

*A guide to creating with the Onion Omega2*



*Volume 1*

# Omega2 Project Book



*onion.io*

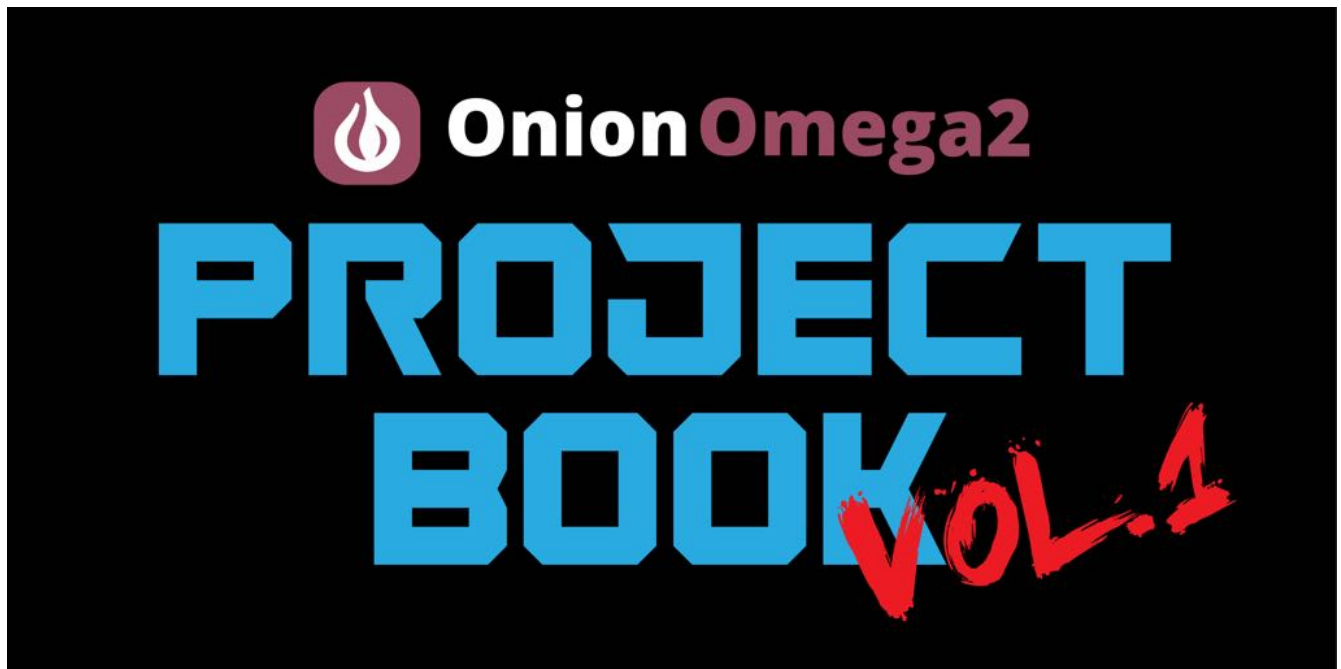


# Contents

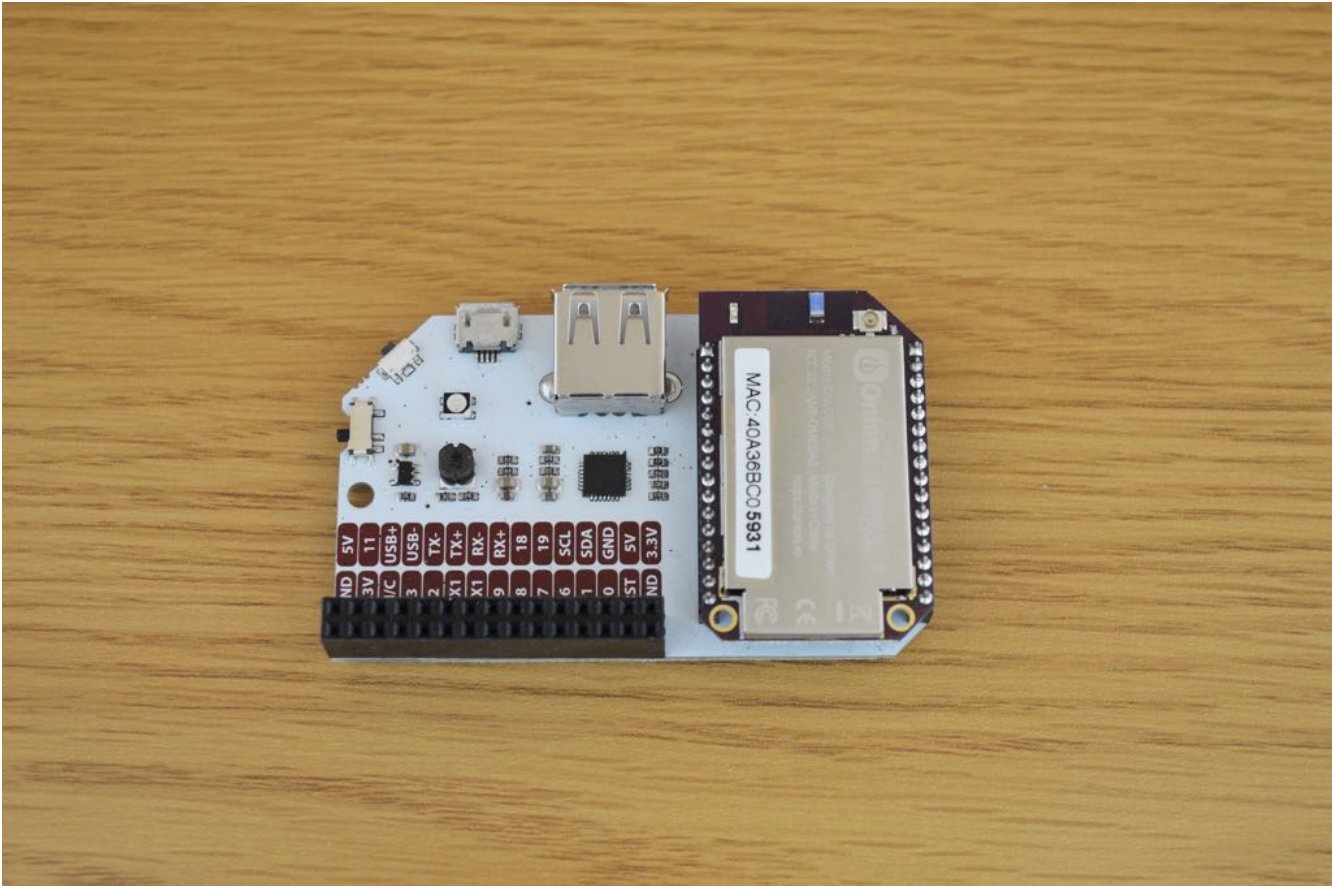
<b>1</b>	<b>Onion Omega2 Project Book Vol. 1</b>	<b>5</b>
	Introduction to the Omega2 . . . . .	7
	Getting Started . . . . .	23
	The Command Line . . . . .	23
	File Editing on the Omega . . . . .	28
	Intro to Python . . . . .	29
	Where Can I Learn More? . . . . .	32
	Where to Get More Onion Products . . . . .	32
	Reporting Issues . . . . .	33
<b>2</b>	<b>Starter Projects</b>	<b>35</b>
	Morse Code on an LED . . . . .	36
	LED Traffic Light . . . . .	41
<b>3</b>	<b>OLED Expansion Projects</b>	<b>47</b>
	Ambient Temperature Monitor . . . . .	49
	QR Code Generator . . . . .	62
	News Flash Headlines . . . . .	66
	Stock Ticker . . . . .	73
	Twitter Feed Display . . . . .	78
<b>4</b>	<b>IoT Projects</b>	<b>85</b>
	Weather Station . . . . .	87
	Time-Lapse Camera . . . . .	99
	Alarms based on an Online Calendar . . . . .	107
	Thermal Printer . . . . .	114
	Thermal Printer - A Compact Version . . . . .	121
	Smart Plant - Measuring Plant Data . . . . .	133
	Smart Plant - Visualizing Plant Data . . . . .	147
	Smart Plant - Twitter Alerts . . . . .	172
	Smart Plant - Automatic Plant Watering . . . . .	198
	Smart Plant - A Single Power Supply . . . . .	226
	Temperature-Based Smart Fan . . . . .	236
	IoT Lock . . . . .	247
	IoT Lock - Control with a Tweet . . . . .	256
<b>5</b>	<b>Audio Projects</b>	<b>267</b>
	AirPlay Speaker . . . . .	268
	Bluetooth Speaker . . . . .	277

<b>6 Wireless Projects</b>	<b>287</b>
Mobile WiFi Network Scanner . . . . .	288
OctoPrint 3D Printing Server . . . . .	297
Mobile Network File Server . . . . .	304
Omega WiFi Router . . . . .	309
Omega WiFi Range Extender . . . . .	316
Omega WiFi Ethernet Bridge . . . . .	320

# 1 | Onion Omega2 Project Book Vol. 1



Welcome to the Volume 1 of the Omega2 Project Book!



Not sure what to do with your Omega? Take a look at some of our project tutorials, you'll learn how to use the Omega AND make a whole bunch of IoT gadgets!

## The Projects

This Project Book Volume consists of 22 projects that are outlined through the course of 28 tutorials. The projects are split up according to five categories:

1. **Starter Projects**
  - Two projects to get you comfortable working with the Omega
2. **OLED Expansion Projects**
  - Projects that use the OLED Expansion to communicate various information to users
3. **IoT Projects**
  - True Internet of Things projects that use connectivity to bring intelligence to everyday objects or existing technology
4. **Audio Projects**
  - Take advantage of the fact that the Omega is well equipped to handle sound input and output since it runs a Linux Operating System
5. **Wireless Projects**
  - Use the Omega's extensive networking capabilities to make a variety of computer networking tools

## Contributors

The contributors for Project Book Vol. 1:

- Lazar Demin
- Gabriel Ongpauco
- Zheng Han
- James Liu

## Introduction to the Omega2

Wondering what exactly the Omega2 is all about? You've come to the right place!



## What is an IoT Computer?

An IoT computer is a Linux computer designed specifically for use in building connected hardware applications meant for IoT. As a refresher, IoT (which stands for the Internet of Things) is the next big technological wave. It involves providing intelligence and internet connectivity to physical devices of all sorts, this includes home appliances, cars, buildings, really anything. The newly smart devices will be able to collect and exchange data as well as receive instructions. All with the goal of having all of our devices work together to improve our lives, whether through automating your sprinkler system to keep your lawn or garden green without any intervention, having your coffee machine brew automatically and sounding your alarm early when there's heavy traffic, or improving the efficiency of your manufacturing and shipping business.

So then, what is an IoT computer used for, exactly? The Omega2 IoT computer is meant to be a development platform for all things IoT, whether you want to experiment, build yourself some sweet gadgets for fun, or prototype and create an IoT product.

What makes the Omega an IoT computer:

- Small form factor
- Power efficiency
- Processing, networking, and encryption capabilities
- Flexibility that comes from running a Linux OS
  - Support for many programming languages and many simultaneous processes

## Comparisons

If you're familiar with existing development boards, the Omega2 can be categorized as **something in between an Arduino and a Raspberry Pi**.

When compared to an **Arduino Uno**, the Omega has several advantages since it is a computer while the Uno is a microcontroller:

- It is powered by a full processor, not a microcontroller
- Runs a full Linux (soft) real-time operating system
  - Supports many programming languages
  - Has a filesystem with storage
- Networking (wireless and wired) support built-in, can be programmed

Some things the Arduino Uno can do that the Omega cannot:

- Support for analog inputs and outputs
- Provide cycle accurate signals for controlling very low level hardware

The Omega is more similar to the **Raspberry Pi**, since they're both computers. Being an IoT computer, the Omega does some things differently from the Raspberry Pi single-board computer:

- Comes with on-board storage and the OS preloaded - getting it up and running for the first time takes about two minutes
- Built-in WiFi radio and capabilities (not all Raspberry Pi models come with this on-board)
- Lower power consumption

Since the Omega is not a general purpose computer, there are a few things the Raspberry Pi family can do that the Omega cannot:

- Output HD video



- The Omega’s IoT-centric purpose means it can drive smaller screens, but not a computer monitor or TV
- Run a graphical Linux desktop
  - The Omega is not a general purpose computer; it is meant for use-cases that prioritize connectivity and lower power consumption
- Generally, the Raspberry Pi’s SoCs have more processing power
  - The Omega prioritizes power efficiency over processing performance

So, the Omega is more powerful and flexible than the Arduino Uno, and it provides internet & network connectivity right out of the box. However, it is not as powerful as the Raspberry Pi and cannot output video to TVs or monitors, since it is geared towards power efficiency and out-of-the-box usage with the built-in storage, preloaded OS, and WiFi networking. Best of all, the Omega is a very affordable module, with prices that are lower than or very closely rival its neighbors in the development board space.

Note that we at Onion use and love both the Arduino Uno (as well as other Arduino products) and all Raspberry Pi models. We wanted to provide a development platform specifically geared towards IoT, and that is how the Omega came into being.

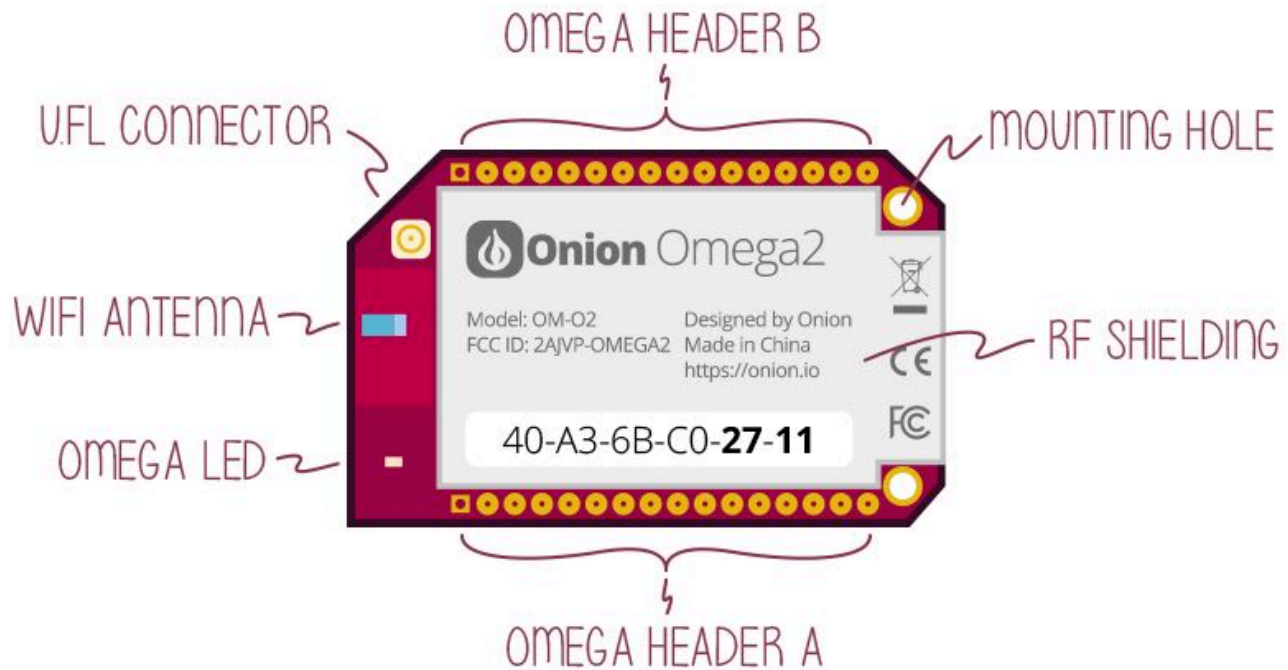
## History

Back in late 2014, we were trying to make a thermal printer automatically print our To Do lists from Evernote. We struggled quite a bit during this endeavour and realized the need for an IoT computer. So we got to work and in the spring of 2015, Onion launched the [original Omega](#) on Kickstarter. The response was great! About 4,400 makers, coders, and tinkerers backed the campaign. We spent a considerable amount of time in Shenzhen, China and made the Omega and it’s assorted Docks and Expansions into a reality, with a variety of speed-bumps along the way. This process provided many great lessons about product design, manufacturing, and delivery.

After listening to feedback from the community and taking a hard look at the state of IoT, Onion decided to launch the Omega2 in 2016. The Omega2 would have two models with different specs, be even lower cost than the original, and be fully FCC certified! The [Omega2 Kickstarter](#) was successful beyond belief, with over 16 thousand backers pledging to the campaign!

## Hardware

The Omega2 IoT Computer:



The Specs:

	Omega2	Omega2+
Processor	580MHz MIPS CPU	580MHz MIPS CPU
Memory	64MB Memory	<b>128MB Memory</b>
Storage	16MB Storage	<b>32MB Storage</b>
USB	USB 2.0	USB 2.0
MicroSD Slot	No	<b>Yes</b>
WiFi adapter	b/g/n Wi-Fi	b/g/n Wi-Fi
Operating Voltage	<b>3.3V</b>	<b>3.3V</b>

## SoC

The Omega2 uses the Mediatek MT7688AN System-on-a-Chip (SoC). The processor is MIPS 24KEc, little-endian, 32-bit RISC core that operates at 580 MHz. For the truly curious, it has a 64 KB I-Cache and 32 KB D-Cache.

While this family of SoCs has traditionally been used in routers, this is very much a real CPU (and not a microcontroller) like you would find in a smartphone or laptop. It's just a different architecture (MIPS as opposed to ARM in smartphones or x86 in laptops & desktops) and operates at a lower frequency: about a quarter of the speed of a modern laptop CPU.

The lower clock speed and the MIPS architecture of the SoC lend to the Omega's low power consumption and low heat generation. This makes it ideal for use in the space and energy constrained use cases common for IoT applications.

SoC	
SoC	MediaTek MT7688AN
Architecture	MIPS 24KEc (RISC, 32-bit)

---

SoC	
Endianness	Little
Clock Speed	580 MHz
I-Cache	64 KB
D-Cache	32 KB

---

## Memory

The Omega2 comes with **128 MB** of memory and the Omega2+ with **256 MB**. Both models use DDR2 DRAM (Dynamic Random Access Memory).

## Storage

While technically still memory, we refer to the Omega's onboard SPI flash memory as storage since it provides persistent, non-volatile memory that will not be destroyed when the Omega is powered off. The flash storage is where the Operating System (OS), programs, and all other files are stored. This flash storage is to the Omega what a hard-drive is to a laptop computer.

The Omega2 comes with **16 MB** flash storage while the Omega2+ has **32 MB**.

## Micro-SD Card Slot

The Omega2+ additionally has a Micro-SD card slot on the underside that can be used to extend the storage available to the system. It is also possible to boot the Omega from the SD card.

## Networking

The Omega2 has support for both wireless and wired networking.

## WiFi

The Omega supports 2.4 GHz IEEE 802.11 b/g/n WiFi with a 150 Mbps PHY data rate. The antenna is 1T1R, meaning that it is used for both transmitting and receiving by virtue of time-multiplexing. By default, the Omega uses the on-board ceramic chip antenna. However, there is also a u.FL connector onboard for those who wish to use external antennas.

The Omega's WiFi interface supports hosting its own WiFi Access Point, connecting to existing WiFi networks, or both simultaneously.

## Ethernet

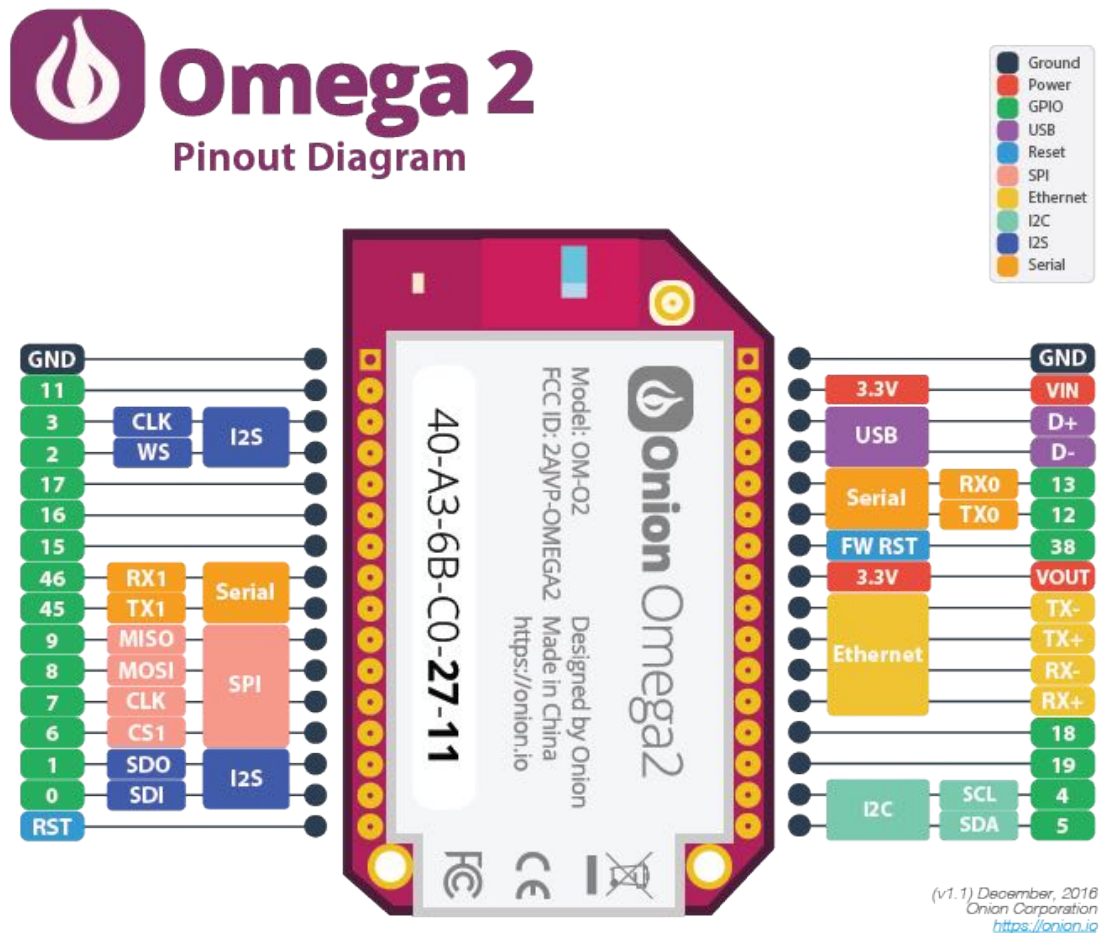
The Omega supports 10M/100M wired ethernet network connectivity as well when used with a Dock and an Ethernet Expansion.

## GPIOs

The Omega2 has twelve General Purpose Input/Output (GPIO) pins that can be controlled by the user.

## GPIO Mapping

On the Omega, the GPIOs are laid out in two banks:



## Electrical Characteristics

Before interfacing with external electronics, the electrical characteristics of the Omega should be noted. The most important part is that the Omega's GPIOs are **3.3V** and **are not** 5V tolerant. Interfacing with 5V devices directly may damage your Omega's GPIOs!

## Digital Signal Voltage Level

**The Omega2's GPIOs are not 5V input tolerant!**

See the table below for the GPIOs' operating voltages:

Parameter	Minimum (V)	Maximum (V)
Input HIGH	2.0	3.6
Input LOW	-0.3	0.8
Output HIGH	2.4	3.3
Output LOW	—	0.4

**Warning: Connecting a signal to an input pin below the minimum LOW or above the maximum HIGH voltages may damage your Omega!**

Standard 5V logic devices typically accept 3.3V as a logical HIGH, however, they output logical HIGH in the range of 4.4V to 5V. This means that the Omega can *output* to a 5V logical device, but *input* from the 5V logic device would damage the GPIO input circuitry.

## Current Limits

The Omega's GPIOs have current limitations: a GPIO can source or sink up to 8mA. This is not a high current, so try to limit the current demands on the GPIOs to avoid damaging your device. Use external circuits controlled by the Omega's GPIOs if your project has higher current demands.

## USB

The Omega supports the USB 2.0 protocol as a host. Use USB devices to extend the functionality of your system. Most Docks provide a USB Type-A socket for easier connectivity.

## Serial Protocols

The Omega supports several useful serial communication protocols.

## I2C

Support is included for the ubiquitous Inter-Integrated Circuit interface, also known as I2C. I2C is a master-slave bus protocol that allows a master device to interact with and control multiple slave devices. It is fast and reliable and only uses two data lines: SCL for the bus clock and SDA for the serial data.

Signal	Purpose	Omega GPIO
SCL	Clock Lane	4
SDA	Data Lane	5

The Omega acts as an I2C bus master, issuing commands and reading responses from other devices and chips. For example, the Omega controls the Servo (PWM), Relay, and OLED Expansions using I2C.

See our [Documentation on I2C](#) for more details.

## UART

The Universal Asynchronous Receiver Transmitter (UART) protocol is supported as well. A UART is meant for direct communication between two devices, with no concept of a master or a slave. It uses two data lines: one for transmitting and one for receiving. Note that the transmitting (TX) pin of one device, needs to be connected to the receiving (RX) pin of the other device, and vice versa.

The Omega has two separate UARTs, meaning it can be connected via UART to two separate devices. By default, `UART0` is configured to provide a serial interface to the Omega's command line. On the Expansion and Mini Docks, `UART0` is connected to a USB-to-Serial chip that allows access to the command line through the Micro-USB connection. The other one, `UART1` is exposed on the Expansion Header, and is free to be used to communicate with other devices. On the Arduino Dock 2, it is hard-wired to the ATmega microcontroller for direct and reliable communication.

Signal	Omega2 GPIO
UART0 TX	12
UART0 RX	13
UART1 TX	45
UART1 RX	46

See our [Documentatation on UART](#) for more details.

## SPI

Finally, the Omega also supports the Serial Peripheral Interface (SPI) protocol. SPI is a four-wire, master-slave, synchronous communication protocol that can run at high speeds and transfer lots of data. It is generally used to connect microprocessors or microcontrollers to sensors, memory, and other peripherals. The SPI Master can have multiple connected SPI slaves, but each requires it's own *Slave Select* (also known as *Chip Select*) signal that indicates that specific devices is the current subject of communication.

The Omega uses SPI to communicate with the on-board flash memory that is used as storage for the Operating System and all of the files.

Signal	Description	Omega GPIO
SCK	System Clock	7
MOSI	Master Out, Slave In - Data sent from the Master to the Slave	8
MISO	Master In, Slave Out - Data sent from the Slave to the Master	9
CS0	Chip Select 0	Internally connected to flash storage
CS1	Chip Select 1	6

See our [Documentatation on SPI](#) for more details.

## Software

The Omega2 runs an Onion-customized version of the [LEDE \(Linux Embedded Development Environment\) operating system](#), a distribution based on OpenWRT.

The Omega's OS comes equipped as a web server by default, so that other devices on the local network can interact with the Omega through a browser.

## Supported Programming Languages

- C & C++
- Python
- NodeJS
- Rust
- Ruby
- PHP
- Perl
- GoLang

## Installing Software

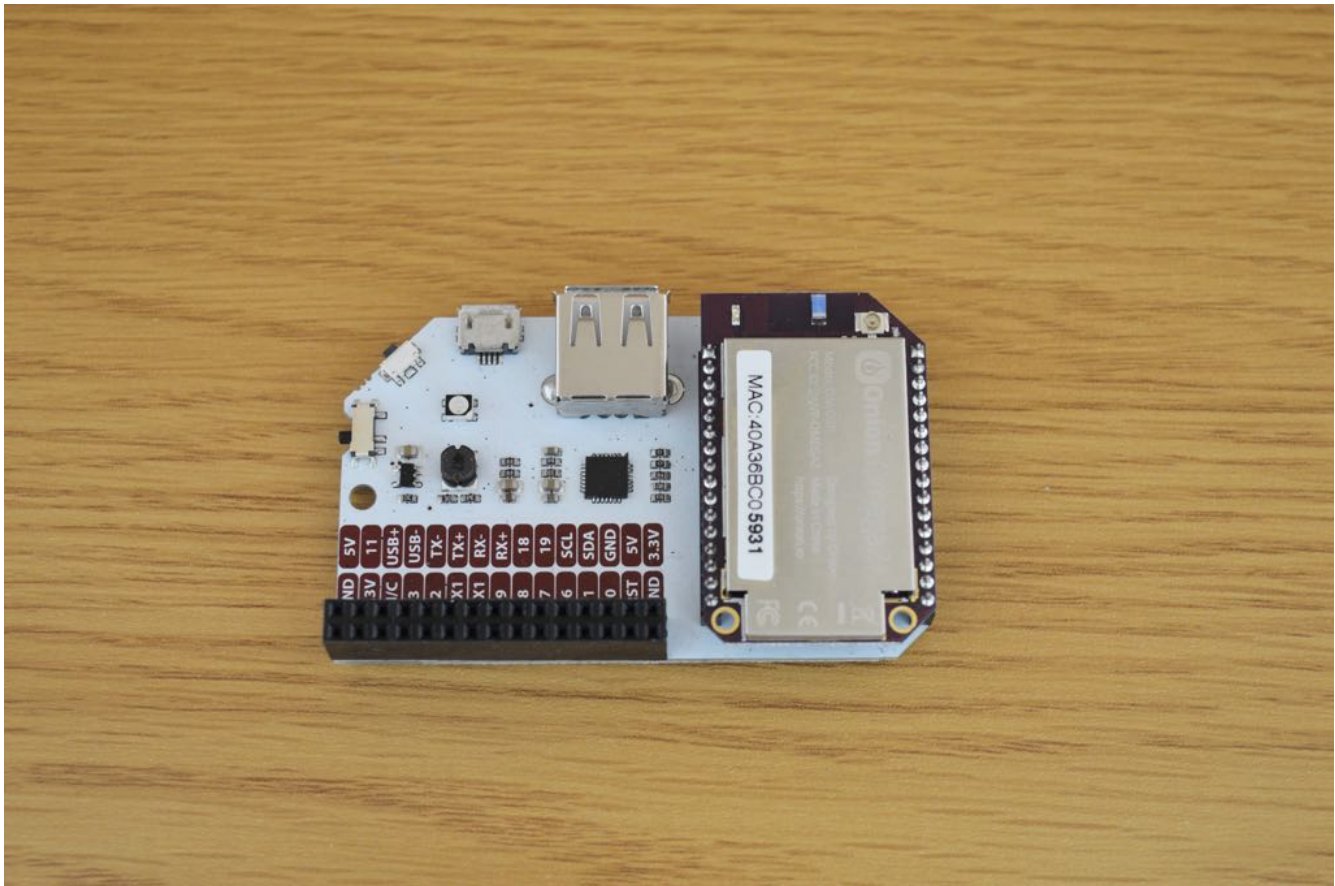
Since the processor's architecture is MIPS, all applications need to be compiled specifically for the MIPS architecture. So you unfortunately can't just download regular Linux installation packages and run them on the Omega. Luckily, LEDE has it's own package management system in the `opkg` program. The `opkg` utility allows users to access online software package repositories and install the packages. By default, it only looks at Onion's package repository, but the main LEDE package repositories can be enabled easily.

## Docks

To make the Omega incredibly easy to use, it can be plugged directly into any of a number of **Docks** provided by Onion. All Docks can be powered with a regular Micro-USB cable, they contain a regulator circuit to safely provide 3.3V to power the Omega safely. Each Dock adds unique functionality to the Omega, including exposing the Omega's GPIOs, supporting the plug and play Omega Expansions, provide USB connectivity, among other things.

## Expansion Dock

The **Expansion Dock** can be considered the main Dock for the Omega. As the name implies, it supports Omega Expansions since it has the Expansion Header that exposes the Omega's GPIOs.



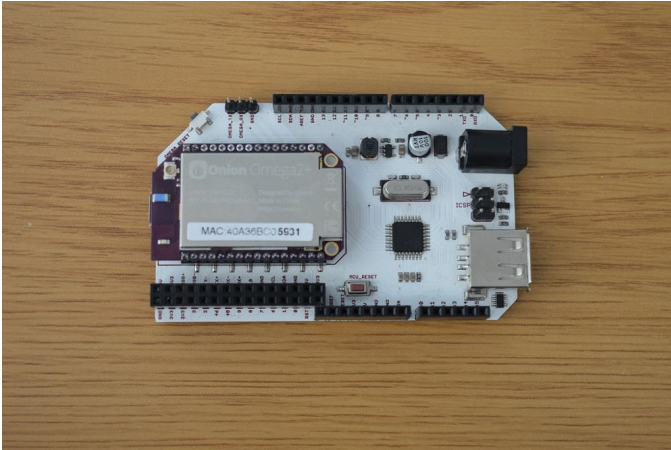
It's meant to be the main platform for the development phase of your project or product. It features an on-board USB-to-Serial chip that allows serial connectivity to the Omega's command line terminal through the Micro-USB port. The serial terminal can be used to access the bootloader in case the Omega's

OS is corrupted or cannot successfully boot. It also provides a USB Type-A plug for connecting USB devices to the Omega.

See the Expansion Dock [hardware overview](#) and [usage guide](#) for more details.

## Arduino Dock

The [Arduino Dock 2](#) features the ATmega328P microcontroller (the very same one used on the Arduino Uno) that can be programmed by the Omega to work in tandem as a co-processor. The Omega and ATmega328P can communicate using I2C and a serial UART connection.

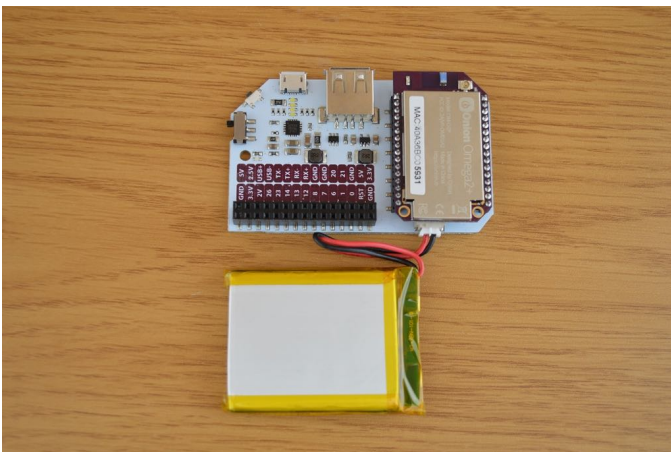


The Arduino Dock 2 exposes the microcontroller pins in the same header design as the Arduino Uno, so it can be used with any existing Arduino Shields. It also features the Expansion Header that exposes the Omega's GPIOs, allowing the use of all Omega Expansions. Also present is a USB Type-A socket for connecting USB devices to the Omega.

See the Arduino Dock 2 [hardware overview](#) and [usage guide](#) for more details.

## Power Dock

The Power Dock offers the mobility that comes with powering the Omega with any 3.7V LiPo (Lithium ion Polymer) battery. When plugged into power with a Micro-USB cable, it will charge up the battery. If no battery is present, the Omega can be powered with just the Micro-USB, similar to the Expansion Dock.



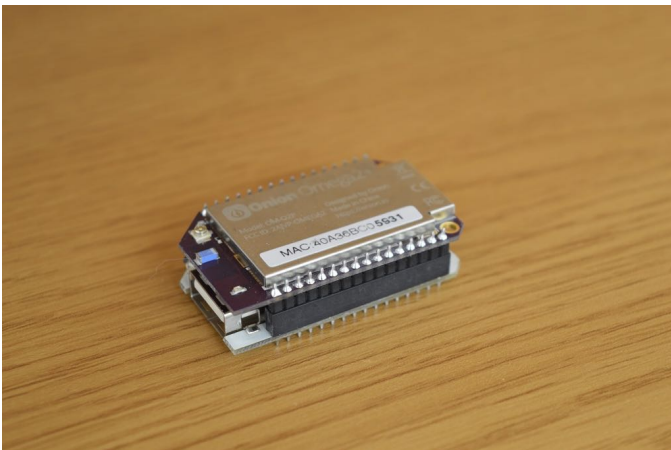


Use the Power Dock to deploy your developed projects and/or provide a battery back-up to maintain power to mission-critical Omega systems. Also featured is the Expansion header, exposing the Omega's GPIOs and providing support for all Omega expansions. Like the other Docks, it has a USB Type-A socket for connecting USB devices to the Omega.

See the Power Dock [hardware overview](#) and [usage guide](#) for more details.

## Mini Dock

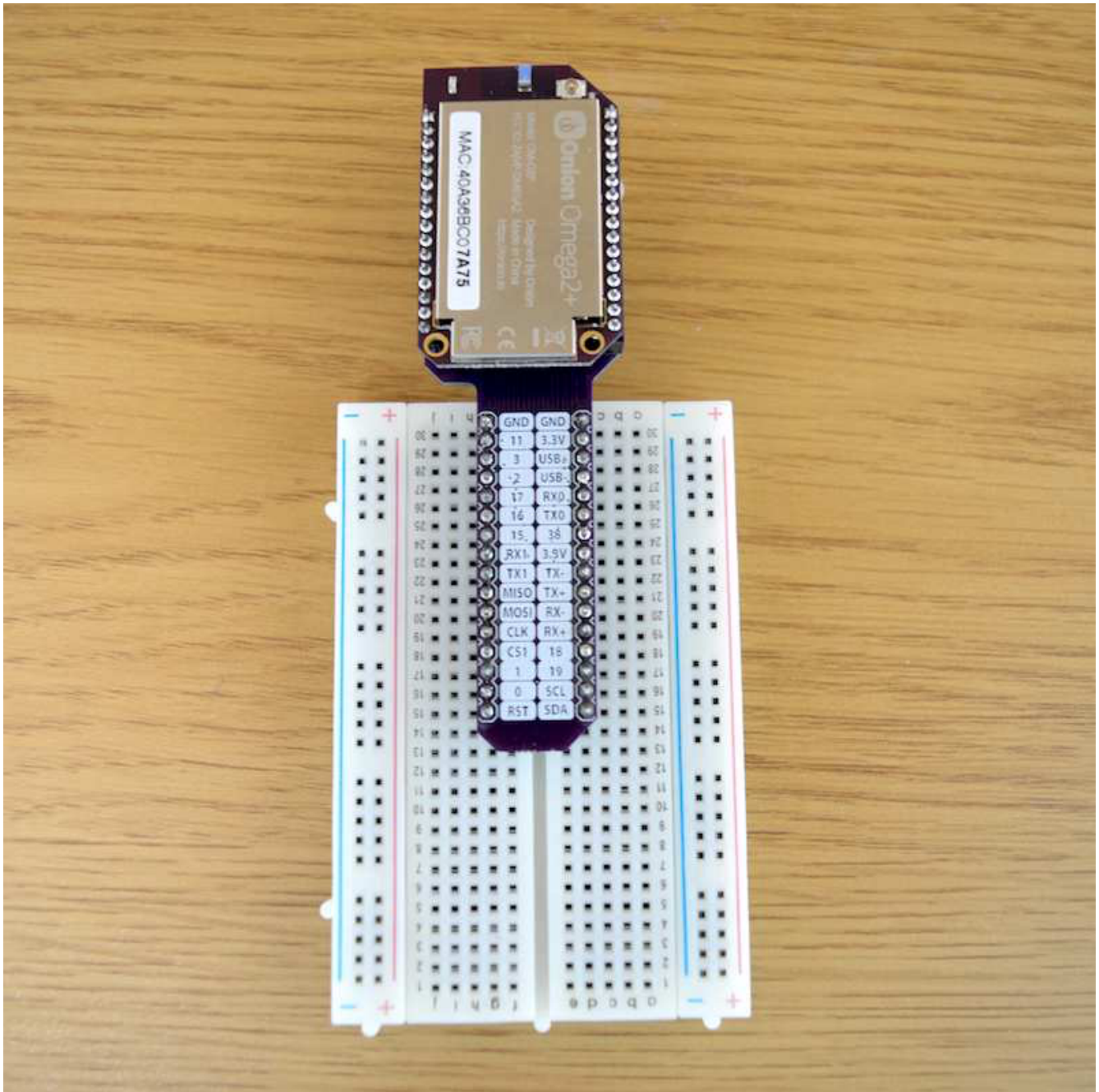
As the name implies, the [Mini Dock](#) is the smallest form-factor Dock for the Omega. It does not provide direct access to the GPIOs, but exposes the USB port, making it ideal for networking or USB-based use-case.



