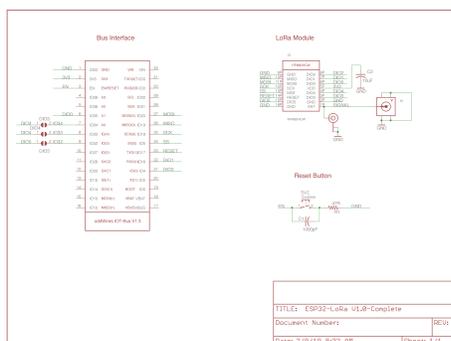
IOT-Bus Pin	Description
4	DIO2
5	SS
16	DIO1
17	LoRa RESET
18	SCK
19	MISO
23	MOSI
32	DIO5(J)
33	DIO4(J)
34	DIO3(J)
35	DIO0(J)
3V3	Power
GND	Digital Ground

Note: LoRa cannot be used at the same time as CAN Bus.

18.2 Libraries

Name	Framework	Description
arduino-Lora	Arduino	Sandeep Mistry's arduino-LoRa library

18.3 Schematic



Click image to enlarge.

18.4 Platforms

Name	Description
Espressif32	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

18.5 Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

IoT-Bus Examples Index

You can find the examples on the [iot-bus GitHub page](#). The examples are in two repositories:

iot-bus-examples-platformio Examples in c++ format and platformio.ini files that contain library dependencies.

iot-bus-examples-arduino Examples in ino format. You will need to separately install library dependencies described in each example.

The examples that are described in this section are a subset of the examples in the repository. If you run through these examples you'll discover how to use each of them.

Blink Get started with the ubiquitous Blink and flash the on-board LED.

Hello World A simple WiFi web-server that responds to HTTP requests.

CAN Bus How to send and receive data using CAN Bus.

Relay Demonstrates how to switch the relay on and off.

LoRa How to send and receive data using LoRa.

Motor How to control a stepper motor.

Display Graphics and touch screen example.

SD Card A test for the SD Card on the IoT-Bus Display.

CHAPTER 20

IoT-Bus Blink Example

This very simple example needs little explanation. The first line includes the Arduino framework.

```
#include <arduino.h>
```

The next line defined the GPIO of the pin of the onboard LED.

```
#define LEDPin 5
```

This line sets the GPIO pin into output mode.

```
pinMode(LEDPin, OUTPUT);
```

This line turns on the LED.

```
digitalWrite(LEDPin, HIGH);
```

This line turns off the LED.

```
digitalWrite(LEDPin, LOW);
```

This line creates 1 second delay.

```
delay(1000);
```

The full example.

```
#include <arduino.h>

#define LEDPin 5

// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin as an output.
    pinMode(LEDPin, OUTPUT);
```

(continues on next page)

(continued from previous page)

```
}  
  
// the loop function runs over and over again forever  
void loop() {  
  digitalWrite(LEDPin, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000); // wait for a second  
  digitalWrite(LEDPin, LOW); // turn the LED off by making the voltage LOW  
  delay(1000); // wait for a second  
}
```

CHAPTER 21

IoT-Bus Hello World Example

This example shows the simplest way to create a web-server and uses a truly asynchronous approach.

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
```

The WiFi library is required to be able to connect to WiFi. ESPAsyncWebServer provides the asynchronous web-server and AsyncTCP is required for an asynchronous TCP stack to support it.

```
const char* ssid = ".....";
const char* password = ".....";
```

Enter the ssid and password of the local WiFi network you want to connect to.

```
Serial.begin(115200);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
}

Serial.println(WiFi.localIP());
```

Start serial output, startup wireless and wait for connection. Print out the ip so you can connect to it from your browser. This loop will never end if you do not have a correct ssid and password.

```
server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request){
  request->send(200, "text/html", "<p>Hello World!</p>");
});

server.on("/html", HTTP_GET, [] (AsyncWebServerRequest *request){
```

(continues on next page)

(continued from previous page)

```
request->send(200, "text/html", "<p>Some HTML!</p>");
});
```

This section tells the server that on receiving an HTTP get request for '/' or for '/html' respond with a normal HTTP response (200) and provide some html response.

The examples provided with ESPAsyncWebServer are somewhat more sophisticated. Take a look at them.

```
server.begin();
```

Start the web-server.

The full example.

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

const char* ssid = ".....";
const char* password = ".....";

AsyncWebServer server(80);

void setup(){
  Serial.begin(115200);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }

  Serial.println(WiFi.localIP());

  server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request){
    request->send(200, "text/html", "<p>Hello World!</p>");
  });

  server.on("/html", HTTP_GET, [] (AsyncWebServerRequest *request){
    request->send(200, "text/html", "<p>Some HTML!</p>");
  });

  server.begin();
}

void loop(){
}
```

IoT-Bus Touch Draw Example

This example demonstrates the use of both the IoT-Bus Display and the touchscreen. In this example we are using a forked version of Bodmer's [TFT_eSPI](#) library with `User_Setup` already setup for IoT-Bus Display which you can find [here](#). We are also using a forked version of Paul Stoffgren's [XPT2046_Touchscreen](#) library which has been enhanced to perform mapping of raw touch Data to screen coordinates. It has also been modified to disable the PENIRQ line so that we use one less pin. You will find other examples in the examples repository that use Adafruit libraries that work in a very similar way.

```
// include touchscreen library
#include <XPT2046_Touchscreen.h>

// Call up the TFT driver library
#include <TFT_eSPI.h> // Hardware-specific library
#include <SPI.h>

// Invoke custom TFT driver library
TFT_eSPI tft = TFT_eSPI(); // Invoke custom library

// These pins are defined in User_Setup.h and have already been setup to be correct_
↳for the IoT-Bus Display

//#define TFT_MISO 19
//#define TFT_MOSI 23
//#define TFT_SCLK 18
//#define TFT_CS 5 // Chip select control pin
//#define TFT_DC 27 // Data Command control pin
//#define TFT_RST -1 // Set TFT_RST to -1 if display RESET is connected to ESP32_
↳board RST

/* Create an instance of the touch screen library */
#define CS_PIN 16

XPT2046_Touchscreen ts(CS_PIN); // Param 2 - NULL - No interrupts
```

In the initial section of the example we include the libraries required, declare both the display and touch screen for

use. We also define the CS_PIN as required for the touchscreen to be GPIO 16.

There is no need to define the rest of the pins as that has been done for you in User_Setup.h which is part of the TFT_eSPI library.

```
// touchscreen calibration values
#define X_MIN 256
#define X_MAX 3632
#define Y_MIN 274
#define Y_MAX 3579
```

These calibration values will vary from display to display. You can use the draw-corners example to find out the values for each corner and change them for your displays. The values here are likely to be OK for initial testing but you will get more accurate results by performing your own calibration.

```
pinMode(33, OUTPUT);
digitalWrite(33, HIGH);
```

These lines are critical as GPIO 33 is used to power on the back-light. The back-light is driven through a transistor to avoid drawing too much current from the GPIO.

```
ts.begin();
ts.setCalibration(X_MIN, X_MAX, Y_MIN, Y_MAX);

tft.init();
```

These lines initialize and calibrate the touchscreen and display.

```
// Set the TFT and touch screen to landscape orientation
tft.setRotation(1);
ts.setRotation(1);
```

Rotation 0 and 2 are portrait and 1 and 3 are landscape. This example will work in any of the rotations so you can change them and see the effect. Remember to keep them in sync or you will get strange effects.

```
tft.setTextSize(1);
tft.fillRect(TFT_BLACK);
tft.setTextColor(TFT_GREEN);
```

This sets the default font-size, sets the background to black and sets the current text color. Note that much better smooth fonts are easily usable - take a look at the TFT_eSPI library documentation.

```
swatchWidth = ts.getWidth()/10;
swatchHeight = 34;
```

This example dynamically determines the display width so that it works in any orientation.

```
tft.fillRect(i * swatchWidth, 0, swatchWidth, swatchHeight, colors[i]);
```

This fills a rectangle with a color.

```
tft.setCursor(ts.getWidth()-swatchWidth*1.5, 3, 2); // x,y,font
tft.setTextColor(TFT_WHITE);
tft.print("Clear");
```

These lines position the cursor, set the text color and write some text at the cursor position.

```
if (ts.touched())
```

Use the touched function to find out whether the display has been touched.

```
TS_Point p = ts.getMappedPoint();
```

This will get an x, y and z value. Although it is not used here you could see how hard the press was by comparing the z value. Note that x and y are relative to the current origin which will vary by rotation. Note that the origin is always in the top left corner of the display as is traditional with graphical displays.

In the remainder of the loop() function, some hit-testing is performed and if the press is on a color of the palette, the current color is changed. If the clear button is pressed, then the screen is cleared to the current color. Otherwise the point is drawn.

The full example is shown below.

```
// include touchscreen library
#include <XPT2046_Touchscreen.h>

// Call up the TFT driver library
#include <TFT_eSPI.h> // Hardware-specific library
#include <SPI.h>

// Invoke custom TFT driver library
TFT_eSPI tft = TFT_eSPI(); // Invoke custom library

// These pins are defined in User_Setup.h and have already been setup to be correct_
↳for the IoT-Bus Display

//#define TFT_MISO 19
//#define TFT_MOSI 23
//#define TFT_SCLK 18
//#define TFT_CS 5 // Chip select control pin
//#define TFT_DC 27 // Data Command control pin
//#define TFT_RST -1 // Set TFT_RST to -1 if display RESET is connected to ESP32_
↳board RST

/* Create an instance of the touch screen library */
#define CS_PIN 16

XPT2046_Touchscreen ts(CS_PIN); // Param 2 - NULL - No interrupts

int color = TFT_WHITE; //Starting paint brush color

// Palette button colour sequence
unsigned int colors[10] = {TFT_RED, TFT_GREEN, TFT_BLUE, TFT_BLACK, TFT_CYAN, TFT_
↳YELLOW, TFT_WHITE, TFT_MAGENTA, TFT_BLACK, TFT_BLACK};

// touchscreen calibration values
#define X_MIN 256
#define X_MAX 3632
#define Y_MIN 274
#define Y_MAX 3579

int swatchWidth;
int swatchHeight;

void setup()
```

(continues on next page)

(continued from previous page)

```
{
Serial.begin(115200);

pinMode(33, OUTPUT);
digitalWrite(33, HIGH);

ts.begin();
ts.setCalibration(X_MIN, X_MAX, Y_MIN, Y_MAX);

tft.init();

// Set the TFT and touch screen to landscape orientation
tft.setRotation(3);
ts.setRotation(3);

tft.setTextSize(1);
tft.fillScreen(TFT_BLACK);
tft.setTextColor(TFT_GREEN);

swatchWidth = ts.getWidth()/10;
swatchHeight = 34;

//Draw the palette
for (int i = 0; i < 10; i++)
{
    tft.fillRect(i * swatchWidth, 0, swatchWidth, swatchHeight, colors[i]);
}

//Draw the clear screen button
tft.setCursor(ts.getWidth()-swatchWidth*1.5, 3, 2); // x,y,font
tft.setTextColor(TFT_WHITE);
tft.print("Clear");
tft.drawRect(0, 0, ts.getWidth()-1, swatchHeight, TFT_WHITE);

// Plot the current colour in the screen clear box
tft.fillRect(ts.getWidth() - swatchWidth, 20, 12, 12, color);
}

/* Main program */
void loop()
{
    // Check if the touch screen is currently pressed
    // Raw and coordinate values are stored within library at this instant

    if (ts.touched())
    {
        Serial.println("touched");
        // Read the current X and Y axis as mapped co-ordinates at the last touch time

        TS_Point p = ts.getMappedPoint();

        // mapped pixel
        Serial.print(p.x); Serial.print(","); Serial.println(p.y);

        // Detect paint brush color change
        if (p.y < swatchHeight + 2)
        {
```

(continues on next page)

(continued from previous page)

```
if (p.x / swatchWidth > 7)
{
    // Clear the screen to current color
    tft.fillRect(0, swatchHeight, ts.getWidth(), ts.getHeight()-1, color);
    Serial.println("clear screen to current color");
}
else
{
    color = colors[p.x / swatchWidth];
    // Update the current color in the clear box
    tft.fillRect(ts.getWidth() - swatchWidth, 20, 12, 12, color);
    Serial.println("Update the current color in the clear box");
}
}
else
{
    // draw a point
    tft.fillCircle(p.x, p.y, 2, color);
    Serial.println("fillcircle");
}
}
}
```

IoT-Bus Relay Example

It is very easy to use the IoT-Bus relay. Simply set the relay pin high to turn on the relay as the relay is active high. The full example is shown below.

```
/*
Turns the Relay on for one second, then off for one second, repeatedly.
This example code is in the public domain.
*/

#include <arduino.h>

#define RelayPin 17

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin Relay_BUILTIN as an output.
  pinMode(RelayPin, OUTPUT);
  Serial.begin(115200);
  Serial.println("Hello world");
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(RelayPin, HIGH); // turn the Relay on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(RelayPin, LOW); // turn the Relay off by making the voltage LOW
  delay(1000); // wait for a second
}
```

IoT-Bus CAN Bus Example

These two examples work together using Sandeep Mistry's excellent arduino-CAN library which can be found [here](#).

24.1 CAN Bus Send

```
#include <CAN.h>

void setup() {
  Serial.begin(115200);

  // start the CAN bus at 1000 kbps
  if (!CAN.begin(1000E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }
}
```

Include the CAN Bus library and start it up at 1Mbps.

```
CAN.beginPacket(0x12);
CAN.write('h');
CAN.write('e');
CAN.write('l');
CAN.write('l');
CAN.write('o');
CAN.endPacket();
```

Send a regular CAN packet.

```
CAN.beginExtendedPacket(0xabcdef);
CAN.write('w');
CAN.write('o');
CAN.write('r');
```

(continues on next page)

(continued from previous page)

```
CAN.write('l');
CAN.write('d');
CAN.endPacket();
```

Send an extended CAN packet. The program loops continuously sending packets. The full example is shown below.

```
// Copyright (c) Sandeep Mistry. All rights reserved.
// Licensed under the MIT license. See LICENSE file in the project root for full
↪license information.

#include <CAN.h>

void setup() {
  Serial.begin(115200);

  // start the CAN bus at 1000 kbps
  if (!CAN.begin(1000E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }
}

void loop() {
  // send packet: id is 11 bits, packet can contain up to 8 bytes of data
  Serial.print("Sending packet ... ");

  CAN.beginPacket(0x12);
  CAN.write('h');
  CAN.write('e');
  CAN.write('l');
  CAN.write('l');
  CAN.write('o');
  CAN.endPacket();

  Serial.println("done");

  delay(1000);

  // send extended packet: id is 29 bits, packet can contain up to 8 bytes of data
  Serial.print("Sending extended packet ... ");

  CAN.beginExtendedPacket(0xabcdef);
  CAN.write('w');
  CAN.write('o');
  CAN.write('r');
  CAN.write('l');
  CAN.write('d');
  CAN.endPacket();

  Serial.println("done");

  delay(1000);
}
```

24.2 CAN Bus Receive

```
#include <CAN.h>

void setup() {
  Serial.begin(115200);
  while (!Serial);

  Serial.println("CAN Receiver");

  // start the CAN bus at 1000 kbps
  if (!CAN.begin(1000E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }
}
```

Include the CAN Bus library and start it up at 1Mbps.

```
// try to parse packet
int packetSize = CAN.parsePacket();

if (packetSize) {
  // received a packet
  Serial.print("Received ");
```

See if we have received a packet and get its size.

```
if (CAN.packetExtended()) {
  Serial.print("extended ");
}
```

Identify an extended packet.

```
if (CAN.packetRtr()) {
  // Remote transmission request, packet contains no data
  Serial.print("RTR ");
}
```

Check if it is a remote transmission request.

```
Serial.print(CAN.packetId(), HEX);
```

Print out the packet id.

```
Serial.println(CAN.packetDlc());
```

If it is a remote transmission request, find out the requested length.

```
Serial.print(" and length ");
Serial.println(packetSize);

// only print packet data for non-RTR packets
while (CAN.available()) {
  Serial.print((char)CAN.read());
}
Serial.println();
```

If it is a regular packet, get its size and read the packet. The full example is shown below.

```
// Copyright (c) Sandeep Mistry. All rights reserved.
// Licensed under the MIT license. See LICENSE file in the project root for full
↳license information.

#include <CAN.h>

void setup() {
  Serial.begin(115200);
  while (!Serial);

  Serial.println("CAN Receiver");

  // start the CAN bus at 1000 kbps
  if (!CAN.begin(1000E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }
}

void loop() {
  // try to parse packet
  int packetSize = CAN.parsePacket();

  if (packetSize) {
    // received a packet
    Serial.print("Received ");

    if (CAN.packetExtended()) {
      Serial.print("extended ");
    }

    if (CAN.packetRtr()) {
      // Remote transmission request, packet contains no data
      Serial.print("RTR ");
    }

    Serial.print("packet with id 0x");
    Serial.print(CAN.packetId(), HEX);

    if (CAN.packetRtr()) {
      Serial.print(" and requested length ");
      Serial.println(CAN.packetDlc());
    } else {
      Serial.print(" and length ");
      Serial.println(packetSize);

      // only print packet data for non-RTR packets
      while (CAN.available()) {
        Serial.print((char)CAN.read());
      }
      Serial.println();
    }

    Serial.println();
  }
}
```

IoT-Bus LoRa Example

These two examples work together using Sandeep Mistry's excellent arduino-LoRa library which can be found [here](#).