See the Mini Dock [hardware overview](#) for more details.

## Breadboard Dock

The [Breadboard Dock](#) rather unsurprisingly allows the user to plug their Omega directly into a breadboard. The Omega can be powered with a Micro-USB 5V supply since the breadboard has a voltage regulator, or it can be provided with 3.3V directly from the breadboard.



The Omega's pins are mapped 1-to-1 to the breadboard headers, so it's about as close as you can get to plugging the Omega directly into a breadboard.

See the Breadboard Dock [hardware overview](#) for more details.

## Expansions

The Expansions are the key to the Omega's modularity and flexibility since they add specific functionality to the system. As long as the Dock used with the Omega has an Expansion Header, it can support plug and play Omega Expansions.

---

**Docks that Support Expansions**

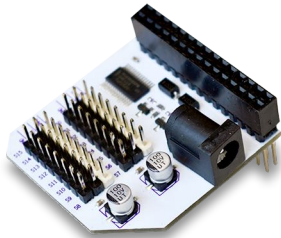
---

Expansion Dock  
Arduino Dock 2  
Power Dock

---

### Servo (PWM) Expansion

Generate up to 16 different, free-running Pulse Width Modulated (PWM) signals with the [Servo \(PWM\) Expansion](#). The PWM signals are driven by the Dock's 5V power supply by default, it also supports using an external supply (up to 12V DC) to drive the PWM signals.



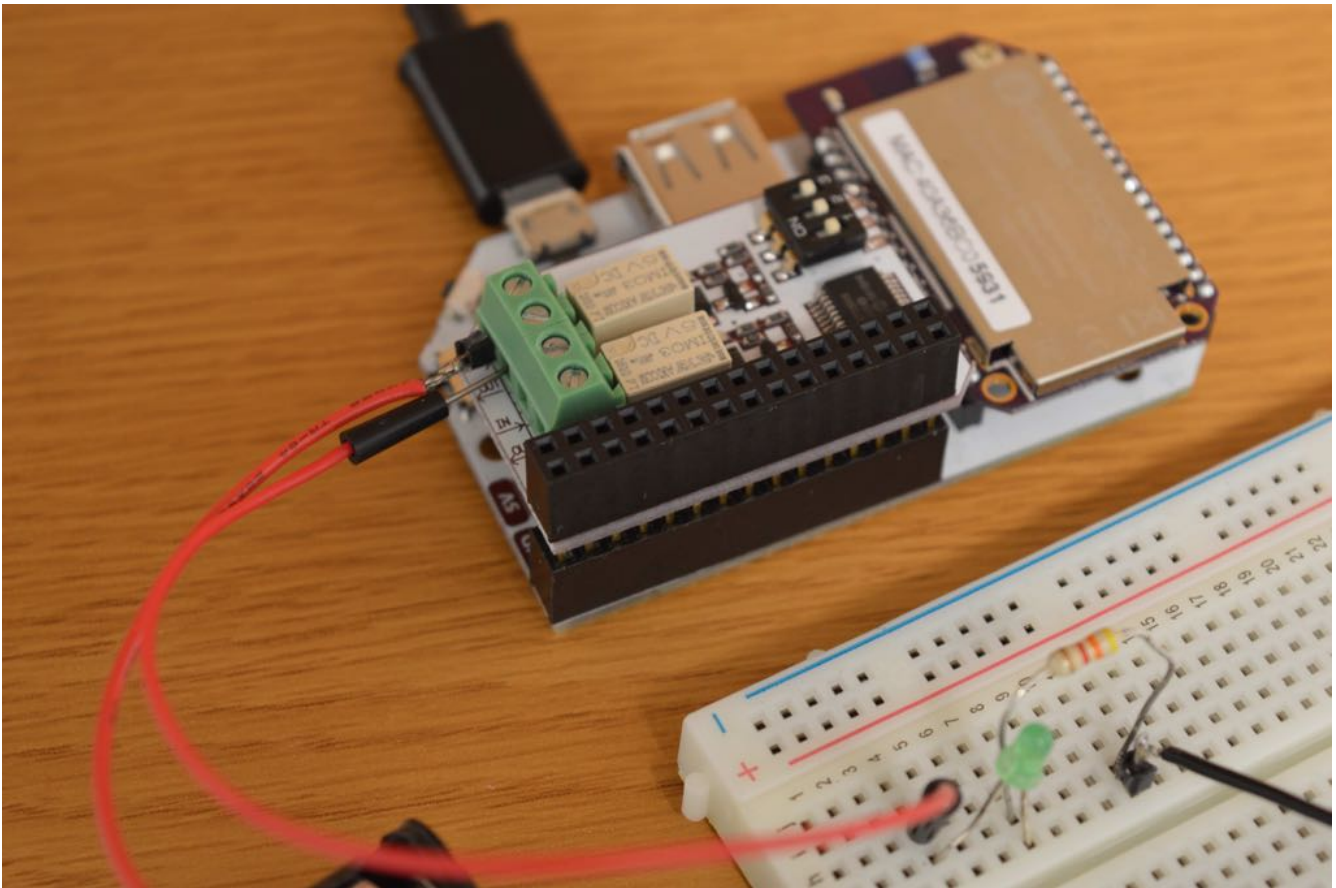
Use it to control Servos, LEDs, transistors, anything that supports PWM. The on-board oscillator supports generating PWM signals at frequencies in the range of 24 Hz to 1526 Hz. The default frequency is 50 Hz for compatibility with most servos. Note that when using multiple servos under a load, use of an external power supply is recommended.

For more details see:

- [PWM Expansion hardware overview](#)
- [Guide to using the PWM Expansion](#)

### Relay Expansion

Use two electromechanical relays to switch external, independent, and potentially much higher-voltage circuits with the [Relay Expansion](#).



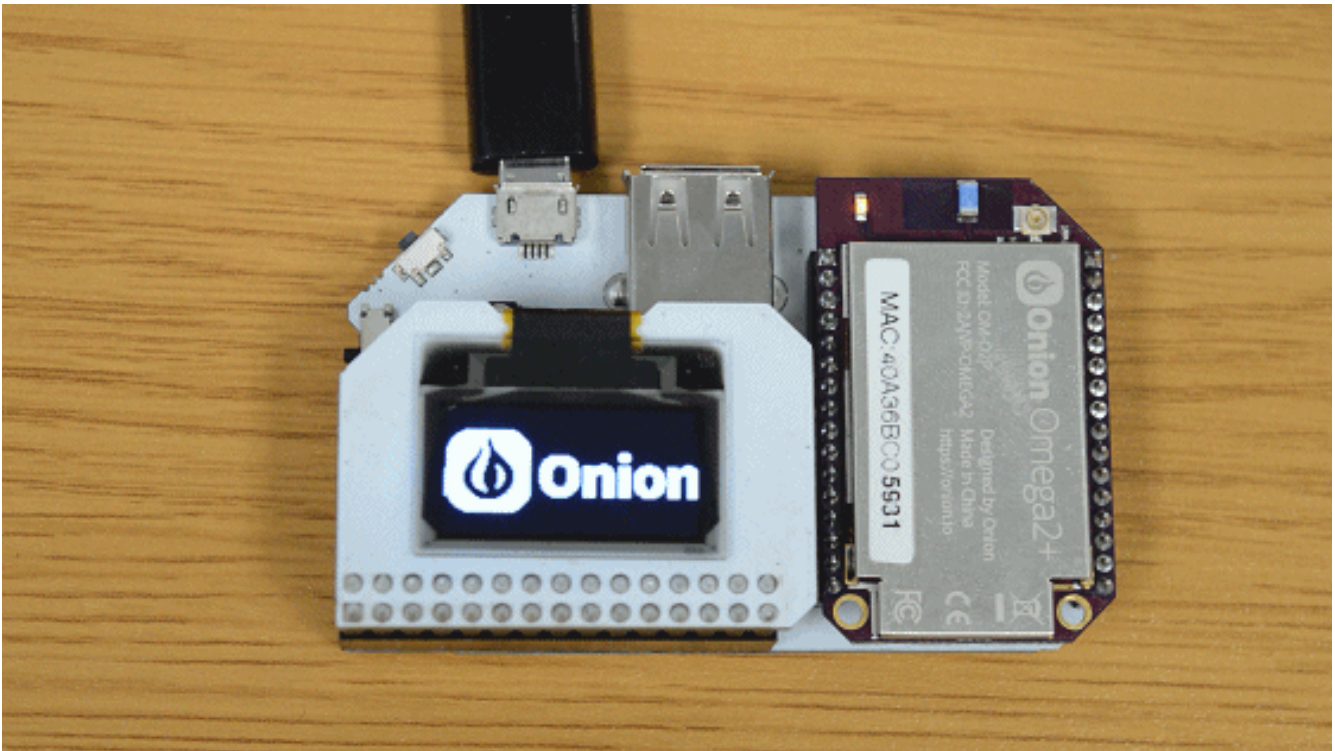
The relays can switch up to 60W, and are rated for a maximum current of 2A and a maximum voltage of 220V DC or 250V AC. It includes an I2C address switch, allowing the use of up to eight (8) Relay Expansions with a single Omega, giving the user control of up to 16 external circuits.

For more details see:

- [Relay Expansion hardware overview](#)
- [Guide to using the Relay Expansion](#)

## OLED Expansion

The [OLED Expansion](#) is a low-power 0.96" monochrome OLED screen with a 128x64 resolution. Use it to display text and images.



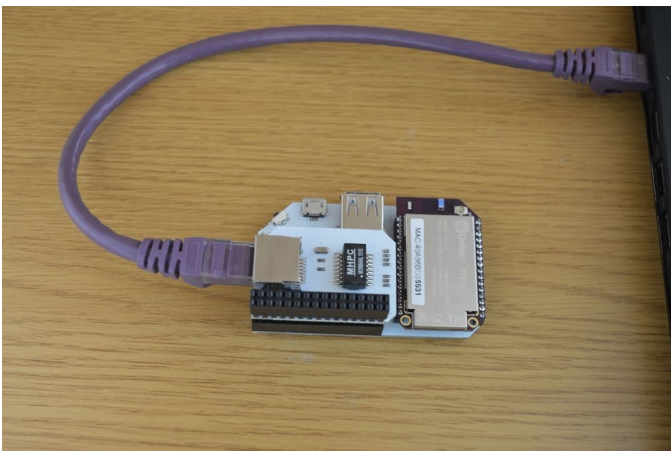
The pixel brightness is adjustable, as well as inversion of white and black. A single command enables built-in scrolling animations, where the contents of the entire display can be scrolled to the left, right, or diagonally left, and right.

For more details see:

- [OLED Expansion hardware overview](#)
- [Guide to using the OLED Expansion](#)

## Ethernet Expansion

Add an ethernet port for wired network connectivity with the [Ethernet Expansion](#).



The Omega has a 10/100 ethernet port, meaning it supports transmissions at 10 Mbps and 100 Mbps. The Omega has extensive networking capabilities, use the Ethernet Expansion to share network access between wired and wireless networks. Or, just provide a wired network connection to the Omega.

For more details see:

- [Ethernet Expansion hardware overview](#)
- [Guide to using the Ethernet Expansion](#)

## GPS Expansion

The USB-based [GPS Expansion](#) provides location data from the Global Positioning System (GPS) satellites.



The on-board antenna is connecting using a u.FL antenna connector, meaning that it can be easily unplugged and replaced with a larger, more powerful antenna.

For more details see:

- [GPS Expansion hardware overview](#)
- [Guide to using the GPS Expansion](#)

## Getting Started



Just got your Omega and wondering how exactly to get to making IoT things? Go through our [Get Started Guide](#) and you'll learn and accomplish the following:

- Properly connecting the Omega to a Dock and providing power
- Learning your Omega's unique name (it's just **Omega-** and the four bolded hex digits on the Omega's cover)
- Connecting the Omega to your WiFi network
- Updating the Omega to the latest firmware

Then you'll be ready to start on the projects!

If you're curious, here's some more recommended reading:

- [Connecting to the Omega's Command Line](#)
- [Using the Console, the Omega's web-based virtual desktop](#)
- [An overview of all of the Onion hardware](#)

If you're excited to get building, go on ahead. Remember to go back to the [Onion Documentation](#) if you're looking for clarification on how something works!

## The Command Line

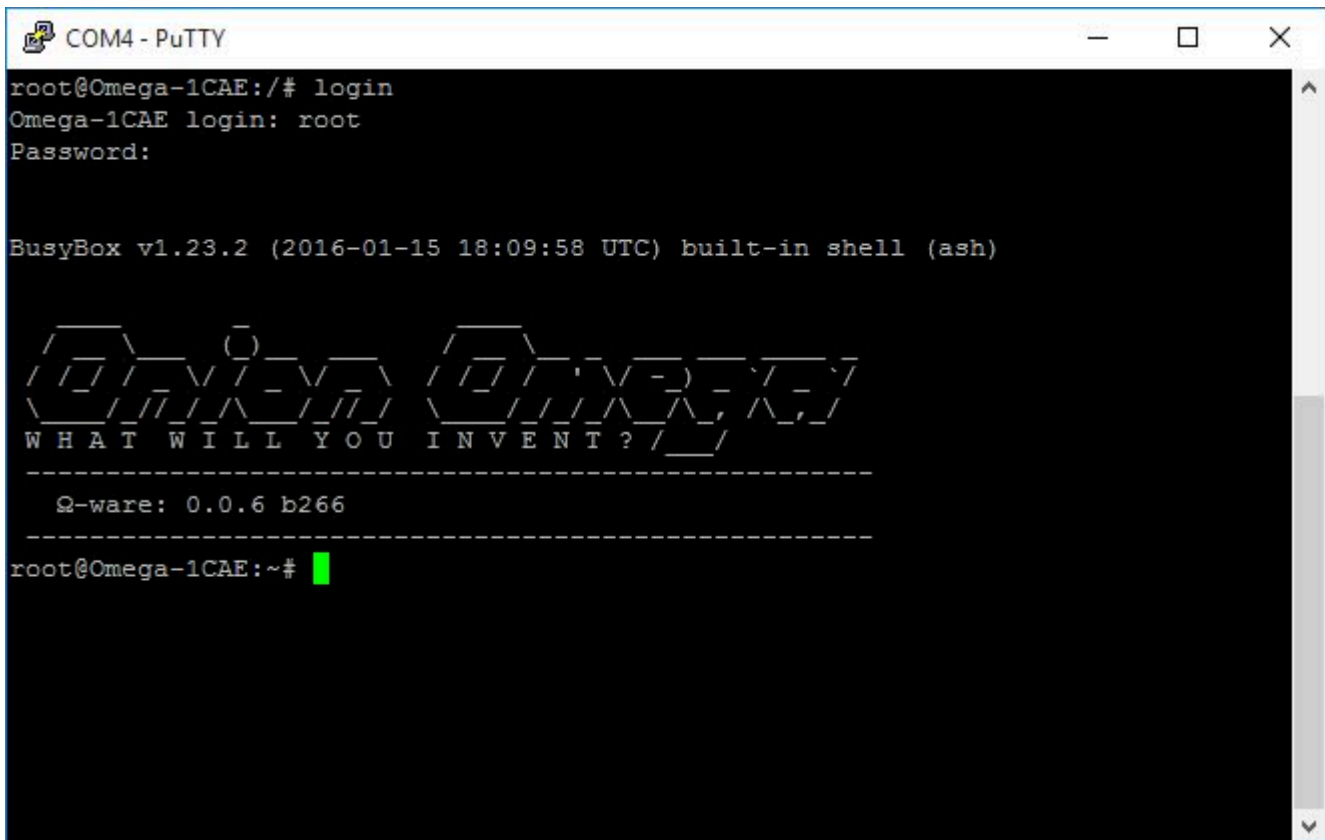
The Omega's operating system (OS) is based on Linux, a popular open-source OS that powers servers and computers all over the world. The version on the Omega is a minimalistic and lightweight distribution

called LEDE, which stands for **L**inux **E**mbedded **D**evelopment **E**nvironment. It supports many programming languages and can run all kinds of complex projects while still being small enough to fit in the Omega's memory.

## The Command Line Interface

We interact and operate the Omega by using the **command line interface** (CLI). The CLI is the user's access point into the operating system using a text-based terminal program. All user interaction is interpreted and executed by the OS through instructions, or **commands**. A user enters a command into a terminal to make something happen.

The CLI can look something like the picture below. In this terminal program on Windows, the green box is where the commands you type will be displayed on the screen.



```
COM4 - PuTTY
root@Omega-1CAE:/# login
Omega-1CAE login: root
Password:

BusyBox v1.23.2 (2016-01-15 18:09:58 UTC) built-in shell (ash)

              O
  _____  _____  _____  _____  _____  _____  _____
 /_ _ _ _ _\/_ _ _ _ _\/_ _ _ _ _\/_ _ _ _ _\/_ _ _ _ _\/_ _ _ _ _\/_ _ _ _ _/
W H A T   W I L L   Y O U   I N V E N T ? / _ _ _

-----
Omega-ware: 0.0.6 b266
-----

root@Omega-1CAE:~# █
```

We'll cover how to do the following:

- Navigating through the filesystem
- Creating (and deleting) files and directories
- Creating and modifying text files

## The Filesystem

In Linux, everything is a file. So naturally, the file system is where a great deal happens. The filesystem of LEDE is organized like a tree. At the very bottom of a tree is the root, and so it is with our filesystem.

`/` is the universal symbol for the very bottom of the filesystem - the root directory. All the files that the OS has access to can be found under some directory under `/`.



This is not to be confused with the **root directory**. In LEDE, every user gets their own ‘home’ directory to store all their personal files. On the Omega, this is located at `/root/` by default. When we connect to Omegas’ command line via `ssh`, we connect as the root and get placed inside the `/root/` folder. (Connecting via serial will place us in `./`.)

In Linux systems, `~` is an alias for the home directory, and can be used in scripts and programs. For example, calling a file in `~/myProject` is equivalent to calling `/root/myProject`.

On the Omega, all the contents in the `/root/` directory will be preserved through any firmware updates. So for our experiments, we’ll try to store our files in there so they stay put!

The home directory on other Linux systems may look like `/home/<username>`

## Navigating the Filesystem

Navigation usually consists of finding our location, looking for landmarks, and then setting a course for our destination. To that end, we’ll look at the `pwd`, `ls`, and `cd` commands to let us do just that.

### Locating Ourselves in the Filesystem

When we first drop into the command line, we don’t get a lot of information about where we are. We only really know that we’re in `root@omega-ABCD`. The `pwd` command will tell us exactly where we are relative to `/`

Immediately after logging in, `pwd` should return something like this:

```
root@Omega-ABCD:~# pwd
/root
```

This tells us we’re in the `/root/` folder, one level down from `/`. Notice that the prompt (all the things before `#`) already tells us where we are, the `~` is exactly our working directory `/root`.

### What’s in a Directory?

Now that we know where we are, we should take a look at our surroundings. To list out every file in a directory, the `ls` command is our go to.

For example, on one of Onion’s office Omegas, this is what `ls` outputs:

```
root@Omega-ABCD:~# ls
MAK01-dimmingLeds.py  checkAS6200Temp.sh
binify.sh              checkTemp.py
```

Awesome, now we know there’s four files in the working directory.

But wait, there’s more! Not only can `ls` list the working directory, if we give it a **path**, it can also peek into that path.

A **path** is the full location of something in the filesystem, starting from the root.

Let’s say we want to take a look into our `/` directory, we can append `/` as an argument like so:

```
ls /
```

Which might return something like this:

```
root@Omega-7CCB:~# ls /
bin          lib          rom          tmp
mnt          root         usr          dev
overlay     sbin        var          etc
proc        sys         www
```

The prompt tells us we're still in `~`, but those are folders in `/`.

## You can Get There from Here

Of course we can't stay in the same working directory forever. We can move around directories using the `cd` command. It stands for **change directory**.

The most frequent use of `cd` is in conjunction with a path as the argument, like so:

```
cd /root
```

This command changes the present working directory to `/root`.

Earlier, we mentioned the idea of a 'path' in passing. A path (or 'absolute path') is like the full address to something in the file system. For example, if you have a directory called `kittens` in your `/root` folder, the path of that directory is `/root/kittens`. Similarly, for a file called `adorable.jpg` in `/root/kittens`, the path would be `/root/kittens/adorable.jpg`.

Typing out the full path everything can become tedious, so there are many shortcuts that `cd` can understand in the form of 'relative paths'. There are some path aliases that change where it leads depending on the context.

- To go to the `/root` on the Omega (or the home folder as a different user) `cd` with no arguments will take us there.
- The relative path for the 'directory above' is `..`, so `cd ..` will take us up one level no matter where we are.
- To get to any sub-directories in the working directory, we can `cd <name of directory>` instead of the absolute path.

All of the shortcuts above work with each other too!

Let's say we want to move two directories up:

```
cd ../../
```

The first `..` expands to 'up one level', and in that directory, `..` again will of course take us back up once more.

We can also move to other directories contained in parent directories. If `..` had a directory called `puppies`, we could run:

If we want to `cd` up and sideways into the `kittens` directory, we can use the path `../kittens` - up one level, and into `kittens` below.

## Interacting with the Filesystem

Navigating is very useful, but doing things with files is what gets projects working! So to do that, we'll go over commands to create and delete directories, and creating and removing files.

We'll cover the `mkdir` and `touch` commands to create things, and the `rm` command to get rid of them.

## Creating Directories

The `mkdir` command allows us to create empty directories. Run it like so:

```
mkdir <DIRECTORY>
```

You can use both relative and absolute paths. See the example below:

```
root@Omega-ABCD:~# mkdir hello          # relative path
root@Omega-ABCD:~# ls
hello
root@Omega-ABCD:~# mkdir /root/hello/world # absolute path
root@Omega-ABCD:~# cd hello
root@Omega-ABCD:~/hello# ls
world
```

## Creating Files

The `touch` command can be used to create files. The syntax looks like this:

```
touch <FILENAME>
```

- If the file doesn't exist yet, it creates an empty file.
- If the file already exists, it updates the time it was last modified to when you ran the command.

You can check the file's last modified time using `ls -l` like in the example below:

```
root@Omega-ABCD:~# touch hello.txt
root@Omega-ABCD:~# ls -l
-rw-r--r--  1 root  root           0 Mar 23 23:38 hello.txt  # we've created our file
root@Omega-ABCD:~# cat hello.txt
root@Omega-ABCD:~#                               # nothing happens, the file is empty
...
(wait a minute or two)
...
root@Omega-ABCD:~# touch hello.txt                # update the time it was last modified
root@Omega-ABCD:~# ls -l
-rw-r--r--  1 root  root           0 Mar 23 23:41 hello.txt  # the modified time updated
```

## Deleting Files and Directories

Delete a file using the `rm` command like so:

```
rm <FILENAME>
```

To delete a directory and all of the files inside it, run:

```
rm -rf <DIRECTORY>
```

These two options are explained below:

- **-r - recursive mode**
  - This means to go into a directory and delete all of the files inside.
  - If it finds more directories, it enters them and deletes their contents as well.

- This is required when deleting directories, otherwise it will return an error.
- **-f - force**
  - This will make the program continue if it runs into an error when trying to delete a file.

## File Editing on the Omega

The projects are based on code, which live in files, so it stands to reason that we'll be doing some file editing for each of our projects.

On the Omega, there's one text editor by default - Vim. It's different from more familiar, visual editors in how it works, so we'll cover the basics super quickly to get to making cool things faster!

### Creating or Opening a File

To work with a file, simply call `vim <filename>`. It will work if you do `vi <filename>` as well.

If the file already exists, it will open it in vim. If it doesn't, it will open an empty temporary file with the given name.

The temporary file won't be permanent until you save it!

If you type things immediately when vim starts, nothing will come out. So how do we get text in?

### Writing Text

Hit `i` to get into 'insert mode', now we can start entering text!

The controls of insert mode should be quite familiar if you've used notepad before. The keys will insert characters, the arrow keys provide navigation, and `home/end/pgup/pgdn` will behave accordingly.

Vim is based on different modes. Entering characters and pasting with `ctrl-shift-v` will only work in **insert mode**.

Vim begins in **normal mode**, where all keystrokes are interpreted as commands. So all the text you want to paste will be interpreted as commands, causing lots of unpredictable changes to the file.

If mistakes were made, what can we do?

### Undo and Redo

Undo is a normal mode function, hit `ESC` and press `u` to undo the last bit of changes made. To redo hit `ctrl-r` in normal mode.

The reason we hit `ESC` first is to return to normal mode. So instead of inserting `u`, vim will undo the last changes we made.

Once the code is fixed, it's time to save.

## Saving Changes

Hit **ESC** to enter normal mode, then `:w` and enter.

Entering `:` calls the ‘command line’ of vim, where we can run vim’s less often used functions. The command we give it is `w` which stands for ‘**w**rite to file’. Since `:` works like a command like, we have to **enter** to send our command to it.

The more commonly used commands in vim are related to navigating, copy/pasting, and editing chunks of text - like `u` for undo.

Now that we’ve saved, we’ll have to exit vim to test the script.

## Quitting and Saving changes

**ESC**, then `:wq` enter.

The quitting process is very similar to saving - return to normal mode with **ESC**, type `:` to enter the `q` command to **q**uit vim.

Vim also has the ability to accept multiple commands and execute them in order. So we can save **and** quit by typing `:wq`.

## Quitting without Saving Changes

`:q!`

Forcing vim to quit can be done by adding `!` to the end of the `:q` command.

Normally when trying to exit vim with unsaved changes, vim will deny the attempt as a safe measure. But quitting without saving is also useful to revert really big changes.

## More on Vi

There’s a lot more than meets the eye with vim. It’s actually a very powerful editor under the hood and can be installed in all major operating systems. If you want to learn how to make the most out of it, [Open Vim](#) has a fantastic tutorial that can get you started.

## Intro to Python

Python is the programming language of choice for this project book since it’s powerful, flexible, and generally easy to use.

As an introduction, this article will cover some of the basics so we can hit the ground running!

Python is a high-level, interpreted language designed for scripting. It supports Object Oriented Programming - which is a handy framework that we’ll be using quite a bit.

The high-level part means that Python reads closer to English than machine instructions and abstracts away many of the low-level hardware intricacies. The interpreted part means that Python code is executed line-by-line by the computer as it’s being run - as opposed to languages like `C` where it is first compiled before it can be run.

## How we'll use Python

The experiments will provide code that works out of the box that can be directly copied to the Omega and run. We'll also discuss more interesting sections of the code in detail.

## Python Syntax Overview

```
import time

# print ('This line will not be printed')

greeting = 'Hello world!'

print (greeting)

for count in range(0,10):
    if (count % 2 == 0):
        print ('Tick')
    else:
        print ('Tock')
    print (count)
    time.sleep(0.5)
```

The above is a block of Python code with all the basic building blocks of the language. Let's go through it bit by bit.

### Variables

```
greeting = 'Hello world!'
```

The equals sign (=) assigns the string 'Hello world!' to the variable named `greeting`.

All variables in Python are created this way with the assignment operator.

### Comments

```
# This is a comment
```

Comments in Python start with a #. Any text after the hash in the same line will be ignored by the interpreter.

### Functions

```
print ('just a function')
```

Function statements

Functions in Python have two parts: a function name and a list of arguments that are sent to it.

---

Name	Argument
print	'just a function'

---

The number of arguments that a function takes can be zero. Some functions return a value that you can assign to a variable.

## Logic

```
if (count % 2 == 0):
    print ('Tick')
else:
    print ('Tock')
```

The if/else structure is used to evaluate variables and make decisions based on them. All indented lines after the if statement will be executed **if the condition is met**. The first un-indented line after the if statement ends the statement. For **else**, all indented lines after the **else** line will be executed **when the condition of the if statement is unmet**. So in the code above, only one option gets executed (either 'Tick' or 'Tock') when the interpreter gets to this part. Which one depends on the value of `count`.

Extra evaluation statements can be inserted as an `elif` block like so:

```
if (count % 2 == 0):
    print ('Tick')
elif (count == 3):
    print ('Tack')
else:
    print ('Tock')
```

This adds a third option - all indented code after the `elif` statement will be executed if:

- The if condition is **unmet**
- AND the `elif` condition is **met**

Now one of **three** options will be executed, printing either 'Tick', 'Tock', or 'Tack' depending on value of `count`.

## Looping

```
# For-loop with a counter variable
for count in range(0,10):
    print (count)

# While-loop - checks condition first, then starts the loop
while (count <= 10):
    print (count)
```

The for-loop can iterate over any list, executing all indented code after the `for` statement as many time as the number of elements in the list. The range function returns a list of integers in the given range. So

the for-loop above will run 10 times, same as the example up top. However this loop will print the count of each cycle instead of ‘Tick’ and ‘Tock’.

The while loop checks a single condition every loop, so it’s useful for infinite loops and checking unique conditions. Since `count` doesn’t change during the while loop, it will run forever assuming `count` is no greater than 10, continuously printing an unchanging value of `count`.

## Using Libraries

```
# Importing a library  
import time
```

The `import` statement above adds all the functionality of the `time` standard library to be available in your program.

Calling a function included in the library is done using the `.` notation - `time.sleep()` will call the `sleep()` function in the `time` library.

## Learning More about Python

Python is a very popular language, so there’s a tremendous amount of tutorials out there. If you’re still unsure of how parts of our code work, we recommend taking a look at the guides over at the Python Wiki for [programmers](#) or [non-programmers](#).

## Where Can I Learn More?

To learn more about the Omega and the Docks & Expansions, what can be done, and much more, check out the Onion Omega2 Documentation:

<https://docs.onion.io>

If you encounter issues, are looking for help with debugging, or just general Omega talk, check out the Onion Community Forum:

<https://community.onion.io>

## Where to Get More Onion Products

Glad you asked! Onion products can be purchased from the [Onion online store!](#)



They're also available from a variety of retailers around the world. In either case, check out the [Onion online store](#) to find the best place to get more Onion gear!

## Reporting Issues

While we've done our best to make sure everything in this project book is 100% correct, it's possible that we've made mistakes, typos, or left stuff out. We would also love to hear any comments or suggestions you have!

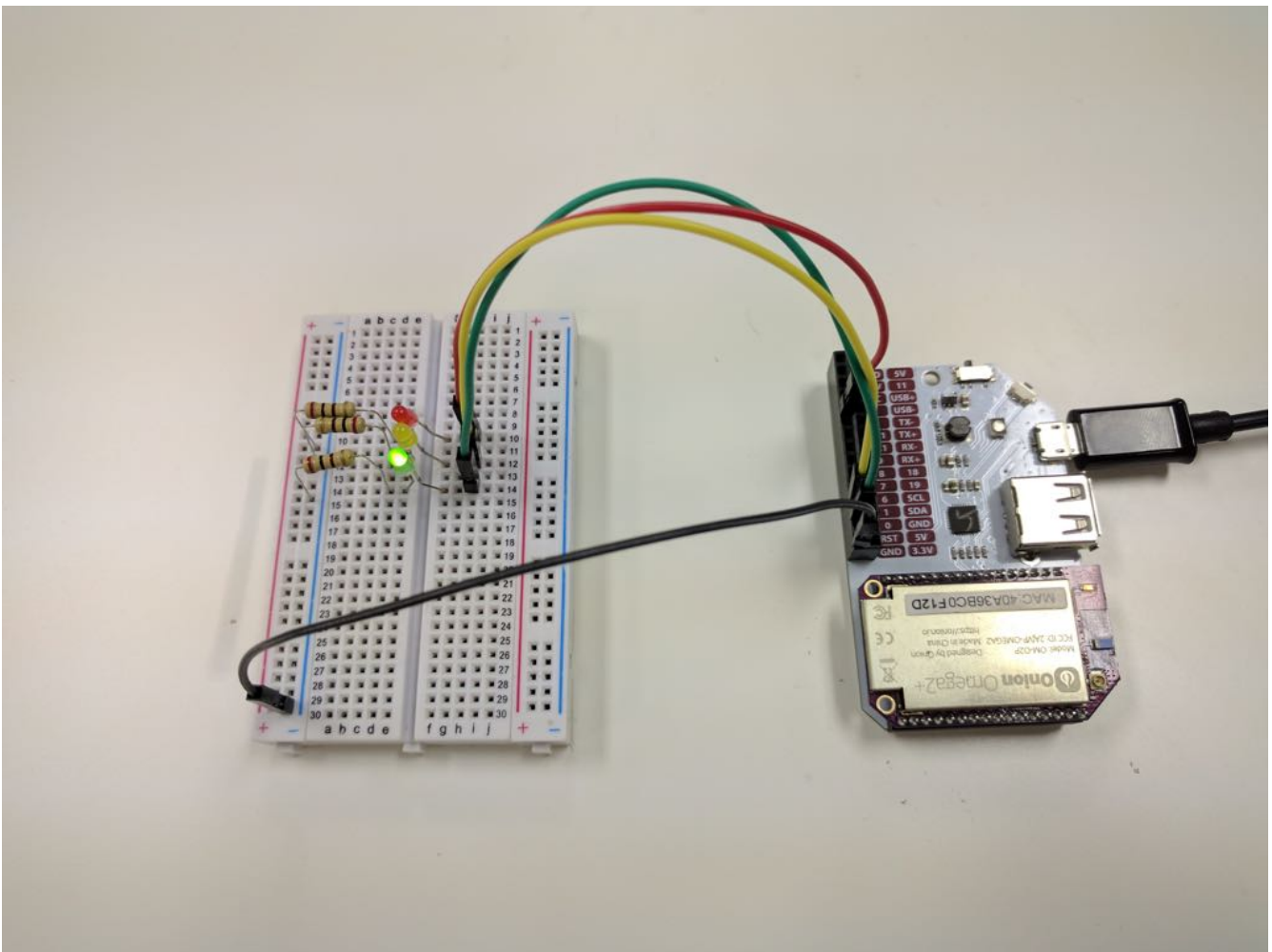
If you find anything amiss with the project tutorials, please let us know by [creating an issue on GitHub](#).

If you find a problem in the code of a project, all projects have their own GitHub repository, please create an issue on the corresponding repository.



## 2 | Starter Projects

The Omega is a great platform to learn how to work with electronics, how to write programs, and how to interact with Linux systems. Let's start with a few projects to get acquainted with using the Omega.



### Concepts

The concepts covered in the starter projects:

- Using scripts to implement logic and perform specific actions
- Controlling external electronics using the Omega

## Projects

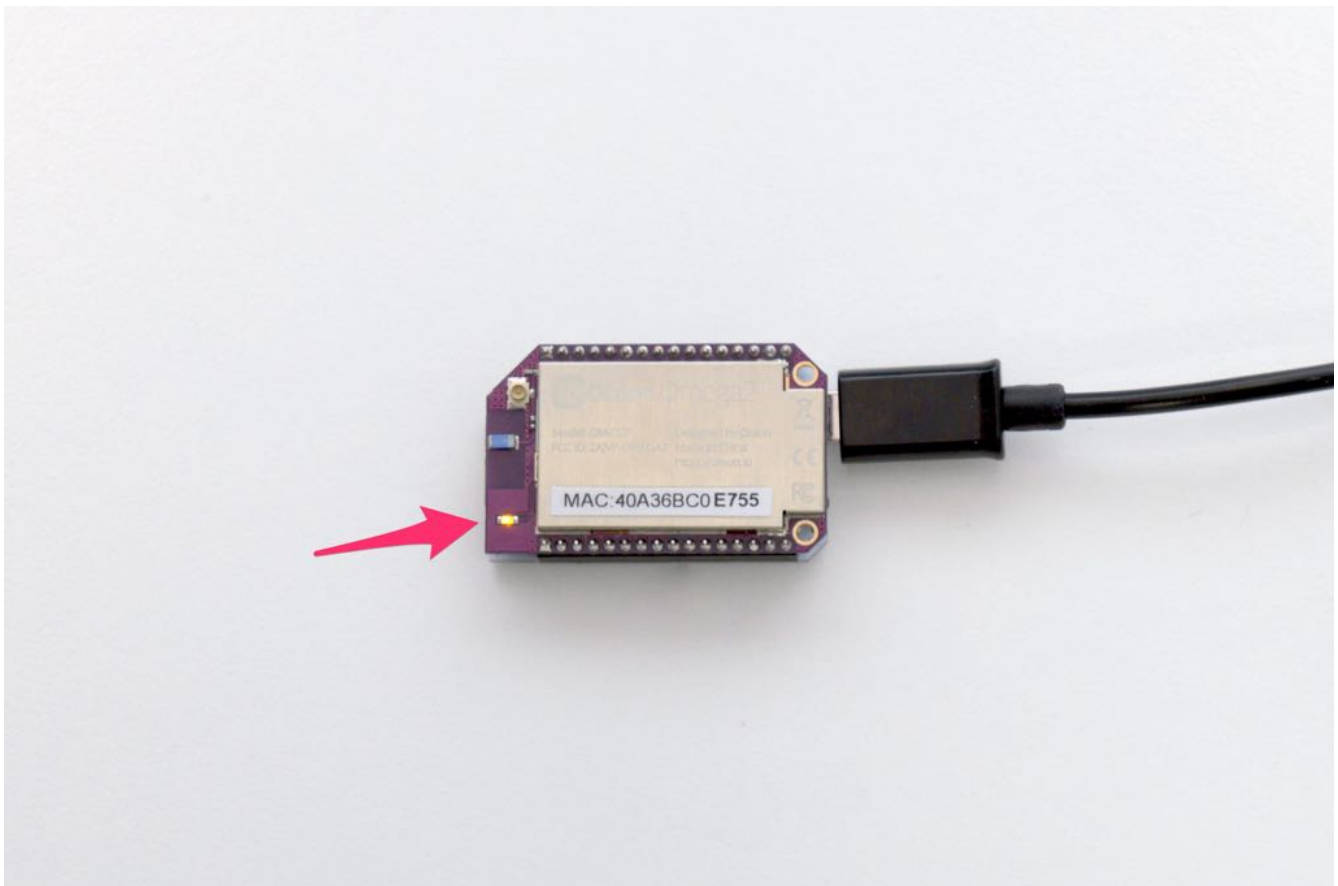
Projects to get you started working with the Omega:

1. **Morse Code on the Omega's LED**
  - For our very first project, we'll write a script that takes text, converts it into morse code, and shows the morse code message on the Omega's built-in LED
2. **Traffic Light using LEDs**
  - Get started making projects with external circuits by creating a miniature traffic light with a few LEDs

## Morse Code on an LED

Morse code is an old binary encoding that transmits messages letter-by-letter through sound or a flashing light.

In this project we're going to write a script that will blink the Omega's LED in morse code based on the user's input using the Omega's command-line interface.



## Overview

**Skill Level:** Beginner

**Time Required:** 15 minutes

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any [Onion Dock](#) - we like the [Mini Dock](#) for this project!

## Step-by-Step

This project will take advantage of a `morse` LED utility that comes pre-loaded on the Omega. We'll test out how it works, then write a script to translate text live as we enter it!

### 1. Prepare the ingredients

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

This project requires the use of the Omega's command-line, so we'll have to either SSH into the Omega's command-line, or connect serially.

To learn more on how to connect to the Omega's command-line you can read our comprehensive [guide to connecting to the Omega](#).

### 2. Controlling the LED with the 'morse' command

The Omega comes ready with a kernel module that can translate text to Morse code and blink an LED, but you'll need to tell the kernel which LED you want to blink. The kernel exposes a lot of hardware status and configuration options through a virtual filesystem under `/sys`.

The files under `/sys` aren't *actually* files, but they look and act like files to make it very easy to access them from the command line and in scripts or programs.

To tell the kernel that we are going to use the Morse code module, set the LED trigger condition for the Omega system LED to `morse` by using the `echo` command to write the setting into the virtual file:

```
echo morse > /sys/class/leds/omega2\:amber\:system/trigger
```

If you're using an **Omega2+**, the LED will be named `omega2p:amber:system`

So the command will look like this instead:

```
echo morse > /sys/class/leds/omega2p\:amber\:system/trigger
```

You can verify that it worked by using `cat` to look at the virtual file:

```
root@Omega-2757:~# cat /sys/class/leds/omega2\:amber\:system/trigger
none mmc0 timer default-on netdev transient gpio heartbeat [morse] oneshot
```

The square brackets indicate that the `morse` trigger is currently selected. The text in that file shows the other available options that this particular bit of the kernel can be set to.

Anyway, now we have everything set up!

Once the `morse` option is selected, the kernel creates a new virtual file for that called `message`. We will `echo` the text we want to it:

```
echo Hello, Onion > /sys/class/leds/omega2\:amber\:system/message
```

Now watch your LED!

The message will keep looping forever or until you change it. To stop it, you can either clear the message entirely:

```
echo > /sys/class/leds/omega2\:amber\:system/message
```

or change the LED trigger to something else:

```
echo default-on > /sys/class/leds/omega2\:amber\:system/trigger
```

### Adjusting the Delay

If it's too fast or too slow, you can change the speed with the `delay` file:

```
root@Omega-12D9:~# cat /sys/class/leds/omega2\:amber\:system/delay
50
```

To slow it down a bit, we `echo` a bigger number:

```
root@Omega-12D9:~# echo 100 > /sys/class/leds/omega2\:amber\:system/delay
```

### 3. Using a Shell Script instead of 'echo'

Create a file called `morse.sh` in the root directory using the following command:

```
vi /root/morse.sh
```

You'll open an empty file. To start typing you can enter `a`.

Copy the code below into the terminal.

```
#!/bin/sh

# find the name of the board, to be used in the name of the LED
. /lib/ramips.sh
board=$(ramips_board_name)

# define the function that will set the LED to blink the arguments in morse code
_MorseMain () {

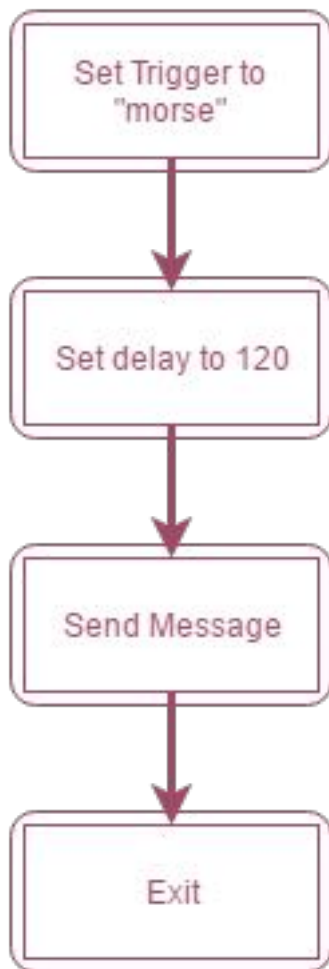
    echo morse > /sys/class/leds/$board\:amber\:system/trigger
    echo 120 > /sys/class/leds/$board\:amber\:system/delay
    echo $* > /sys/class/leds/$board\:amber\:system/message
}

##### Main Program #####

# run the function and pass in all of the arguments
_MorseMain $*
```

```
exit
```

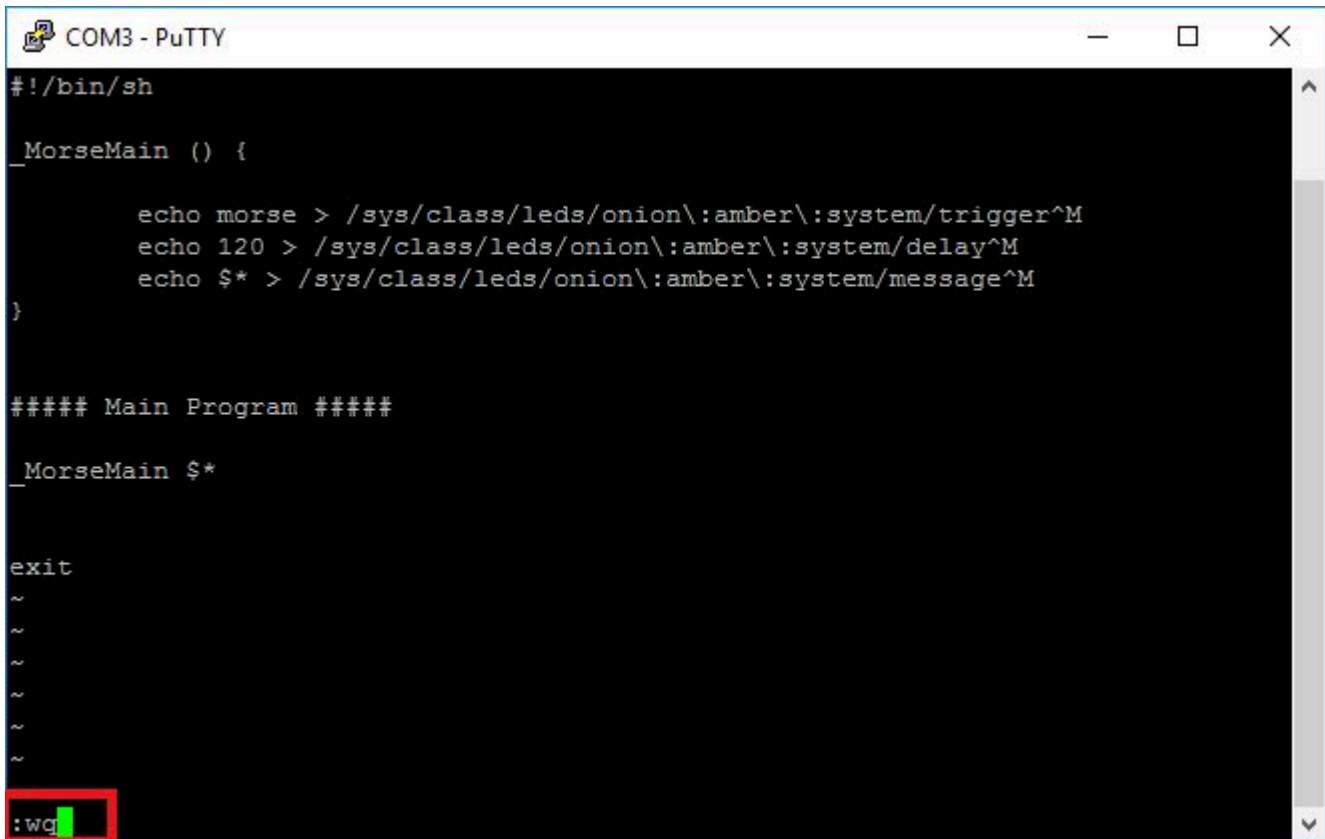
This block diagram shows the steps the `_MorseMain` function will perform:



Now to save the file you'll have to enter the Command Mode by hitting the **ESC** button on your keyboard.

The Command Mode of `vi` allows you to enter keys to perform commands such as saving and exiting, exiting without saving, or deleting lines.

Type `:wq` and hit enter to save and exit your file.



```
#!/bin/sh

_MorseMain () {

    echo morse > /sys/class/leds/onion\:amber\:system/trigger^M
    echo 120 > /sys/class/leds/onion\:amber\:system/delay^M
    echo $* > /sys/class/leds/onion\:amber\:system/message^M
}

##### Main Program #####

_MorseMain $*

exit
~
~
~
~
~
~
:wg
```

You are now ready to convert text to morse code!

#### 4. Run it!

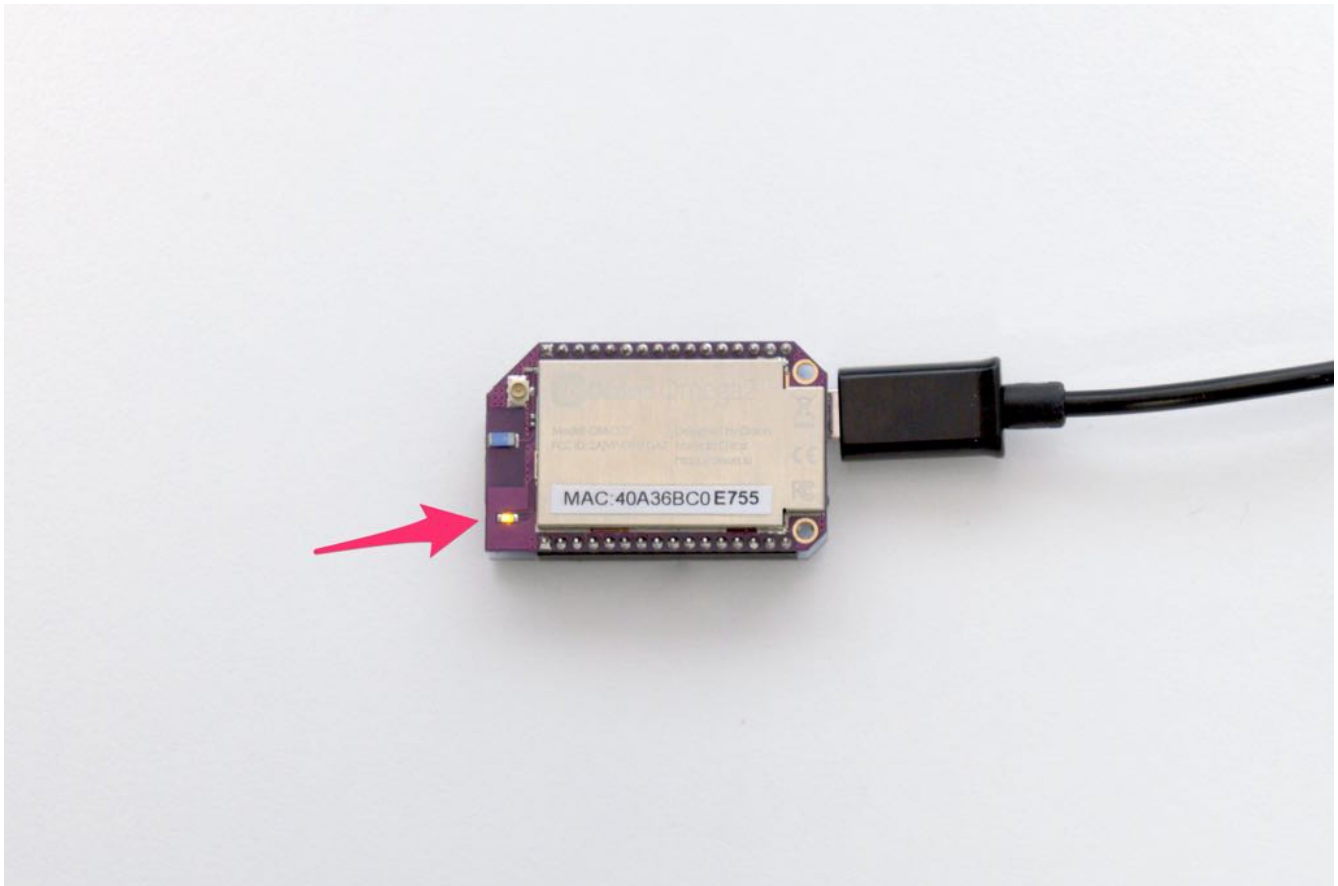
To run your Script enter the following command to run your script:

```
sh /root/morse.sh <YOUR MESSAGE HERE>
```

Enter a message that you would like to blink in morse code:

```
root@Omega-2757:~# sh /root/morse.sh Hello Union
```





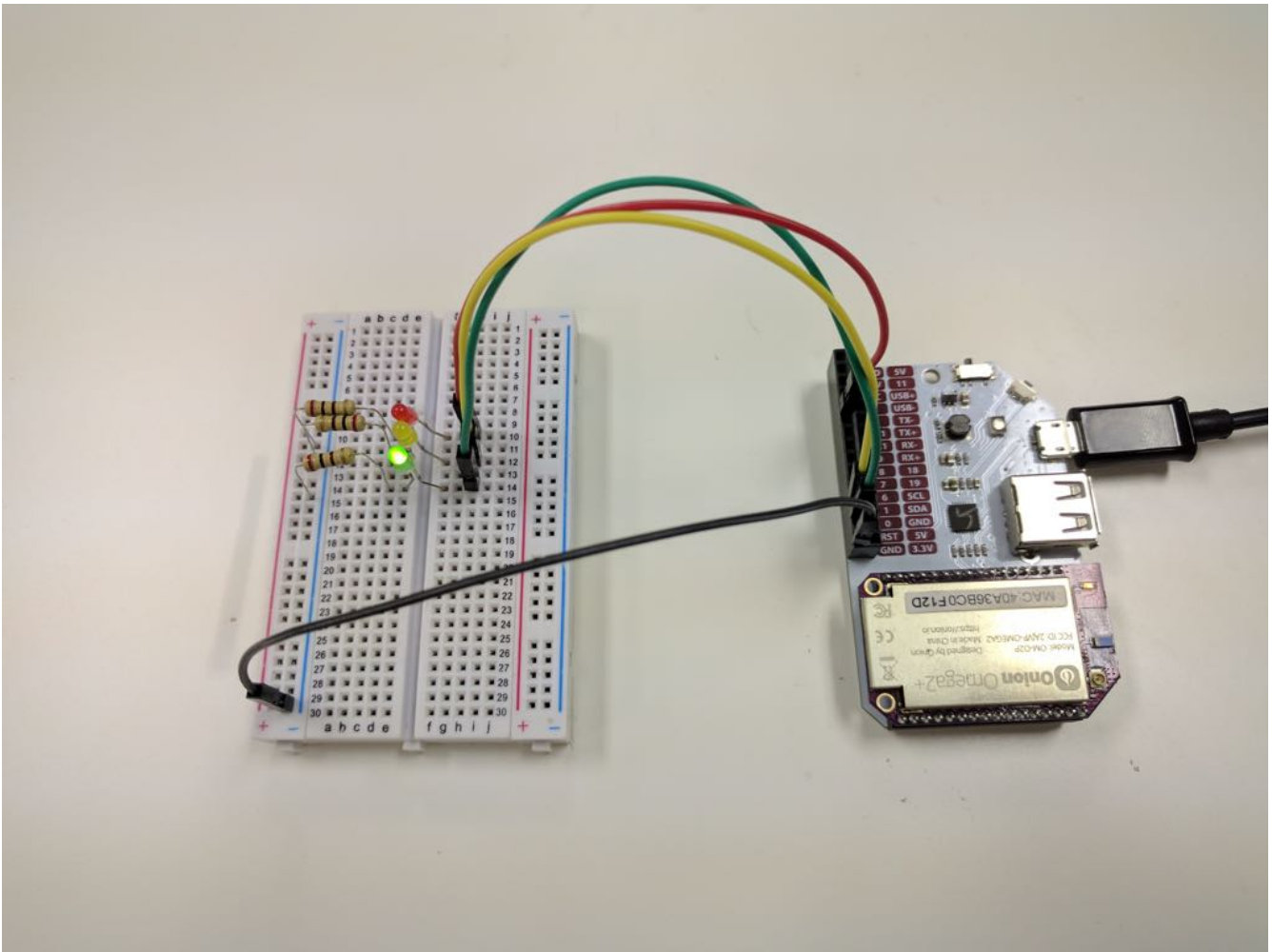
Once you're done, you can set the blinking back to `default-on` with the following command:

```
echo default-on > /sys/class/leds/omega2\:amber\:system/trigger
```

Remember, on an Omega2+, the LED will be named `omega2p:amber:system` as opposed to `omega2:amber:system` so you will have to pipe the above command to `/sys/class/leds/omega2p\:amber\:system/trigger`

## LED Traffic Light

For this project, we will be building a working, miniature traffic light using a few LEDs and the Omega. We'll also introduce the basics of controlling the Omega's GPIOs with a Python program.



## Overview

**Skill Level:** Beginner

**Time Required:** 10 minutes

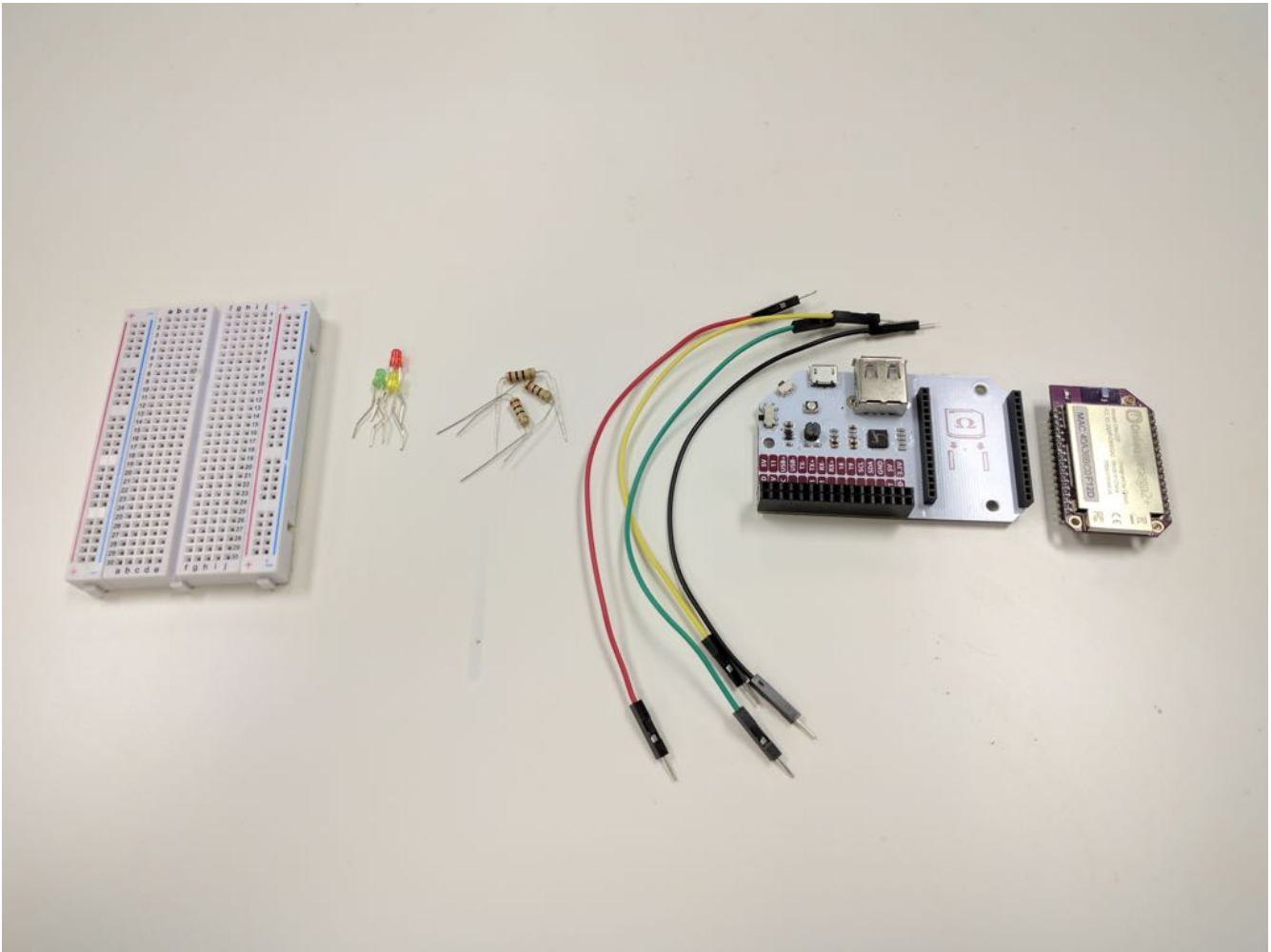
We'll be using the [Onion GPIO Python module](#). This module is the bread and butter of any project where you will need to control and interface with other circuits!

The complete project code can be found in Onion's [starter-traffic-light repo on GitHub](#).

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that exposes the Omega's GPIOs: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#), [Breadboard Dock](#)
- 1x [Breadboard](#)
- 3x [LEDs](#)
  - 1x red
  - 1x yellow/amber
  - 1x green

- 4x Jumper Wires (M-M)
- 3x 200 $\Omega$  Resistors



## Step-by-Step

Follow these instructions to set this project up on your very own Omega!

### 1. Prepare the ingredients

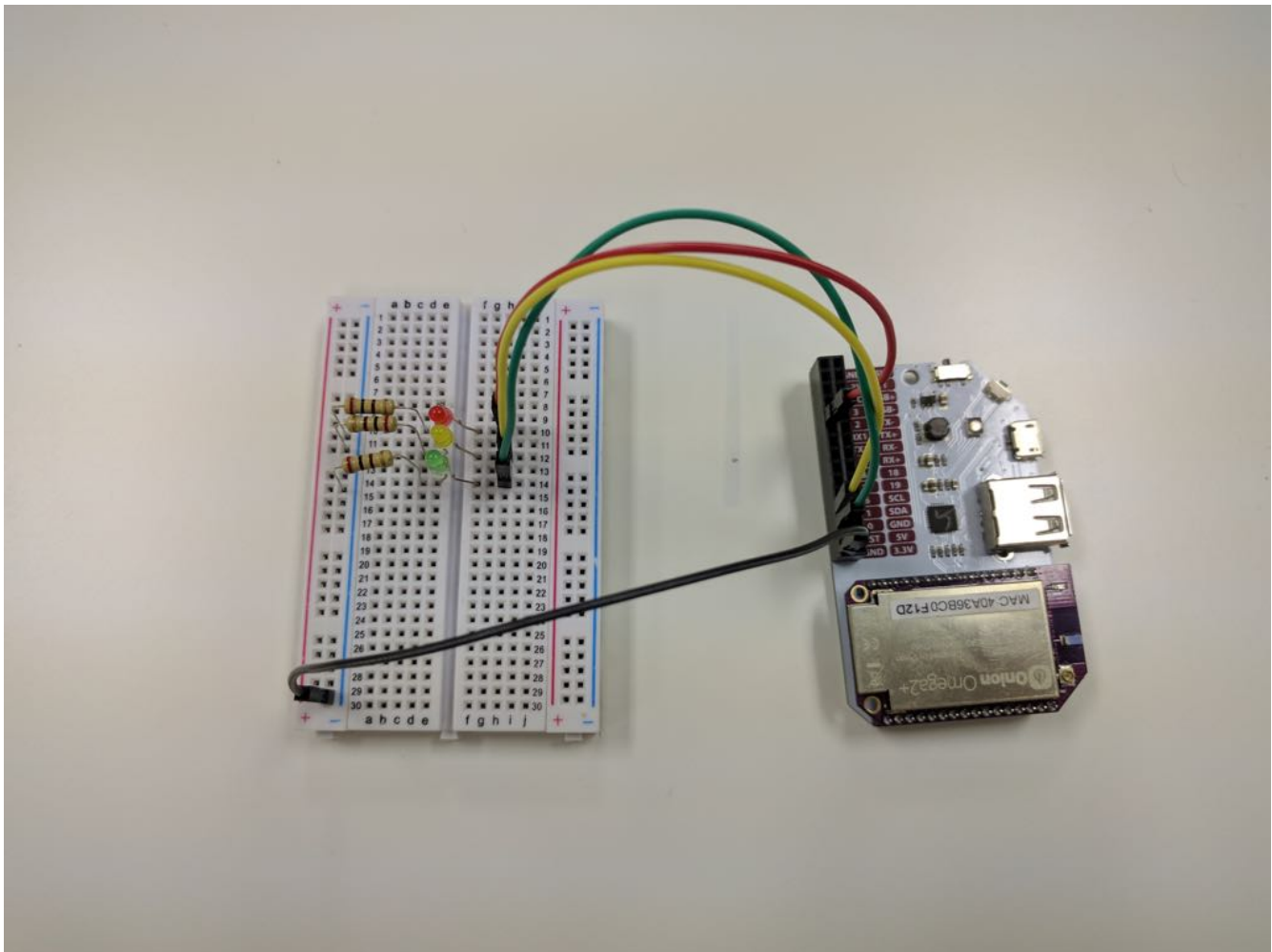
You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

This project will need the Omega's command-line, so we'll have to either SSH into the Omega's command-line, or connect serially.

To learn more on how to connect to the Omega's command-line you can read our comprehensive [guide to connecting to the Omega](#).

## Wire Up the LEDs

1. Plug in the LEDs across the breadboard, with the cathode on the left side of the gap and the anode on the right.
  - Make sure red is above amber, and amber above green!
2. Connect one end of a 200 $\Omega$  resistor to the cathode row, and the other end to the negative rail marked - on the left side of the board.
3. Connect the - rail to one of the GND pins on the Omega.
4. Connect the GPIOs to the LEDs in the following manner:
  - Red to GPIO2
  - Amber to GPIO1
  - Green to GPIO0



### 1. Install Python

Install Python, the required Python module, and Git by running the following commands:

```
opkg update
opkg install python-light pyOnionGpio git git-http ca-bundle
```

## 2. Download the Project Code

The code for this project can be found in Onion's [starter-traffic-light](#) repo on GitHub. Use [git to download the code to your Omega](#): navigate to the /root directory, and clone the GitHub repo:

```
cd /root
git clone https://github.com/OnionIoT/starter-traffic-light.git
```

Since the project code is only a single file, you can download the code directly to avoid installing and using git:

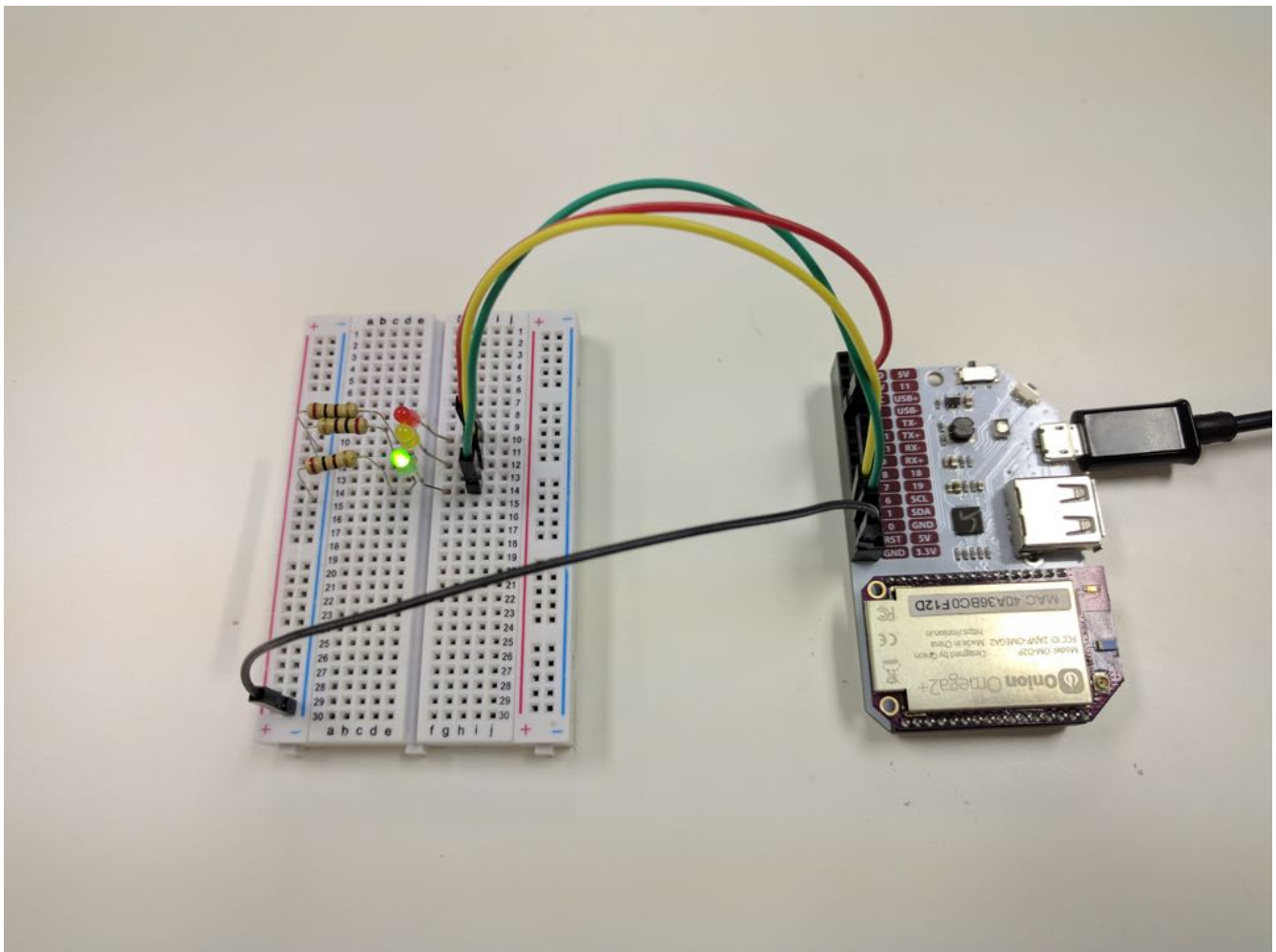
```
mkdir /root/starter-traffic-light
cd /root/starter-traffic-light
wget https://raw.githubusercontent.com/OnionIoT/starter-traffic-light/master/main.py
```

## 3. Running the Project

Enter the project directory and run the main.py file:

```
cd starter-traffic-light
python main.py
```

You should see the lights changing color!



## Code Highlight

We use the Onion GPIO Python module to control the GPIOs. It provides an object with convenient functions such as `setOutputDirection()` and `setValue()` that allow us to control the Omega's GPIOs and abstract a lot of the work under the hood.

For some insight in how the GPIO class works, take a look the `main.py` file:

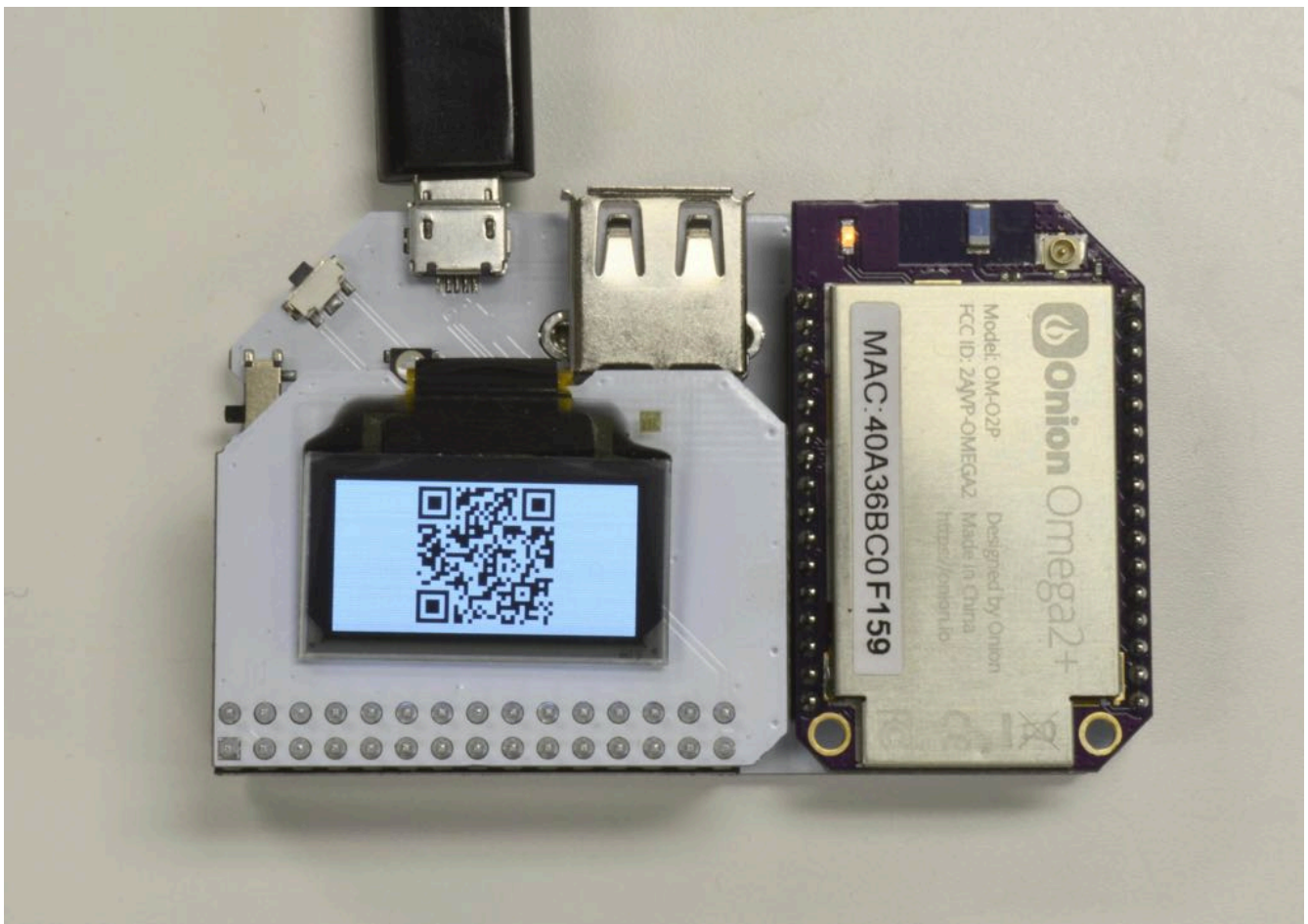
- First, three `OnionGpio` objects are instantiated, with the GPIO number passed in. Now there are three `OnionGpio` objects, each controlling one of the Omega's GPIOs
- Next, all three GPIOs are set to the output direction with a default value of 0 or OFF by calling the `setOutputDirection()` function on each of the `OnionGpio` objects
- Then, the values on the GPIOs can be changed at any time by calling the `setValue()` function on the objects. The `setSignal()` function sets the value of all three GPIOs depending on the color the miniature traffic light is meant to be showing, as dictated by the function argument.

For more details, take a look at the [Onion GPIO Python module documentation](#).

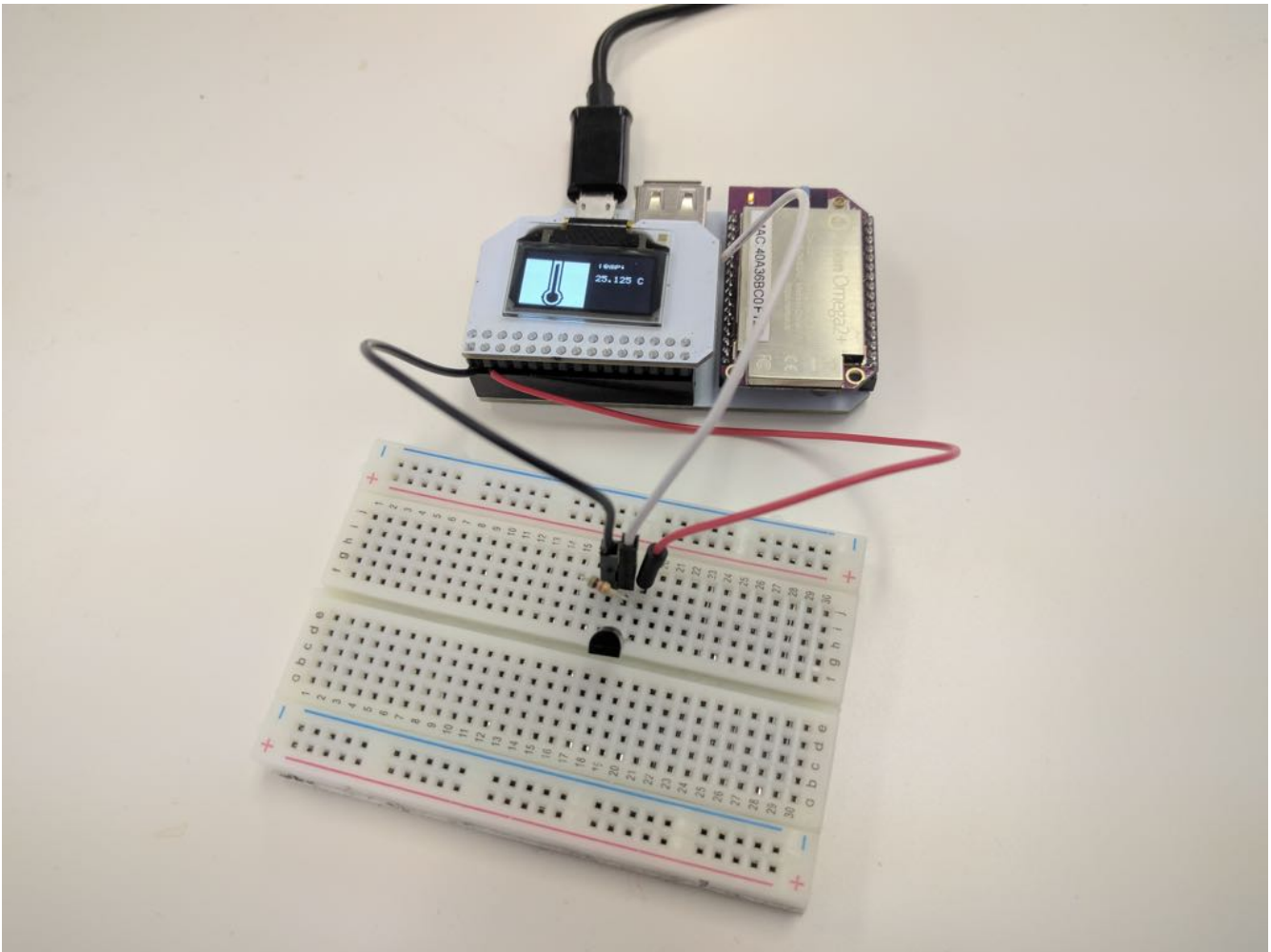
### 3 | OLED Expansion Projects

Now that you're a little more familiar with using the Omega, let's take advantage of the Omega's wireless connectivity and processing power to do create some more intricate projects. Since the Omega2 IoT computer doesn't have video output like a traditional computer, we'll use the **OLED Expansion** as the primary way to communicate information to users.