25.1 LoRa Sender

```
#include <SPI.h>
#include <LoRa.h>

#define DIO0 35
#define SS 5
#define RESET 17

int counter = 0;

void setup() {
  LoRa.setPins(SS, RESET, DIO0);
```

These lines include the libraries required and define the DIO0, SS and RESET pins required to use the IoT-BUs LoRa board.

```
if (!LoRa.begin(915E6)) {
  Serial.println("Starting LoRa failed!");
  while (1);
}
```

These lines attempt to start LoRa and will go no further if failure.

```
// send packet
LoRa.beginPacket();
LoRa.print("hello ");
LoRa.print(counter);
LoRa.endPacket();
```

These lines send a packet. The packet is transmitted on endPacket(). The complete example is shown below.

```
#include <SPI.h>
#include <LoRa.h>

#define DIO0 35
#define SS 5
#define RESET 17

int counter = 0;

void setup() {
  LoRa.setPins(SS, RESET, DIO0);
  Serial.begin(115200);
  while (!Serial);

  Serial.println("LoRa Sender");

  if (!LoRa.begin(915E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
}

void loop() {
  Serial.print("Sending packet: ");
  Serial.println(counter);

  // send packet
  LoRa.beginPacket();
  LoRa.print("hello ");
  LoRa.print(counter);
  LoRa.endPacket();

  counter++;

  delay(5000);
}
```

25.2 LoRa Receiver

```
#include <SPI.h>
#include <LoRa.h>

#define DIO0 35
#define SS 5
#define RESET 17

int counter = 0;

void setup() {
  LoRa.setPins(SS, RESET, DIO0);
```

These lines include the libraries required and define the DIO0, SS and RESET pins required to use the IoT-BUs LoRa board.

```

if (!LoRa.begin(915E6)) {
  Serial.println("Starting LoRa failed!");
  while (1);
}

```

These lines attempt to start LoRa and will go no further if failure.

```

// try to parse packet
int packetSize = LoRa.parsePacket();
if (packetSize) {
  // received a packet
  Serial.print("Received packet ");
}

```

These lines will check if a packet has been received and get its size.

```

// read packet
while (LoRa.available()) {
  Serial.print((char) LoRa.read());
}

```

These lines will read the packet until there's no more data.

```

Serial.println(LoRa.packetRssi());

```

This line will print out the signal strength indicator.

```

#include <SPI.h>
#include <LoRa.h>

#define DIO0 35
#define SS 5
#define RESET 17

void setup() {

  LoRa.setPins(SS, RESET, DIO0);
  Serial.begin(115200);
  while (!Serial);

  Serial.println("LoRa Receiver");

  if (!LoRa.begin(915E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
}

void loop() {
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    // received a packet
    Serial.print("Received packet ");

    // read packet
    while (LoRa.available()) {

```

(continues on next page)

(continued from previous page)

```
        Serial.print((char)LoRa.read());
    }

    // print RSSI of packet
    Serial.print(" with RSSI ");
    Serial.println(LoRa.packetRssi());
}
}
```

CHAPTER 26

IoT-Bus Motor Example

This example use the Adafruit_Motorshield library which you can find [here](#) to control a stepper motor on the IoT-Bus Motor board.

```
/*
  This is a test sketch for the iot-bus motor board
 */

#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWM_ServoDriver.h"

// Create the motor shield object with the default I2C address

Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x5F);
```

Include necessary libraries and create a motor controller at address 0x5F. You can change the solder jumpers on the board for another address between 0x40 and 0x5F.

```
// Connect a stepper motor with 200 steps per revolution (1.8 degree)
// to motor port #2 (M3 and M4)
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 1);
```

Create a motor to refer to motor 2 on the motor board.

```
AFMS.begin(); // create with the default frequency 1.6KHz
//AFMS.begin(1000); // OR with a different frequency, say 1KHz
```

Start up the motor.

```
Serial.println("Single coil steps");
myMotor->step(100, FORWARD, SINGLE);
myMotor->step(100, BACKWARD, SINGLE);

Serial.println("Double coil steps");
```

(continues on next page)

(continued from previous page)

```
myMotor->step(100, FORWARD, DOUBLE);
myMotor->step(100, BACKWARD, DOUBLE);

Serial.println("Interleave coil steps");
myMotor->step(100, FORWARD, INTERLEAVE);
myMotor->step(100, BACKWARD, INTERLEAVE);

Serial.println("Microstep steps");
myMotor->step(50, FORWARD, MICROSTEP);
myMotor->step(50, BACKWARD, MICROSTEP);
```

Move the stepper motor forward and backward in different ways. The full example is shown below.

```
/*
This is a test sketch for the iot-bus motor board
*/

#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWM_ServoDriver.h"

// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x5F);

// Connect a stepper motor with 200 steps per revolution (1.8 degree)
// to motor port #2 (M3 and M4)
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 1);

void setup() {
  Serial.begin(115200);
  Serial.println("Stepper test!");

  AFMS.begin(); // create with the default frequency 1.6KHz
  //AFMS.begin(1000); // OR with a different frequency, say 1KHz

  myMotor->setSpeed(10); // 10 rpm
}

void loop() {
  Serial.println("Single coil steps");
  myMotor->step(100, FORWARD, SINGLE);
  myMotor->step(100, BACKWARD, SINGLE);

  Serial.println("Double coil steps");
  myMotor->step(100, FORWARD, DOUBLE);
  myMotor->step(100, BACKWARD, DOUBLE);

  Serial.println("Interleave coil steps");
  myMotor->step(100, FORWARD, INTERLEAVE);
  myMotor->step(100, BACKWARD, INTERLEAVE);

  Serial.println("Microstep steps");
  myMotor->step(50, FORWARD, MICROSTEP);
  myMotor->step(50, BACKWARD, MICROSTEP);
}
```

IoT-Bus SD_MMC Card Example

This example show how to test an SD Card on the IoT-Bus Display board.

```
if(!SD_MMC.begin("/sdcard")){
    Serial.println("Card Mount Failed");
    return;
}
uint8_t cardType = SD_MMC.cardType();

if(cardType == CARD_NONE){
    Serial.println("No SD_MMC card attached");
    return;
}

Serial.print("SD_MMC Card Type: ");
if(cardType == CARD_MMC){
    Serial.println("MMC");
} else if(cardType == CARD_SD){
    Serial.println("SDSC");
} else if(cardType == CARD_SDHC){
    Serial.println("SDHC");
} else {
    Serial.println("UNKNOWN");
}

uint64_t cardSize = SD_MMC.cardSize() / (1024 * 1024);
Serial.printf("SD_MMC Card Size: %lluMB\n", cardSize);
```

These lines start up the SD_MMC card and mount it, then determine what type of card (which can be unknown) and then get the card size.

```
listDir(SD_MMC, "/", 0);
createDir(SD_MMC, "/mydir");
listDir(SD_MMC, "/", 0);
removeDir(SD_MMC, "/mydir");
```

(continues on next page)

(continued from previous page)

```
listDir(SD_MMC, "/", 2);
writeFile(SD_MMC, "/hello.txt", "Hello ");
appendFile(SD_MMC, "/hello.txt", "World!\n");
readFile(SD_MMC, "/hello.txt");
deleteFile(SD_MMC, "/foo.txt");
renameFile(SD_MMC, "/hello.txt", "/foo.txt");
readFile(SD_MMC, "/foo.txt");
testFileIO(SD_MMC, "/test.txt");
```

These lines list, create and remove directories. After that a file is written, data appended and read. If “/foo.txt” exists it is deleted and “/hello.txt” is renamed. Then it is read. Lastly some read and write timings are performed.

```
/*
   This is a test of the iot-bus SD_MMC
*/

#include "FS.h"
#include "SD_MMC.h"
#include "driver/gpio.h"
#include "SPI.h"

void listDir(fs::FS &fs, const char * dirname, uint8_t levels){
    Serial.printf("Listing directory: %s\n", dirname);

    File root = fs.open(dirname);
    if(!root){
        Serial.println("Failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println("Not a directory");
        return;
    }

    File file = root.openNextFile();
    while(file){
        if(file.isDirectory()){
            Serial.print("  DIR : ");
            Serial.println(file.name());
            if(levels){
                listDir(fs, file.name(), levels -1);
            }
        } else {
            Serial.print("  FILE: ");
            Serial.print(file.name());
            Serial.print("  SIZE: ");
            Serial.println(file.size());
        }
        file = root.openNextFile();
    }
}

void createDir(fs::FS &fs, const char * path){
    Serial.printf("Creating Dir: %s\n", path);
    if(fs.mkdir(path)){
        Serial.println("Dir created");
    }
}
```

(continues on next page)

(continued from previous page)

```
    } else {
        Serial.println("mkdir failed");
    }
}

void removeDir(fs::FS &fs, const char * path){
    Serial.printf("Removing Dir: %s\n", path);
    if(fs.rmdir(path)){
        Serial.println("Dir removed");
    } else {
        Serial.println("rmdir failed");
    }
}

void readFile(fs::FS &fs, const char * path){
    Serial.printf("Reading file: %s\n", path);

    File file = fs.open(path);
    if(!file){
        Serial.println("Failed to open file for reading");
        return;
    }

    Serial.print("Read from file: ");
    while(file.available()){
        Serial.write(file.read());
    }
}

void writeFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file){
        Serial.println("Failed to open file for writing");
        return;
    }
    if(file.print(message)){
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
}

void appendFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file){
        Serial.println("Failed to open file for appending");
        return;
    }
    if(file.print(message)){
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
}
```

(continues on next page)

(continued from previous page)

```

}

void renameFile(fs::FS &fs, const char * path1, const char * path2){
    Serial.printf("Renaming file %s to %s\n", path1, path2);
    if (fs.rename(path1, path2)) {
        Serial.println("File renamed");
    } else {
        Serial.println("Rename failed");
    }
}

void deleteFile(fs::FS &fs, const char * path){
    Serial.printf("Deleting file: %s\n", path);
    if(fs.remove(path)){
        Serial.println("File deleted");
    } else {
        Serial.println("Delete failed");
    }
}

void testFileIO(fs::FS &fs, const char * path){
    File file = fs.open(path);
    static uint8_t buf[512];
    size_t len = 0;
    uint32_t start = millis();
    uint32_t end = start;
    if(file){
        len = file.size();
        size_t flen = len;
        start = millis();
        while(len){
            size_t toRead = len;
            if(toRead > 512){
                toRead = 512;
            }
            file.read(buf, toRead);
            len -= toRead;
        }
        end = millis() - start;
        Serial.printf("%u bytes read for %u ms\n", flen, end);
        file.close();
    } else {
        Serial.println("Failed to open file for reading");
    }

    file = fs.open(path, FILE_WRITE);
    if(!file){
        Serial.println("Failed to open file for writing");
        return;
    }

    size_t i;
    start = millis();
    for(i=0; i<2048; i++){
        file.write(buf, 512);
    }
}

```

(continues on next page)

(continued from previous page)

```
end = millis() - start;
Serial.printf("%u bytes written for %u ms\n", 2048 * 512, end);
file.close();
}

void setup() {
  Serial.begin(115200);

  if(!SD_MMC.begin("/sdcard")) {
    Serial.println("Card Mount Failed");
    return;
  }
  uint8_t cardType = SD_MMC.cardType();

  if(cardType == CARD_NONE) {
    Serial.println("No SD_MMC card attached");
    return;
  }

  Serial.print("SD_MMC Card Type: ");
  if(cardType == CARD_MMC) {
    Serial.println("MMC");
  } else if(cardType == CARD_SD) {
    Serial.println("SDSC");
  } else if(cardType == CARD_SDHC) {
    Serial.println("SDHC");
  } else {
    Serial.println("UNKNOWN");
  }

  uint64_t cardSize = SD_MMC.cardSize() / (1024 * 1024);
  Serial.printf("SD_MMC Card Size: %lluMB\n", cardSize);

  listDir(SD_MMC, "/", 0);
  createDir(SD_MMC, "/mydir");
  listDir(SD_MMC, "/", 0);
  removeDir(SD_MMC, "/mydir");
  listDir(SD_MMC, "/", 2);
  writeFile(SD_MMC, "/hello.txt", "Hello ");
  appendFile(SD_MMC, "/hello.txt", "World!\n");
  readFile(SD_MMC, "/hello.txt");
  deleteFile(SD_MMC, "/foo.txt");
  renameFile(SD_MMC, "/hello.txt", "/foo.txt");
  readFile(SD_MMC, "/foo.txt");
  testFileIO(SD_MMC, "/test.txt");
}

void loop() {
}
```

IoT-Bus Mozilla IoT Examples

These examples are designed to run on IoT-Bus boards by oddWires. In each case, they demonstrate how to create a mozilla-iot “thing” and expose it through the mozilla-iot gateway running on a raspberry pi on the same Wi-Fi network. We will refer to a mozilla-iot thing as a Thing in this document. To setup a gateway see [this link](#). You can find the examples on GitHub at [IoT-Bus](#). There are two repositories:

iot-bus-mozilla-iot-examples-platformio Examples in PlatformIo format. A platformio.ini file is included with library dependencies.

iot-bus-mozilla-iot-examples-arduino Examples in ino format. You will need to install required libraries separately.

In each example you will have to enter your ssid and password within the main cpp file otherwise you will not reach the gateway. You can find the code for all the examples on github [here](#).

```
//TODO: Hardcode your wifi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";
```

The typical platformio.ini for mozilla-iot contains:

```
[env:iotbusio]
platform = espressif32
board = iotbusio
framework = arduino

; Serial Monitor options
monitor_speed = 115200

; Library dependencies
lib_deps = ArduinoJson
           https://github.com/me-no-dev/ESPAsyncWebServer
           https://github.com/mozilla-iot/webthing-arduino
```

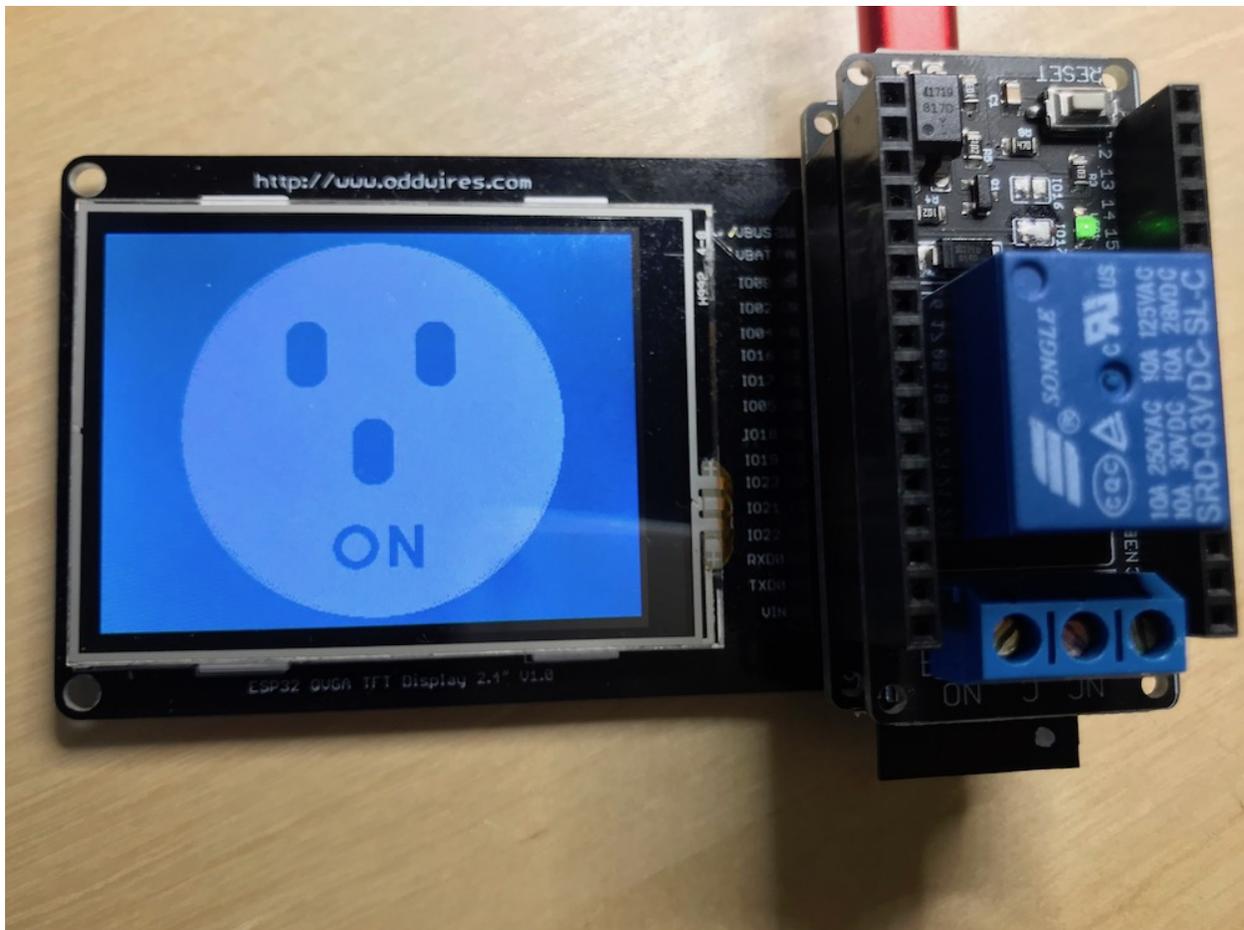
If you are using a display, this would be the platformio.ini required:

```
[env:iotbusio]
platform = espressif32
board = iotbusio
framework = arduino

; Serial Monitor options
monitor_speed = 115200

; Library dependencies including tft and touchscreen
lib_deps = ArduinoJson
           https://github.com/me-no-dev/ESPAsyncWebServer
           https://github.com/mozilla-iot/webthing-arduino
           https://github.com/iot-bus/XPT2046_Touchscreen
           https://github.com/iot-bus/TFT_eSPI
```

Of course if you are using other libraries for sensors or your own, you can include them here and PlatformIO will locate and install them.