We can generate images to display:



Or display data from online sources or attached hardware:



## Concepts

A highlight of the concepts that will be covered in these projects:

- Acquiring data from a one-wire sensor
- Displaying sensor data on the OLED
- Generating images to display on the OLED
- Installing additional software on the Omega
- Using APIs to get data from online services
- Reporting data to the Ubidots IoT Platform

## Projects

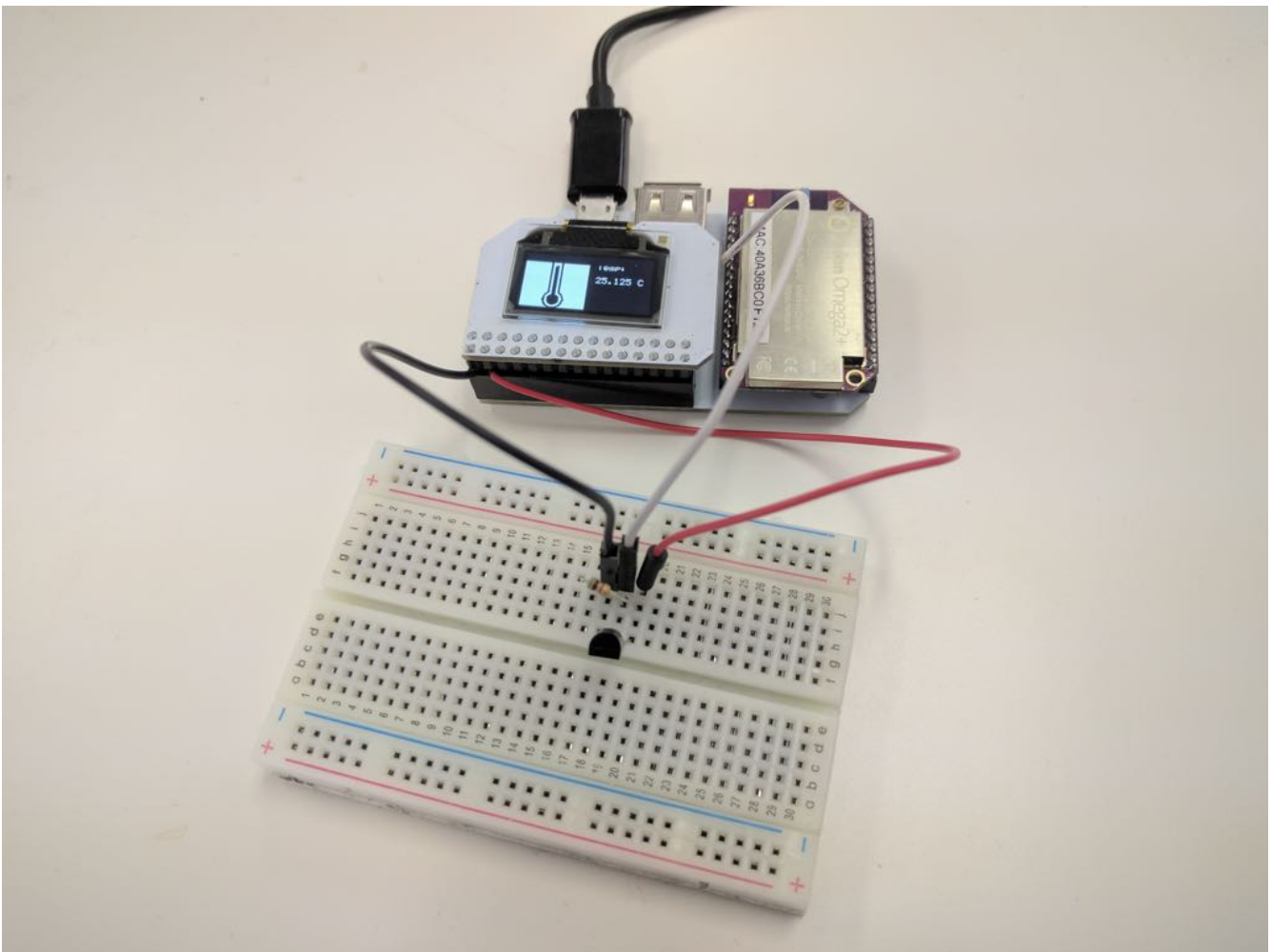
1. **QR Code Generator**
  - Create QR codes with your very own text and display them on the OLED screen
2. **News Flash Headlines**
  - Show the latest news headlines using your Omega
3. **Stock Ticker**
  - Get stock ticker data for your favorite companies
4. **Twitter Feed Display**



- Stay up to date with the latest Tweet of any Twitter user
5. **Ambient Temperature Monitor**
- Measure and display the ambient temperature. Push the data to the Ubidots IoT Platform and view it from anywhere!

## Ambient Temperature Monitor

This project will allow you to read temperature from a digital sensor, display it on the OLED Expansion, and also push the data to [Ubidots IoT Platform](#) for logging and monitoring!



Visualize and track the data on Ubidots:



## Overview

**Skill Level:** Beginner-Intermediate

**Time Required:** 20 minutes

We'll be using a software 1-Wire bus to read the temperature from a sensor. The code will then write the value on the OLED Expansion and push the data to your [Ubidots](#) account. We're also using [Onion's pyOledExp module](#) to provide control of the OLED Expansion.

This project also shows how, with a little bit of craftiness, it's possible to use the Omega's GPIOs to control external circuits while still using the OLED Expansion.

The complete project code can be found in [Onion's temperature-monitor repo on GitHub](#).

## Ingredients

- [Onion Omega2](#) or [Omega2+](#)
- Any [Onion Dock](#) that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
- [Onion OLED Expansion](#)
- 1x [DS18B20 1-Wire Temperature Sensor](#)
- 1x [5.1 kΩ Resistor](#)
  - We used 5.1 kΩ but anything between 4 and 6 kΩ will work just as well
- 3x [Male-to-Female](#) or [Male-to-Male Jumper Wires](#)
  - Make sure they use threaded wire on the inside
- 1x [Breadboard](#)

Tools:

- [Wire Cutter](#)
- [Wire Stripper](#)

## Step-by-Step

Follow these instructions to setup the Ambient Temperature Monitor project on your very own Omega!

### 1. Prepare

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

**Do not plug in your OLED Expansion just yet.**

### 2. Install Software

[Connect to the Omega's Command line](#) and install Python as well as some of the packages we need:

```
opkg update
opkg install python-light python-urllib3 pyOledExp ubidots-client git git-http ca-bundle
```

The `python-urllib3` package will allow us to make HTTP requests in Python, while the `pyOledExp` package gives us control of the OLED Expansion.

The `ubidots-client` package will allow us to push and pull data from the Ubidots IoT Platform.

The `git`, `git-http`, and `ca-bundle` packages will allow us to download the project code form GitHub.

### 3. Download the Project Code

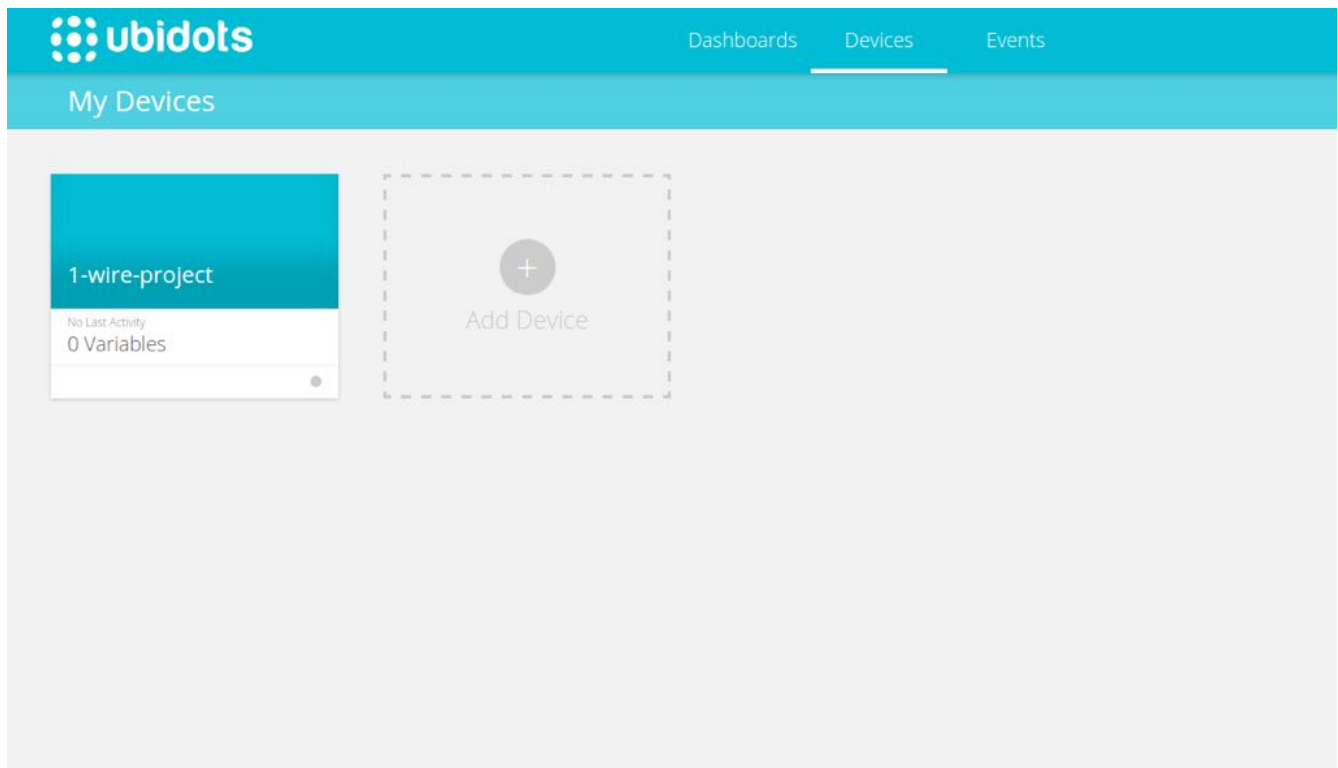
The code for this project is all done and can be found in Onion's [temperature-monitor repo](#) on GitHub. Use [git to download the code to your Omega](#): navigate to the `/root` directory, and clone the GitHub repo:

```
cd /root
git clone https://github.com/OnionIoT/oled-temperature-monitor.git
```

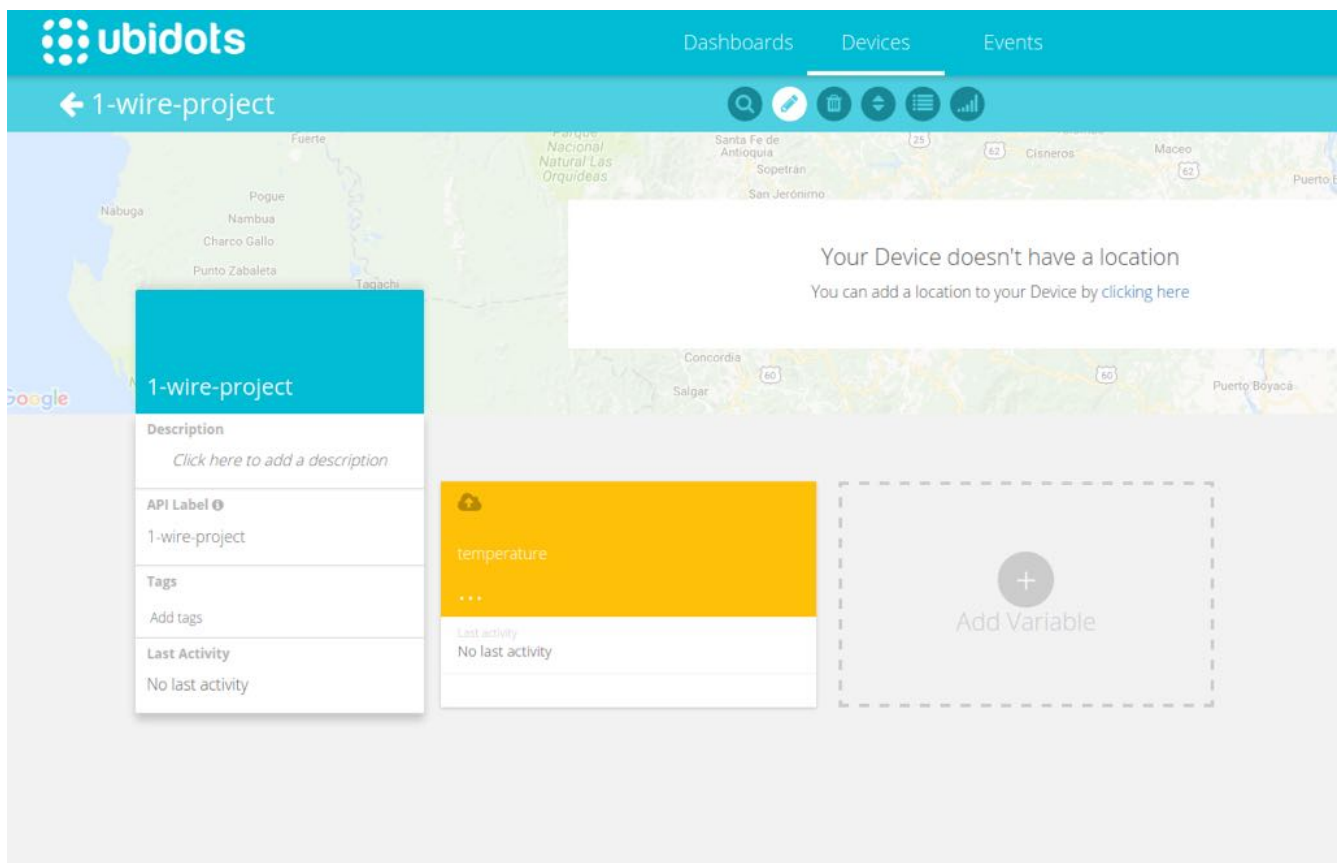
### 4. Setup Ubidots

First, sign up for a [Ubidots](#) account. At the time this was written, you should have 5000 credits in your account available for trial and testing. This is more than enough to get this project running!

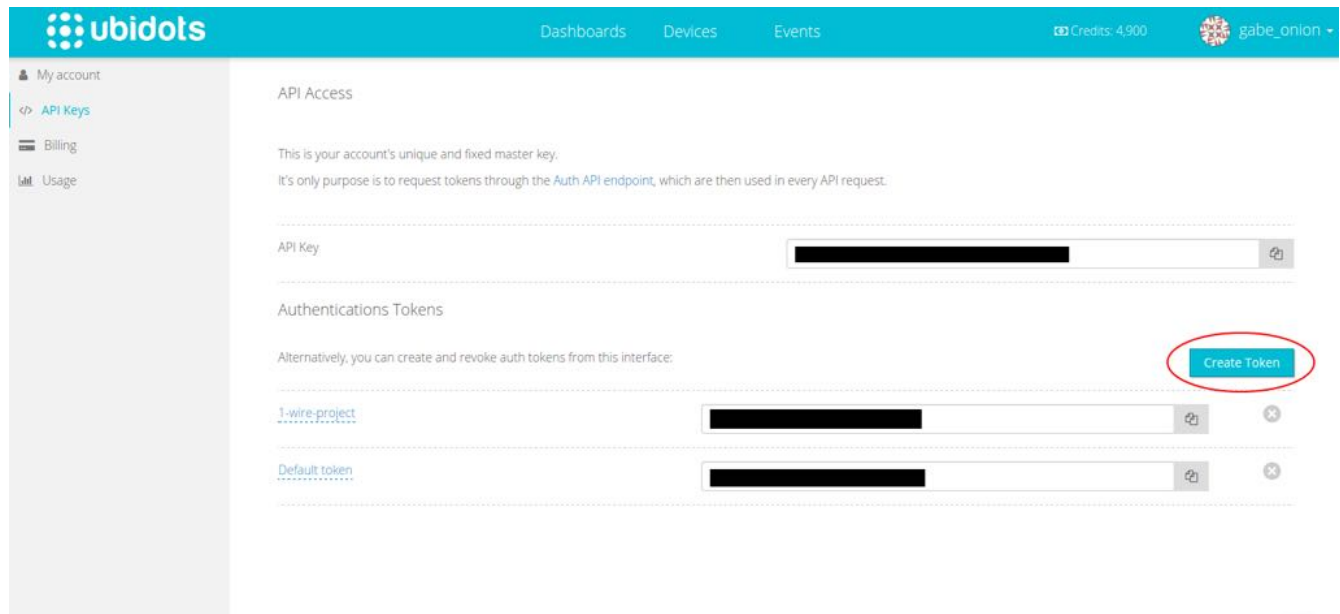
Then go to the [account homepage](#), and click on Devices at the top, and click on the grey **Add Device** button. Call it `1-wire-project` like so:



Now we need to add a **variable** to store our data. Click on the device's blue card to go to its device page. Then click on the grey Add Variable button, then click Default. Call the new variable `temperature` (case sensitive) like so:



Now we need to create an **API key** for this project. Click on your username in the top right of the screen, then click My Profile. In the profile menu, click on API Keys on the left. Then click on the blue Create Token button to generate a token; click on the `newToken` text to rename it to `1-wire-project`.



In order to authenticate your requests to Ubidots, you will need to put this long string of text into the `config.json` file in the project directory on the Omega. Replace the `yourTokenHere` placeholder with the key you just created:

```
{
  "token": "yourTokenHere",
  "deviceName": "1-wire-project"
}
```

Your software is now ready to run!

## 5. Prepare the Wires

Next you will need to prepare the wires. The OLED Expansion does not have female headers to connect wires or Expansions because they may block the screen. To deal with this, do the following for each of the 3 wires:

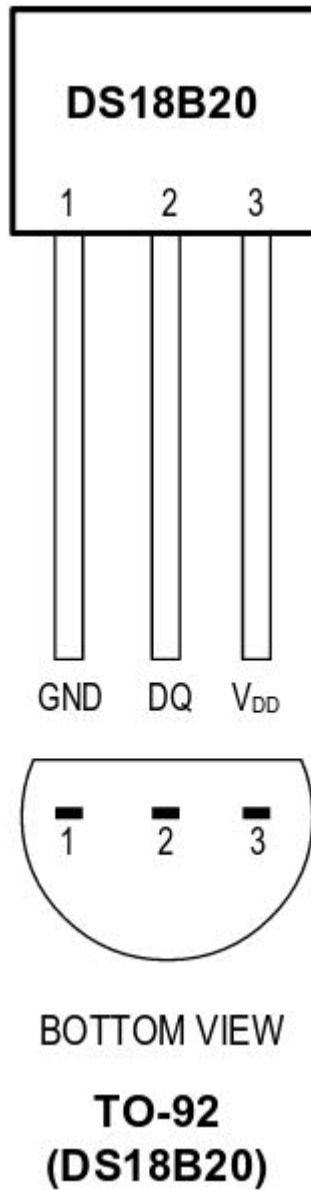
1. Using the wire cutter, cut one connector of the jumper wire off while leaving a male end intact.
  - One male end is needed to connect to the breadboard!
2. Using the wire stripper, strip about 10mm of insulation from the freshly cut end.
3. Pinch the exposed wire with one hand and twist it several times until the threads are thoroughly wound around each other.
  - This is so they don't fray.
4. Take the twisted wire and bend it 180 degrees backwards in half to make a thin hook-like shape.
5. Twist the hook again so it closes and doesn't fray.

Your wires should look like this:



## 6. Connect the Sensor

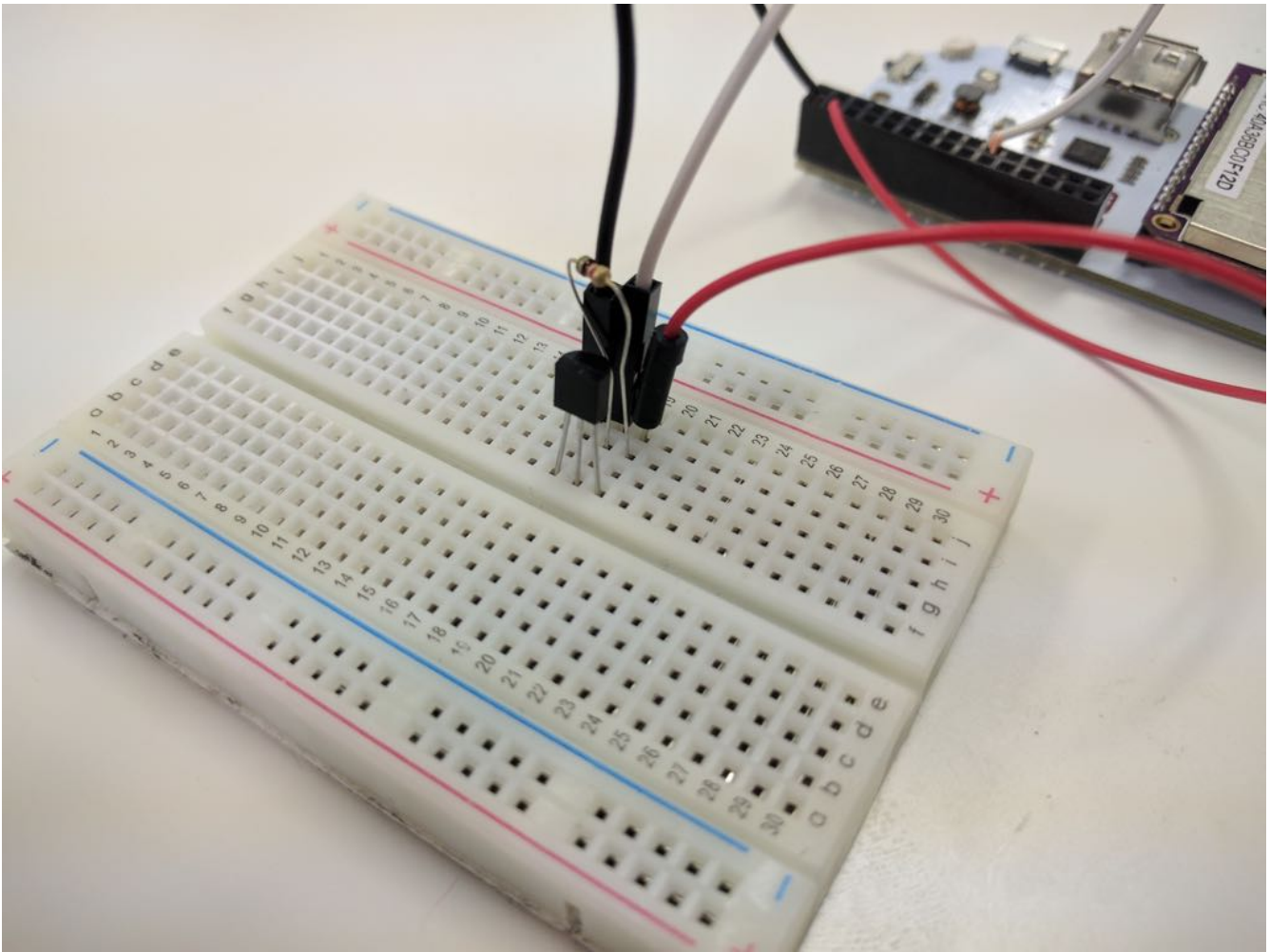
Use this diagram for reference when wiring up the sensor:



We will treat the flat side as the front.

1. With the front of the sensor facing the middle gap of the breadboard, insert the three pins across 3 adjacent rows.
2. Connect the 5.1k $\Omega$  resistor to both DQ (pin 2) and V<sub>DD</sub> (pin 3).

Your sensor should look like this:



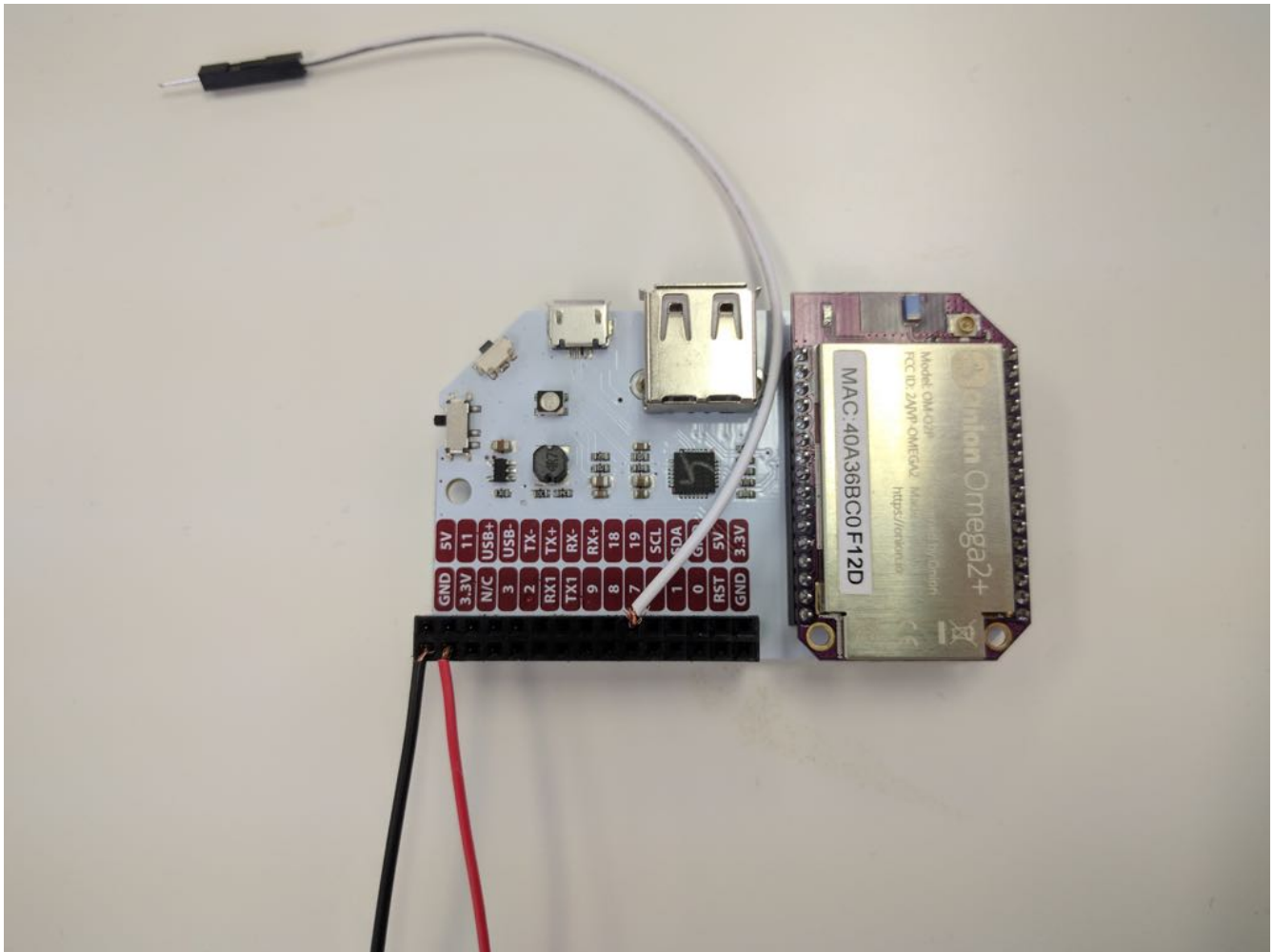
## 7. Sensor -> Omega

Our sensor is now ready and we need to connect it to the Omega using the wires we just prepared. The male end of the wire will plug into the breadboard while the bare ends will go into the Dock's Expansion pins.

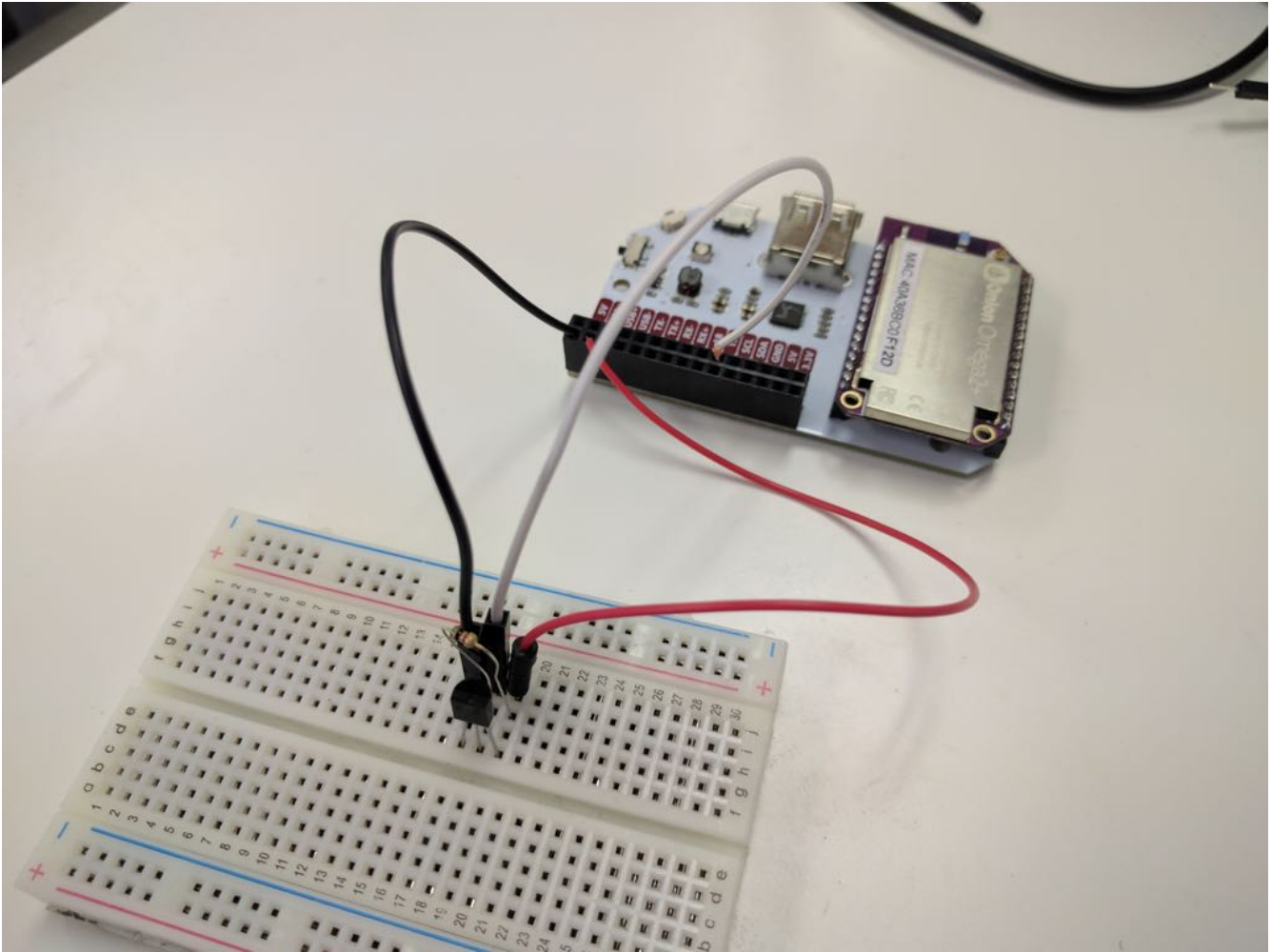
1. Connect GND (pin 1) to the Omega's GND pin.
2. Connect DQ (pin 2) to the Omega's GPIO19.
3. Connect Vdd (pin 3) to the Omega's 3.3V pin.

Insert the bare ends of the wire into the Expansion Dock like this:



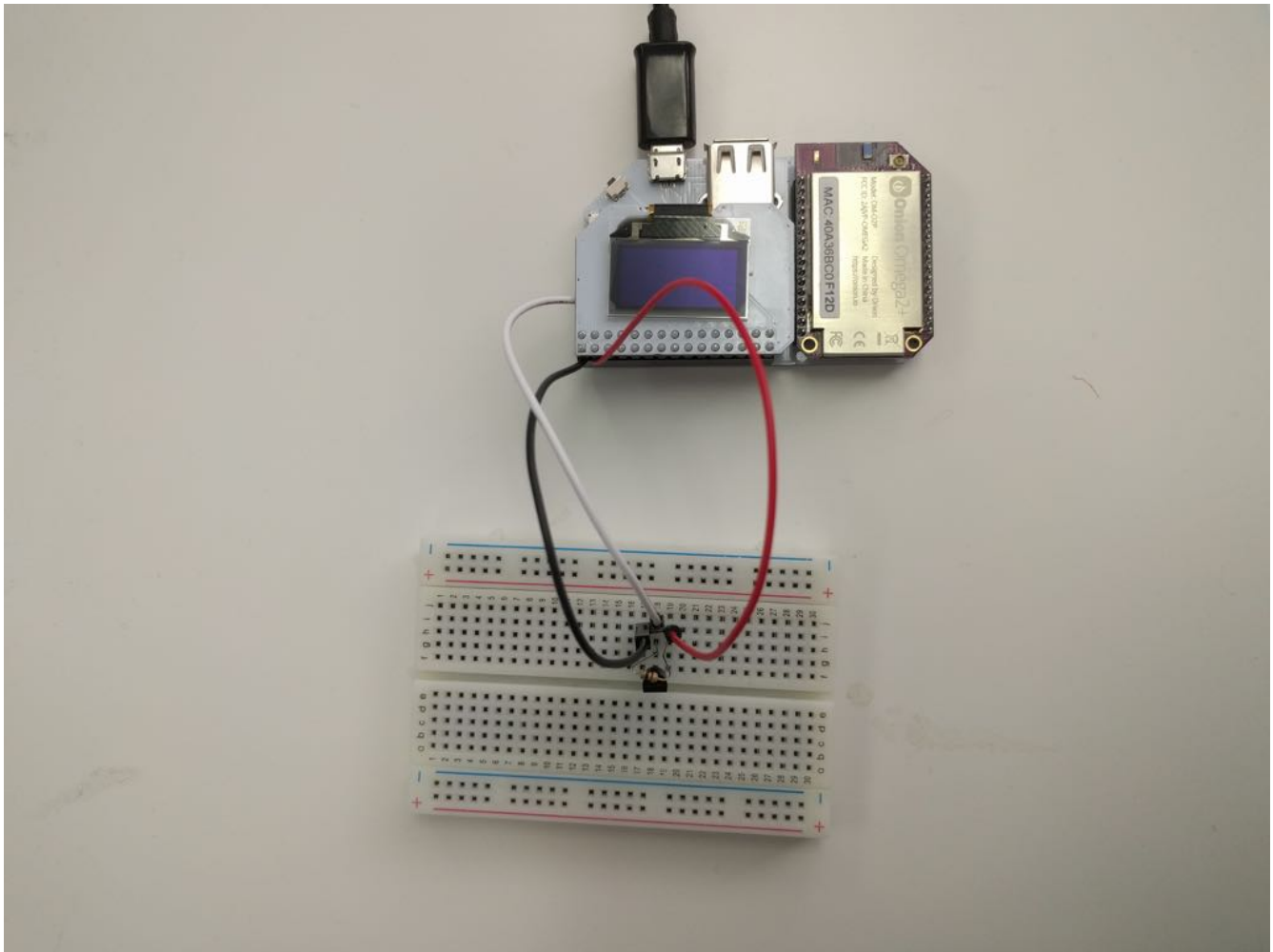


Your circuit should now look like this so far:



## 8. Connect OLED Expansion

The OLED Expansion will then plug in on top of the wires; it might be a little bit of a tight squeeze but you will definitely be able to successfully plug in the Expansion. Plug it in and it should look like this:



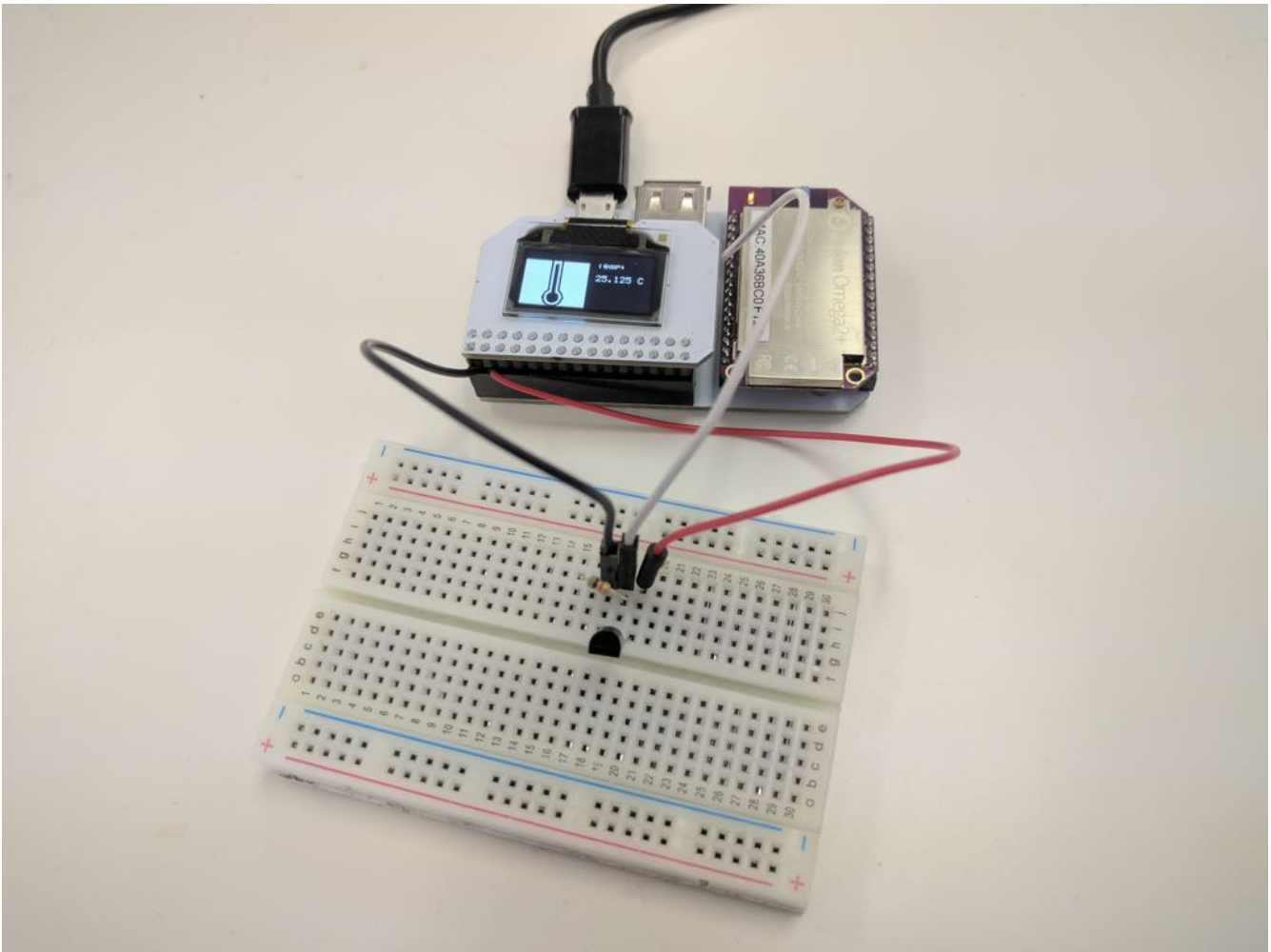
Good work! You've just done a little bit of physical hacking to use a sensor alongside the OLED Expansion.