LED Thing This example creates a Thing that enables the built-in LED on GPIO5 to be set through the mozilla-iot domain.

LED Lamp Thing This example creates a Thing that enables the status and brightness of the built-in LED on GPIO5 to be set through the mozilla-iot domain.

DHT11 Thing This example creates a Thing that exposes temperature and humidity properties and also displays the temperature locally.

Connect a DHT11 Temperature Sensor and run:

```
DATA -> GPIO4
VCC -> VUSB (VIN is OK if using battery(3.7-42V) but not 3V3 as not high enough)
GND -> GND
```

HC-SR04 Thing This example creates a Thing that exposes the current distance reading or the HC-SR04 ultrasonic distance sensor and also displays the distance locally.

Connect an HC-SR04 Ultrasonic Distance Sensor and run:

```
TRIG -> GPIO2
ECHO -> GPIO4
VCC -> VUSB
GND -> GND (VIN is OK if using battery(3.7-42V) but not 3V3 as not high enough)
```

HC-SR501 PIR Thing This example creates a motion sensor Thing that triggers on movement. It sets the on-board LED on triggering and updates the mozilla-iot interface appropriately. Connect an HC-SR501 Passive Infrared Sensor or any similar device and run:

```
OUT -> GPIO4
VCC -> VUSB (Typically VIN is OK if using battery(3.7-42V) but not 3V3 as not
↔high enough)
GND -> GND
```

Touch Thing This example creates a touch switch Thing that triggers when one of the capacitive touch pins on the ESP32 are touched. It sets the on-board LED on triggering and updates the mozilla-iot interface appropriately.

Connect a wire to GPIO4 and run.

Calculator Thing This example is a simple integer calculator that creates a Thing that exposes the two numbers, the last function and the result. It requires the IoT-Bus display.

Door Sensor Thing This example shows how to use a typical magnetic door sensor. Just connect one side of the contacts to GPIO4 and the other to GND. When the contacts are open the door sensor will show open in the mozilla interface and when they are shut the door will show shut.

Relay Thing This example uses an IoT-Bus relay board together with an IoT-Bus Io processor to expose the relay status and to enable the user to change through your mozilla-iot domain.

Relay Display & Touch Switch Thing In this second relay example the current status of the relay is also shown on the display. The touchscreen is enabled so it can be switched on and off locally. The status will be reflected by mozilla-iot.

IoT-Bus LED Touch Thing

```
#include <Arduino.h>
#include <Thing.h>
#include <WebThingAdapter.h>

/*
Simple binary sensor example using ESP32 capacitive touch input
This example code is in the public domain.
*/

//TODO: Hardcode your wifi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";

// Uses a touch sensor to detect input and turn on a LED
int ledPin = 5; // choose the pin for the LED
int touchPin = 4; // choose the input pin - T0 is the same as GPIO4

WebThingAdapter* adapter;

const char* sensorTypes[] = {"binarySensor", nullptr};
ThingDevice touch("Touch", "ESP32 Touch Input", sensorTypes);
ThingProperty touched("true", "", BOOLEAN, "BooleanProperty");
ThingPropertyValue sensorValue;

int threshold = 40;

void setup() {
  Serial.begin(115200);

  // Start WiFi
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");
```

(continues on next page)

```
// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

// Initialize MOZ IoT thing
adapter = new WebThingAdapter("adapter", WiFi.localIP());
touch.addProperty(&touched);
adapter->addDevice(&touch);
adapter->begin();

pinMode(ledPin, OUTPUT); // declare LED as output
}

void loop() {

    int val = touchRead(T0); // get value using T0 / GPIO4

    Serial.println(val);
    if (val < threshold){
        sensorValue.boolean = true;
        digitalWrite(ledPin, HIGH);
    }
    else{
        sensorValue.boolean = false;
        digitalWrite(ledPin, LOW);
    }
    touched.setValue(sensorValue);
    adapter->update();
    delay(300);
}
```

CHAPTER 30

IoT-Bus LED Thing

```
/**
 * Simple server compliant with Mozilla's proposed WoT API
 * Originally based on the HelloServer example
 * Tested on ESP8266, ESP32, Arduino boards with WINC1500 modules (shields or
 * MKR1000)
 *
 * This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/.
 */

#include <Arduino.h>
#include "Thing.h"
#include "WebThingAdapter.h"

//TODO: Hardcode your wifi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";

const int ledPin = 5; // manually configure LED pin

WebThingAdapter* adapter;

const char* ledTypes[] = {"OnOffSwitch", "led", nullptr};
ThingDevice led("LED", "LED", ledTypes);
ThingProperty ledOn("on", "", BOOLEAN, "OnOffProperty");

bool lastOn = false;

void setup(void) {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
  Serial.begin(115200);
  Serial.println("");
}
```

(continues on next page)

```

Serial.print("Connecting to \");
Serial.print(ssid);
Serial.println("\");
#ifdef ESP8266 || defined(ESP32)
WiFi.mode(WIFI_STA);
#endif
WiFi.begin(ssid, password);
Serial.println("");

// Wait for connection
bool blink = true;
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    digitalWrite(ledPin, blink ? LOW : HIGH); // active low led
    blink = !blink;
}
digitalWrite(ledPin, HIGH); // active low led

Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
adapter = new WebThingAdapter("w25", WiFi.localIP());

led.addProperty(&ledOn);
adapter->addDevice(&led);
adapter->begin();
Serial.println("HTTP server started");
Serial.print("http://");
Serial.print(WiFi.localIP());
Serial.print("/things/");
Serial.println(led.id);
}

void loop(void) {
    adapter->update();
    bool on = ledOn.getValue().boolean;
    digitalWrite(ledPin, on ? HIGH : LOW); // active high led
    if (on != lastOn) {
        Serial.print(led.id);
        Serial.print(": ");
        Serial.println(on);
    }
    lastOn = on;
}

```

IoT-Bus LED Lamp Thing

```
/**
 * Simple server compliant with Mozilla's proposed WoT API
 * Originally based on the HelloServer example
 * Tested on ESP8266, ESP32, Arduino boards with WINC1500 modules (shields or
 * MKR1000)
 *
 * This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/.
 */

#include <Arduino.h>
#include <Thing.h>
#include <WebThingAdapter.h>

#define MIN(a,b) (((a)<(b))?(a):(b))

const char* ssid = ".....";
const char* password = ".....";

#ifdef LED_BUILTIN
const int lampPin = LED_BUILTIN;
#else
const int lampPin = 5; // manually configure LED pin
#endif

// use first channel of 16 channels (started from zero)
#define LEDC_CHANNEL_0 0

// use 13 bit precision for LEDC timer
#define LEDC_TIMER_13_BIT 13

// use 5000 Hz as a LEDC base frequency
#define LEDC_BASE_FREQ 5000
```

(continues on next page)

(continued from previous page)

```

WebThingAdapter* adapter;

const char* lampTypes[] = {"OnOffSwitch", "Light", nullptr};
ThingDevice lamp("lamp", "My Lamp", lampTypes);

ThingProperty lampOn("on", "Whether the lamp is turned on", BOOLEAN, "OnOffProperty");
ThingProperty lampLevel("level", "The level of light from 0-100", NUMBER,
↳"BrightnessProperty");

// Arduino like analogWrite
// value has to be between 0 and valueMax
void ledcAnalogWrite(uint8_t channel, uint32_t value, uint32_t valueMax = 255) {
    // calculate duty, 8191 from 2 ^ 13 - 1
    uint32_t duty = (8191 / valueMax) * MIN(value, valueMax);

    // write duty to LEDC
    ledcWrite(channel, duty);
}

void setup(void) {
    pinMode(lampPin, OUTPUT);
    digitalWrite(lampPin, LOW); // initially off

    // Setup timer and attach timer to a led pin
    ledcSetup(LEDC_CHANNEL_0, LEDC_BASE_FREQ, LEDC_TIMER_13_BIT);
    ledcAttachPin(lampPin, LEDC_CHANNEL_0);

    Serial.begin(115200);
    #if defined(ESP8266) || defined(ESP32)
    WiFi.mode(WIFI_STA);
    #endif
    WiFi.begin(ssid, password);
    Serial.println("");

    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    adapter = new WebThingAdapter("led-lamp", WiFi.localIP());

    lamp.addProperty(&lampOn);
    lamp.addProperty(&lampLevel);
    adapter->addDevice(&lamp);
    adapter->begin();
    Serial.println("HTTP server started");

    #ifdef analogWriteRange
        analogWriteRange(255);
    #endif

```

(continues on next page)

(continued from previous page)

```
}  
  
void loop(void) {  
  adapter->update();  
  if (lampOn.getValue().boolean) {  
    int level = map(lampLevel.getValue().number, 0, 100, 0, 255);  
    // Serial.println(lampLevel.getValue().number);  
    ledcAnalogWrite(LED_CHANNEL_0, level);  
  
    // analogWrite(lampPin, level);  
  } else {  
    ledcAnalogWrite(LED_CHANNEL_0, 0);  
    // analogWrite(lampPin, 255);  
  }  
}
```


CHAPTER 32

IoT-Bus Relay Thing

```
#include <Arduino.h>
#include "Thing.h"
#include "WebThingAdapter.h"

/*
Turns on/off a relay using MOZ IoT.
This example code is in the public domain.
*/

//TODO: Hard-code your WiFi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";

const int relayPin = 17; // IoT-Bus relay

WebThingAdapter* adapter;

const char* relayTypes[] = {"SmartPlug", nullptr};
ThingDevice relay("relay", "IoT-Bus Relay", relayTypes);
ThingProperty relayOn("on", "", BOOLEAN, "OnOffProperty");

bool lastOn = false;

// the setup function runs once when you press reset or power the board

void setup() {
  Serial.begin(115200); //Use serial monitor for debugging

  // initialize relay pin as an output.
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, LOW);

  // Start WiFi
  WiFi.mode(WIFI_STA);
```

(continues on next page)

```

WiFi.begin(ssid, password);
Serial.println("");

// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

// Initialize MOZ IoT thing
adapter = new WebThingAdapter("adapter", WiFi.localIP());
relay.addProperty(&relayOn);
adapter->addDevice(&relay);
adapter->begin();
Serial.println("HTTP server started");
Serial.print("http://");
Serial.print(WiFi.localIP());
Serial.print("/things/");
Serial.println(relay.id);
}

void loop() {
    // update status
    adapter->update();
    bool on = relayOn.getValue().boolean;
    digitalWrite(relayPin, on ? HIGH : LOW); // active high
    if (on != lastOn) {
        Serial.print(relay.id);
        Serial.print(": ");
        Serial.println(on);
    }
    lastOn = on;
}

```

IoT-Bus Relay Display-Touch Thing

```
#include <Arduino.h>
#include <Thing.h>
#include <WebThingAdapter.h>

/*
 * Turns on/off a relay using MOZ IoT.
 * This example code is in the public domain.
 */

//TODO: Hardcode your wifi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";

const int relayPin = 17; // IoT-Bus relay 16 or 17 supported. If using touchscreen,
↳ then relay must be on GPIO17 and PENIRQ not used. TFT OK either way.

/*
 * Graphics and Touch
 */
#include <TFT_eSPI.h>
#include <XPT2046_Touchscreen.h> // We use modified version that maps from raw to,
↳ pixel and matches screen orientation

// declare display and touchscreen helpers below
void drawPlug(bool on);
bool touched(bool hit);

//Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST,
↳ TFT_MISO); //TFT_eSPI display = TFT_eSPI(); // Invoke custom library
TFT_eSPI tft = TFT_eSPI(); // Invoke custom library

const int width = 320;
const int height = 240;
#define ILI9341_MOZCYAN 0x0595
```

(continues on next page)

(continued from previous page)

```

#define ILI9341_LIGHTCYAN 0x4DF5

/* Create an instance of the touch screen library */
#define CS_PIN 16 //PENIRQ not used as it would drive relay on same pin

// These are the pins used to interface between the 2046 touch controller and ESP32
// Hardware SPI pins as used above for the display are the same
// DOUT(MISO) 19 Data out pin (T_DO) of touch screen */
// DIN(MOSI) 23 Data in pin (T_DIN) of touch screen */
// DCLK(CLK) 18 Clock pin (T_CLK) of touch screen */

XPT2046_Touchscreen ts(CS_PIN, 255); // Param 2 - No interrupts

/*
 * MOZ IOT
 */
WebThingAdapter* adapter;

const char* relayTypes[] = {"SmartPlug", nullptr};
ThingDevice relay("Relay", "IoT-Bus Relay", relayTypes);
ThingProperty relayOn("on", "", BOOLEAN, "OnOffProperty");

// remember last state
bool lastOn = false;

// the setup function runs once when you press reset or power the board
void setup() {
    Serial.begin(115200); //Use serial monitor for debugging

    // initialize relay pin as an output.
    pinMode(relayPin, OUTPUT);
    digitalWrite(relayPin, LOW);

    //Start WiFi
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.println("");

    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    // Turn on display
    pinMode(33, OUTPUT); // turn on the IoT-Bus Display
    digitalWrite(33, HIGH);

    // Start display and draw plug
    tft.begin();
    tft.setRotation(1);

```

(continues on next page)

(continued from previous page)

```

// background
tft.fillScreen(ILI9341_MOZCYAN); // closest 16 bit color to MOZ IOT

// draw initial state
drawPlug(false);

// start the touch screen
ts.begin();

// Initialize MOZ IoT thing
adapter = new WebThingAdapter("adapter", WiFi.localIP());
relay.addProperty(&relayOn);
adapter->addDevice(&relay);
adapter->begin();
Serial.println("HTTP server started");
Serial.print("http://");
Serial.print(WiFi.localIP());
Serial.print("/things/");
Serial.println(relay.id);
}

ThingPropertyValue value;

void loop() {
bool on = relayOn.getValue().boolean;
if(touched(true)){
    on = !on;        // invert status
    lastOn = !on;    // invert status
    value.boolean = on;
    relayOn.setValue(value);
    delay(300); // avoid touch bounce
}
// update status
adapter->update();
on = relayOn.getValue().boolean;
digitalWrite(relayPin, on ? HIGH : LOW); // active high
if (on != lastOn) {
    Serial.print(relay.id);
    Serial.print(": ");
    Serial.println(on);
    drawPlug(on);
}
lastOn = on;
}

// helper to draw the plug state on/off

void drawPlug(bool on=false){
#define RADIUS 110
#define SOCKET_WIDTH 24
#define SOCKET_HEIGHT 40
#define SOCKET_RADIUS 10
#define LEFT_SOCKET_X 108
#define LEFT_SOCKET_Y 55
#define RIGHT_SOCKET_X 185
#define RIGHT_SOCKET_Y 55
#define BOTTOM_SOCKET_X 147

```

(continues on next page)

(continued from previous page)

```

#define BOTTOM_SOCKET_Y 115

uint16_t bgColor;
uint16_t socketColor;
uint16_t textColor;
uint16_t outlineColor;
String text;

if(on){
    bgColor = ILI9341_WHITE;
    outlineColor = ILI9341_WHITE;
    socketColor = ILI9341_MOZCYAN;
    textColor = ILI9341_MOZCYAN;
    text = " ON";
}
else{
    bgColor = ILI9341_LIGHTCYAN;
    outlineColor = ILI9341_WHITE;
    socketColor = ILI9341_WHITE;
    textColor = ILI9341_WHITE;
    text = "OFF";
}

// outline
for (int i=0; i>-5;i--){
    tft.drawCircle(width/2, height/2, RADIUS-i, outlineColor);
}
tft.fillCircle(width/2, height/2, RADIUS+1, bgColor);

// sockets
tft.fillRoundRect(LEFT_SOCKET_X, LEFT_SOCKET_Y, SOCKET_WIDTH, SOCKET_HEIGHT,
↳SOCKET_RADIUS, socketColor);
tft.fillRoundRect(RIGHT_SOCKET_X, RIGHT_SOCKET_Y, SOCKET_WIDTH, SOCKET_HEIGHT,
↳SOCKET_RADIUS, socketColor);
tft.fillRoundRect(BOTTOM_SOCKET_X, BOTTOM_SOCKET_Y, SOCKET_WIDTH, SOCKET_HEIGHT,
↳SOCKET_RADIUS, socketColor);

// We're simply drawing the text because loading fonts is more complex for this
↳example
// and the base fonts are not smooth

#define TEXT_O_X 140
#define TEXT_O_Y 192
#define TEXT_O_RADIUS 13

#define TEXT_HEIGHT 25
#define STROKE_WIDTH 5

#define TEXT_2_X 160
#define TEXT_2_Y 180
#define TEXT_WIDTH 15

#define TEXT_3_X 180
#define TEXT_3_Y 180

#define OFFSET 8

```

(continues on next page)