## 9. Run the Code

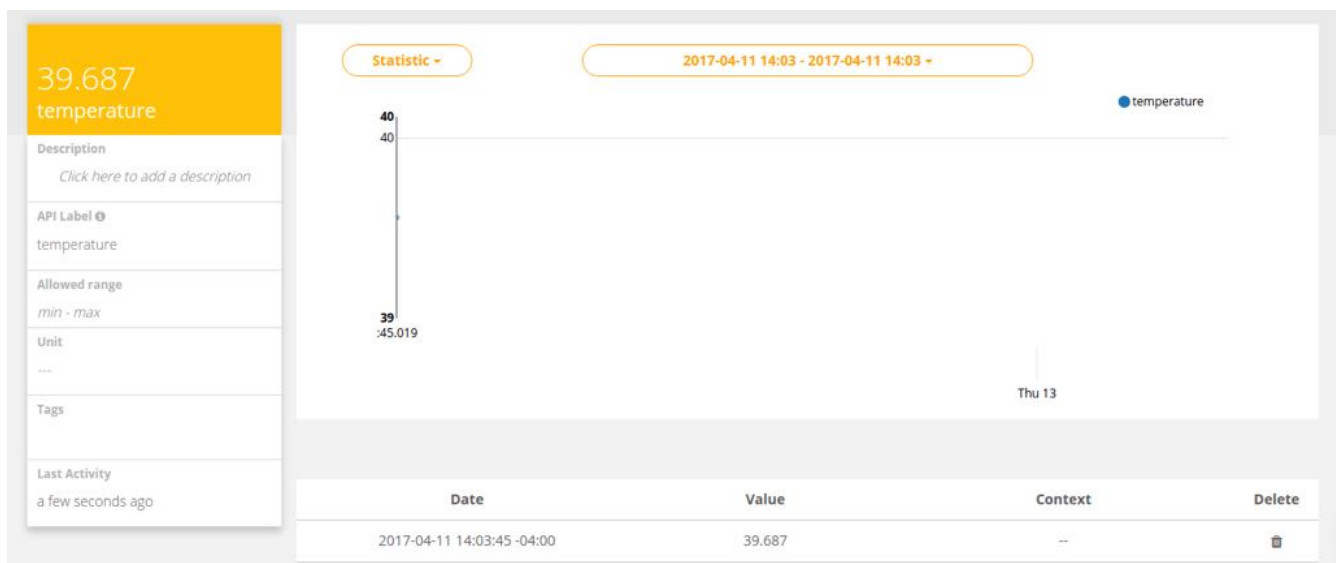
On the Omega, launch the program:

```
cd /root/temperature-monitor  
python main.py
```

You should see something like this:



Now go to your Ubidots account page and check on your temperature variable in the 1-wire-project device. You should see your new reading:



## 10. Automate the Program to Run Periodically

The program will read the temperature, display it on the OLED, push the value to Ubidots, then promptly exit. We'll use `cron`, a super useful Linux utility, to have the program run periodically.

Enter `crontab -e` to add a task to the `cron` daemon, it will open a file in `vi`, enter in the following:

```
* * * * * python /root/temperature-monitor/main.py
#
```

This assumes that your project code is located in `/root/oled-temperature-monitor`

Now, we'll restart `cron`:

```
/etc/init.d/cron restart
```

And **the code will run once every minute**, pushing data to your Ubidots account so that you can view the changes over time!



Check out the Omega documentation for more info on [using cron](#)

## Code Highlight

This project makes use of two main interfaces: 1-Wire and Ubidots.

### 1-Wire

The 1-Wire protocol is a bus-based protocol that, as the name implies, uses one data wire to transmit data between devices. The `main.py` script uses some functions from the `oneWire.py` module to automatically do the following:

- setup a 1-Wire bus on the Omega
- scan for the temperature sensor's address
- use the sensor in subsequent calls without you having to probe it yourself!

## Ubidots

The Ubidots requests are handled by the `ubidots-client` command line utility that the `Ubidots` class calls. This is the same as running the command below:

```
ubidots -t (TOKEN) -d (DEVICENAME) set '{"variableOne":12, "variableTwo":10, ...}'
```

In the case of this project, the equivalent command would be:

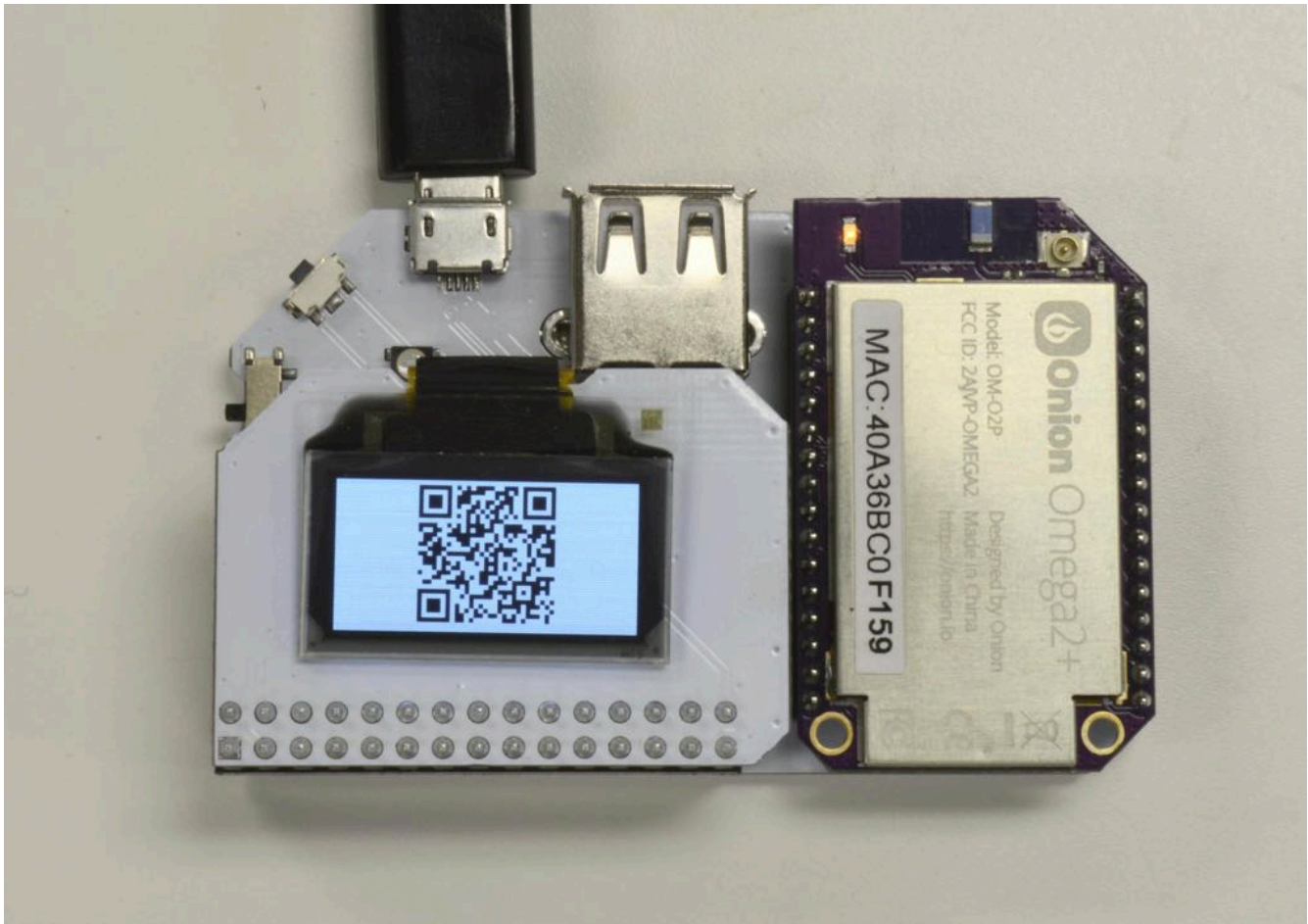
```
ubidots -t (TOKEN) -d (DEVICENAME) set '{"temperature": (VALUE READ FROM SENSOR)}'
```

## QR Code Generator

QR Codes are essentially two dimensional barcodes that can easily be scanned with any camera and a little bit of processing power. The average smartphone will make short work of any QR code it comes across.



In this tutorial, we'll go through how you can use Python to encode text into a QR Code and display it on your OLED Expansion:



The resulting code can then be scanned to read the encoded text. If it's a URL that's encoded, most smartphone QR code readers will open the browser to this URL. Useful if you have a complicated URL!

## Overview

**Skill Level:** Beginner

**Time Required:** 10 minutes

This project is mostly code based, all of the code can be found on this Onion GitHub Repo: <https://github.com/OnionIoT/oledQrCodeGenerator>

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
- Onion [OLED Expansion](#)

## Step-by-Step

Ok, here we go! First we'll install some required packages to make everything run smoothly, and then we'll grab the code for this tutorial from GitHub.

## 1. Prepare your Ingredients

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

## 2. Installing Required Packages

We will need to have support for git, Python, and the [Onion OLED Expansion Python Module](#). [Connect to the Omega's command line](#) and run the following command:

```
opkg update
opkg install python-light python-codecs pyOledExp git git-http ca-bundle
```

## 3. Downloading the Code

Now we need to download the Python code from GitHub that actually does all the work. [Connect to the Omega's command line](#) and [clone the project repo from GitHub](#) with the following command:

```
cd /root
git clone https://github.com/OnionIoT/oledQrCodeGenerator.git
```

## 4. Running the Code

Finally, we get to make some QR codes! Navigate into the repo directory:

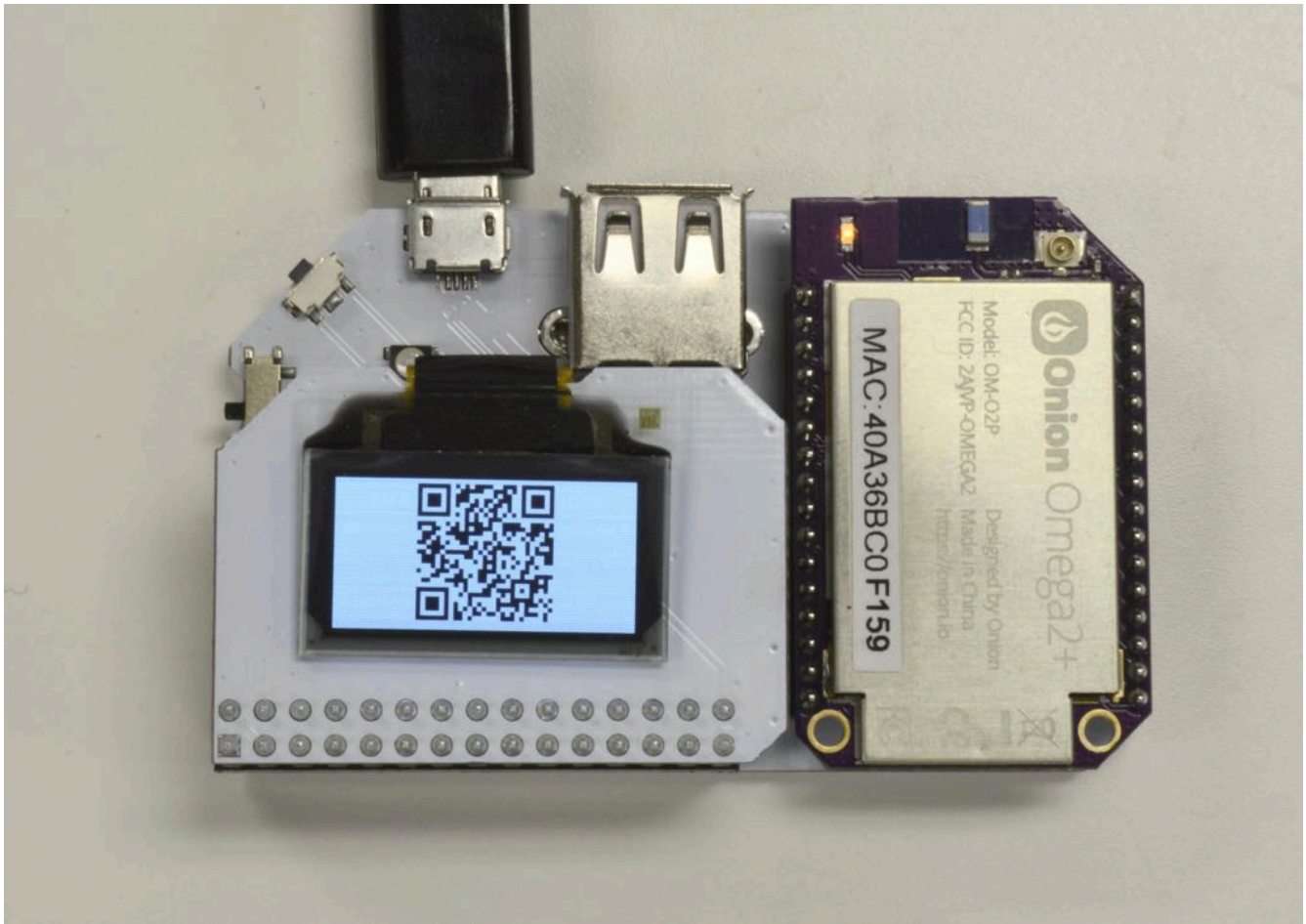
```
cd oledQrCodeGenerator
```

And run the program, the argument to the script is the text that will be encoded in the QR code pattern:

```
root@Omega-18C2:~/oledQrCodeGenerator# python main.py 'Wow, my first QR Code'
> Encoding 21 characters
> Generated QR Code: 31x31 pixels
> Doubled QR Code size: 62x62
> Initializing display
> Setting display mode to inverted
> Writing buffer data to display
```

This will encode the data and display the resulting QR code on the OLED Expansion:





## Program Details

Behind the scenes, the Python code does the following:

- Encodes the input text into a matrix representing the QR Code
  - The size of the QR code is based on the amount of input text
- Converts the QR code matrix into data that can be displayed on the OLED
- Displays the resulting image on the OLED display
  - Performs display initialization
  - Inverts the display colors
  - Displays the generated image file

An additional feature was added for easier scanning: if the QR code is small (less than half the height of the OLED display), the image will be doubled in size so that each QR code pixel shows up as four pixels on the OLED display.

The default generated QR code will be a Version 3 code with the Low error correction setting and a one-pixel border, creating a code that is 31x31 pixels. If the amount of text to be encoded cannot fit in a Version 3 code, the program will select the next version that will fit the amount of data to be encoded. Check out the [Wikipedia entry on QR Codes](#) for more details on QR code versions.

## 5. Using the code as a Python Module (Optional)

The `oledQrCodeGenerator` code can also be imported as a module into your own Python projects!

The `dispQrCode()` function will perform the same actions described above.

### Example Code

A small example showing how to use this module:

```
import sys
sys.path.append("/root")
import oledQrCodeGenerator

print 'Now using the oledQrCodeGenerator'
oledQrCodeGenerator.dispQrCode('Hello!')

print 'All done!'
```

Note that the above code assumes the project code can be found at `/root/oledQrCodeGenerator`. It appends `/root` to the `sys.path` list that Python uses when looking for modules that need to be imported. If the `sys.path.append("/root")` line is not present, Python will return an error saying `ImportError: No module named oledQrCodeGenerator` since it cannot find the module in the usual places it looks.

## Reading QR Codes

It's no fun to just display QR codes and not be able to read them, right?

Don't worry, your smartphone is perfectly capable of reading the code from the OLED:

- On Android, we've used the [QR Code Reader](#) and [QR Barcode Scanner](#) apps successfully
- On iOS, we've had success with the [QR Reader App](#)

For QR codes that encode a lot of text, your phone might take a little while longer to scan the code. Trial and error works best in this scenario: try moving your phone to different distances and angles from the OLED.

## Acknowledgements

The code in the `qrcode` directory is a stripped-down version of `lincolnloop`'s `python-qrcode` repo: <https://github.com/lincolnloop/python-qrcode>

## News Flash Headlines

This project will pull a fresh headline from News API and display it to the OLED screen. [News API](#) is a news aggregator that returns headlines from a variety of news sources as JSON data.



## Overview

**Skill Level:** Beginner

**Time Required:** 20 minutes

The project will use a Python script to call the News API `/articles` endpoint for headline data.

The complete project code can be found in Onion's [oled-news-flash repo on GitHub](#).

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
- Onion [OLED Expansion](#)

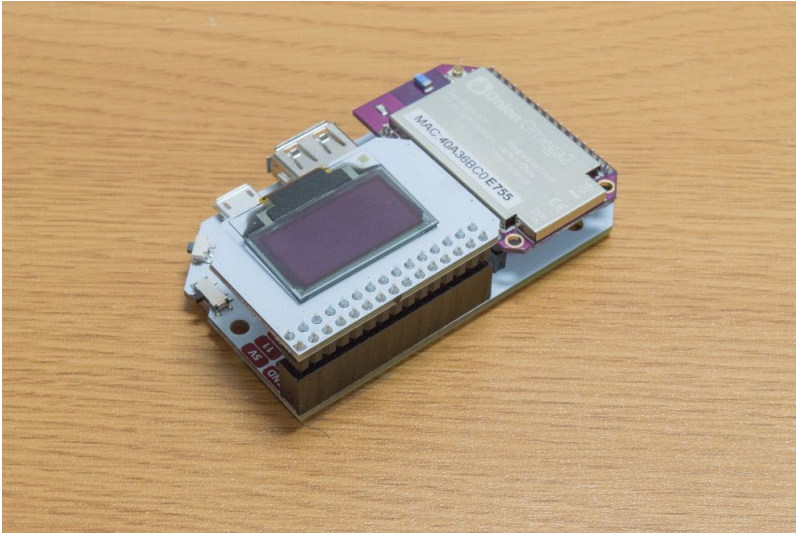
## Step-by-Step

Here's how to get your own headlines screen running on your Omega!

## 1. Prepare your Ingredients

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

Once that's done, plug in your OLED Expansion:



## 2. Install Python

[Connect to the Omega's command line](#) and install Python and some additional packages we need:

```
opkg update
opkg install python-light python-urllib3 pyOledExp
```

The `python-urllib3` package will allow us to make HTTP requests in Python, while the `pyOledExp` package gives us control of the OLED Expansion.

## 3. Download the Project Code

All the code from the project can be found in the [oled-news-flash repo on GitHub](#).

This project only has two files, so you can download it directly to your Omega without much hassle.

```
mkdir /root/oled-news-flash
cd /root/oled-news-flash
wget https://raw.githubusercontent.com/OnionIoT/oled-news-flash/master/oledNewsFlash.py
wget https://raw.githubusercontent.com/OnionIoT/oled-news-flash/master/config.json
```

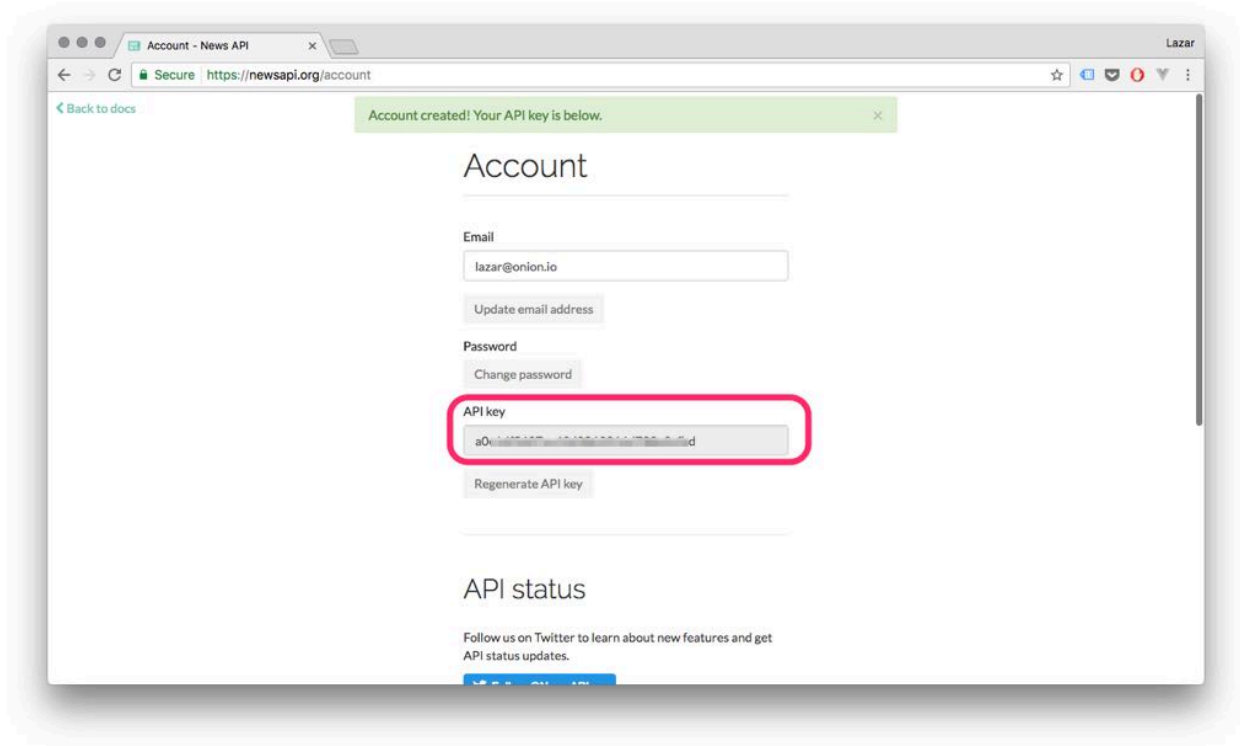
If you'd like to use git **instead**, [install Git on your Omega](#), navigate to the `/root` directory, and clone the GitHub repo:

```
cd /root
git clone https://github.com/OnionIoT/oled-news-flash.git
```

## 4. Obtain a News API Key

We need an API key in order to access the News API endpoints. The simplest way is to create an account which will give us access to the News API key generator.

1. Register at <https://newsapi.org/register> and copy your API Key:

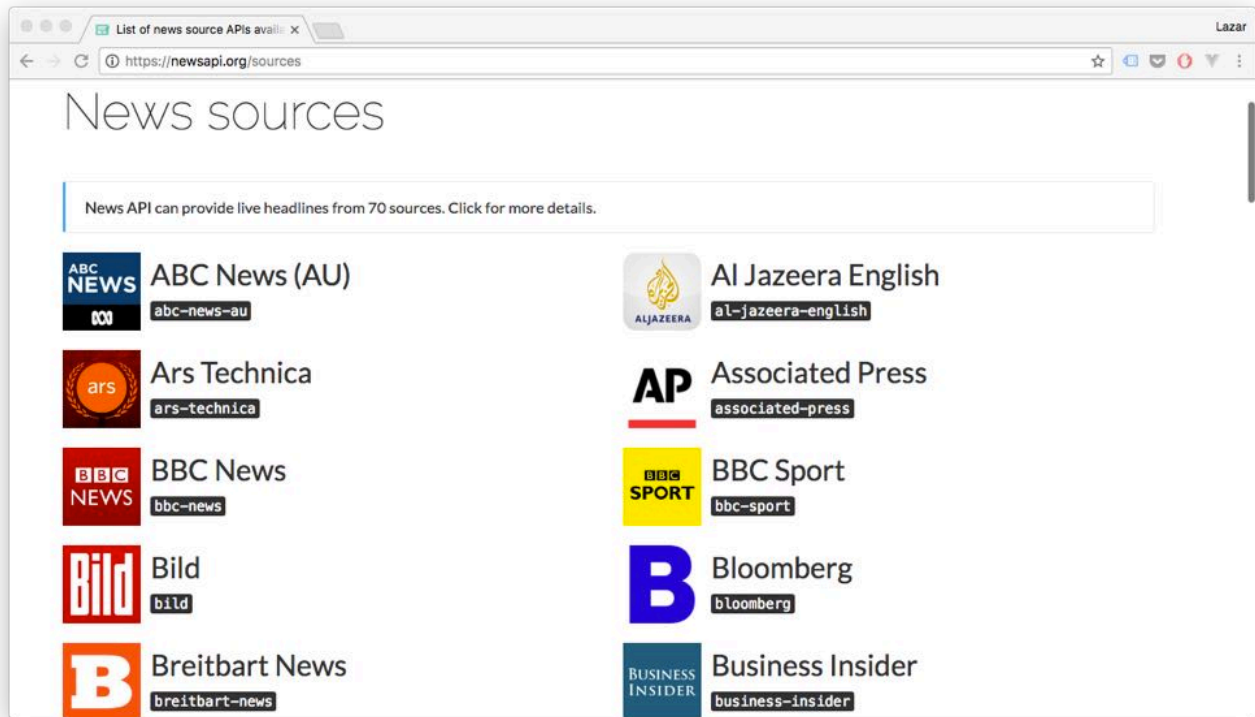


2. Open up `config.json` and paste the API key generated as the `X-API-KEY` value - replacing your api key here:

```
root@Omega-2729:~/oled-news-flash# cat config.json
{
  "X-API-KEY" : "a0b1c2d3e4f5g6h7i8j9k0l1m2n3o4p5q6r7s8t9u0v1w2x3y4z5",
  "source" : "reuters",
  "sortBy" : "latest"
}
```

## 5. Choose Your Source

News API gets headlines from 70 different news sources. We've set the default source to Reuters, but you can easily change the source of your headlines. Head over to [News API's sources page](#) and pick your source:



Open up `config.json` and copy the text under any source you wish as the value for `source` - replacing `reuters`.

```
root@Omega-2729:~/oled-news-flash# cat config.json
{
  "X-API-KEY" : "a0[REDACTED]d",
  "source" : "the-verge",
  "sortBy" : "latest"
}
```

## 6. Run it!

On your Omega's command line, run the following:

```
python oledNewsFlash.py
```

And you should see the latest news headline on your OLED screen.



## 7. And Beyond

Now we can automate this script with `cron` to keep the headlines updated on the OLED screen.

Enter `crontab -e` to add a task to the `cron` daemon, it will open a file in `vi`, enter in the following:

```
*/15 * * * * python /root/oled-news-flash/oledNewsFlash.py  
#
```

This assumes that your project code is located in `/root/oled-news-flash/` - if it's not, don't forget to change the directory!

Now, we'll restart `cron` to update it with our new task:

```
/etc/init.d/cron restart
```

And **the code will run once every 15 minutes**, updating the OLED screen with the latest headline.

Check out the Omega documentation for more info on [using cron](#)

## Code Highlight

Many web sites and services provide Application Programming Interfaces (API) to allow others to call on the data they provide without a whole webpage to bog it down.

Calling an API is all about knowing what the API needs, and how to deliver that data.

To contact any API, we need to know at least two major things:

1. URL - the address we need to look up
2. Method - what method the URL accepts, and what do they do

## Endpoints

For this project, our URL is `https://newsapi.org/v1/articles`. The URL has two bits to it, first is the the actual API's location - `newsapi.org/v1/`. The second is the **endpoint**, kind of like a specific apartment number of the address. Here it's `/articles`.

Together, they're often referred as an **endpoint** of the API.

News API has a `/sources` endpoint as well which provide different services when called.

## Methods

Now that we have our endpoint, we need know how to request data from it.

When sending a request, it must be made with a request **method** to let the server know what we need at a broad level from the endpoint.

HTTP supports at least nine different request methods to accommodate different needs. The most common ones are 'GET', 'POST', and 'DELETE'.

Logically, to get data from the `/articles` endpoint, we need to send it a 'GET' request.

## Parameters, Headers and Bodies

Often, APIs provide personalized data - calendars, emails, and other user-specific data. To implement this kind of interaction, requests are sent with **parameters**, **headers** and possibly a **body** for 'POST' requests.

Parameters are strings that get appended to the request URL with details about our request. This is the most basic way of communicating additional information to the server.

In this case, the `source` and `sortBy` values are sent to the server through URL parameters. Meaning, the request from the news flash code is to the following URL:  
`https://newsapi.org/v1/articles?source=reuters&sortBy=latest`

The API key is a way to identify and authenticate a user of the service, allowing APIs to pull up user-specific data. For an API serving general information like News API, an API key is mostly useful in identifying the user's level of access.

Generally, the API key is passed through the HTTP request's header - a list of key-value pairs that is sent with our request. What goes in the header depends on the particular API being used. [The documentation](#) should always specify what kind of things should be put in the header to correctly get information.



In the news flash code, the headers only contain the {'X-API-KEY':<API KEY>} pair.

The body is typically a JSON list of key-value pairs used to store content for 'POST' requests. Again, the specifics of the body depends on what the API needs.

## Stock Ticker

For this project, we'll be obtaining the latest stock data from an online API for a configurable list of stocks and displaying the data:



**Disclaimer:** It's important to note that most stock data APIs provide data that is delayed and does not represent the latest information. Use this project for informational purposes only, it is not meant to offer any investment advice. Orion is not responsible for any investment decisions or their outcomes, please invest responsibly.

### Overview

**Skill Level:** Beginner

**Time Required:** 10 minutes

This code will be written in Python and we'll be making use of a [Google Finance](#) API to grab stock data. It will print the following data to the OLED:

1. Date and time (UTC)
2. Stock symbol (up to 4 characters)
3. Current trading price (USD)
4. Percentage change since last closing

Specifically, the code uses the `info` endpoint; this is technically [deprecated](#), but still seems to remain active and returns up-to-date finance data.

We also use the [Onion's pyOledExp module](#) to provide control of the OLED Expansion.

The complete project code can be found in Onion's [oled-stock-ticker repo on GitHub](#).

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
- Onion [OLED Expansion](#)

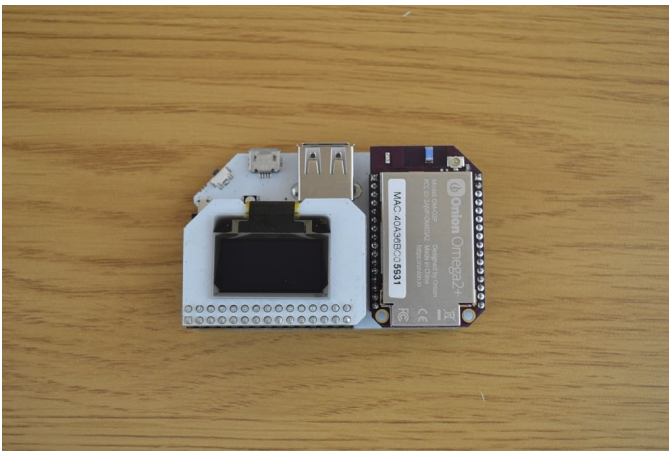
## Step-by-Step

Follow these instructions to set this project up on your very own Omega!

### 1. Prepare

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

Once that's done, plug in your OLED Expansion:



### 2. Install Python and Git

[Connect to the Omega's command line](#) and install Python as well as some of the packages we need:

```
opkg update
opkg install python-light python-urllib3 pyOledExp git git-http ca-bundle
```

The `python-urllib3` package will allow us to make HTTP requests in Python, while the `pyOledExp` package gives us control of the OLED Expansion.

The `git`, `git-http`, and `ca-bundle` packages will allow us to download the project code form GitHub.

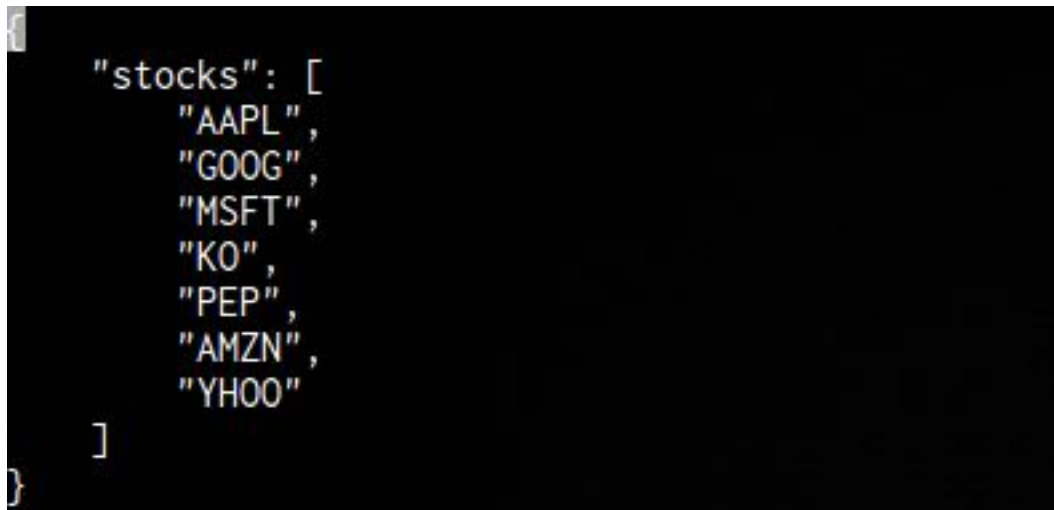
### 3. Download the Project Code

The code for this project is all done and can be found in Onion's [oled-stock-ticker repo](#) on GitHub. We'll use [git to download the code to your Omega](#): navigate to the `/root` directory, and clone the GitHub repo:

```
cd /root
git clone https://github.com/OnionIoT/oled-stock-ticker.git
```

### 4. Setup the Ticker

The `config.json` file holds all of the settings for the project. Populate the `stocks` array with the symbols of the stocks you wish to track.



```
{
  "stocks": [
    "AAPL",
    "GOOG",
    "MSFT",
    "KO",
    "PEP",
    "AMZN",
    "YHOO"
  ]
}
```

Some notes about the stock symbols:

- The OLED has 8 rows and the 1st will be used for the date and time, so only the first seven will be shown.
- Due to space constraints on the OLED, the stock ticker can properly display only stocks with 4 letters or less.
- The code assumes the stocks are traded in USD.

### 4. Run the Code

Now run the code: `python main.py`



If you're interested in how the `pyOledExp` code can be used to control the OLED Expansion, take a look at how it's used in [the `oledDriver.py` file in the project code](#) and also check out the [pyOledExp Module documentation](#).

## 6. Automate the Program to Run Periodically

The program will grab and display the latest stock info, then promptly exit. We'll use `cron`, a super useful Linux utility, to have the program run periodically.

Enter `crontab -e` to add a task to the `cron` daemon, it will open a file in `vi`, enter in the following:

```
* * * * * python /root/oled-stock-ticker/main.py
#
```

This assumes that your project code is located in `/root/oled-stock-ticker`

Now, we'll restart `cron`:

```
/etc/init.d/cron restart
```

And the code will run once every minute, generating *literally* up-to-the-minute stock information on your OLED!

**Again, remember that the stock information is likely delayed data and shouldn't be used to inform investment decisions!**

Check out the Omega documentation for more info on [using cron](#)

## Code Highlight

This code does the following:

1. Load the list of stocks from the configuration file
2. Creates a timestamp of when this script was called
3. Sends a GET request with the given stocks to the Google Finance API
4. Cleans up and stores the response into a variable
5. Formats relevant information such as symbol and price for displaying on the OLED
6. Prints the timestamp and stock information to the OLED

## Going Further

You can customize the formatting or which information you want to display on the OLED by changing the `formatGoogleStockInfo()` function in `stocks.py`.

To see all of the available information, query the API for a single stock by running `stocks.py` and the symbol as the first argument:

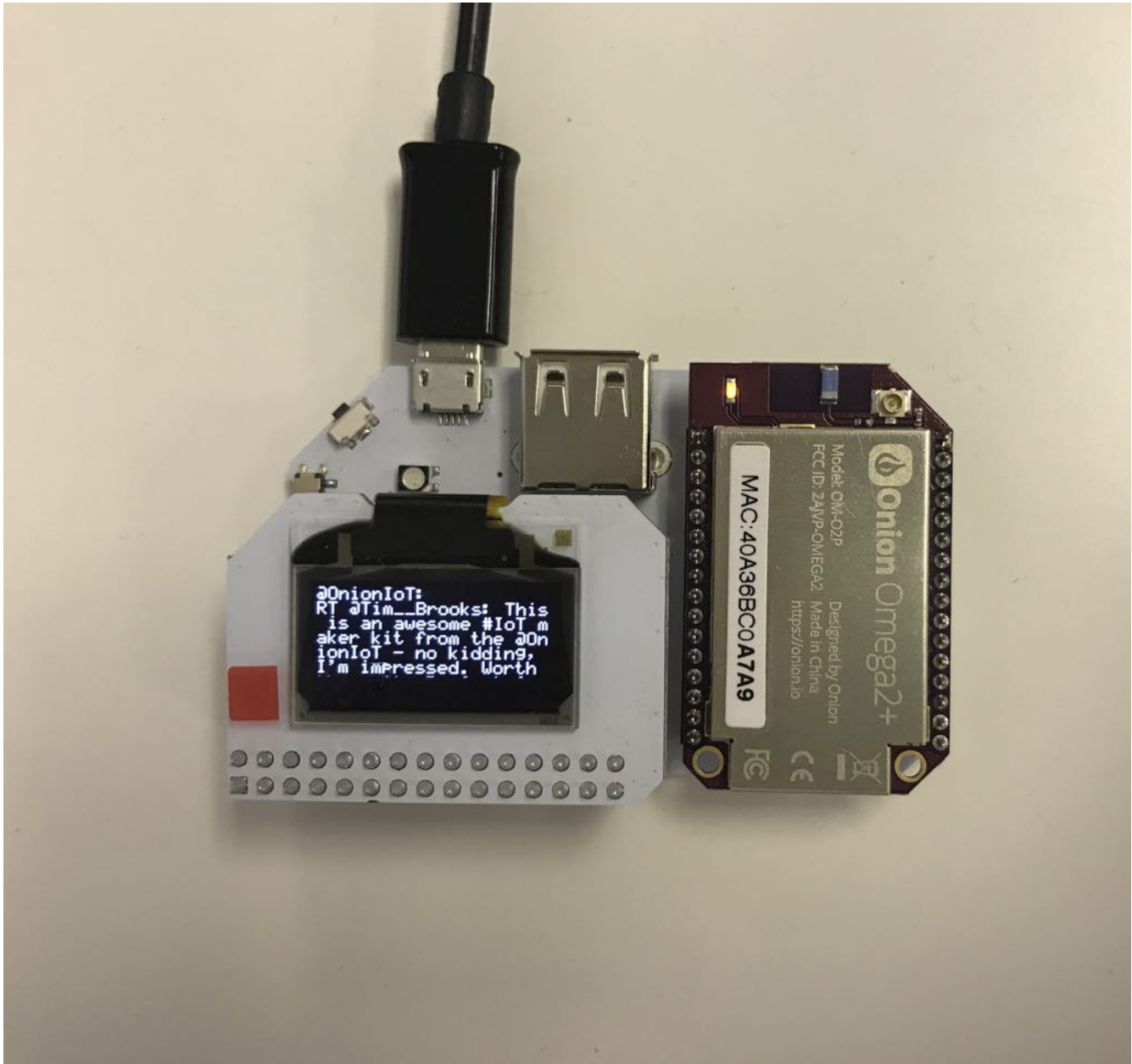
```
python stocks.py BB
```

```
[
  {
    "c": "+0.01",
    "ccol": "chg",
    "e": "TSE",
    "ltt": "4:00PM EDT",
    "cp_fix": "0.09",
    "c_fix": "0.01",
    "l": "10.64",
    "s": "0",
    "lt": "Apr 6, 4:00PM EDT",
    "pcls_fix": "10.63",
    "t": "BB",
    "lt_dts": "2017-04-06T16:00:00Z",
    "l_fix": "10.64",
    "cp": "0.09",
    "id": "674819",
    "l_cur": "CA$10.64"
  }
]
```

Add any information that's relevant to you to the data that's displayed on the OLED screen. Keep in mind that the screen can display 21 characters per line, any additional characters will be automatically relegated to the next line.

## Twitter Feed Display

For this project, we'll be displaying the latest Tweet of a specified Twitter user on the OLED Expansion:



### Overview

**Skill Level:** Beginner

**Time Required:** 20 minutes

The code will be written in Python and we'll be making use of [Twitter's REST APIs](#) to grab Tweet data. Specifically, the code uses the [statuses/user\\_timeline](#) endpoint. Also in use is [Onion's pyOledExp module](#) to provide control of the OLED Expansion.

The complete project code can be found in Onion's [oled-twitter-display repo on GitHub](#).