(continued from previous page)

```

if (on) {
    // O
    tft.fillCircle(TEXT_O_X+OFFSET, TEXT_O_Y, TEXT_O_RADIUS, textColor);
    tft.fillCircle(TEXT_O_X+OFFSET, TEXT_O_Y, TEXT_O_RADIUS-STROKE_WIDTH,
↳bgColor);

    // N
    tft.fillRect(TEXT_2_X+OFFSET, TEXT_2_Y, STROKE_WIDTH, TEXT_HEIGHT, textColor);
    tft.fillRect(TEXT_2_X+TEXT_WIDTH+OFFSET, TEXT_2_Y, STROKE_WIDTH, TEXT_HEIGHT,
↳textColor);
    for (int i=0; i<STROKE_WIDTH; i++) {
        tft.drawLine(TEXT_2_X+i+OFFSET, TEXT_2_Y, TEXT_2_X+TEXT_WIDTH+i+OFFSET, TEXT_
↳2_Y+TEXT_HEIGHT, textColor);
    }
}
else {

    // O
    tft.fillCircle(TEXT_O_X, TEXT_O_Y, TEXT_O_RADIUS, textColor);
    tft.fillCircle(TEXT_O_X, TEXT_O_Y, TEXT_O_RADIUS-STROKE_WIDTH, bgColor);

    // F
    tft.fillRect(TEXT_2_X, TEXT_2_Y, STROKE_WIDTH, TEXT_HEIGHT, textColor);
    tft.fillRect(TEXT_2_X, TEXT_2_Y, TEXT_WIDTH, STROKE_WIDTH, textColor);
    tft.fillRect(TEXT_2_X, TEXT_2_Y+TEXT_HEIGHT/2, TEXT_WIDTH, STROKE_WIDTH,
↳textColor);

    // F
    tft.fillRect(TEXT_3_X, TEXT_3_Y, STROKE_WIDTH, TEXT_HEIGHT, textColor);
    tft.fillRect(TEXT_3_X, TEXT_3_Y, TEXT_WIDTH, STROKE_WIDTH, textColor);
    tft.fillRect(TEXT_3_X, TEXT_3_Y+TEXT_HEIGHT/2, TEXT_WIDTH, STROKE_WIDTH,
↳textColor);
}
}

// returns true if touched in socket image
bool touched(bool hit) {

if (ts.touched())
{
    Serial.println("touched");
    // Read the current X and Y axis as co-ordinates at the last touch time
    TS_Point p = ts.getMappedPoint();
    Serial.print(p.x); Serial.print(" "); Serial.println(p.y); Serial.print(" "); Serial.
↳println(p.z);
    if (sqrt((p.x-width/2)*(p.x-width/2) + (p.y-height/2)*(p.y-height/2)) < RADIUS) {
        Serial.println("Hit");
        return true;
    }
}
return false;
}

```


IoT-Bus Window and Door Sensor Thing

```
#include <Arduino.h>
#include <Thing.h>
#include <WebThingAdapter.h>

/*
Simple binary sensor example using ESP32 capacitive touch input
This example code is in the public domain.
*/

//TODO: Hardcode your wifi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";

// Uses a window/door sensor and turns on a LED when open
int ledPin = 5;    // choose the pin for the LED
int sensorPin = 4; // choose the input pin - we'll use a pullup on this pin to keep_
↳high
                    // Connect one end of the door switch to this pin.
                    // Connect the other end to GND

WebThingAdapter* adapter;

const char* sensorTypes[] = {"binarySensor", nullptr};
ThingDevice sensor("DoorSensor", "Window/Door Sensor", sensorTypes);
ThingProperty openProperty("Sensor Open", "", BOOLEAN, "OpenProperty", "Door", "",
↳false);
ThingPropertyValue sensorValue;

void setup() {
  Serial.begin(115200);

  // Start WiFi
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
```

(continues on next page)

```

Serial.println("");

// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

// Initialize MOZ IoT thing
adapter = new WebThingAdapter("adapter", WiFi.localIP());
sensor.addProperty(&openProperty);
adapter->addDevice(&sensor);
adapter->begin();

pinMode(sensorPin, INPUT_PULLUP);
pinMode(ledPin, OUTPUT); // declare LED as output
}

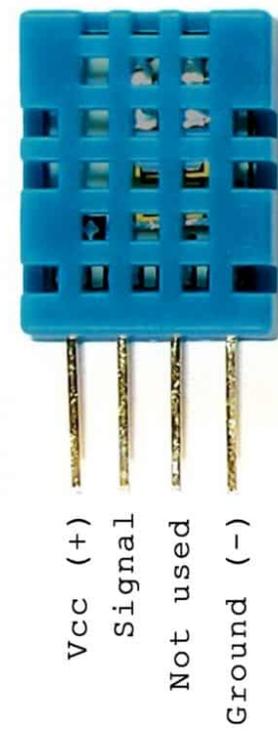
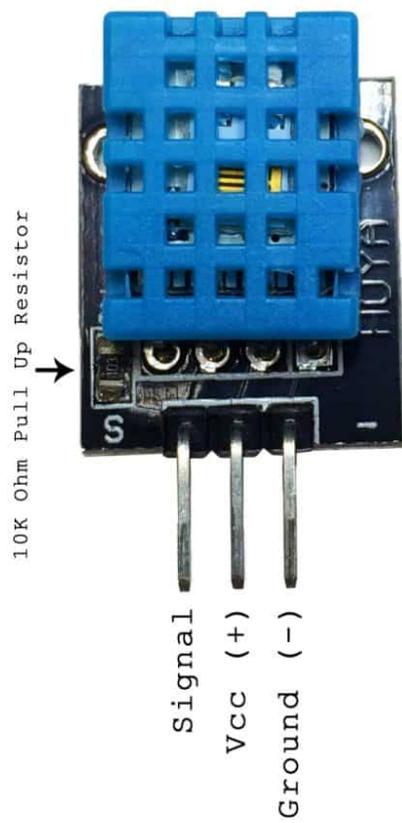
void loop() {

    int val = digitalRead(sensorPin); // get value

    Serial.println(val);
    if (val == LOW){
        sensorValue.boolean = false; // pulled to GND
        digitalWrite(ledPin, LOW);
    }
    else{
        sensorValue.boolean = true; // pullup active
        digitalWrite(ledPin, HIGH);
    }
    openProperty.setValue(sensorValue);
    adapter->update();
    delay(300);
}

```

IoT-Bus DHT11 Thing



```
#include <arduino.h>  
/*
```

(continues on next page)

(continued from previous page)

```

* Graphics
*/
#include <TFT_eSPI.h>

TFT_eSPI display = TFT_eSPI();

/*
* MOZ IoT
*/
#include <Thing.h>
#include <WebThingAdapter.h>

WebThingAdapter* adapter;

const char* sensorTypes[] = {"MultiLevelSensor", nullptr};

ThingDevice dht11("thermometer", "DHT11 Thermometer & Hygrometer", sensorTypes);
// ThingProperty variable_name( id, description, type, @type, label, unit, writable );
↪ // writable not supported yet

ThingProperty temperature("temperature", "The temperature from x to y", NUMBER,
↪ "LevelProperty", "Temperature", "F", false);
ThingProperty humidity("humidity", "The humidity from 0 to 100%", NUMBER,
↪ "LevelProperty", "Humidity", "percent", false);

ThingPropertyValue reading;

/*
* WiFi ssid and password
*/
const char* ssid = ".....";
const char* password = ".....";

/*
* DHT11
*/
#include "DHTesp.h"

DHTesp dht;

/** Comfort profile */
ComfortState cf;

int dhtPin = 4;

/**
* getTemperature
* Reads temperature from DHT11 sensor
* @return bool
* true if temperature could be aquired
* false if aquisition failed
*/

TempAndHumidity newValues;

bool getTemperature() {
    // Reading temperature for humidity takes about 250 milliseconds!

```

(continues on next page)

(continued from previous page)

```

// Sensor readings may also be up to 2 seconds 'old' (it's a very slow sensor)
newValues = dht.getTempAndHumidity();
// Check if any reads failed and exit early (to try again).
if (dht.getStatus() != 0) {
    Serial.println("DHT11 error status: " + String(dht.getStatusString()));
    return false;
}

float heatIndex = dht.computeHeatIndex(newValues.temperature, newValues.humidity);
float dewPoint = dht.computeDewPoint(newValues.temperature, newValues.humidity);
float cr = dht.getComfortRatio(cf, newValues.temperature, newValues.humidity);

String comfortStatus;
switch(cf) {
    case Comfort_OK:
        comfortStatus = "Comfort_OK";
        break;
    case Comfort_TooHot:
        comfortStatus = "Comfort_TooHot";
        break;
    case Comfort_TooCold:
        comfortStatus = "Comfort_TooCold";
        break;
    case Comfort_TooDry:
        comfortStatus = "Comfort_TooDry";
        break;
    case Comfort_TooHumid:
        comfortStatus = "Comfort_TooHumid";
        break;
    case Comfort_HotAndHumid:
        comfortStatus = "Comfort_HotAndHumid";
        break;
    case Comfort_HotAndDry:
        comfortStatus = "Comfort_HotAndDry";
        break;
    case Comfort_ColdAndHumid:
        comfortStatus = "Comfort_ColdAndHumid";
        break;
    case Comfort_ColdAndDry:
        comfortStatus = "Comfort_ColdAndDry";
        break;
    default:
        comfortStatus = "Unknown:";
        break;
};

Serial.println(" T:" + String(newValues.temperature) + " H:" + String(newValues.
↪humidity) + " I:" + String(heatIndex) + " D:" + String(dewPoint) + " " + ↪
↪comfortStatus);
return true;
}

/*
 * displayString helper function to draw text on
 * the TFT display
 */
const int textHeight = 12;

```

(continues on next page)

(continued from previous page)

```

const int textWidth = 12;
const int width = 320;
const int height = 240;

String last, current; // current and last values of text

void displayString(const String& str, int color) {
    int len = str.length()+1;
    int strWidth = len * textWidth;
    int strHeight = textHeight;
    Serial.println(strWidth);
    int scale = width / strWidth;
    Serial.println(scale);
    if (scale < 1)
        scale = 1;

    int x = width / 2 - strWidth * scale / 2;
    int y = height / 2 + strHeight * scale / 2;

    display.setFreeFont(&FreeSans18pt7b);
    display.setRotation(1);
    display.setTextColor(color);
    display.setTextSize(scale);
    display.setCursor(x, y);
    display.println(str);
}

void setup()
{
    Serial.begin(115200);
    dht.setup(dhtPin, DHTesp::DHT11);

    // Turn on display
    pinMode(33, OUTPUT); // turn on the IoT-Bus Display
    digitalWrite(33, HIGH);

    // Start display and clear
    display.begin();
    display.fillScreen(ILI9341_BLACK);

    // Start WiFi
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.println("");

    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
}

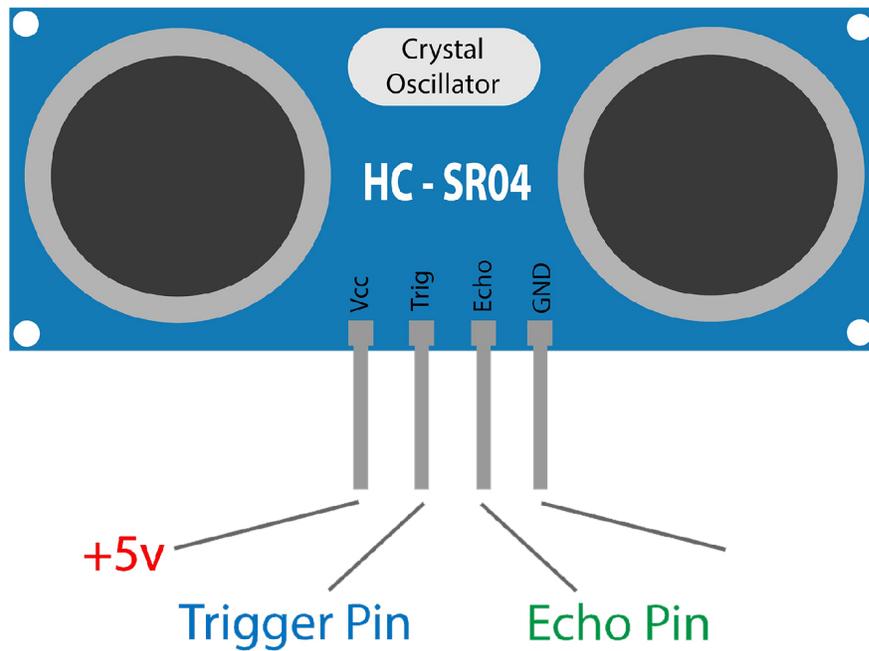
```

(continues on next page)

(continued from previous page)

```
// Initialize MOZ IoT thing
adapter = new WebThingAdapter("multilevelsensor", WiFi.localIP());
dht11.addProperty(&temperature);
dht11.addProperty(&humidity);
adapter->addDevice(&dht11);
adapter->begin();
}

void loop() {
  getTemperature();
  current = String(dht.toFahrenheit(newValues.temperature)) + "°F  " +
↳String(newValues.humidity) + "%";
  if (current != last){
    displayString(last, ILI9341_BLACK); // clear old text by writing it black
    displayString(current, ILI9341_WHITE); // write the new value
    reading.number = dht.toFahrenheit(newValues.temperature); // needs to be a
↳PropertyValue
    temperature.setValue(reading); // now set the property
    reading.number = newValues.humidity; // needs to be a PropertyValue
    humidity.setValue(reading); // now set the property
    adapter->update(); // update the MOZ adapter
    last = current; // remember the last write to be able
↳to clear it
  }
  delay(1200);
}
```

```
/*  
 * Arduino framework  
 */  
  
#include <arduino.h>  
  
/*  
 * MOZ IoT  
 */
```

(continues on next page)

(continued from previous page)

```

#include <Thing.h>
#include <WebThingAdapter.h>

/*
 * Graphics
 */
#include "TFT_eSPI.h" // Hardware-specific library

TFT_eSPI display = TFT_eSPI();

WebThingAdapter* adapter;

const char* sensorTypes[] = {"LevelSensor", nullptr};
ThingDevice hcsr04("HC-SR04", "HC-SR04", sensorTypes);
ThingProperty distance("distance", "Distance in cm", NUMBER, "LevelProperty",
↳"Distance", "in", "false");
ThingPropertyValue measurement;

/*
 * WiFi ssid and password
 */
const char* ssid = ".....";
const char* password = ".....";

/*
 * HC-SR04
 */
int trigPin = 2; // Trigger
int echoPin = 4; // Echo
long duration, cm, inches;

/*
 * displayString helper function to draw text on
 * the TFT display
 */
const int textHeight = 18;
const int textWidth = 18;
const int width = 320;
const int height = 240;

String last, current; // current and last values of text

void displayString(const String& str, int color) {
    int len = str.length()+1;
    int strWidth = len * textWidth;
    int strHeight = textHeight;
    int scale = width / strWidth;

    int x = width / 2 - (strWidth * scale / 2);
    int y = height / 2 + (strHeight * scale / 2);

    display.setFreeFont(&FreeSans18pt7b);
    display.setRotation(1);
    display.setTextColor(color);
    display.setTextSize(scale);
    display.setCursor(x, y);
    display.println(str);

```

(continues on next page)

(continued from previous page)

```
    Serial.println(str);
}

/*
 * First-time initialization
 */

void setup() {
    // Start serial monitor - make sure same speed as monitor
    Serial.begin (115200);

    // HC-SR04 pins
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);

    // Turn on display
    pinMode(33, OUTPUT); // turn on the IoT-Bus Display
    digitalWrite(33, HIGH);

    // Start display and clear
    display.begin();
    display.fillScreen(ILI9341_BLACK);

    // Start WiFi
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.println("");

    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    // Initialize MOZ IoT thing
    adapter = new WebThingAdapter("textdisplayer", WiFi.localIP());
    measurement.number = -1;
    distance.setValue(measurement);
    hcsr04.addProperty(&distance);
    adapter->addDevice(&hcsr04);
    adapter->begin();
}

void loop() {
    // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
    // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
    digitalWrite(trigPin, LOW);
    delayMicroseconds(5);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
}
```

(continues on next page)

```

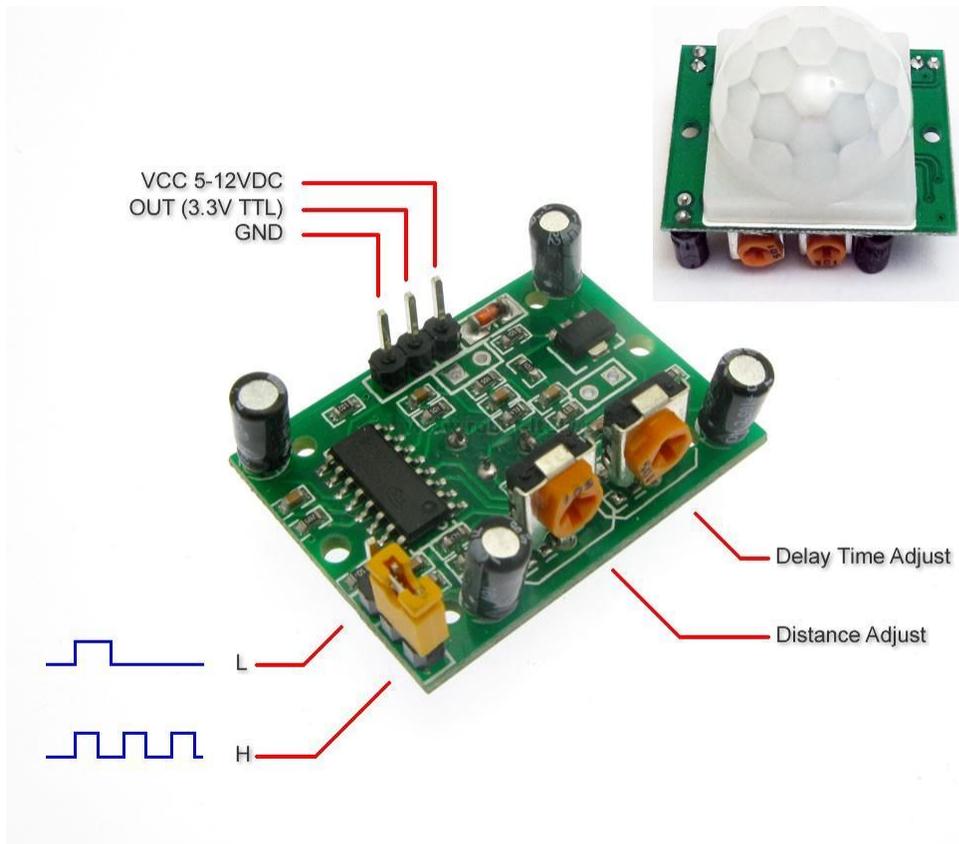
// Read the signal from the sensor: a HIGH pulse whose
// duration is the time (in microseconds) from the sending
// of the ping to the reception of its echo off of an object.
pinMode(echoPin, INPUT);
duration = pulseIn(echoPin, HIGH);

// Convert the time into a distance
cm = (duration/2) / 29.1;    // Divide by 29.1 or multiply by 0.0343
inches = (duration/2) / 74; // Divide by 74 or multiply by 0.0135

// Uncomment to display on serial monitor
// Serial.print(inches);
// Serial.print("in, ");
// Serial.print(cm);
// Serial.print("cm");
// Serial.println();
if(cm >= 400 || cm < 2){
    current = "Out of range";
}
else{
    current = String(inches) + " in";
}
if (current != last){
    displayString(last, ILI9341_BLACK); // clear old text by writing it black
    displayString(current, ILI9341_WHITE); // write the new value
    measurement.number = inches;
    distance.setValue(measurement);
    adapter->update(); // update the MOZ IoT thing
    last = current; // remember the last write to be able
↳to clear it
    delay(500); // vary to suit
}
}

```

IoT-Bus HC-SR501 PIR Thing



```
#include <Arduino.h>

/*
Simple motion sensor example using HC-S501
```

(continues on next page)

(continued from previous page)

```

This example code is in the public domain.
*/

#include <Arduino.h>
#include <Thing.h>
#include <WebThingAdapter.h>

//TODO: Hardcode your wifi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";

// Uses a PIR sensor to detect movement and turn on a LED
int ledPin = 5; // choose the pin for the LED
int inputPin = 4; // choose the input pin (for PIR sensor)

WebThingAdapter* adapter;

const char* sensorTypes[] = {"MotionSensor", nullptr};
ThingDevice hcsr501("HC-SR501", "HC-SR501 PIR Sensor", sensorTypes);
ThingProperty sensorOn("on", "", BOOLEAN, "MotionProperty");
ThingPropertyValue sensorValue;

void setup() {

    Serial.begin(115200); //Use serial monitor for debugging

    // Start WiFi
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    Serial.println("");

    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    // Initialize MOZ IoT thing
    adapter = new WebThingAdapter("adapter", WiFi.localIP());
    hcsr501.addProperty(&sensorOn);
    adapter->addDevice(&hcsr501);
    adapter->begin();

    pinMode(ledPin, OUTPUT); // declare LED as output
    pinMode(inputPin, INPUT); // declare sensor as input
}

void loop(){
    int val = digitalRead(inputPin); // read input value
    if (val == HIGH) { // check if the input is HIGH
        if (sensorOn.getValue().boolean != true) {

```

(continues on next page)

(continued from previous page)

```
    // Turned on
    Serial.println("Motion detected!");
    // Update on the output change, not state
    sensorValue.boolean = true;
    digitalWrite(ledPin, HIGH);
  }
} else {
  if (sensorOn.getValue().boolean == true){
    // Turned off
    Serial.println("Motion ended!");
    // Update on the output change, not state
    sensorValue.boolean = false;
    digitalWrite(ledPin, LOW);
  }
}
// update the adapter status
sensorOn.setValue(sensorValue);
adapter->update();
delay(300); // need to retain state long enough to be visible
}
```


IoT-Bus Calculator Thing

```

/*_____Import Libraries_____*/
#include "TFT_eSPI.h" // Hardware-specific library
#include "colors.h" // extended colors
#include <XPT2046_Touchscreen.h>
#include <FS.h>
#include <WiFi.h>
/*
 * MOZ IoT
 */
#include <Thing.h>
#include <WebThingAdapter.h>
/*_____End of Libraries_____*/

/*
 * WiFi ssid and password
 */
const char* ssid = ".....";
const char* password = ".....";

// simple window style class used for the response box and buttons
enum alignment { LEFT, RIGHT, MIDDLE };

class Button{
public:
Button(TFT_eSPI& _tft): tft(_tft){}
void setRect(uint16_t _x, uint16_t _y, uint16_t _width, uint16_t _height){
    x = _x;
    y = _y;
    width = _width;
    height = _height;
}
void setText(String _text, uint16_t _color){text = _text;color = _color;}
String& getText(){return text;}
void setType(uint8_t _type){type = _type;}

```

(continues on next page)

(continued from previous page)

```

uint8_t getType(){return type;}
void setBgColor(uint16_t _bgColor){bgColor = _bgColor;}
void setTextAlign(uint8_t _alignment){alignment = _alignment;}
void draw(){
    tft.fillRect(x, y, width, height, bgColor);
    switch(alignment){
    default:
    case MIDDLE:
        tft.setCursor(x+width/3, y+height/3); // center text
        break;
    case LEFT:
        tft.setCursor(x+10, y+height/3);
        break;
    case RIGHT:
        tft.setCursor(x+width, y+height/3);
        break;
    }
    // draw the outline
    tft.drawFastHLine(x, y, width, TFT_WHITE);
    tft.fillRect(x, y+height-5, width, 5, TFT_DARKGREY);
    tft.drawFastHLine(x, y+height-1, width, TFT_WHITE);
    tft.drawFastVLine(x, y, height, TFT_WHITE);
    tft.fillRect(x+width-5, y, 5, height, TFT_DARKGREY);
    tft.drawFastVLine(x+width-1, y, height, TFT_WHITE);

    // draw the text
    tft.setTextSize(4);
    tft.setTextColor(color);
    tft.println(text);
}
bool hit(TS_Point p){
    if (p.x<x+width && p.x>x){
        if (p.y<y+height && p.y>y){
            // draw the outline as pushed
            tft.drawFastHLine(x, y, width, TFT_WHITE);
            tft.fillRect(x, y, width, 5, TFT_DARKGREY);
            tft.drawFastHLine(x, y+height-1, width, TFT_WHITE);
            tft.drawFastVLine(x, y, height, TFT_WHITE);
            tft.fillRect(x, y, 5, height, TFT_DARKGREY);
            tft.drawFastVLine(x+width-1, y, height, TFT_WHITE);
            delay(300); // enough time to be visible
            draw(); // redraw
            return true;
        }
    }
    return false;
}

private:
TFT_eSPI& tft;
    uint16_t x = 0, y = 0, width = 240, height = 320, color = TFT_WHITE, bgColor =
↳TFT_BLACK;
uint8_t type; // user type
uint8_t alignment = MIDDLE;
String text = "";
};
    
```

(continues on next page)

(continued from previous page)

```

/*_____Calibration values TFT TS_____*/
#define TS_MINX 256
#define TS_MINY 274
#define TS_MAXX 3632
#define TS_MAXY 3579
/*_____End of Calibration_____*/

class Calculator{
private:
    TFT_eSPI& tft;
    XPT2046_Touchscreen ts;

    String symbol[4][4] = {
        { "7", "8", "9", "/" },
        { "4", "5", "6", "*" },
        { "1", "2", "3", "-" },
        { "C", "0", "=", "+" }
    };

    unsigned int buttonColors[4][4] = {
        { TFT_LIGHTGREY, TFT_LIGHTGREY, TFT_LIGHTGREY, TFT_LIGHTBLUE },
        { TFT_LIGHTGREY, TFT_LIGHTGREY, TFT_LIGHTGREY, TFT_LIGHTBLUE },
        { TFT_LIGHTGREY, TFT_LIGHTGREY, TFT_LIGHTGREY, TFT_LIGHTBLUE },
        { TFT_LIGHTSALMON, TFT_LIGHTGREY, TFT_LIGHTGREEN, TFT_LIGHTBLUE }
    };

    enum buttonType{ NUM, CLEAR, EQUALS, DIVIDE, MULTIPLY, ADD, SUBTRACT };

    uint8_t buttonTypes[4][4] = {
        { NUM, NUM, NUM, DIVIDE },
        { NUM, NUM, NUM, MULTIPLY },
        { NUM, NUM, NUM, SUBTRACT },
        { CLEAR, NUM, EQUALS, ADD }
    };

    #define BUTTONCOUNT 16
    Button* buttons[BUTTONCOUNT];

    long Num1, Num2, Accumulator;
    char action;

    Button* resultBox;
    String function = "";

public:
    Calculator(TFT_eSPI& _tft, XPT2046_Touchscreen _ts): tft(_tft), ts(_ts){
        // initialize display
        tft.init();
        tft.setRotation(0);
        ts.setRotation(0);
        ts.setCalibration(TS_MINX, TS_MAXX, TS_MINY, TS_MAXY);
    }

    long getAccumulator(){return Accumulator;}
    long getNum1(){return Num1;}
    long getNum2(){return Num2;}
    String getFunction(){return function;}

```

(continues on next page)

```

// wait for a button press
TS_Point waitTouch() {
    while (!ts.touched()){
        // do nothing
    }
    Serial.println("touched");
    // Read the current X and Y axis as co-ordinates at the last touch time
    // The values were captured when Pressed() was called!
    TS_Point p = ts.getPoint();
    Serial.println("-----");
    // print raw data
    Serial.print("raw:");Serial.print(p.x);Serial.print(",");Serial.println(p.y);
    p = ts.getMappedPoint();
    // print data mapped to width and height of TFT depending on rotation
    Serial.print("mapped by library:");Serial.print(p.x);Serial.print(",");Serial.
↪println(p.y);
    Serial.println("-----");
    return p;
}

void draw()
{
    #define resultBoxHeight 80
    #define buttonHeight 60
    #define buttonWidth 60

    resultBox = new Button(tft);

    // we are using the touchscreen to getwidth as we know it changes with
↪orientation - need to check tft
    resultBox->setRect(0, 0, ts.getWidth(), resultBoxHeight);
    resultBox->setBgColor(TFT_BLACK);
    resultBox->setText(String("IoT-Bus"), TFT_WHITE);
    resultBox->setTextAlign(LEFT);
    resultBox->draw();

    // Create keypad
    for (int j=0;j<4;j++) {
        for (int i=0;i<4;i++) {
            buttons[(j*4)+i] = new Button(tft);
            buttons[(j*4)+i]->setRect( 60*i, resultBoxHeight + (60*j), buttonWidth,
↪buttonHeight );
            buttons[(j*4)+i]->setBgColor(buttonColors[j][i]);
            buttons[(j*4)+i]->setText(String(symbol[j][i]), TFT_BLACK);
            buttons[(j*4)+i]->setType(buttonTypes[j][i]);
            buttons[(j*4)+i]->draw();
        }
    }

    // Handle each button
    void handleButton(Button* button){
        uint8_t type = button->getType();
        Serial.println(type);
        switch(type){
            default:

```

(continues on next page)

(continued from previous page)

```

        case NUM:
            handleNumber(button);
            displayResult();
            Serial.println("number");
            break;
        case CLEAR:
            Serial.println("clear");
            Accumulator=Num1=Num2=0;
            displayResult();
            break;
        case EQUALS:
            handleEquals(button);
            displayResult();
            Serial.println("=");
            break;
        case MULTIPLY:
        case DIVIDE:
        case ADD:
        case SUBTRACT:
            action = button->getType();
            Serial.println("function");
            Num1 = Accumulator;
            Accumulator = 0;
            function = button->getText();
            break;
    }
}
// Hittest for all buttons
Button* hitTest(TS_Point p){
    for(int i=0;i<BUTTONCOUNT;i++){
        if(buttons[i]->hit(p)){
            Serial.println(buttons[i]->getText());
            handleButton(buttons[i]);
            return buttons[i];
        }
    }
    return nullptr;
}

void displayResult(){
    resultBox->setText(String(Accumulator), TFT_WHITE);
    resultBox->draw();
}

// Handle number
void handleNumber(Button* button){
    if (Accumulator==0)
        Accumulator= button->getText().toInt(); // First press
    else
        Accumulator = (Accumulator*10) + button->getText().toInt(); // Second press
}

// Handle =
void handleEquals(Button* button){
    Num2=Accumulator;
    switch(action){
        default:

```

(continues on next page)

(continued from previous page)

```

        case ADD:
            Accumulator = Num1+Num2;
            Serial.println("+");
            break;
        case SUBTRACT:
            Accumulator = Num1-Num2;
            Serial.println("-");
            break;
        case MULTIPLY:
            Accumulator = Num1*Num2;
            Serial.println("*");
            break;
        case DIVIDE:
            Accumulator = Num1/Num2;
            Serial.println("/");
            break;
    }
}
};

// Globals

Calculator* calculator;

// MOZ IoT
WebThingAdapter* adapter;
ThingPropertyValue value;
String iotbus ("IoT-Bus");
String accumulator, num1, num2;

const char* textDisplayTypes[] = {"TextDisplay", nullptr};
ThingDevice textDisplay("textDisplay", "IoT-Bus Calculator", textDisplayTypes);
ThingProperty accumulatorProperty("Accumulator", "", NUMBER, nullptr);
ThingProperty num1Property("Number 1", "", NUMBER, nullptr);
ThingProperty num2Property("Number 2", "", NUMBER, nullptr);
ThingProperty functionProperty("Last Function", "", STRING, nullptr);

// Create TFT
TFT_eSPI tft = TFT_eSPI();

// Create Touchscreen
#define CS_PIN 16 // touch pin CS
XPT2046_Touchscreen ts(CS_PIN, 255); // No IRQ Pin

void setup() {
    Serial.begin(115200); //Use serial monitor for debugging

    Serial.println("Starting Caclulator");
    ts.setSize(240,320); // set width, height
    ts.begin();

    // Turn on display
    pinMode(33, OUTPUT); // turn on the IoT-Bus Display
    digitalWrite(33, HIGH);

    // Start WiFi
    WiFi.mode(WIFI_STA);

```

(continues on next page)

(continued from previous page)

```

WiFi.begin(ssid, password);
Serial.println("");

// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

// Initialize MOZ IoT thing
adapter = new WebThingAdapter("textdisplayer", WiFi.localIP());
value.string = &iotbus;
accumulatorProperty.setValue(value);
textDisplay.addProperty(&accumulatorProperty);
textDisplay.addProperty(&num1Property);
textDisplay.addProperty(&num2Property);
functionProperty.setValue(value);
textDisplay.addProperty(&functionProperty);
adapter->addDevice(&textDisplay);
adapter->begin();

// Create and draw calculator
calculator = new Calculator(tft, ts);
calculator->draw();
}

String function; // needs to be around for MOZ IoT

void loop() {
    // wait for a button press
    TS_Point p = calculator->waitTouch();
    calculator->hitTest(p);

    // update MOZ IoT
    value.number = calculator->getAccumulator();
    accumulatorProperty.setValue(value);
    value.number = calculator->getNum1();
    num1Property.setValue(value);
    value.number = calculator->getNum2();
    num2Property.setValue(value);
    function = calculator->getFunction();
    value.string = &function;
    functionProperty.setValue(value);
    adapter->update();

    delay(300);
}

```

IoT-Bus Mozilla IoT Tutorials

We have picked a couple of the simplest mozilla examples to explain in more detail, the “touch switch” thing which uses the capacitive switch capability of the IoT-Bus ESP32 processor and the LED lamp. We will see how to get the working on their own and then use the Mozilla IoT Rules engine to control the on-board LED on the IoT-BUs IO board to reflect the state of the capacitive switch.

- *LED*
- *Touch Switch*
- *Rules Engine*

CHAPTER 40

LED Thing Tutorial

If you haven't already installed your IDE do so now. We are going to use PlatformIo. See the section on *Getting Started with PlatformIO* for details. If you prefer to use Arduino you can follow this: [this:ref:guide <getting-started-with-arduino>](#).

Get the IoT-Bus examples from Github. Download or clone the PlatformIO examples from this [location](#). The Arduino examples can be found on [GitHub](#).

This tutorial requires an IoT-Bus Io ESP32 processor board. Once running you will be able to control the on-board LED on the IoT-Bus Io using the Mozilla-IoT browser interface. You will need to have installed the Mozilla-IoT gateway on a Raspberry PI as described [here](#).

```
#include <Arduino.h>
#include "Thing.h"
#include "WebThingAdapter.h"
```

These lines include the Arduino framework and the Mozilla-IoT device and adapter libraries. If you are using the examples for Platformio, these libraries and their pre-requisites will automatically be installed in your project.

Note: The ESPAsyncWebServer and AsyncTCP libraries are pre-requisites and also need installing.

```
//TODO: Hard-code your WiFi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";
```

Enter your WiFi ssid and password so that the IoT-Bus board can access your WiFi.

```
const int ledPin = 5; // manually configure LED pin
```

The LED is connected to GPIO 5.

```
WebThingAdapter* adapter;
```

This defines a pointer to a Mozilla-IoT adapter which is set when a new adapter is created in setup as below.

```
adapter = new WebThingAdapter("w25", WiFi.localIP());
```

```
const char* ledTypes[] = {"OnOffSwitch", "led", nullptr};
ThingDevice led("LED", "LED", ledTypes);
ThingProperty ledOn("On", "", BOOLEAN, "OnOffProperty");
```

These are the the lines of code that define the Mozilla-IoT Thing and its properties.

```
const char* ledTypes[] = {"OnOffSwitch", "led", nullptr};
```

This line defines an array of types ended by a null pointer. These types are @types, that is they are pre-defined types that the Mozilla-IoT platform understands semantically. That is, what they are and how to render and interface with them.

```
ThingDevice led("LED", "LED", ledTypes);
```

This line defines a LED named LED. Note the reference to ledTypes defining the device type or types. Officially these are described as capabilities. You can find the current list available [here](#).

```
ThingProperty ledOn("on", "", BOOLEAN, "OnOffProperty");
```

This defines a property “on” which has a property type of OnOffProperty. Again, this is a predefined property type.