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
- Onion [OLED Expansion](#)

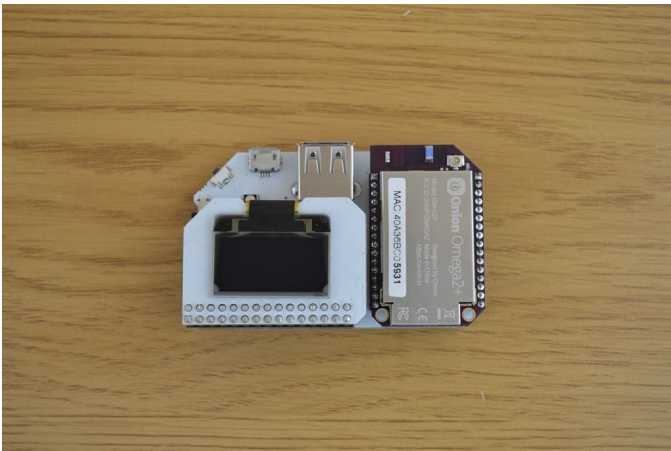
## Step-by-Step

Follow these instructions to set this project up on your very own Omega!

### 1. Prepare

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

Once that's done, plug in your OLED Expansion:



### 2. Install Python

[Connect to the Omega's command line](#) and install Python as well as some of the packages we need:

```
opkg update
opkg install python-light python-urllib3 pyOledExp
```

The `python-urllib3` package will allow us to make HTTP requests in Python, while the `pyOledExp` package gives us control of the OLED Expansion.

### 3. Download the Project Code

The code for this project is all done and can be found in Onion's [oled-twitter-display repo](#) on GitHub. Follow the [instructions on installing Git](#), navigate to the `/root` directory on the Omega, and clone the GitHub repo:

```
cd /root
git clone https://github.com/OnionIoT/oled-twitter-display.git
```

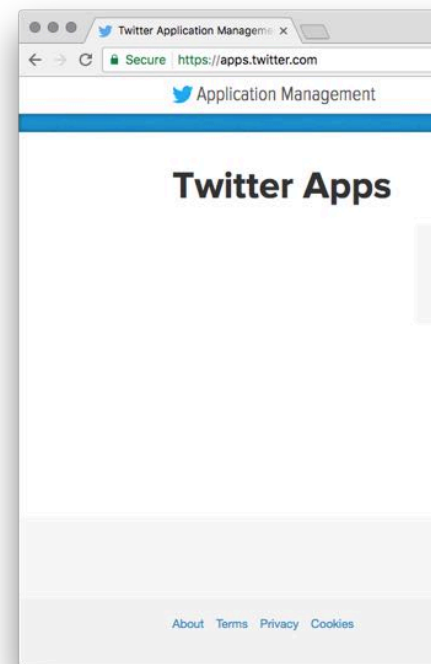
If you're in a hurry, we can download the code directly and avoid installing git. Run the following:

```
mkdir /root/oled-twitter-display
cd /root/oled-twitter-display
wget https://raw.githubusercontent.com/OnionIoT/oled-twitter-display/master/oledTwitterDisplay.py
wget https://raw.githubusercontent.com/OnionIoT/oled-twitter-display/master/config.json
```

We can do this direct download since this GitHub repo is public.

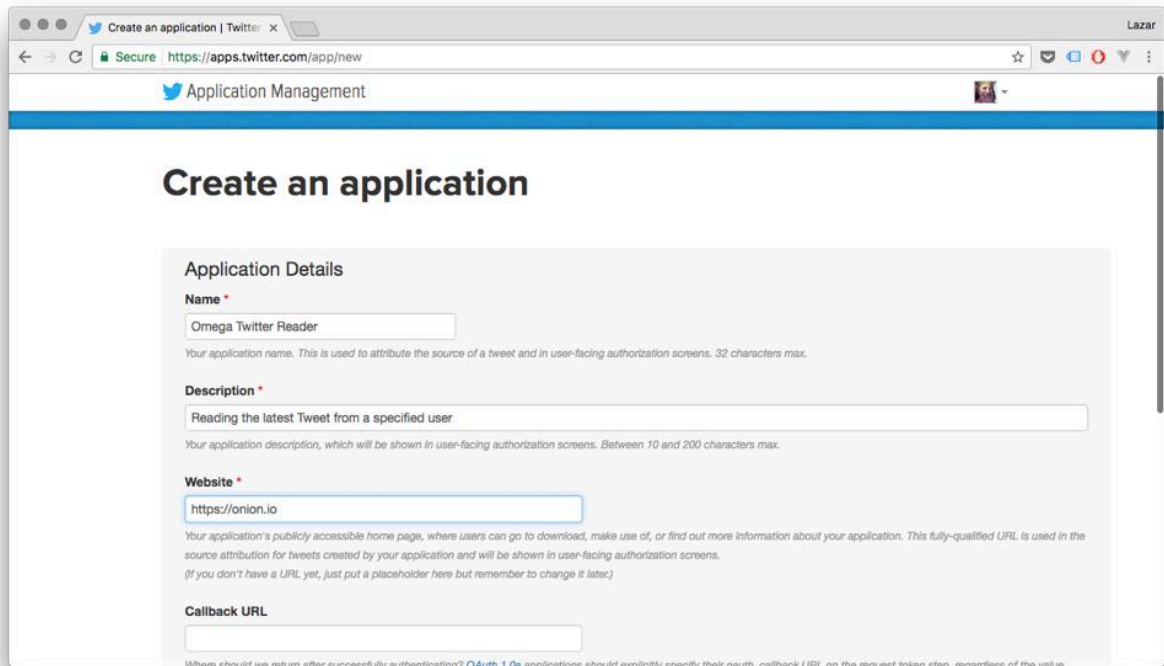
#### 4. Create a Twitter Application

We'll need to create a Twitter Application in order to be able to use Twitter's APIs to grab Tweets. Specifically, our code needs an API Key and API Secret in order to authenticate with Twitter before we can use the APIs:



1. Head over to <https://apps.twitter.com> and sign in with your Twitter handle
2. Fill in the form details for your application. It doesn't really matter what you type in, but a solid name and description goes a long way when you come back to a project after months away from it.

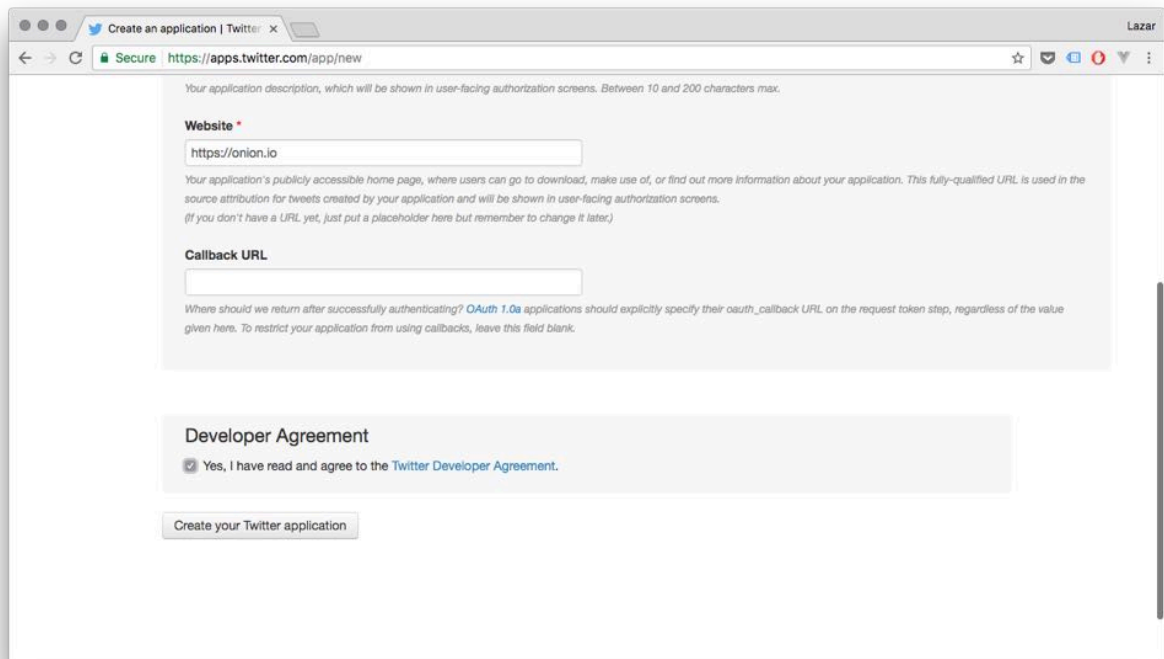




The screenshot shows the 'Create an application' page on the Twitter developer portal. The browser address bar shows 'https://apps.twitter.com/app/new'. The page title is 'Application Management'. The main heading is 'Create an application'. Below this is the 'Application Details' section with the following fields:

- Name \***: Input field containing 'Omega Twitter Reader'. Below it is a note: 'Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.'
- Description \***: Input field containing 'Reading the latest Tweet from a specified user'. Below it is a note: 'Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.'
- Website \***: Input field containing 'https://onion.io'. Below it is a note: 'Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)'
- Callback URL**: An empty input field. Below it is a note: 'Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth\_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.'

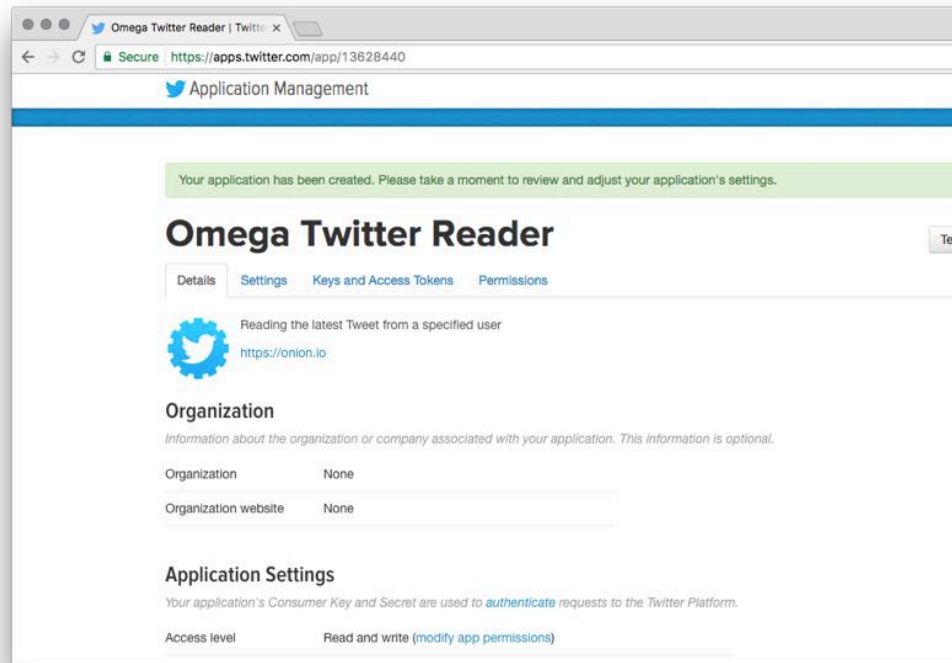
3. Read and agree to the Twitter Developer Agreement and hit Create your Twitter application.



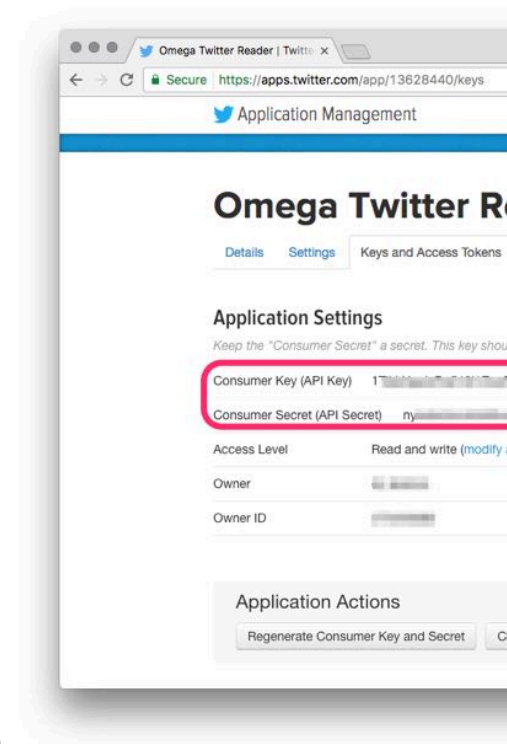
The screenshot shows the 'Create an application' page on the Twitter developer portal, focusing on the 'Developer Agreement' section. The browser address bar shows 'https://apps.twitter.com/app/new'. The page title is 'Create an application | Twitter'. The main heading is 'Create an application'. Below this is the 'Developer Agreement' section with the following fields:

- Developer Agreement**: A checkbox labeled 'Yes, I have read and agree to the [Twitter Developer Agreement](#)' is checked.
- Create your Twitter application**: A button to submit the form.

Note that your Twitter account must have an associated mobile phone number before Twitter will allow you to create an application!



4. Your Application is now created!



5. Head over to the **Keys and Access Tokens** tab to grab the info we need

## 5. Setup and Run the Code

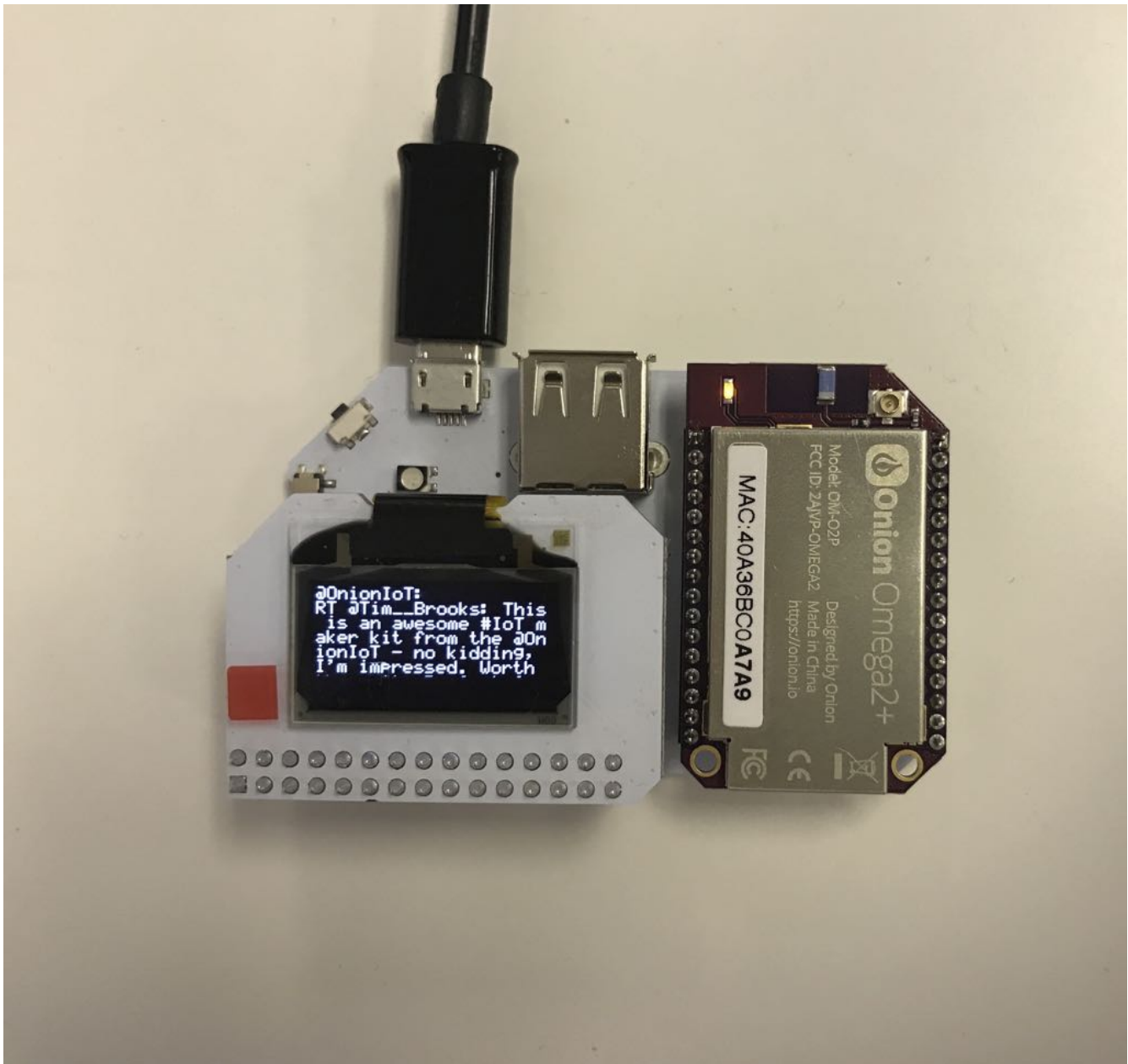
The `config.json` file holds all of the settings for the project. Populate the `authorization` object with the Consumer Key and Consumer Secret from the Twitter app.

And then populate `application.user` with the Twitter handle whose latest tweet you want to be shown

on the OLED Expansion:

```
root@Omega-A7A9:~/oled-twitter-display# cat config.json
{
  "application": {
    "user": "OnionIoT"
  },
  "authorization": {
    "consumerKey": "1",
    "consumerSecret": "n"
  }
}
```

Now run the code: `python oledTwitterDisplay.py`



If you're interested in how the `pyOledExp` code can be used to control the OLED Expansion, take a look at how it's used in [the project code](#) and also check out the [pyOledExp Module documentation](#).

## 6. Automate the Program to Run Periodically

The program will grab and display the latest Tweet, and then promptly exit. We'll use `cron`, a super useful Linux utility, to have the program run periodically.

Enter `crontab -e` to add a task to the `cron` daemon, it will open a file in `vi`, enter in the following:

```
*/5 * * * * python /root/oled-twitter-display/oledTwitterDisplay.py  
#
```

This assumes that your project code is located in `/root/oled-twitter-display`

Now, we'll restart `cron`:

```
/etc/init.d/cron restart
```

And the code will run once every 5 minutes, updating the Tweet shown on your OLED.

Check out the Omega documentation for more info on [using cron](#)

### Code Highlight

All of Twitter's API endpoints require authentication, so that will be the first task of our program. Luckily, Twitter provides [Application-Only Authentication](#), which is why we had to create our own Twitter Application in Step 4 above. Application-Only Authentication is great for a few reasons:

- Your program doesn't include your Twitter username and password
- It allows restricting access, so the application can view/modify only certain things
- The API Key and API Secret (also referred to as Consumer Key and Consumer Secret) can be regenerated if compromised

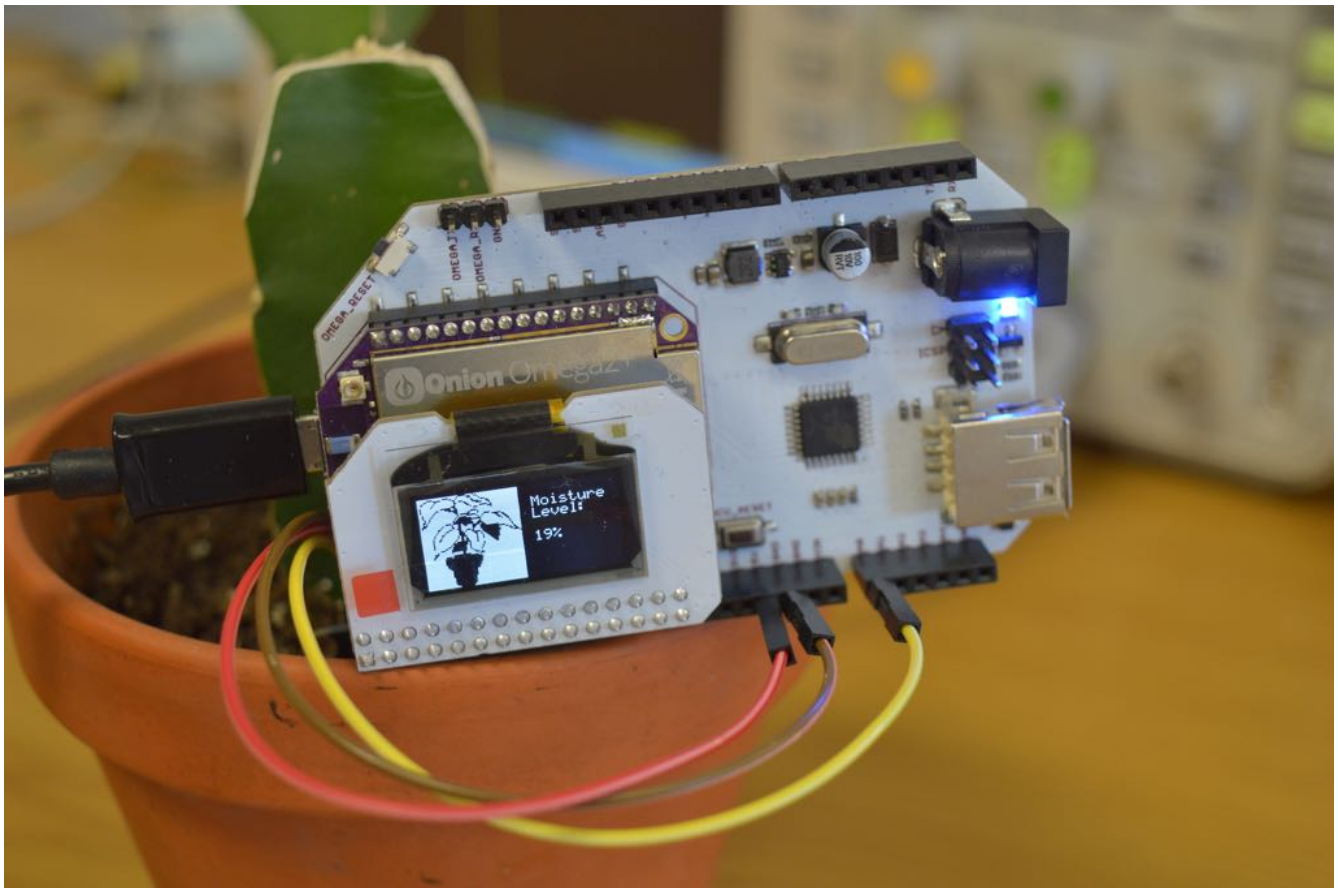
From a high-level, the `twitterApiAuthenticate()` function in the code does an HTTP POST request to `https://api.twitter.com/oauth2/token` with the header containing the base64 encoded Consumer Key and Secret for our application. If the provided Key and Secret are valid, the response will include a Bearer Token. The returned Bearer Token is set to a global variable, and is then used for authorization in the headers of every subsequent request to Twitter's API.

This is a very common authentication practise, see [Twitter's Authentication documentation](#) for more details

## 4 | IoT Projects

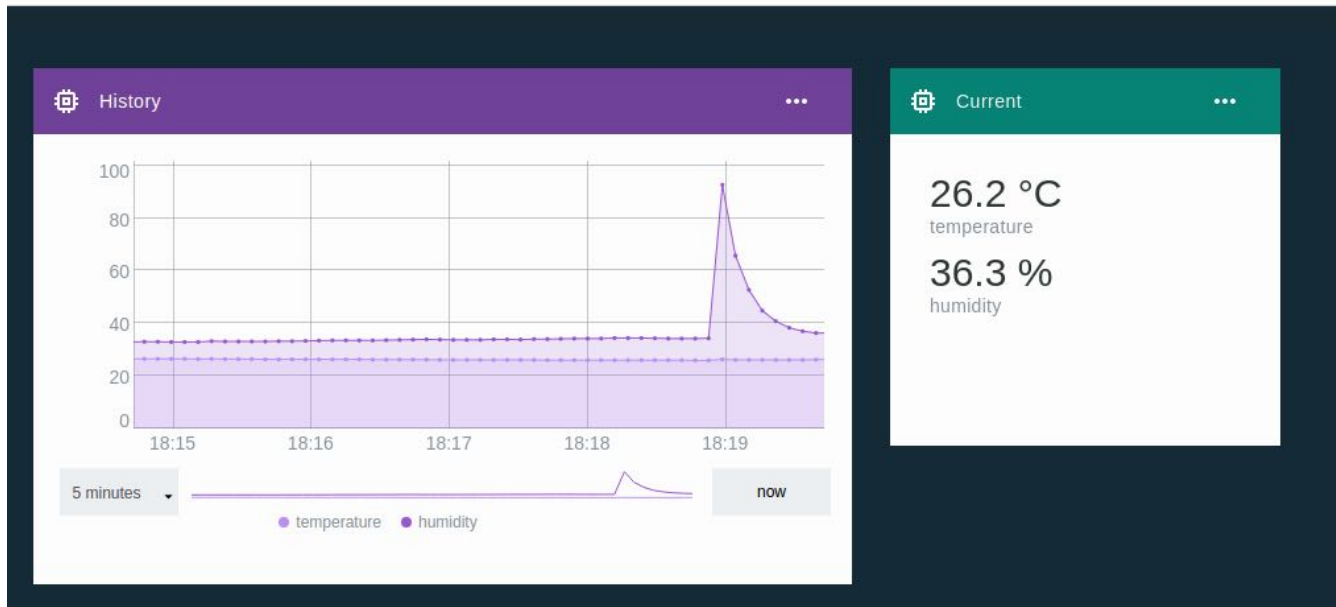
As an IoT computer, the Omega2 really shines when used for IoT projects. The combination of the Omega's small form factor, low power draw, processing and networking capabilities, and the flexibility that comes from running Linux make it ideal for the type of connected and intelligent applications associated with IoT.

Use your Omega to easily add intelligence to everyday objects or existing technology:



and connect them to cloud-based IoT platforms:

## Weather Station



Thereby allowing you to monitor and control your new smart item or device, regardless of your location. This opens the door to creating some really interesting devices in your very own home, school, or workplace.

### Concepts

A highlight of some of the concepts that will be covered in these projects:

- Controlling a webcam
- Generating video from many still images
- Interfacing with an online calendar service
- Acquiring data from analog sensors with the Arduino Dock
- Controlling various attached electronics components and hardware
- Powering the Omega and external, higher voltage components with a single power supply
- Installing additional Python modules
- Adding internet connectivity to existing devices to extend their utility
- Sending data to the IBM Watson IoT platform
- Interfacing with the Losant IoT Platform; sending data and receiving commands
- Using cgi to create quickly create endpoints on the Omega

### Projects

IoT projects for the Omega2 IoT Computer:

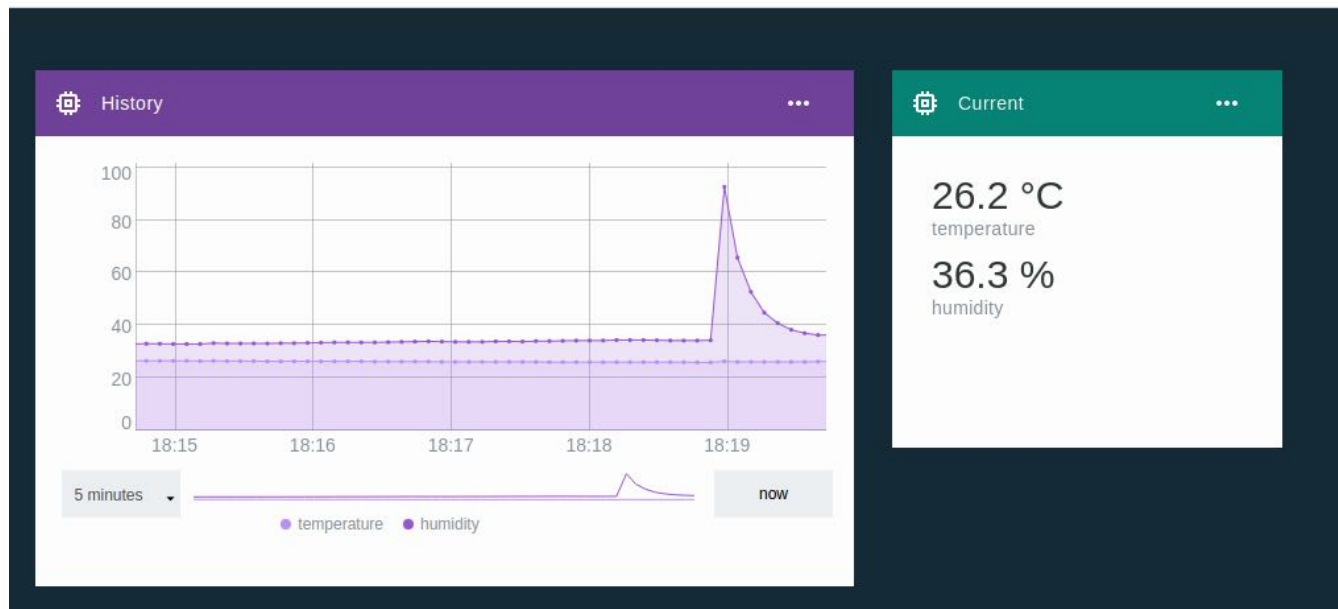
1. **Weather Station**
  - Report collected temperature and humidity data to IBM's Watson IoT Platform
2. **Time-Lapse Camera**
  - Create awesome timelapse videos with your Omega by taking photos at a regular interval and then stitching them together into a video
3. **Alarms based on an Online Calendar**

- Create an event on your online calendar and your Omega will buzz to remind you
4. Thermal Printer
    1. **Your very own Thermal Printer**
      - Wire up a thermal printer and print from your browser
    2. **A Compact Version**
      - Put your soldering skills to use and setup a really compact thermal printer
  5. Smart Plant
    1. **Measure Plant Data**
      - Add smarts to your plant by measuring its soil moisture level
    2. **Visualizing Plant Data**
      - Send plant data to the Losant IoT Platform and check in on your plant from anywhere by looking at the nicely visualized data
    3. **Twitter Alerts**
      - Update the Losant workflow to notify you with a Tweet when your plant needs watering
    4. **Automatic Plant Watering**
      - Add a water pump to your setup and update the Losant workflow to automatically water your plant when it needs watering
    5. **A Single Power Supply**
      - Update the smart plant setup so the Omega and pump can be powered with a single supply
  6. **Temperature-Based Smart Fan**
    - Cool stuff down by spinning up a fan based on temperature readings
  7. IoT Lock
    1. **On your Local Network**
      - Wire up an electric lock and control it from a browser on your local network
    2. **Control the lock with a Tweet**
      - Add to the IoT lock to allow authorized Twitter users to control the lock with specific hashtags in a Tweet

## Weather Station

This project will show you how to collect temperature and humidity data from a DHT22 sensor and send it to [IBM's Watson IoT Platform](#), where you can view it on the web!

## Weather Station



### Overview

**Skill Level:** Intermediate

**Time Required:** 40 minutes

We'll be using the Arduino Dock to read the temperature and humidity data from the DHT22 sensor since the on-board microcontroller will make this a breeze. On the Omega side, we'll use the [Python pySerial module](#) to periodically send commands to the Arduino Dock and trigger a response with the sensor data. The Omega will then read that data back and send it to IBM Watson, where you can visualize the data in real time! We will then automate the project to run automatically when the Omega is turned on.

The device code can be found in Onion's [iot-weather-station repo on GitHub](#), while the Arduino Dock sketch can be found in the Examples of the [Onion Arduino Library](#).

### Ingredients

- Onion [Omega2+](#)
  - or [Omega2](#) standard booting from a USB storage device with at least 16 MB of free memory; see [Booting From External Storage](#)
  - For convenience, this tutorial will assume you are using an Omega2+.
- Onion [Arduino Dock 2](#)
- [Breadboard](#)
- [DHT22 temperature + humidity sensor](#)
- [10kΩ resistor](#)
- [M-M jumper wires](#)
- (Optional) [M-F jumper wires](#)



## Step-by-Step

Follow these instructions to setup the Weather Station project on your very own Omega!

### 1. Prepare

You'll have to have an Omega2+ ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

**Note 1:** The Arduino Dock does not have a USB to serial converter chip, so [all connections to the Omega's command line must be done over SSH](#).

**Note 2:** The Omega's firmware along with this project and its dependencies will require approximately 18 MB of storage space. If you wish to install apps on the Console such as the Editor, we recommend [booting the filesystem from an external SD card or USB drive](#).

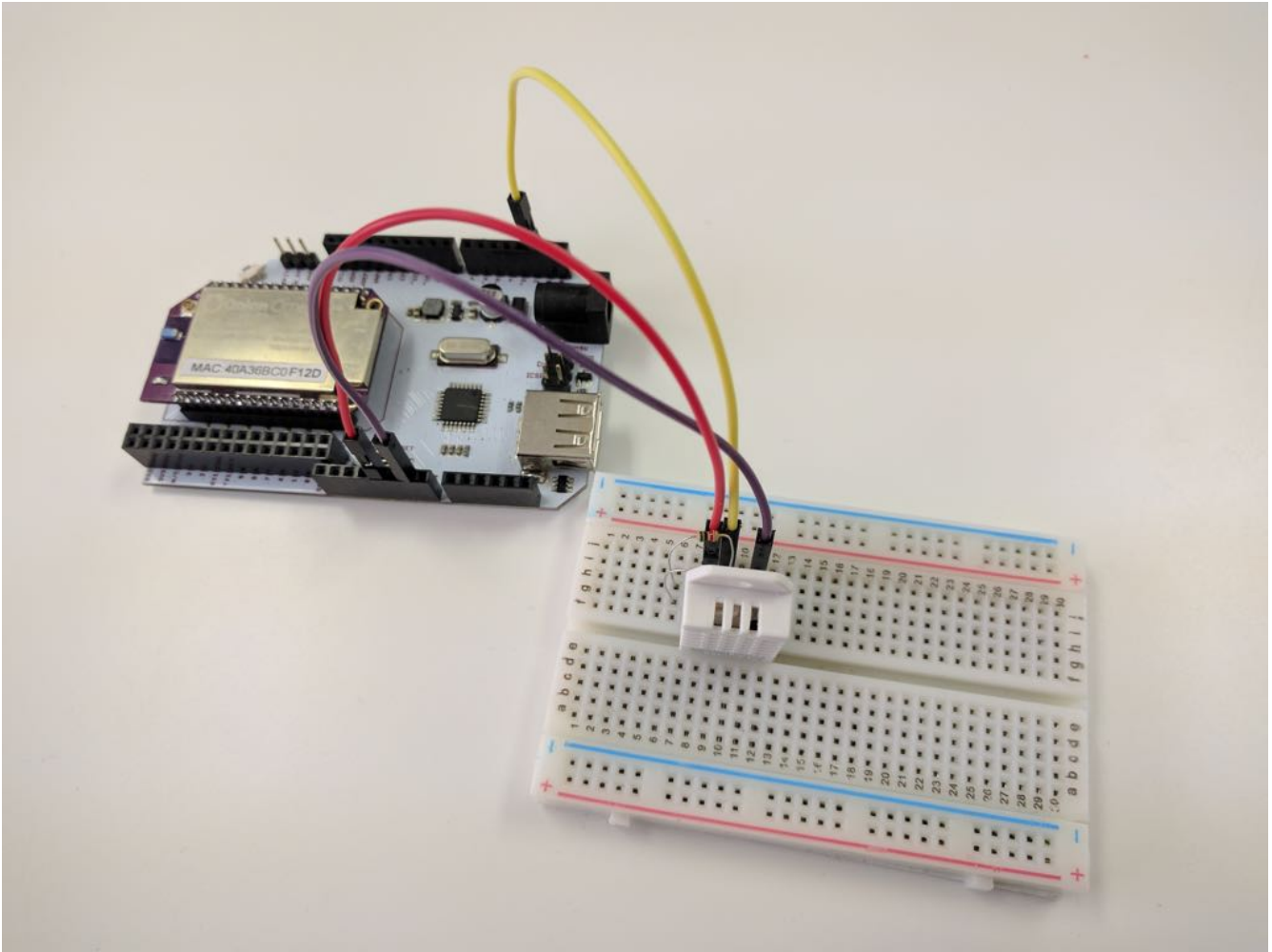
After completing the first time setup, follow the steps in the [Arduino Dock guide](#) to prepare it for flashing Arduino sketches.

### 2. Wire Up the Sensor

We will treat the side of the DHT sensor with the holes as the **front**.

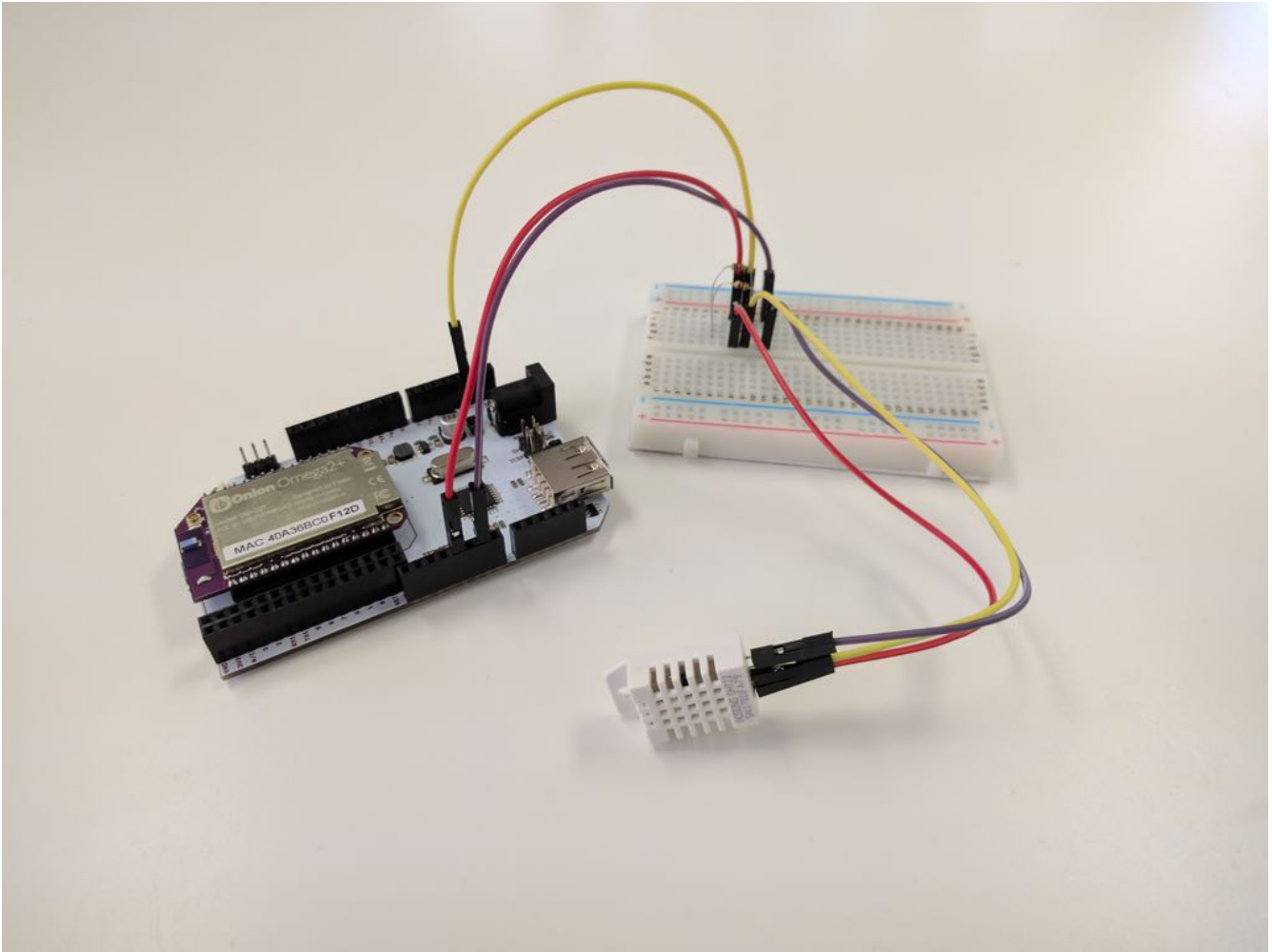
1. Insert the DHT sensor into the breadboard with the front facing the middle gap.
  - The pins will be numbered 1-4 from left to right.
2. Connect Vdd (pin 1) to the Arduino Dock's 5V pin.
3. Connect DATA (pin 2) to the Arduino Dock's digital pin 2.
4. Connect GND (pin 4) to one of the Arduino's GND pins.
  - This is not a typo. Pin 3 is unused on this DHT sensor!
5. Connect the 10k $\Omega$  resistor across Vdd and DATA.