Note: There is no connection between the property and the device or adapter at this point. Although there is no mention of an adapter here, it is an adapter that connects to a gateway and exposes its capabilities.

```
led.addProperty(&ledOn);
adapter->addDevice(&led);
adapter->begin();
```

These three lines add the ledOn property to the led, then add the device to the adapter and then start the adapter. Once the adapter has started it can be recognized by the gateway.

```
{
  "@context": "https://iot.mozilla.org/schemas/",
  "@type": ["Light", "OnOffSwitch"],
  "name": "LED",
  "description": "LED",
  "properties": {
    "on": {
      "@type": "OnOffProperty",
      "type": "boolean",
      "href": "/things/led/properties/on"
    }
  }
},
```

This json extract illustrates What is happening under the covers. The arduino-webthing library is turning the definitions above into JSON when it is polled by the gateway. This json describes the capabilities, properties, actions and events that are possible with this device or devices. In theory there is no limit to what can be described in the schema. However, in practice if you want to control it using Mozilla-IoT it needs to be an @type that it knows how to render or effectively fold into one of those types.

Most of the rest is boiler plate, but do note that you’ll want to take note of the ip address that is displayed on the serial monitor once WiFi has started because you can use that ip address to get the raw json response provided by the device once it is up and running. This is useful for debugging because you can see exactly what will be provided to the gateway.

```

adapter->update();
bool on = ledOn.getValue().boolean;
digitalWrite(ledPin, on ? HIGH : LOW); // active high led
if (on != lastOn) {
    Serial.print(led.id);
    Serial.print(": ");
    Serial.println(on);
}
lastOn = on;

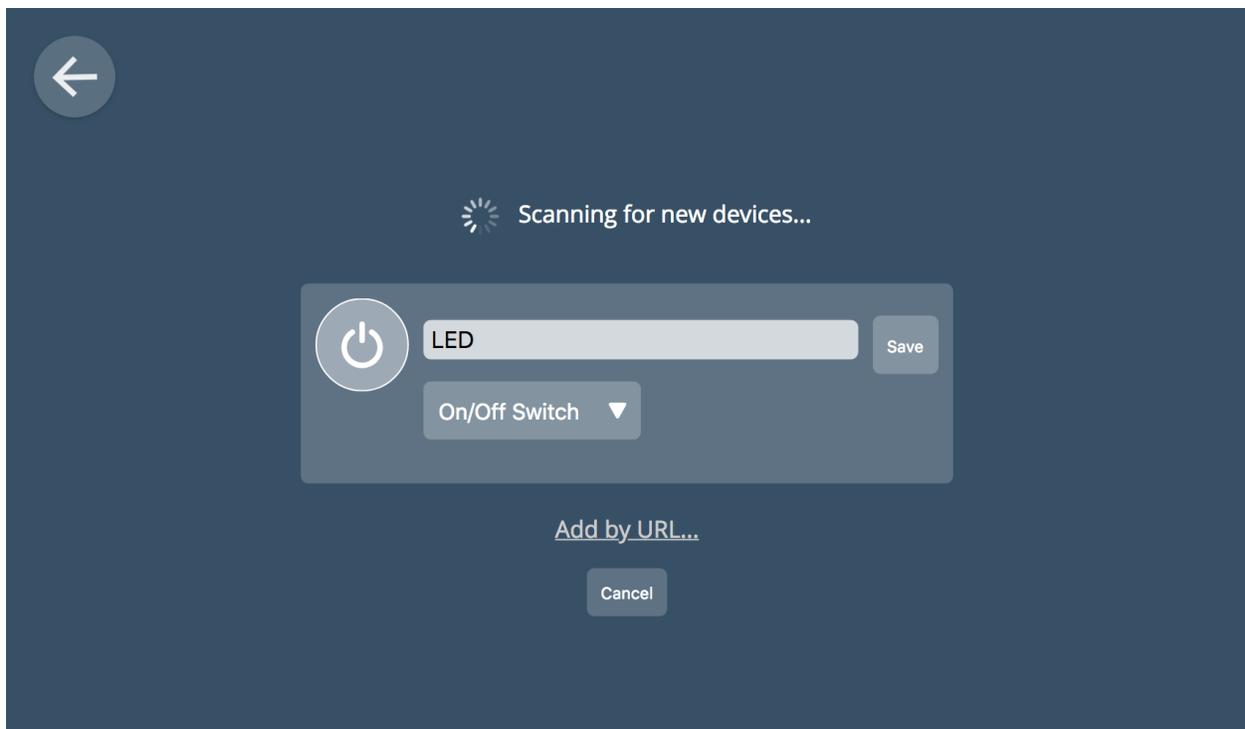
```

In the loop function, the latest state of the adapter is obtained. If we had changed the state of the LED locally we would need to call the update function to have it reflected through the gateway. We get that current value of the property ledOn. And we use it to set the value of the LED. if it has changed we print the changed value to the serial monitor.

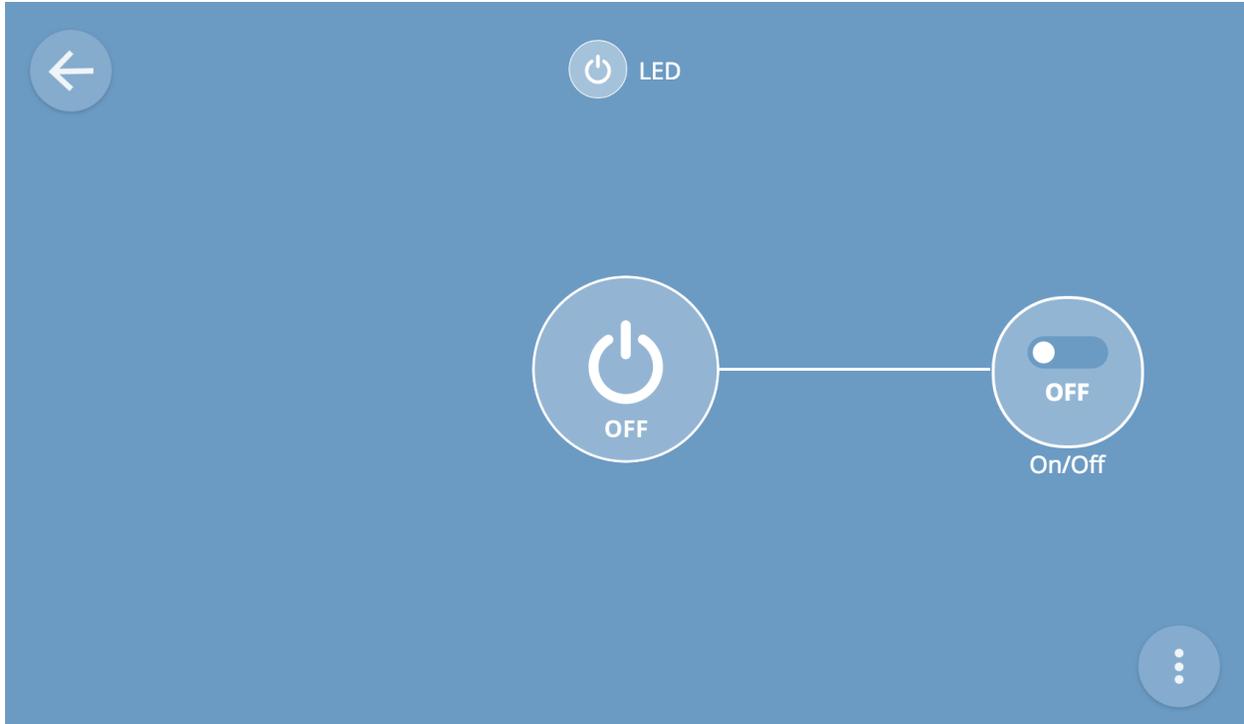
So now we have run through the code, let's create a Thing and add it to the gateway.

40.1 Creating a Thing

Start up the previously installed and configured Mozilla-IoT gateway on your Raspberry Pi and look for this screen.



Your Thing should be found. Save it and click Done. You should now be able to click on the thing and get a display like this:



The LED should respond to you turning it off and on in the Mozilla IoT interface! There are lots more examples in the Github file you downloaded or cloned.

The full code is shown below.

```

/**
 * Simple server compliant with Mozilla's proposed WoT API
 * Originally based on the HelloServer example
 * Tested on ESP8266, ESP32, Arduino boards with WINC1500 modules (shields or
 * MKR1000)
 *
 * This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/.
 */

#include <Arduino.h>
#include "Thing.h"
#include "WebThingAdapter.h"

//TODO: Hard-code your WiFi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";

const int ledPin = 5; // manually configure LED pin

WebThingAdapter* adapter;

const char* ledTypes[] = {"OnOffSwitch", "led", nullptr};
ThingDevice led("LED", "LED", ledTypes);
ThingProperty ledOn("on", "", BOOLEAN, "OnOffProperty");

bool lastOn = false;

```

(continues on next page)

(continued from previous page)

```
void setup(void) {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
  Serial.begin(115200);
  Serial.println("");
  Serial.print("Connecting to ");
  Serial.print(ssid);
  Serial.println("");
  #if defined(ESP8266) || defined(ESP32)
  WiFi.mode(WIFI_STA);
  #endif
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  bool blink = true;
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    digitalWrite(ledPin, blink ? LOW : HIGH); // active low led
    blink = !blink;
  }
  digitalWrite(ledPin, HIGH); // active low led

  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  adapter = new WebThingAdapter("w25", WiFi.localIP());

  led.addProperty(&ledOn);
  adapter->addDevice(&led);
  adapter->begin();
  Serial.println("HTTP server started");
  Serial.print("http://");
  Serial.print(WiFi.localIP());
  Serial.print("/things/");
  Serial.println(led.id);
}

void loop(void) {
  adapter->update();
  bool on = ledOn.getValue().boolean;
  digitalWrite(ledPin, on ? HIGH : LOW); // active high led
  if (on != lastOn) {
    Serial.print(led.id);
    Serial.print(": ");
    Serial.println(on);
  }
  lastOn = on;
}
```

Touch Switch Thing Tutorial

If you haven't already installed your IDE do so now. We are going to use PlatformIo. See the section on *Getting Started with PlatformIO* for details. If you prefer to use Arduino you can follow these *instructions*.

Get the IoT-Bus examples from Github. Download or clone the PlatformIO examples from here [GitHub](#). The Arduino examples can be found on [GitHub](#).

This tutorial requires an IoT-Bus Io ESP32 processor board and a simpler jumper lead connected to GPIO4. Once running you will be able to use a capacitive switch on a GPIO pin on the IoT-Bus Io to be exposed to the Mozilla-IoT browser interface. Holding the jumper wire will cause the switch to trip. You will need to have installed the Mozilla-IoT gateway on a Raspberry PI as described [here](#).

```
#include <Arduino.h>
#include "Thing.h"
#include "WebThingAdapter.h"
```

These lines include the Arduino framework and the Mozilla-IoT device and adapter libraries. If you are using the examples for Platformio, these libraries and their pre-requisites will automatically be installed in your project.

Note: The ESPAsyncWebServer and AsyncTCP libraries are pre-requisites and also need installing.

```
//TODO: Hard-code your WiFi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";
```

Enter your WiFi ssid and password so that the IoT-Bus board can access your WiFi.

```
// Uses a touch sensor to detect input and turn on a LED
int ledPin = 5; // choose the pin for the LED
int touchPin = 4; // choose the input pin - T0 is the same as GPIO4
```

These define the input pin and the local LED to show the status.

```
WebThingAdapter* adapter;
```

This defines a pointer to a Mozilla-IoT adapter which is set when a new adapter is created in setup as below.

```
adapter = new WebThingAdapter("adapter", WiFi.localIP());
```

```
const char* sensorTypes[] = {"binarySensor", nullptr};
ThingDevice touch("Touch", "ESP32 Touch Input", sensorTypes);
ThingProperty touched("Touched", "", BOOLEAN, "BooleanProperty");
```

These are the the lines of code that define the Mozilla-IoT device “Touch” and its property “Touched”.

```
const char* sensorTypes[] = {"binarySensor", nullptr};
```

This line defines an array of types ended by a null pointer. These types are @types, that is they are pre-defined types that the Mozilla-IoT platform understands semantically. That is, what they are and how to render and interface with them. In this case it is a binarySensor.

```
ThingDevice touch("Touch", "ESP32 Touch Input", sensorTypes);
```

This line defines a touch switch named Touch. Note the reference to sensorTypes which is what actually defines the device type or types. So we are saying Touch is a binarySensor. Officially these are described as capabilities. You can find the current list available at [this location](#).

```
ThingProperty touched("Touched", "", BOOLEAN, "BooleanProperty");
```

This defines a property “Touched” which has a property type of BooleanProperty. Again, this is a predefined property type.

Note: There is no connection between the property and the device or adapter at this point. Although there is no mention of an adapter here, it is an adapter that connects to a gateway and exposes its capabilities.

```
touch.addProperty(&touched);
adapter->addDevice(&touch);
adapter->begin();
```

These three lines add the touched property to the led, then add the touch device to the adapter and then start the adapter. Once the adapter has started it can be recognized by the gateway.

Most of the rest is boiler plate, but do note that you’ll want to take note of the ip address that is displayed on the serial monitor once WiFi has started because you can use that ip address to get the raw json response provided by the device once it is up and running. This is useful for debugging because you can see exactly what will be provided to the gateway.

```
int val = touchRead(T0); // get value using T0 / GPIO4

Serial.println(val);
if (val < threshold){
    sensorValue.boolean = true;
    digitalWrite(ledPin, HIGH);
}
else{
    sensorValue.boolean = false;
    digitalWrite(ledPin, LOW);
}
```

(continues on next page)

(continued from previous page)

```
}  
touched.setValue(sensorValue);  
adapter->update();  
delay(300);
```

Note: sensorValue must not be allocated on the stack. It must be in a location that can be referred to asynchronously, so we have allocated it globally for simplicity. You'll get a memory trap if you allocate it on the stack.

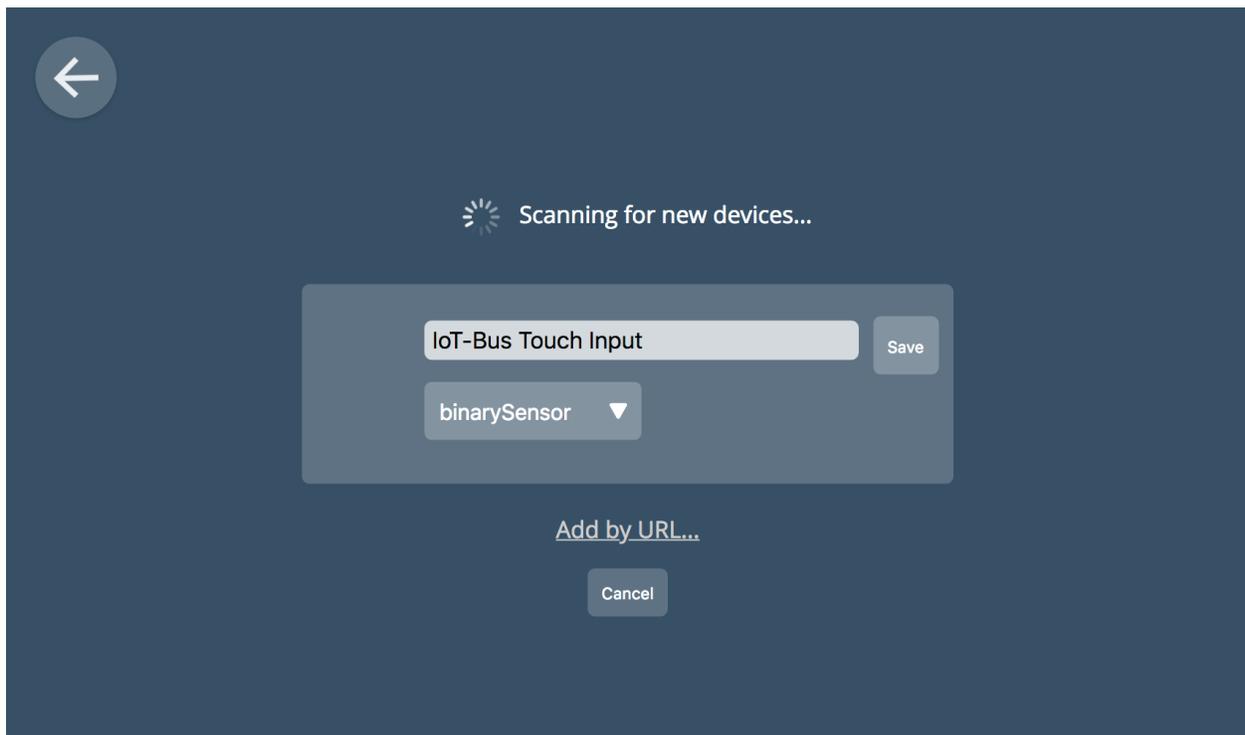
In the main loop we read the touch value and check against a threshold. We could also change this to reflect a multi-level value rather than a binary value if we wished. The ED and sensorValue are set to reflect the reading of T0 and the adapter is updated.

Note: T0 is the same as GPIO 4, either way of referencing the pin is fine.