Your setup should look something like this:



In the above setup, two  $5.1\text{k}\Omega$  resistors were used in series to achieve the  $10\text{k}\Omega$  pullup.

**Optional** - Remove the DHT sensor from the breadboard, and use the M-F jumper wires to connect the pins back to the breadboard. This is so you can easily move the sensor around!



### 3. Download the Project Code on the Omega

The code for this project can be found in Onion's [iot-weather-station repo](#) on GitHub. We'll first need to [connect to the Omega's command line](#) and install the packages we need for git:

```
opkg update
opkg install git git-http ca-bundle
```

Then we'll [use git to download the project code to your Omega](#):

```
cd /root
git clone https://github.com/OnionIoT/iot-weather-station.git
```

Now enter the repo directory and run `install.sh` to install the required packages and dependencies:

```
cd iot-weather-station
sh install.sh
```

This may take several minutes, go grab a drink or a quick snack!

The `install.sh` script will install Python and the additional modules needed to communicate with the Watson IoT platform. Specifically, it will install Python and PIP, the Python package manager. Then, it will use PIP to install IBM's `ibmiotf` module and all of its dependencies.

#### 4. Find your Omega's MAC Address

Then get the Omega's MAC address for use with Watson by running the `watsonHelper.py` script:

```
python watsonHelper.py
```

You will see output that looks something like this:

```
root@Omega-2729:~/iot-weather-station# python watsonHelper.py
=====
Device ID:
42a36b002729
=====
Loaded Device ID into device.cfg.
Enter this Device ID on the IBM Bluemix website when registering this device on the Watson IoT platform.
Don't forget to add your Organization ID, Device Type, and Authorization Token to the device.cfg file!
```

Copy or write down the MAC address underneath “Device ID” for later.

#### 5. Arduino IDE Setup

If you don't already have it, install the [Arduino IDE](#) on your computer. Once you're in the Arduino IDE, install the [Adafruit Unified Sensor](#) and [DHT sensor library](#) libraries from the Library Manager by following [this guide](#) on the Arduino website.

Then you'll need to install the [Onion Arduino Library](#) by doing the following:

1. In your web browser, download the [Onion Arduino Library ZIP file](#).
2. Install the ZIP library by following the instructions in the [Arduino Library Installation guide](#).
3. Restart your Arduino IDE to reload the library.

Finally, follow [our Arduino Dock setup instructions](#) to setup the Arduino IDE to wirelessly flash the Arduino Dock 2.

#### 6. Flash the Arduino Dock's Microcontroller

Flash the weather station sketch to the Arduino Dock by doing the following:

1. Click on `File > Examples > Onion > weatherStation` to open the weather station sketch.
2. Flash it to the Arduino Dock by following the instructions in the [Arduino Dock guide](#).

This sketch will read the temperature and humidity measurements from the DHT22 sensor and will transmit the value via serial of the correct command is received from the other end. So the Omega will have to issue a command through UART1, in this case, the command is just the `r` character, and it will then receive a JSON-formatted string of the sensor data as a response.

#### 7. Setup the Omega on the IBM Watson IoT Platform

We will be using [IBM's guide on registering devices in Watson](#) as a reference for this section. Open the link in your web browser and refer to the *additional* information that we have provided for each step below.

If you're having difficulties in this section, follow these two developer recipes to become familiar with the Watson web interface:

- [Connect an Onion Omega2 to IBM Watson IoT Platform](#)
- [How to Register Devices in IBM Watson IoT Platform](#)

### Step 1 - Introduction

- You can register for an [IBM Bluemix account here](#).

### Step 2 - Create IBM Watson IoT Platform Organization

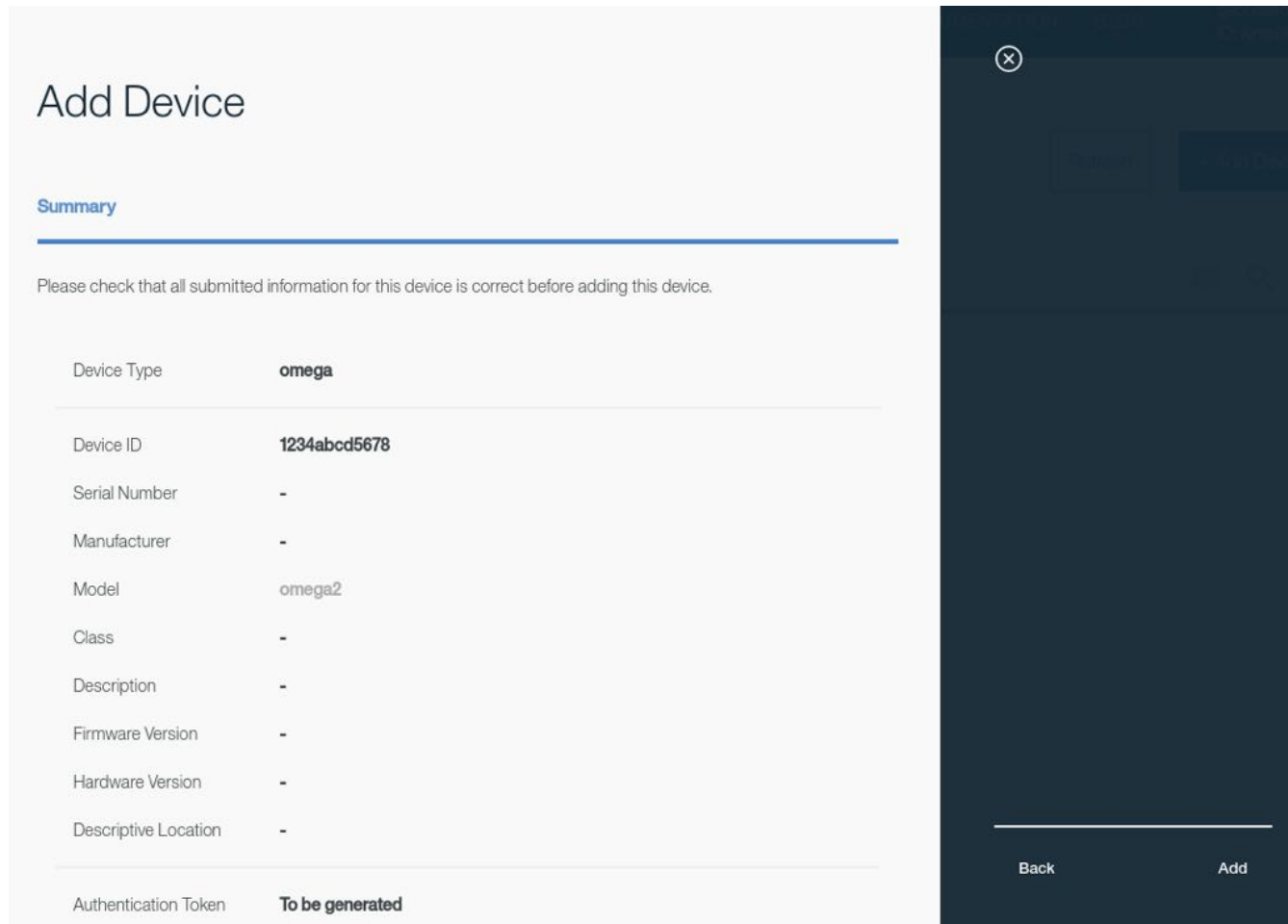
- You can call the service “Onion IoT”, “Watson IoT”, or whatever you like.

### Step 3 - Create Device Type

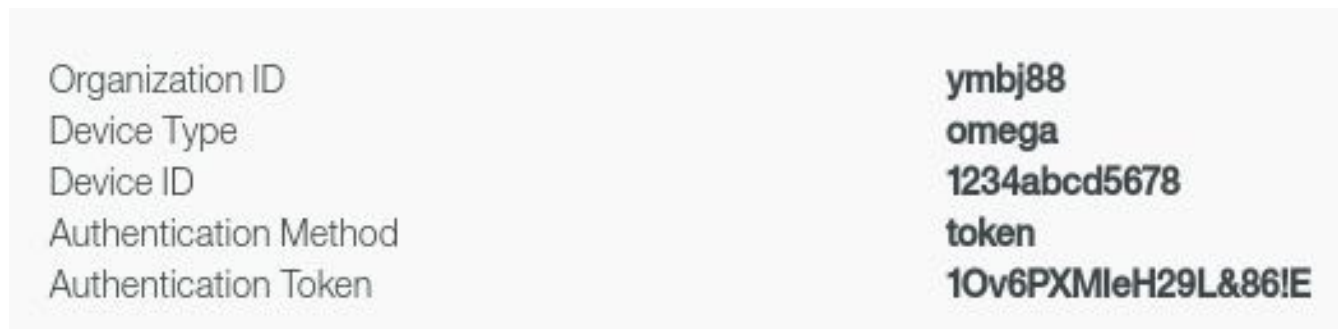
- Call the device type `omega`.
  - The description can be something like “Onion Omega IoT board”.
- In the *Define Template* substep on the Watson website, check off only the *Model* attribute.
- In *Submit Information*, enter `omega2` for the *Model*.
  - New devices by default will have this value for their *Model* attribute unless you specify something else.
- You can leave *Metadata* blank.

### Step 4 - Add Device in IBM Watson IoT Platform

- When adding a device, choose the `omega` device type we just created.
- In the *Device Info* substep, enter the Device ID that we got from the `watsonHelper.py` helper script a few steps back.
- You can leave the *Model* field blank and it will automatically fill it in with `omega2`.
- You can leave the *Metadata* field blank.
- In *Security*, we recommend letting Watson automatically generate an authentication token for you. Click on *Next* without entering anything.
- In the *Summary* substep, review that your information is correct, then click *Add*.



You should see a card containing your Organization ID, Device ID, and more. Don't close this card until you've recorded the token somewhere, because there's no way to view the authentication token for this device again! Take a look at the sample card below:



On the Omega, open the `device.cfg` file for editing and replace the placeholders in ALLCAPS with the information in the fields above like so:

Watson Website	device.cfg
Organization ID	YOURORG
Device ID	YOURDEVICEID
Authentication Token	YOURTOKEN

## Remaining steps

“Step 5 - Generate API Keys” and onwards are not necessary for this project.

## 8. Set Up Visualization Boards and Cards on Watson

Follow the steps in [IBM's guide to configuring cards in Watson](#) with the *additional* information we have provided below:

### Step 2 - Overview to Boards & Cards

- You can make your own board to show the collected data from the sensor. In this example, we've called it `Weather Station`.

### Step 3 - Realtime Data Visualization

- First create a line chart card according to the guide.
- When connecting data sets, set `weather` as the Event.
- Create a data set for temperature following the example below:

# Edit Line chart Card

Connect data set

temperature

Event

weather

Property

temperature

Name

temperature

Type	Unit
Number	°C

Min	Max
-100	200

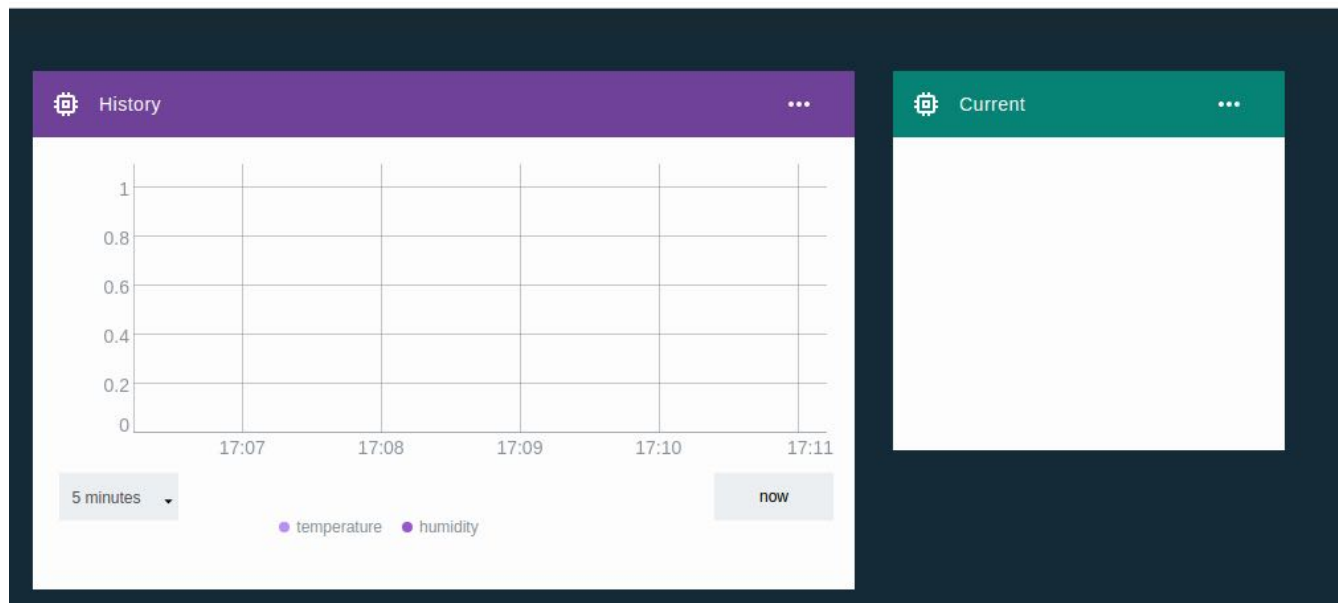
- Repeat for humidity by replacing all instances of **temperature** with **humidity**.
- We recommend using an XL line chart size to be able to see enough data over time.
- Set the title to “History”

You can also add a Value card to clearly display the most recent measurement values.

- When adding the card, click on “Value” as the type.
- Use a M size chart to display both temperature and humidity at the same time.
- You can set the title of this card to “Current”



## Weather Station



Now you've got visualization set up on Watson!

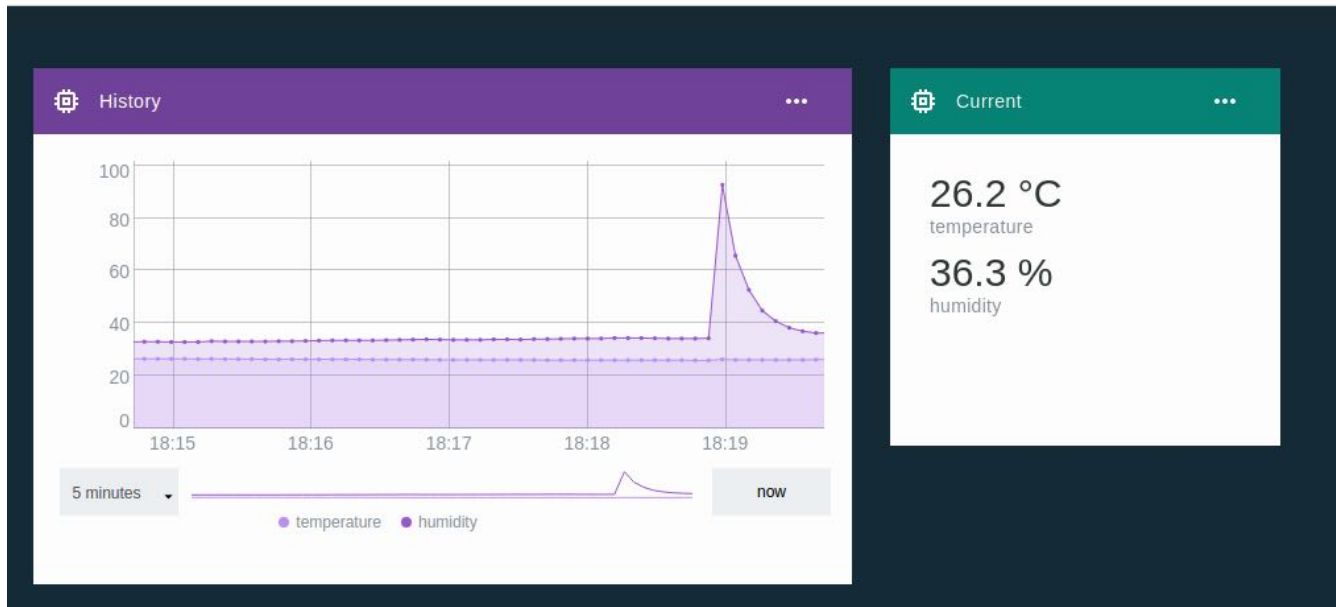
### 9. Running the Weather Station Project

On the Omega, navigate to the `iot-weather-station` directory and run the `main.py` file:

```
python main.py
```

You should see messages being published from the command line, and new data points in your Watson dashboard! Try placing the sensor near sources of cold or hot air, or try breathing over it to change the relative humidity and see what happens on the dashboard.

## Weather Station



### 10. Run the Program on Boot

We can automate this project to run when the Omega is turned on, and we can also make it run in the background so you can use the Omega for other things while it's running! To do this, we'll place a script in `/etc/init.d`.

In the repo folder, make the `weather-station` file executable, copy it to `/etc/init.d`, then enable it to run on boot:

```
chmod +x init.d/weather-station
cp init.d/weather-station /etc/init.d
/etc/init.d/weather-station enable
```

Reboot the Omega, and you will see the dashboard automatically being populated with data while the command line is available for you to use!

The `/etc/init.d/weather-station` script registers the Weather Station Python script as a service with `procd`, the process management daemon of the system. `procd` then ensures that the process gets started at boot and continues to run until the service is disabled.

### Code Highlight

This project makes use of two main interfaces: the Arduino Dock and the IBM Watson IoT platform.

The Arduino Dock sketch is set to read data from the DHT sensor only when an `r` character is sent over the serial bus.

```
if (Serial.available() > 0) {
  // read the input
  int inByte = Serial.read();
```

```
delay(500); // small delay before responding

// respond only if correct command is received
if ((char)inByte == 'r') {
  responder();
  delay(delayMS);
}
}
```

It then sends a response in JSON.

```
void responder() {
  // read the weather sensor
  if (getWeather()) {
    String temperature = String(sensorResponse.temperature);
    String humidity = String(sensorResponse.humidity);

    // encode output to JSON
    String response = "{\"temperature\": " + temperature + ", \"humidity\": " + humidity + "}";
    Serial.println(response);
  }
  else {
    // send false
    Serial.println("false");
  }
}
```

JSON was chosen as it is a widely used data serialization format, and JSON parsers exist in most programming languages. You can use this example to create your own Arduino Dock apps that can collect data and send it back to the Omega, which can then run any programming language you like!

The IBM Watson IoT Python library is also useful for sending data to Watson from the Omega. You can reuse the `device.cfg` and `watsonHelper.py` files in your next IoT projects!

## Time-Lapse Camera

Using a webcam connected to the Omega, we can take photos over time and string them together to make a video of your scene!



## Overview

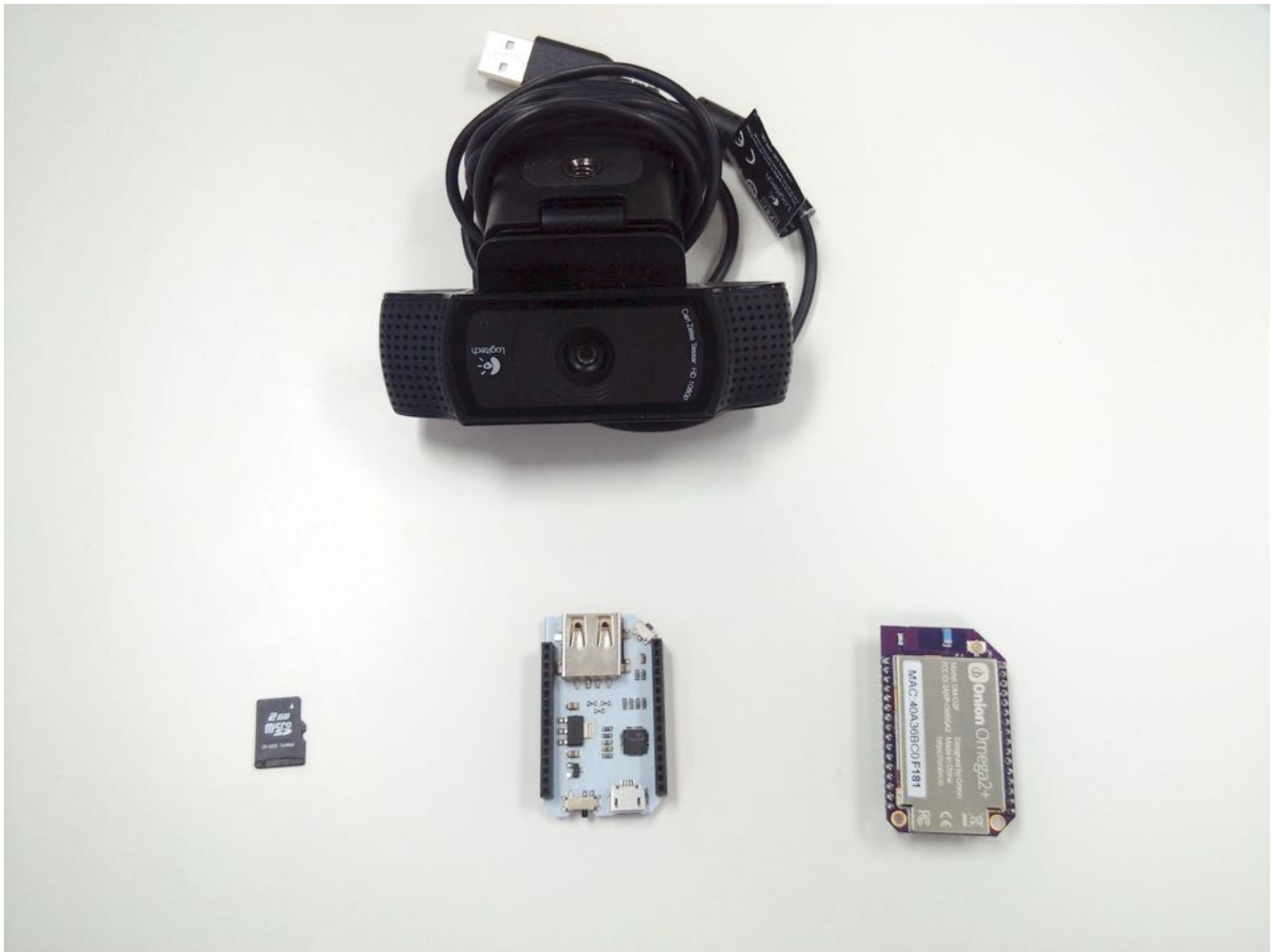
**Skill Level:** Intermediate

**Time Required:** 30 minutes

In this project, we'll use the `fswebcam` utility and `cron` to capture images from the webcam at timed intervals, then we'll convert it all to a video using `ffmpeg`. To keep the amount of typing to a minimum, we will create shell scripts to do the work for us!

## Ingredients

- Orion [Omega2+](#)
- Any Orion Dock with a USB host connector: [Expansion Dock](#), [Power Dock](#), [Mini Dock](#), [Arduino Dock 2](#)
- [Micro SD card](#)
- [USB Camera](#)



## Step-by-Step

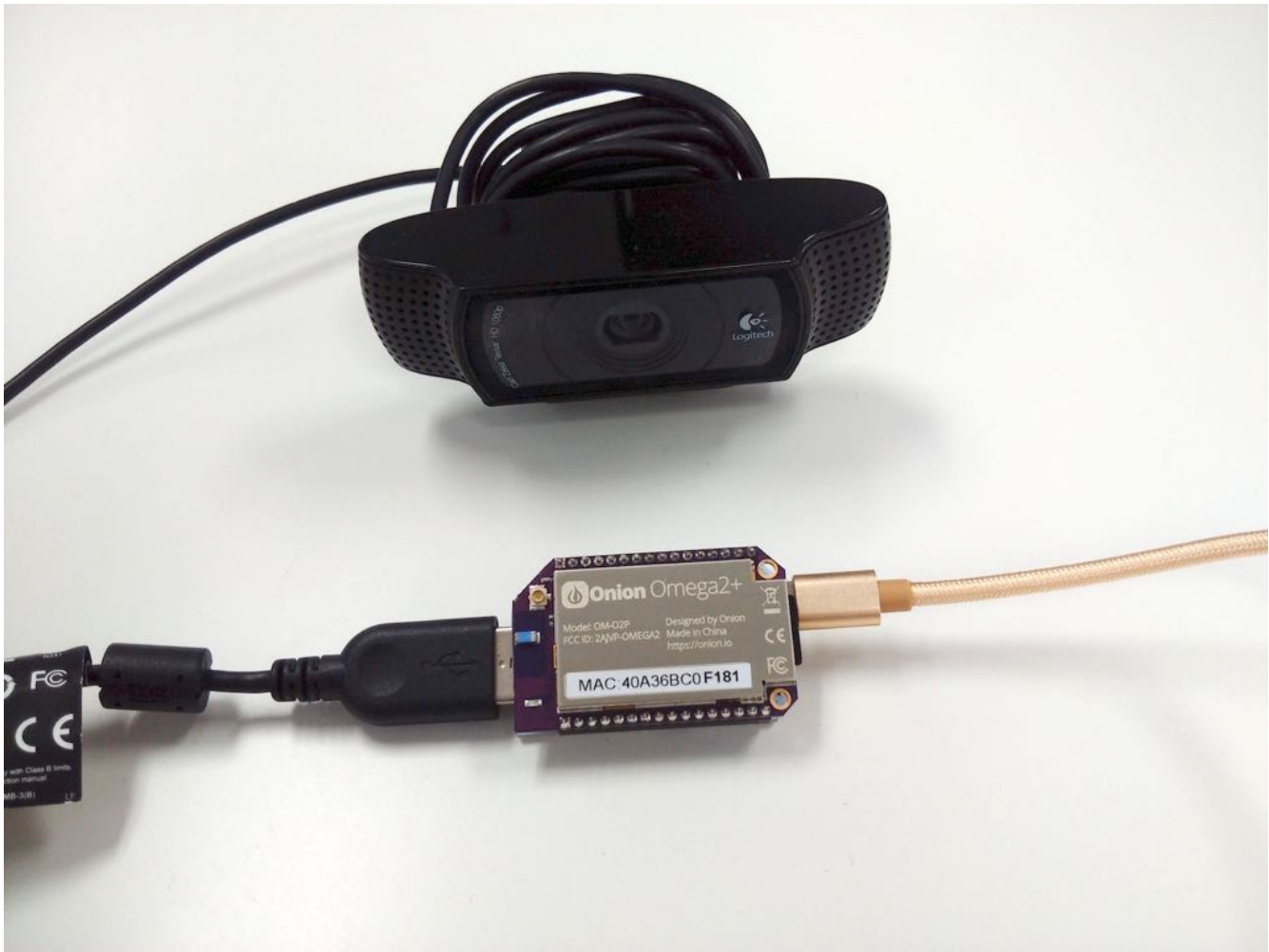
Follow these instructions create sweet time-lapse videos with a webcam on your Omega2+!

All of the code can be found in Onion's [timelapse-camera repo on GitHub](#).

### 1. Get the Hardware Ready

You'll have to have an Omega2+ ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

Insert the MicroSD card, plug in the USB webcam, connect the Omega to power, and we're ready to start.



## 2. Prepare an External Storage Device

The pictures taken with the webcam would fill up the Omega's built-in storage quite quickly. To make sure we have enough space for our pictures, we'll need a MicroSD card to store it all.

Connect to the Omega's Command line and use `df -h` to check the storage on the Omega - and make sure we're using the correct device.

```
root@Omega-F181:~# df -h
Filesystem      Size      Used Available Use% Mounted on
/dev/root        5.5M       5.5M          0 100% /rom
tmpfs            61.4M      84.0K      61.3M   0% /tmp
/dev/mtdblock6  25.1M     756.0K     24.3M   3% /overlay
overlayfs:/overlay 25.1M     756.0K     24.3M   3% /
tmpfs            512.0K          0     512.0K   0% /dev
/dev/mmcblk0p4  1.8G     652.5M     1.2G   35% /tmp/run/mountd/mmcblk0p4
```

Here, the last line shows the MicroSD card is successfully mounted under `/tmp/run/mountd/mmcblk0p4`

Now that we're sure where the MicroSD card is, let's create a soft-link to it for easy access:

```
ln -s /tmp/run/mountd/mmcblk0p1 /root/sd
mkdir sd/timelapse
```

A soft-link is the Linux equivalent of a shortcut. It is just a file that contains a reference to another file or directory. In this case, `/root/sd` contains a reference to `/tmp/run/mountd/mmcblk0p1`

### 3. Install Webcam Software

To get the webcam to take pictures, we'll need the software.

The package we use here isn't included in the Onion package repo so we'll have to use the LEDE package repo to get it.

Open up the source list like so:

```
vim /etc/opkg/distfeeds.conf
```

And comment/uncomment the lines so they look like this:

```
src/gz reboot_core http://downloads.lede-project.org/snapshots/targets/ramips/mt7688/packages
src/gz reboot_base http://downloads.lede-project.org/snapshots/packages/mipsel_24kc/base
# src/gz reboot_onion http://repo.onion.io/omega2/packages
## src/gz reboot_luci http://downloads.lede-project.org/snapshots/packages/mipsel_24kc/luci
src/gz reboot_packages http://downloads.lede-project.org/snapshots/packages/mipsel_24kc/packages
## src/gz reboot_routing http://downloads.lede-project.org/snapshots/packages/mipsel_24kc/routing
## src/gz reboot_telephony http://downloads.lede-project.org/snapshots/packages/mipsel_24kc/telep
# src/gz omega2_core http://repo.onion.io/omega2/packages/core
# src/gz omega2_base http://repo.onion.io/omega2/packages/base
# src/gz omega2_packages http://repo.onion.io/omega2/packages/packages
src/gz omega2_onion http://repo.onion.io/omega2/packages/onion
```

A complete guide on how to do so can be found in our guide on [Using Opkg](#).

Next, we'll run `opkg update` so the changes take effect and then we can go ahead and install the `fswebcam` package:

```
opkg update
opkg install fswebcam
```

### 4. Take a Photo

The `fswebcam` utility lets us take a photo with a webcam with the following command:

```
fswebcam --no-banner -r 1280x720 `date +%Y-%m-%d_%H%M%S` .jpg
```

If you're wondering how we have `date +%Y-%m-%d_%H%M%S/` as the output name of the file but it ends up being the date and time, it's through the wonders of command substitution! The command inside the backticks, ```, will be executed and replaced with the output of the command, and then the rest of the `fswebcam` command will be executed. Try running `date +%Y-%m-%d_%H%M%S` on the command line, you'll see that it outputs the time and date in the same format as the images get named.

### 5. Script it to save our fingers

To save us from typing that out every time, we'll write a short script.

Save the code below to `/root/snapshot.sh`:

```
#!/bin/sh

fswebcam --no-banner -r 1280x720 /root/sd/timelapse/`date +"%Y-%m-%d_%H%M"` .jpg
```

Make the file executable with `chmod`:

```
cd /root
chmod +x snapshot.sh
```

## 6. Automate it

We'll use `cron` to automate the capture image script. `cron` is a utility that will repeatedly execute some command at set times. Here's a quick overview of how `cron`'s syntax works:

```
# * * * * * command to execute
#
#
#
#           day of week (0 - 7) (0 to 6 are Sunday to Saturday, or use names; 7 is Sunday, the sa
#           month (1 - 12)
#           day of month (1 - 31)
#           hour (0 - 23)
#           min (0 - 59)
```

To set up a new task - called a cronjob - we'll need to edit `cron`'s configuration file with:

```
crontab -e
```

This opens an editor (by default, `vim`) to edit the file.

To create a cronjob, append the following to the `crontab`:

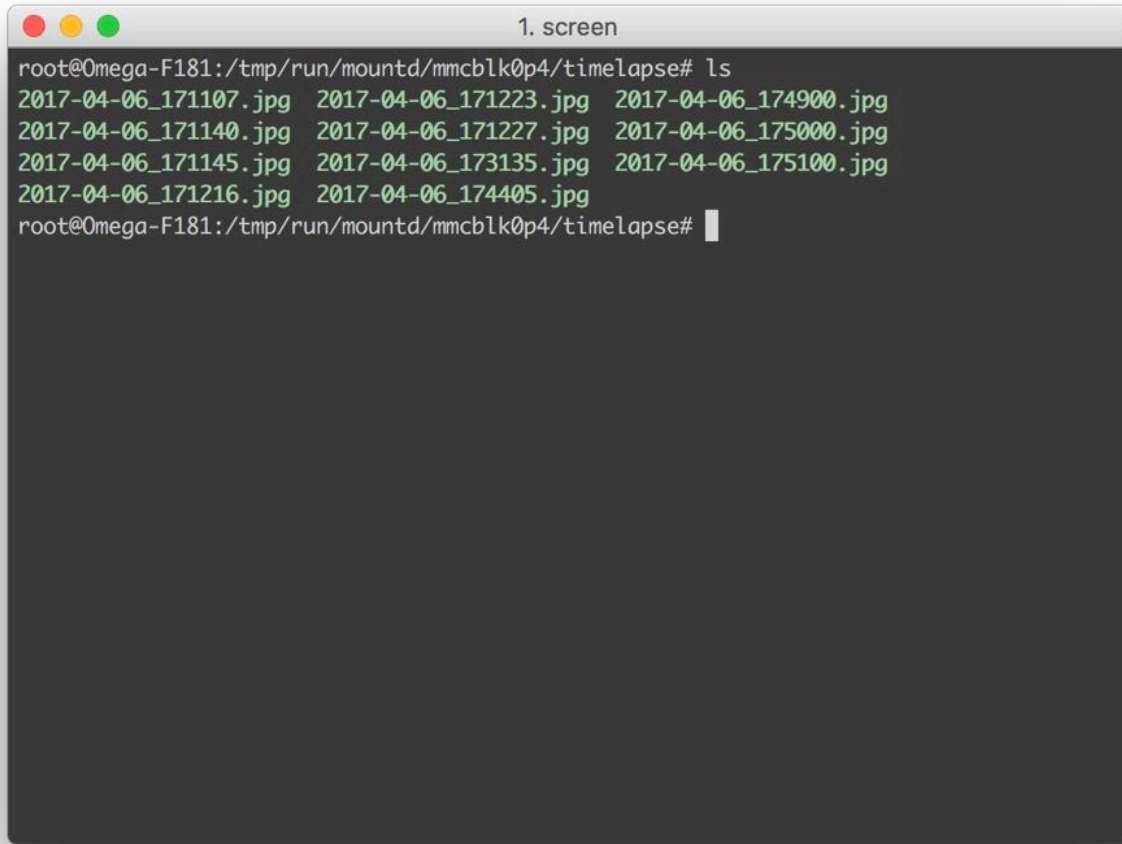
```
#take snapshot once a minute
* * * * * /root/snapshot.sh
#
```

For our new cronjob to take effect, restart the `cron` daemon with:

```
/etc/init.d/cron restart
```

Now wait a bit, if new files are created each minute, then we're good to move on!



A terminal window titled "1. screen" showing the output of the 'ls' command. The terminal prompt is "root@Omega-F181:/tmp/run/mountd/mmcblk0p4/timelapse#". The output lists 10 JPEG files with timestamps: 2017-04-06\_171107.jpg, 2017-04-06\_171140.jpg, 2017-04-06\_171145.jpg, 2017-04-06\_171216.jpg, 2017-04-06\_171223.jpg, 2017-04-06\_171227.jpg, 2017-04-06\_173135.jpg, 2017-04-06\_174405.jpg, 2017-04-06\_174900.jpg, and 2017-04-06\_175100.jpg. The terminal prompt is followed by a cursor.

```
root@Omega-F181:/tmp/run/mountd/mmcblk0p4/timelapse# ls
2017-04-06_171107.jpg 2017-04-06_171223.jpg 2017-04-06_174900.jpg
2017-04-06_171140.jpg 2017-04-06_171227.jpg 2017-04-06_175000.jpg
2017-04-06_171145.jpg 2017-04-06_173135.jpg 2017-04-06_175100.jpg
2017-04-06_171216.jpg 2017-04-06_174405.jpg
root@Omega-F181:/tmp/run/mountd/mmcblk0p4/timelapse#
```

Check out the Omega documentation for more info on [using cron](#)

## 7. Set up FFmpeg

Once cron starts working and we have some images from the webcam, we'll want to create our very first timelapse video. To do that we'll be using the FFmpeg program.

First, grab the package:

```
opkg update
opkg install ffmpeg
```

Next, we'll use a script to rename the photos and create the video. Renaming first makes sending the files to FFmpeg a lot easier.

Copy the code below and save it to `/root/makevideo.sh`.

```
#!/bin/sh

cd /root/sd/timelapse

# rename images to be sequential (according to time)
```

```

a=1
for i in `ls -tv *.jpg`; do
  new=$(printf "%d.jpg" "$a") #04 pad to length of 4
  mv -- "$i" "$new"
  let a=a+1
done

# call ffmpeg to create our video
ffmpeg -r 6 -start_number 1 -i %d.jpg -s 1280x720 -q:v 1 `date +"%Y-%m-%d_%H%M%S"` .mp4

```

Here's the options we used for ffmpeg and what they do:

Option	Effect	Accepted format	Our input
-r	Frame rate of the output	integer	6
-start_number	Number the sequence of files start with	integer	1
-i	Input files	List of file names	%d.jpg
-s	Resolution of the output	Width x Length	1280x720
-q:v	Quality of video	integer between 1~31 (1 is highest quality)	1

## 8. Making the Timelapse Video

Almost there!