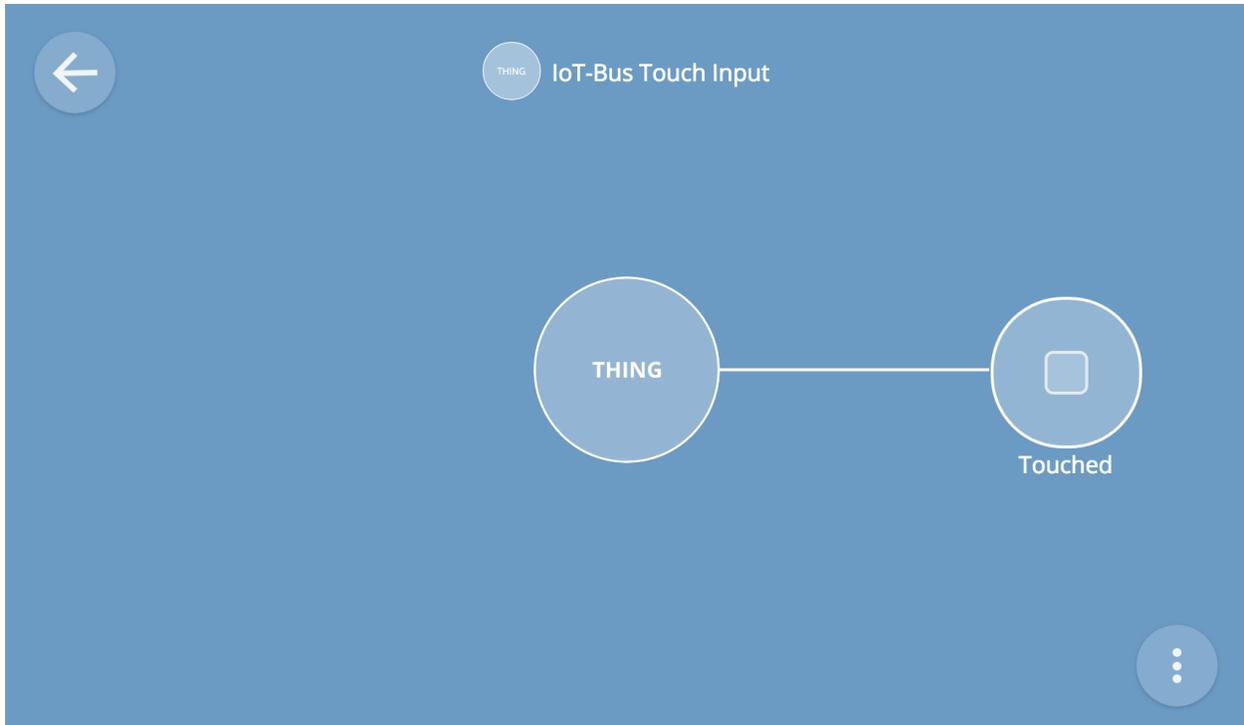
So now we have run through the code, let's create a Thing and add it to the gateway.

41.1 Creating a Thing

Start up the previously installed and configured Mozilla-IoT gateway on your Raspberry Pi and look for this screen.



Your Thing should be found. Save it and click Done. You should now be able to click on the thing and get a display like this:



The onboard LED will light when the wire is touched and turn off when the wire is released. At the same time the Mozilla IoT interface will reflect the status of the wire whether touched or not. There are lots more examples in PlatformIO format [here](#) or in Arduino format.

The full code is shown below.

```
#include <Arduino.h>
#include <Thing.h>
#include <WebThingAdapter.h>

/*
Simple binary sensor example using ESP32 capacitive touch input
This example code is in the public domain.
*/

//TODO: Hard-code your WiFi credentials here (and keep it private)
const char* ssid = ".....";
const char* password = ".....";

// Uses a touch sensor to detect input and turn on a LED
int ledPin = 5; // choose the pin for the LED
int touchPin = 4; // choose the input pin - T0 is the same as GPIO4

WebThingAdapter* adapter;

const char* sensorTypes[] = {"binarySensor", nullptr};
ThingDevice touch("Touch", "ESP32 Touch Input", sensorTypes);
ThingProperty touched("true", "", BOOLEAN, "BooleanProperty");
ThingPropertyValue sensorValue;

int threshold = 40;

void setup() {
```

(continues on next page)

(continued from previous page)

```
Serial.begin(115200);

// Start WiFi
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
Serial.println("");

// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

// Initialize MOZ IoT thing
adapter = new WebThingAdapter("adapter", WiFi.localIP());
touch.addProperty(&touched);
adapter->addDevice(&touch);
adapter->begin();

pinMode(ledPin, OUTPUT); // declare LED as output
}

void loop() {

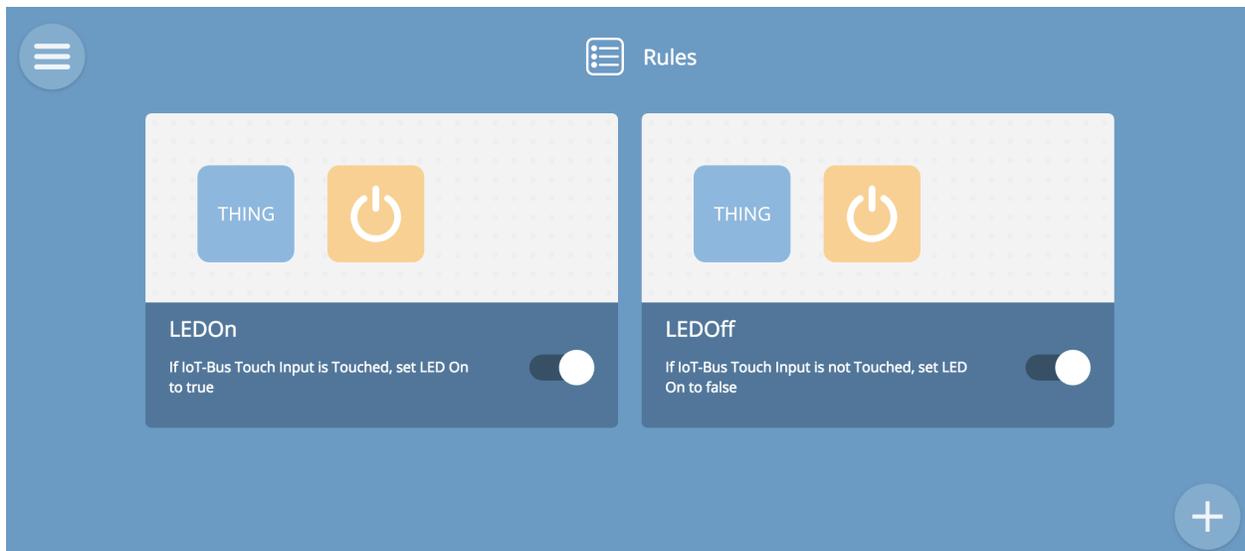
    int val = touchRead(T0); // get value using T0 / GPIO4

    Serial.println(val);
    if (val < threshold){
        sensorValue.boolean = true;
        digitalWrite(ledPin, HIGH);
    }
    else{
        sensorValue.boolean = false;
        digitalWrite(ledPin, LOW);
    }
    touched.setValue(sensorValue);
    adapter->update();
    delay(300);
}
```

Mozilla Rules Engine

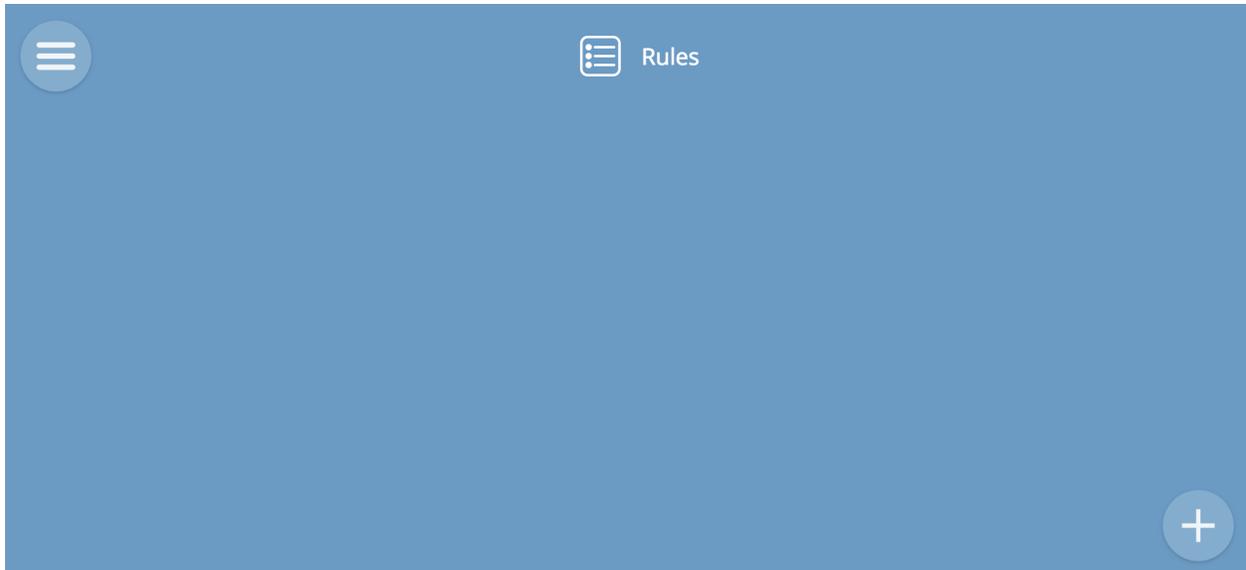
In this tutorial we'll use the the Mozilla Things created in the previous two tutorials, the Touch Thing and the LED Thing. We're going to use the Rules Engine to use the input from one Thing ,the Touch Thing to control the other , the LED Thing. What we want to do is have the LED Thing on when we hold the Touch wire on the other Thing and off when when we release it.

So we are going to create two rules.

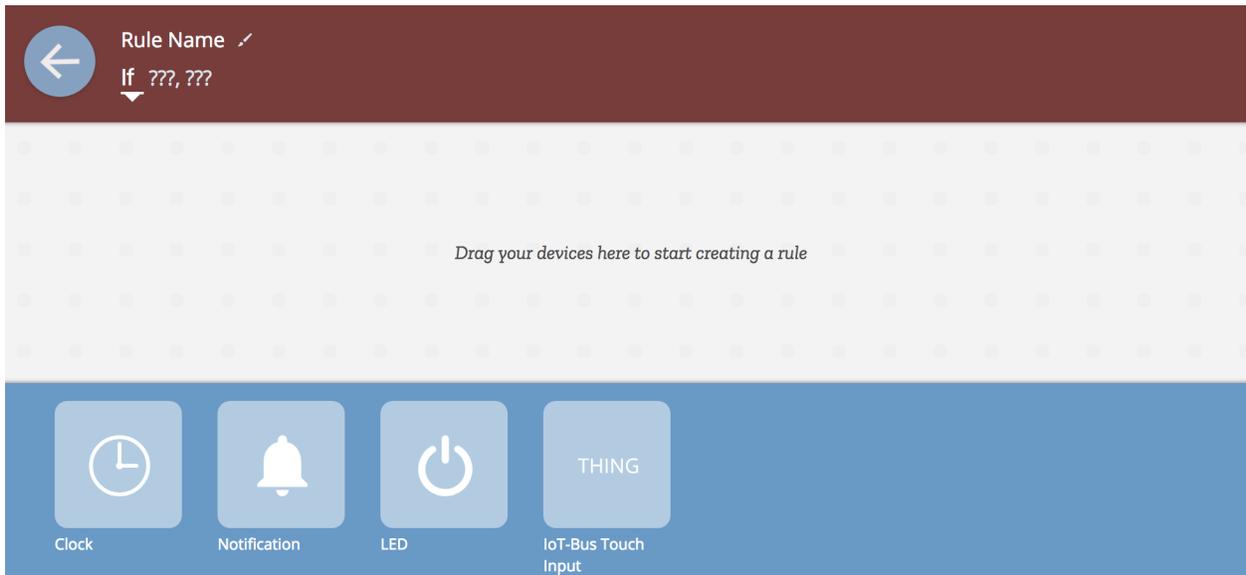


This is what the rules engine will look like when we're finished.

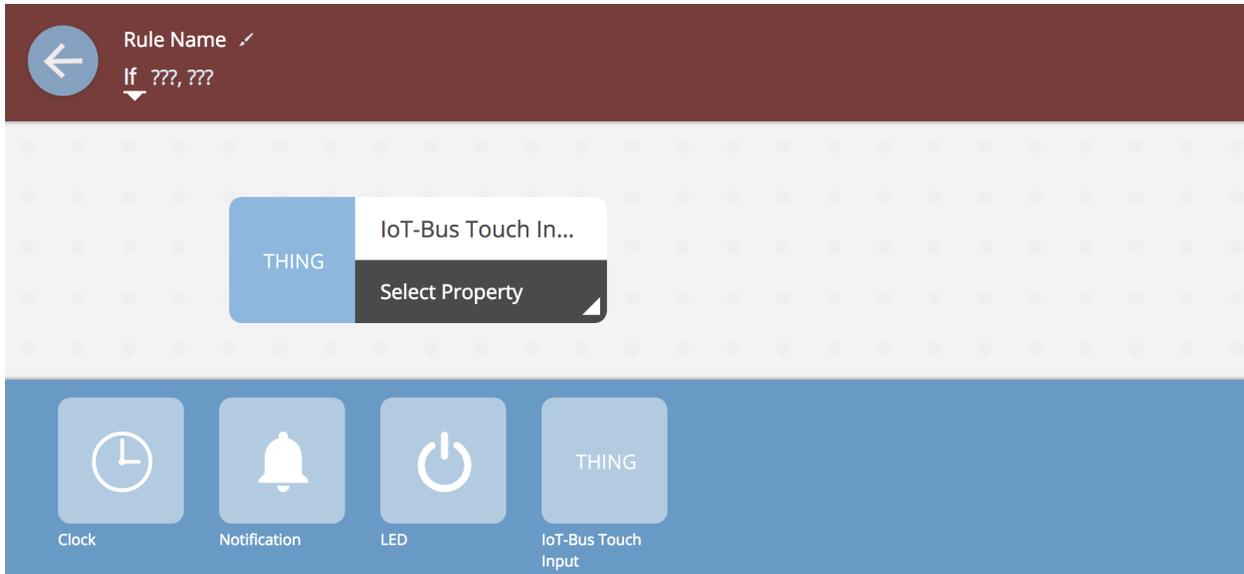
Let's get started. It's a very easy process. Use the menu icon in the top left corner to select the rules engine. You'll see this screen:



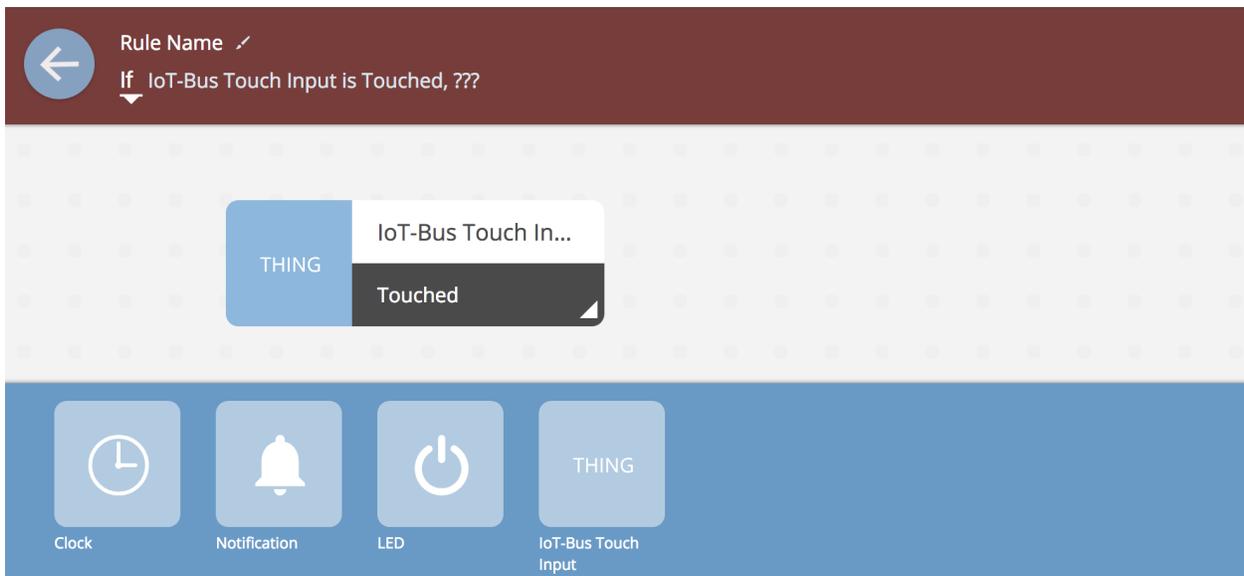
Click the add button on the bottom right and you'll see this:



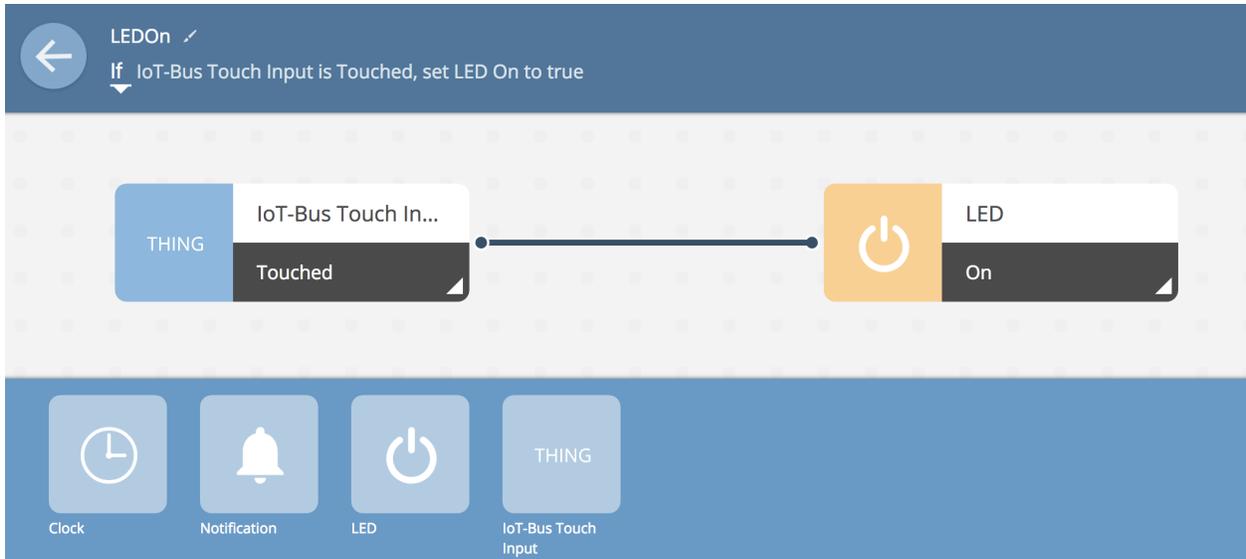
You should be able to see the devices LED and IoT-Bus Touch Input. If you can't, you'll need to go back and add them according to the two previous tutorials.



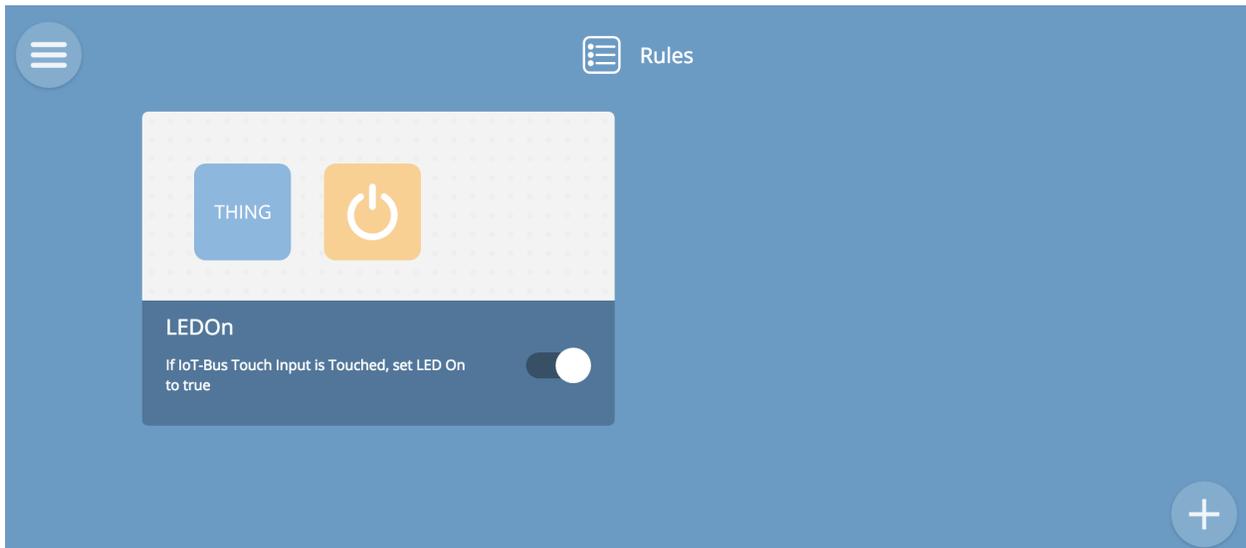
Creating a rule is a drag and drop or a point and click section process. Drag the Touch Input icon from the bar on the bottom and move it up and to the left into the input box.



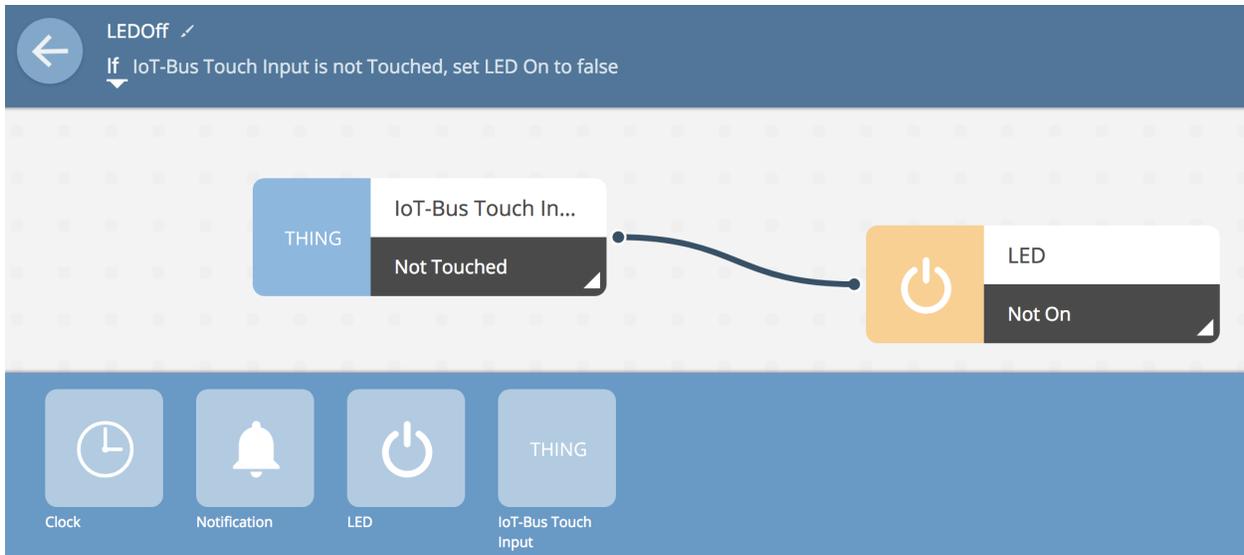
The select the Touched property from the choices given when you select the bottom right hand corner of the rule icon. In a similar fashion drag the LED icon to thr right, output box and select the On property. When you've done that you've connected the two properties on their respective devices into a rule. Name the rule something useful like LEDOn or Touched or whatever you like.



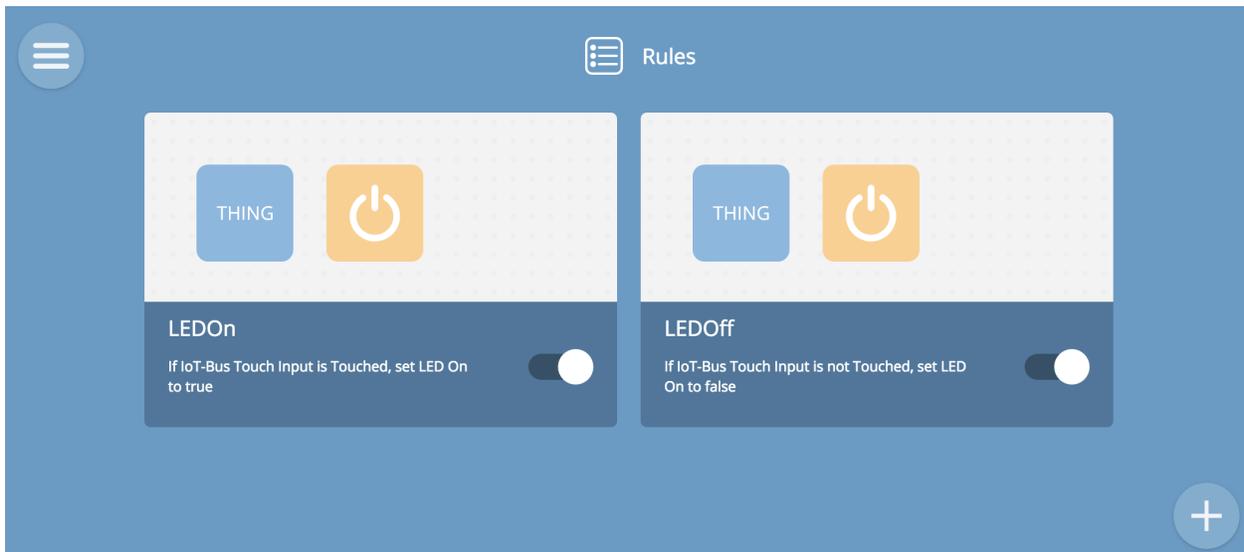
If you step back out of rule editing, the rule should look like this:



In a similar fashion add the second rule for not touched / LED off.



Once you have done that, you should see the finished result. Now when you touch the wire on one Thing, the status is reflected on the other Thing by the on-board LED. Easy!



43.1 Arduino

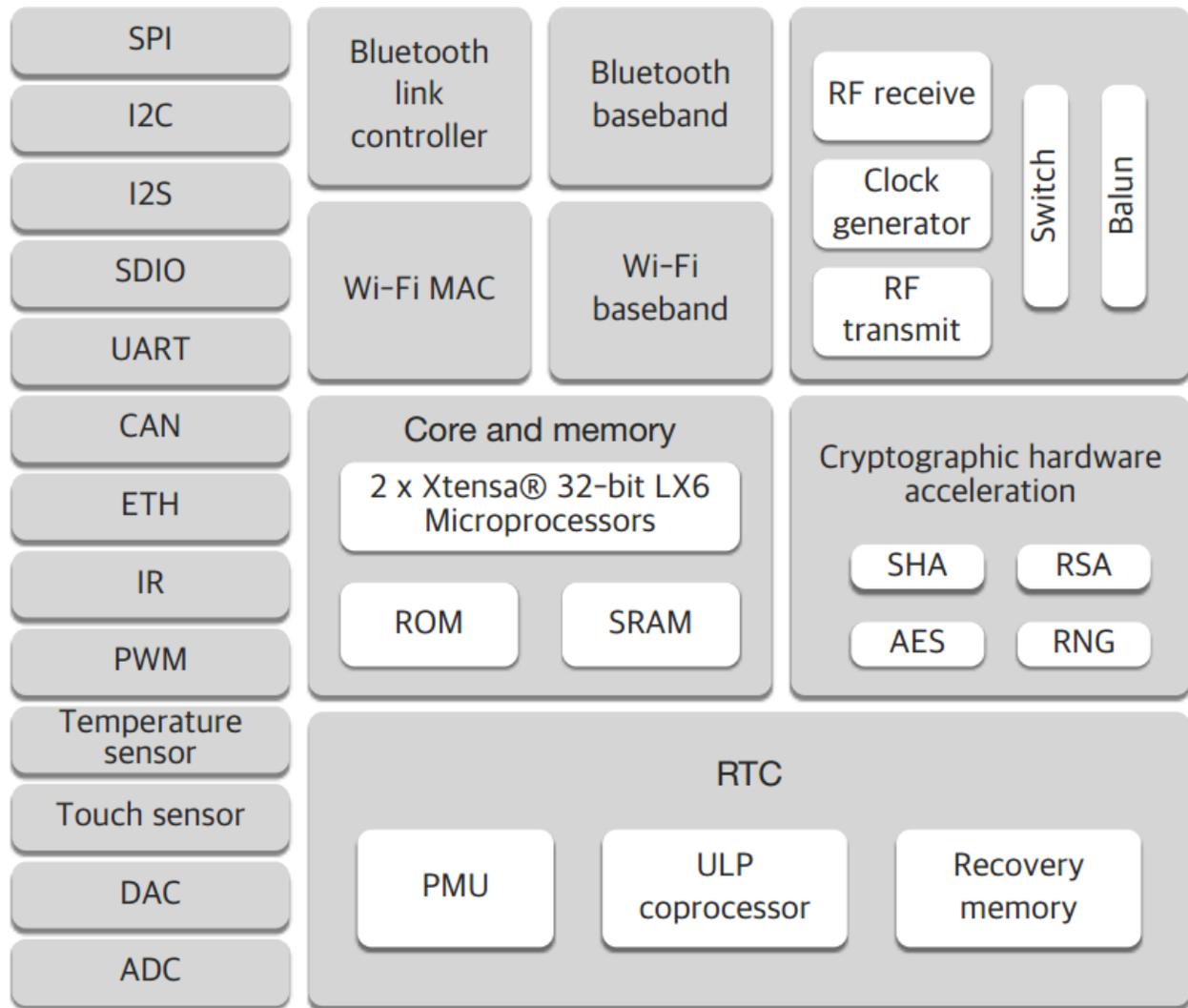
Arduino framework enables cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

43.2 ESP-IDF

Espressif IoT Development Framework. Official development framework for ESP32.

44.1 Espressif32

ESP-WROOM32 WiFi Bluetooth Combo Module



The ESP-WROOM-32 WiFi Bluetooth combo module is ultra high performance and ultra low-power consumption FCC-approved (2AC7Z-ESPWROOM32) Wi-Fi and Bluetooth combo wireless platform based on ESPRESSIF ESP32 chipset. It integrates a dual-core processor, 448 KByte ROM, 520 KByte SRAM, 16 KByte SRAM in RTC, 802.11 b/g/n/e/I Wi-Fi, Bluetooth v4.2 BR/EDR & BLE, clocks & Times, abundant peripheral Interfaces and security mechanism.

The ESP-WROOM-32 WiFi Bluetooth combo module provides SDK Firmware for fast on-line programming and open source toolchains based on GCC for development support. It is designed for Generic low power IoT sensor hub, loggers, video streaming for camera, Wi-Fi & Bluetooth enabled devices, Home automation and mesh network applications, aimed at makers, hardware engineers, software engineers and solution providers. ESP32 is a single chip 2.4 GHz Wi-Fi and Bluetooth combo chip designed with TSMC ultra low power 40 nm technology. It is designed and optimized for the best power performance, RF performance, robustness, versatility, features and reliability, for a wide variety of applications, and different power profiles.

ESP32 is the most integrated solution for Wi-Fi + Bluetooth applications in the industry with less than 10 external components. ESP32 integrates the antenna switch, RF balun, power amplifier, low noise receive amplifier, filters, and power management modules. As such, the entire solution occupies minimal Printed Circuit Board (PCB) area. ESP32 is designed for mobile, wearable electronics, and Internet of Things (IoT) applications. It has many features of the state-of-the-art low power chips, including fine resolution clock gating, power modes, and dynamic power scaling.

Espressif32 Key Features

Core

- CPU and Memory: Xtensa® 32-bit LX6 Dual-core processor, up to 600 DMIPS.
- 448 KByte ROM
- 520 KByte SRAM
- 16 KByte SRAM in RTC.
- QSPI can connect up to 4* Flash/SRAM, each flash should be less than 16 MBytes.
- Supply Voltage: 2.2V~3.6V

WiFi

- 802.11 b/g/n/e/i
- 802.11 n (2.4 GHz), up to 150 Mbps
- 802.11 e: QoS for wireless multimedia technology.
- WMM-PS, UAPSD
- MPDU and A-MSDU aggregation
- Block ACK
- Fragmentation and de-fragmentation
- Automatic Beacon monitoring/scanning
- 802.11 i security features: pre-authentication and TSN
- Wi-Fi Protected Access (WPA)/WPA2/WPA2-Enterprise/Wi-Fi Protected Setup (WPS)
- Infrastructure BSS Station mode/SoftAP mode
- Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode and P2P Power Management
- UMA compliant and certified
- Antenna diversity and selection

Bluetooth

- Compliant with Bluetooth v4.2 BR/EDR and BLE specification
- Class-1, class-2 and class-3 transmitter without external power amplifier
- Enhanced power control
- +10 dBm transmitting power
- NZIF receiver with -98 dBm sensitivity
- Adaptive Frequency Hopping (AFH)
- Standard HCI based on SDIO/SPI/UART ? High speed UART HCI, up to 4 Mbps
- BT 4.2 controller and host stack
- Service Discover Protocol (SDP)
- General Access Profile (GAP)
- Security Manage Protocol (SMP)
- Bluetooth Low Energy (BLE)
- ATT/GATT
- HID
- All GATT-based profile supported
- SPP-Like GATT-based profile
- BLE Beacon
- A2DP/AVRCP/SPP, HSP/HFP, RFCOMM
- CVSD and SBC for audio codec
- Bluetooth Piconet and Scatternet

Clocks and Timers

- Internal 8 MHz oscillator with calibration
- Internal RC oscillator with calibration
- External 2 MHz to 40 MHz crystal oscillator
- External 32 kHz crystal oscillator for RTC with calibration
- Two timer groups, including 2 x 64-bit timers and 1 x main watchdog in each group
- RTC watchdog

Peripheral Interface

- 12-bit SAR ADC up to 18 channels
- 2 x 8-bit D/A converters
- 10 x touch sensors
- Temperature sensor
- 4 x SPI, 2 x I2S, 2 x I2C, 3 x UART
- 1 host (SD/eMMC/SDIO), 1 slave (SDIO/SPI)
- Ethernet MAC interface with dedicated DMA and IEEE 1588 support

- CAN 2.0
- IR (TX/RX)
- Motor PWM, LED PWM up to 16 channels
- Hall sensor
- Ultra low power analog pre-amplifier

Security

- IEEE 802.11 standard security features all supported, including WPA, WPA/WPA2 and WAPI
- Secure boot
- Flash encryption
- 1024-bit OTP, up to 768-bit for customers
- Cryptographic hardware acceleration: -AES-HASH(SHA-2) library-RSA-ECC-Random Number Generator (RNG)