Let's make the script executable so we don't have to call `sh` every time we want to make the video:

```
chmod +x /root/makevideo.sh
```

To make the video, we directly call the script:

```
/root/makevideo.sh
```

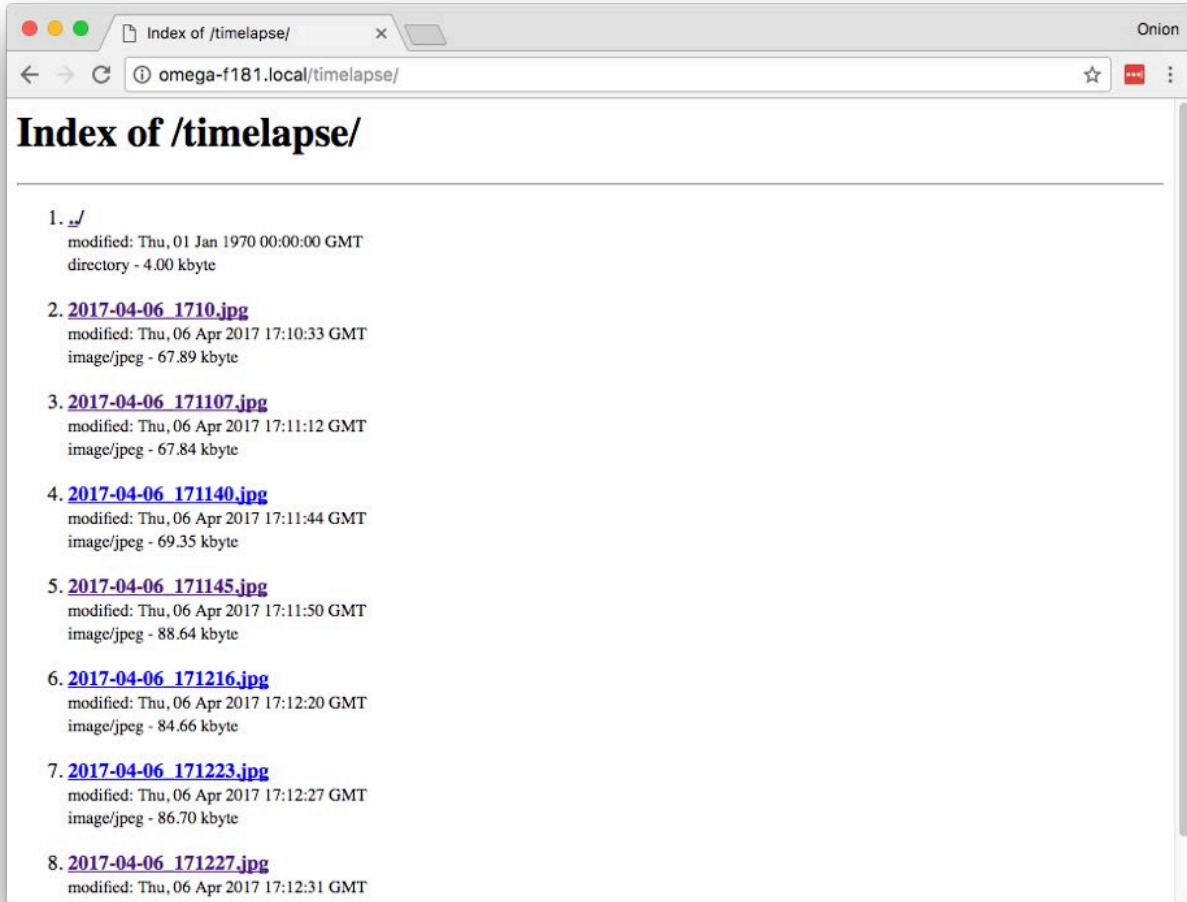
Voila! A time-lapsed video of the photos your webcam was taking!

### Bonus points: Accessing the Images through a Browser

Link the images to the web directory of the Omega, and `uhttpd` will automatically serve them up.

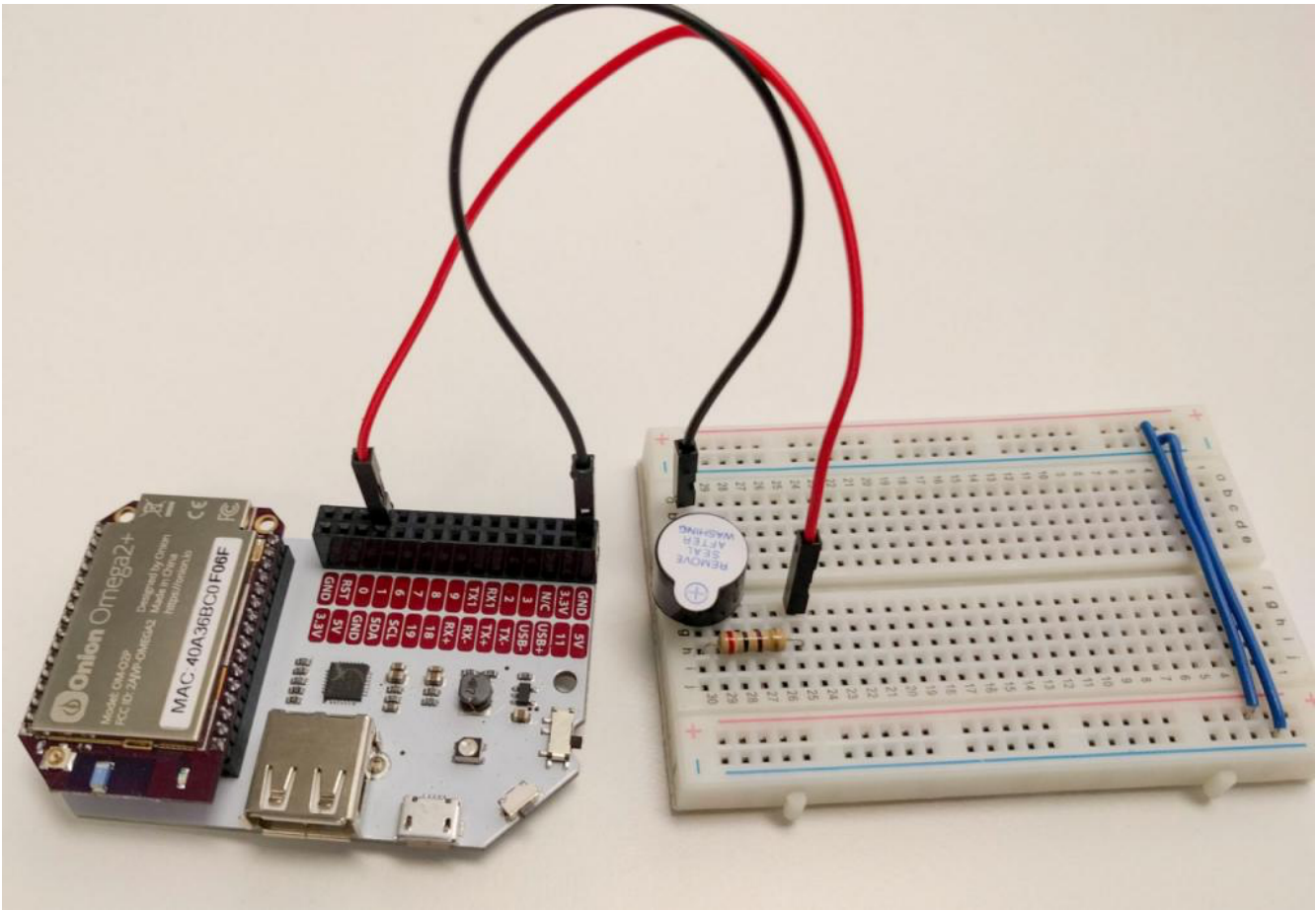
```
ln -s /tmp/run/mountd/mmcblk0p4/timelapse /www/timelapse
```

To get to our images, go to <http://omega-abcd/timelapse> in your browser. Remember to replace `omega-abcd` with your Omega's unique name!



## Alarms based on an Online Calendar

This project will create a real-world alarm clock that can be setup from an online calendar of your choice. Just create a calendar event with a specific word in it, and your Omega will act as an alarm based on the event's time and date.



## Overview

**Skill Level:** Intermediate

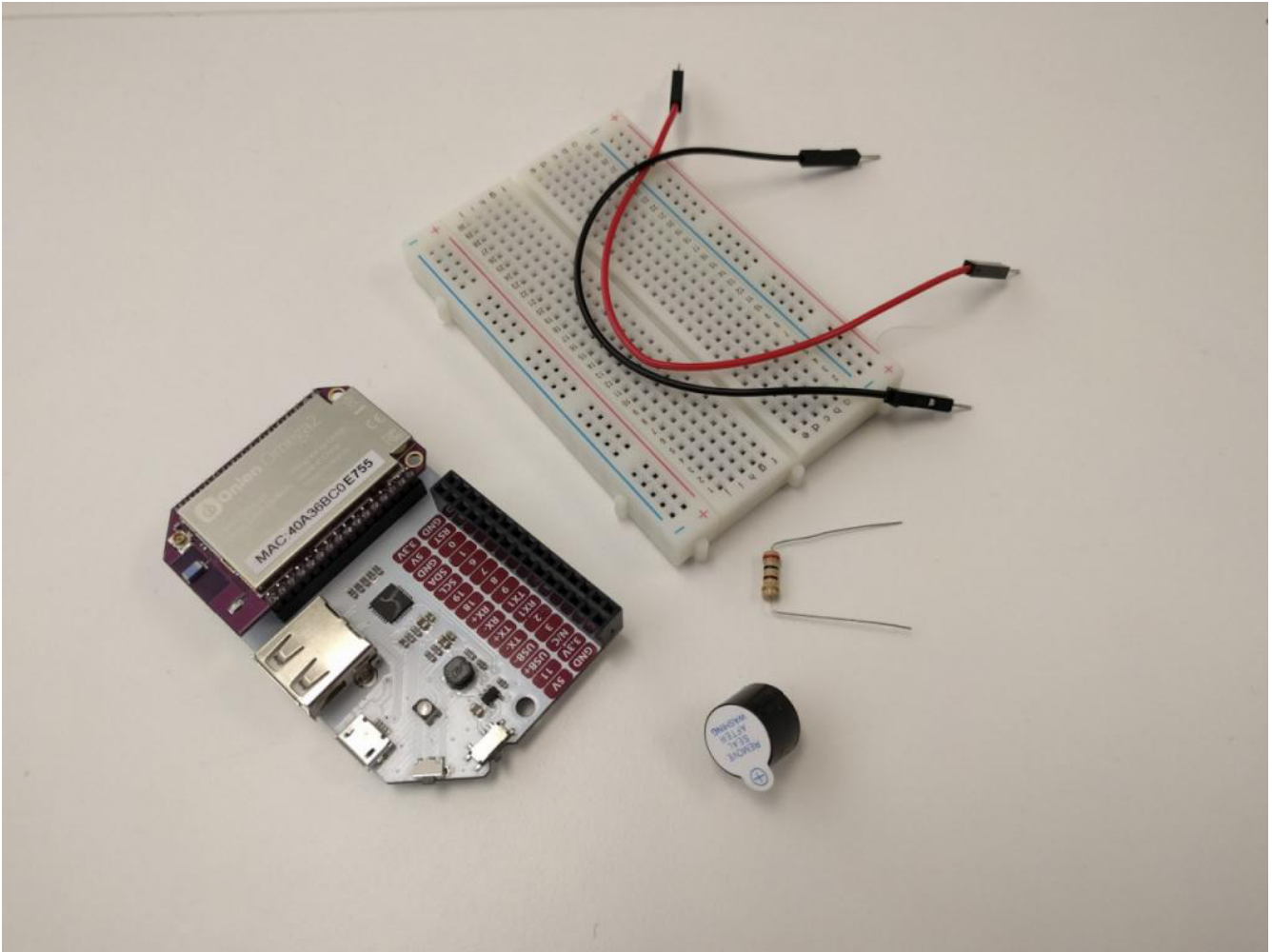
**Time Required:** 30 minutes

This project requires an online calendar source in addition to the ingredients below. We'll be using Google calendar in our Step-By-Step, but you can pick any compatible iCalendar source.

The complete project code can be found in Onion's [iot-gcal-alarm repo](#) on GitHub.

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that exposes the Omega's GPIOs: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#), [Breadboard Dock](#)
- A [breadboard](#) (optional, but recommended)
- 1x [100Ω Resistor](#)
- [Buzzer](#)
- 2x [Jumper wires \(M-M\)](#)



## Step-by-Step

Follow these instructions to setup your very own alarm!

### 1. Prepare

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 2. Install Required Software on the Omega

[Connect to the Omega's Command line](#) and install Python as well as some of the packages we need:

```
opkg update
opkg install python python-pip git git-http ca-bundle
```

We'll use pip to install some additional Python module:

```
pip install --upgrade setuptools
pip install urllib3 python-crontab icalendar
```

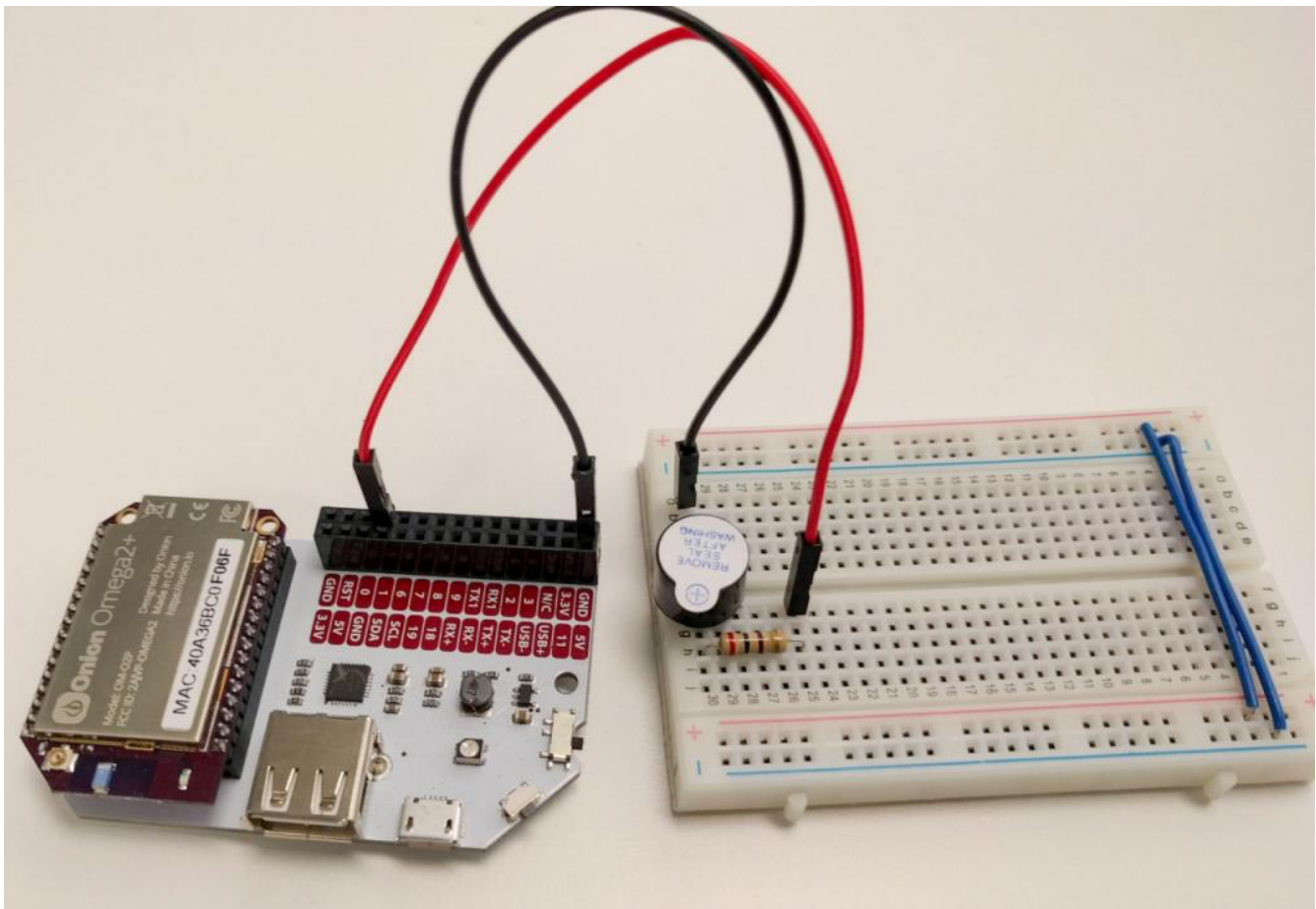
The `urllib3` module will help make HTTP requests to connect to our calendar source. `icalendar` will help parse the calendar data, and `crontab` will allow us to create and remove cron jobs for our alarm.

### 3. Build the Buzzer Circuit

To keep it straightforward, the buzzer will be powered directly by a GPIO. We'll build it on our breadboard for ease of setup. Feel free to do some soldering or electrical tape if you'd like a more compact alarm.

1. Plug the Buzzer across the channel of your breadboard.
2. Using a jumper, connect the negative end of the buzzer (the pin WITHOUT a plus sign) to a GND pin on the expansion headers
3. Plug in a  $100\Omega$  current limiting resistor across the (+) row of the buzzer and an empty row.
4. Finally, connect the resistor to GPIO1 on the Dock's Expansion Headers

Once we're done, it should look a little like this:



### 4. Download the Project Code

The code for the Calendar Alarm can be found in Onion's [iot-gcal-alarm](https://github.com/OnionIoT/iot-gcal-alarm) repo on Github. You can [use Git to clone it](#) to your Omega:

```
cd /root
git clone https://github.com/OnionIoT/iot-gcal-alarm.git
```

Or use `wget` to download the three files directly to your Omega:

```
mkdir /root/iot-gcal-alarm
cd /root/iot-gcal-alarm
wget https://raw.githubusercontent.com/OnionIoT/iot-gcal-alarm/master/iotGcalAlarm.py https://raw
```

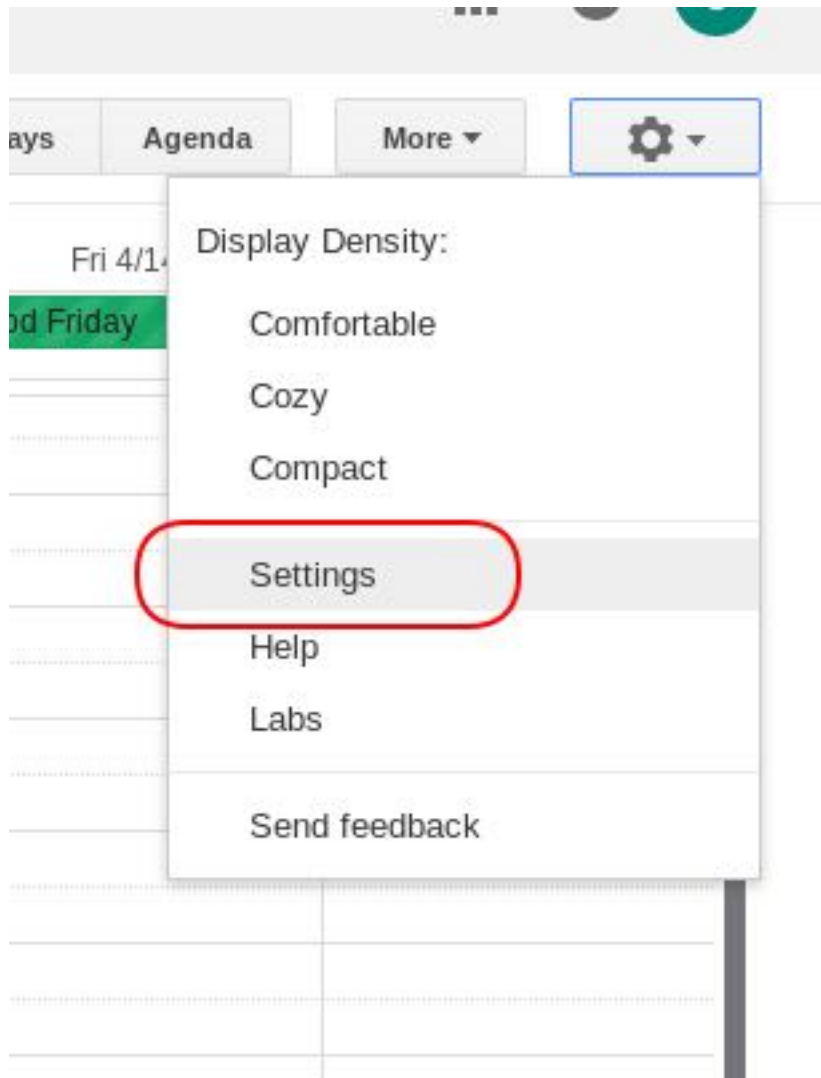
Now all of the code will be in the new `/root/iot-gcal-alarm/` directory on your Omega.

## 5. Setting Up Your Calendar

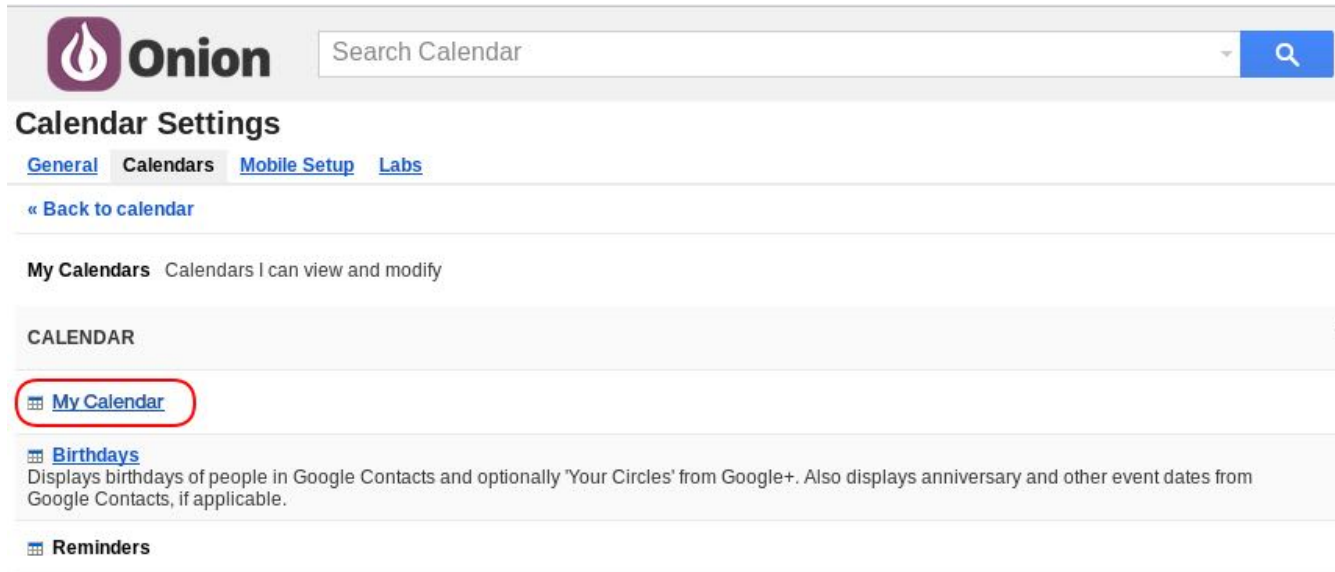
The calendar can be any calendar you wish, but for the events to be recognized, they have to include `SET_BUZZER` string in the event's title. More specifically, the 'SUMMARY' field of the iCalendar event.

We'll use a Google calendar as an example.

First, go to the settings page of your calendar:



Then navigate to the 'Calendars' tab in the settings page. Here, click which calendar your event will be in:



**Onion** Search Calendar

## Calendar Settings

[General](#) [Calendars](#) [Mobile Setup](#) [Labs](#)

[« Back to calendar](#)


**My Calendars** Calendars I can view and modify

CALENDAR

- [My Calendar](#)
- [Birthdays](#)  
Displays birthdays of people in Google Contacts and optionally 'Your Circles' from Google+. Also displays anniversary and other event dates from Google Contacts, if applicable.
- [Reminders](#)

The green 'ICAL' button next to 'Private Address' will be a direct link to your up-to-date calendar in the .ics format - this is it! To get the link, right click, and hit the 'Copy Link Location' button.



<b>Organization:</b>	Onion Corporation
<b>Description:</b>	<input type="text"/>
<b>Location:</b>	<input type="text"/> e.g. "San Francisco" or "New York"
<b>Calendar Time Zone:</b>	This calendar uses your current time zone
<p><b>Embed This Calendar</b></p> <p>Embed this calendar in your website or blog by pasting this code into your web page. To embed multiple calendars, click on the Customize Link</p>	
<p><b>Calendar Address:</b></p> <p><a href="#">Learn more</a> <a href="#">Change sharing settings</a></p>	<p><b>ICAL</b> <b>HTML</b> (Calendar ID: jamr...) This is the address for your calendar</p>
<p><b>Private Address:</b></p> <p><a href="#">Learn more</a></p>	<p><b>ICAL</b> <b>Reset Private URLs</b> This is the private address for this calendar</p>
<p><b>Export Calendar:</b></p> <p><a href="#">Learn more</a></p>	<p><a href="#">Export this calendar</a> <b>Export:</b> All events in this calendar</p>
<p><b>Delete calendar:</b></p> <p><a href="#">Learn more</a></p>	<p><a href="#">Delete all events in this calendar</a> <b>Delete:</b> All events in this calendar</p>
<p><a href="#">« Back to calendar</a></p>	<p><input type="button" value="Save"/> <input type="button" value="Cancel"/></p>

Open up config.json from the repo, and paste the link as the value to the "icalAddr" key - replacing your-calendar-address:

```
root@Omega-F13B:~/iot-gcal-alarm# cat config.json
{
  "icalAddr" : "https://calendar.google.com/calendar/ical/[redacted]/basic.ics"
}
```

## 6. Run the Code

Let's run the code!

```
python /root/iot-gcal-alarm/iotGcalAlarm.py
```

The script will read the calendar data from your source, and add a cron job for every event in the future with the keyword `SET_BUZZER`. It will also clear any cron jobs that have already been run - only for up to a year, since cron does not keep track of year data.

## 7. Schedule the Code to Run Once a Day

Now we'll use the trusty `cron` Linux utility to schedule the script to run once a day at midnight to update our other cron jobs.

Run `crontab -e` to add the task, it will open the crontab file in `vi`, add the following lines:

```
0 0 * * * python /root/iot-gcal-alarm/iotGcalAlarm.py
#
```

restart the cron daemon for the changes to take effect:

```
/etc/init.d/cron restart
```

Now the `iotGcalAlarm` script will run every day at midnight, updating your alarms based on calendar events that have the `SET_BUZZER` keyword.

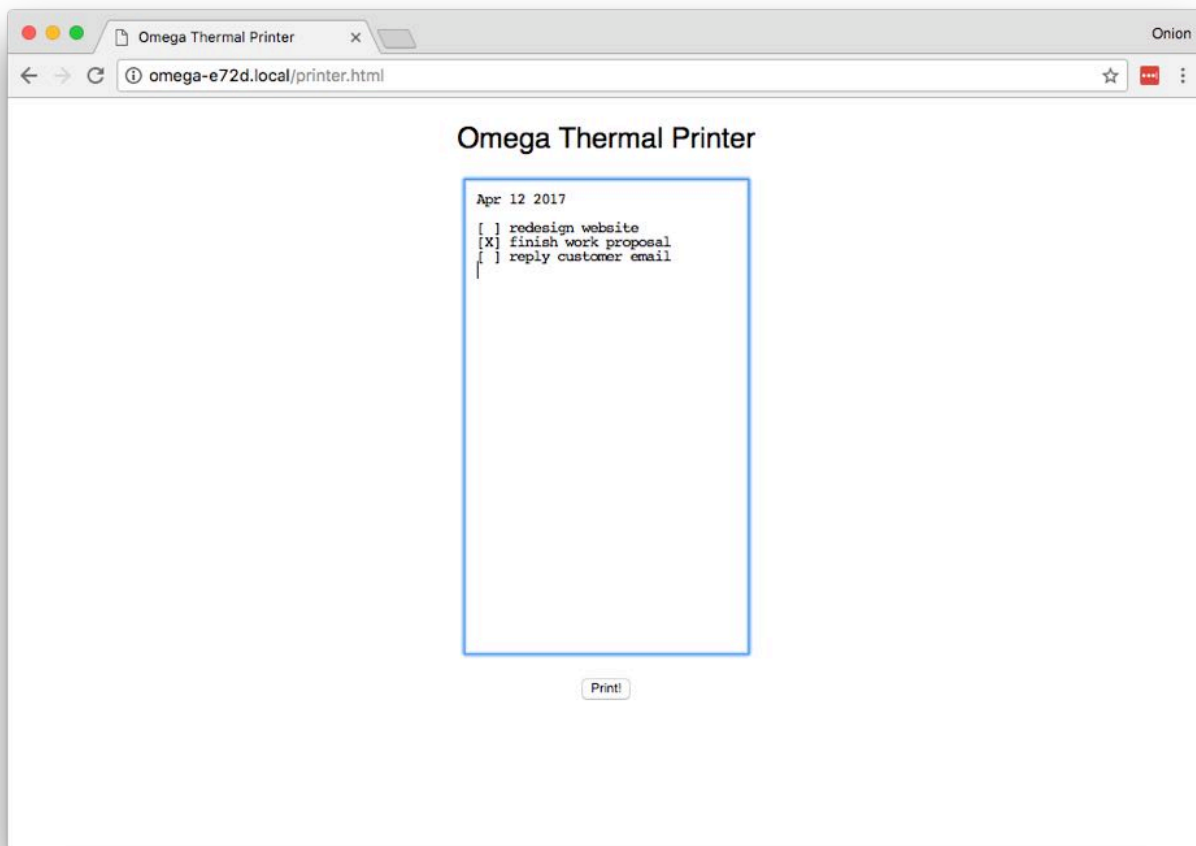
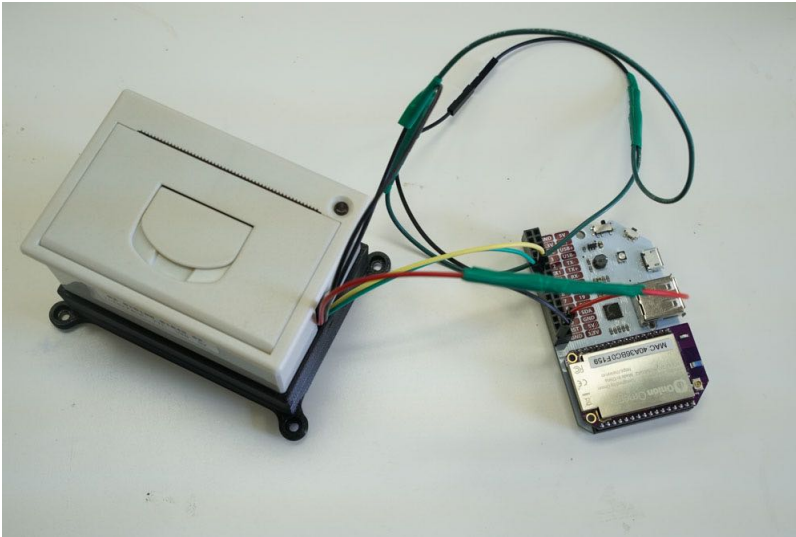
Check out the Omega documentation for more info on [using cron](#)

## 8. Alarming!

You're all set, your Omega will now automatically set off alarms based on your calendar events!

## Thermal Printer

In this project, we'll be using the Omega to control a thermal printer via a web interface. Simply type text in a box, and click Print to print it out in real life!





## Overview

**Skill Level:** Intermediate

**Time Required:** 20 minutes

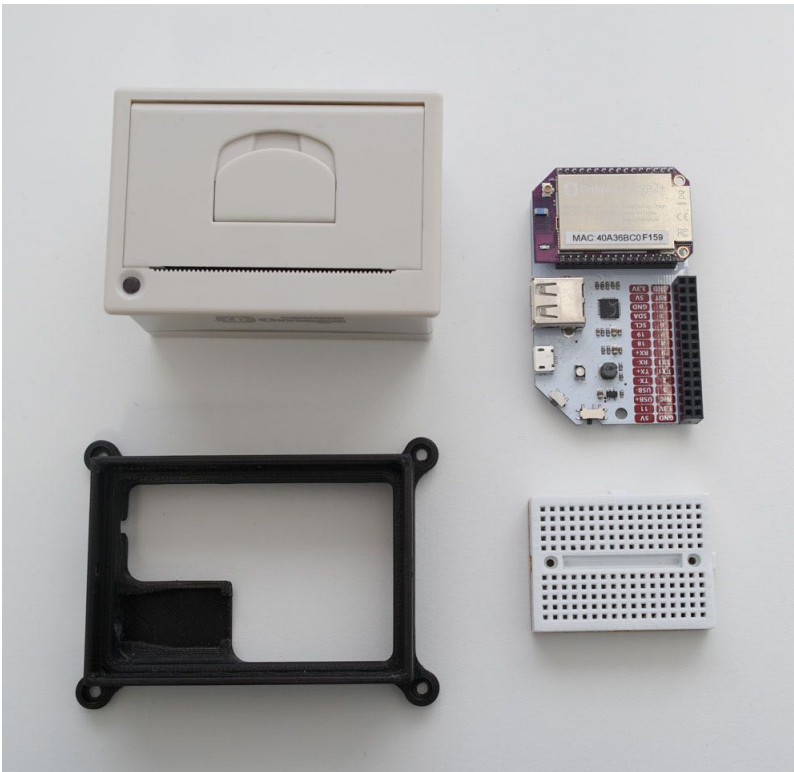
This tutorial will use the Omega to control a thermal printer - most often seen at cash registers and restaurant checkouts. We'll be using connectors and cables that come with the printer to provide it with power and communicate serially.

Optionally, we'll 3D print a base to clean up the cabling and give the printer some polish.

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that exposes the Omega's GPIOs: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#), [Breadboard Dock](#)
- [Thermal Printer](#)
  - comes with a 2-pin JST power cable and a 5-pin TTL cable
- [DC barrel jack adapter](#)
- [5V 2A DC Power Supply](#)

- 3D printed base
- Male-to-Male Jumper Wires



## Step-by-Step

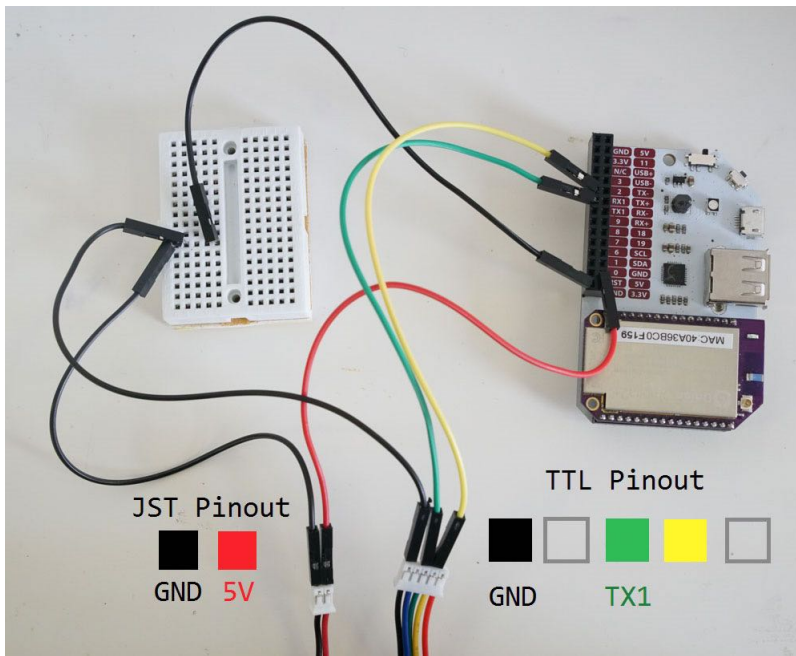
Follow these steps to turn your Omega into a web-based printer!

### 1. Prepare

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 2. Wire Up the Thermal Printer

We'll be doing the following to connect the Omega to the printer:



Let's dive in:

1. First make sure the Omega is off and seated in the Expansion Dock.
2. Then, plug in the 2-pin power cable into the left side of the bottom of the printer above.
3. Route the black wire to the GND pin on the Expansion Dock headers.
4. Next we'll connect the serial wires:
  1. First plug one end of the 5-pin TTL cable into the socket at the bottom of the printer.
  2. Using a jumper (preferably green to keep it consistent) connect the green wire pin on the TTL connector to the UART1 TX pin on the Omega Expansion header.
  3. Same goes for the yellow wire pin on the TTL connector, except this one goes to the UART1 RX pin on the Expansion Header.
  4. Lastly, do the same for the black wire to the GND pin on the Expansion Dock header - we used a breadboard intermediary in the diagram to show how the connection is supposed to go.
5. Finally, connect the red wire from the JST connector to a 5V pin on the Expansion Dock headers.

Note that we used a breadboard to connect the two GND pins from the printer to a single GND pin on the Omega. It would have been equally ok to connect the two printer GND pins to two GND pins on the Omega.

## 2. Download the Project Code

The code for this project is all done and can be found in Onion's [iot-thermal-printer](#) repo on GitHub.

First, [connect to the Omega's Command line](#) and install git:

```
opkg update
opkg install git git-http ca-bundle
```

And then [use git to download the project code to your Omega](#):

```
cd /root
git clone https://github.com/OnionIoT/iot-thermal-printer.git
```

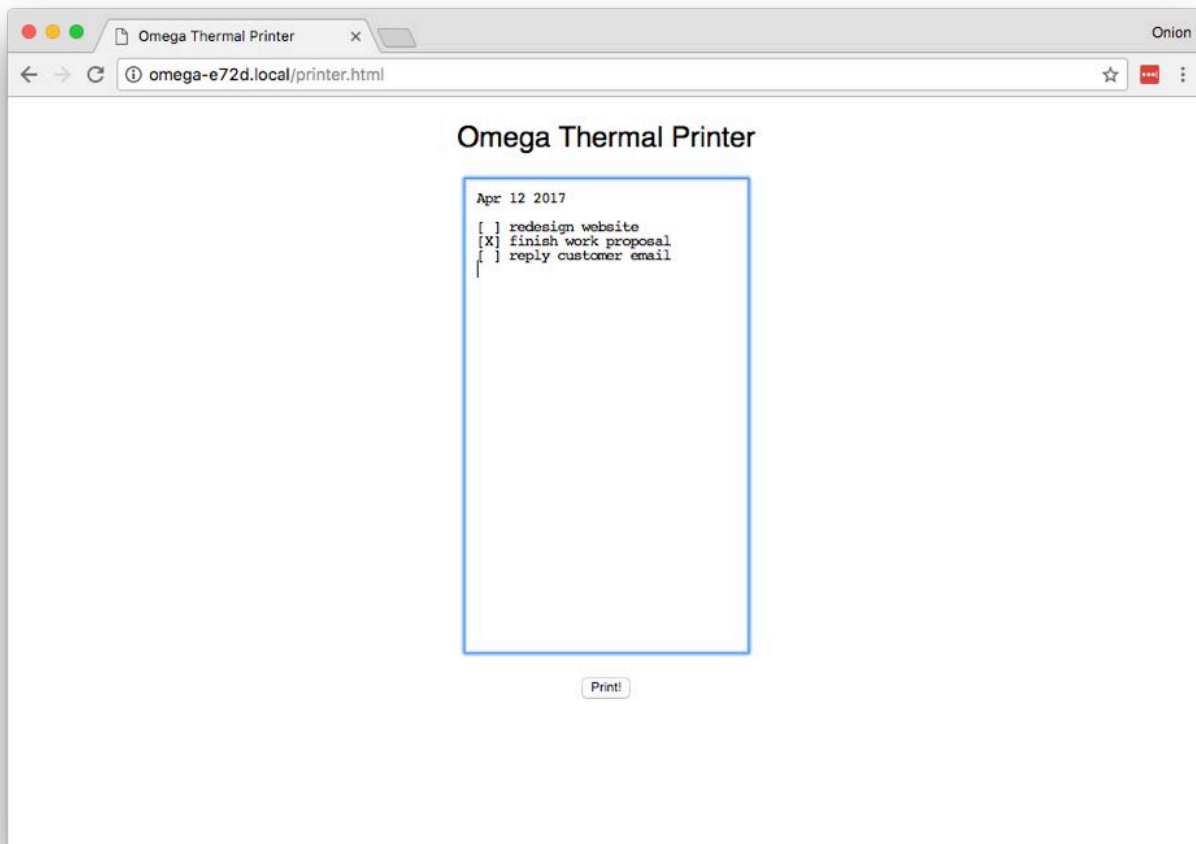
After cloning the repo, enter the repo directory and copy the contents of the `www` to the `/www` directory on the Omega:

```
cd iot-thermal-printer
cp -r www/ /
```

By virtue of `uhttpd`, the HTTP server running on the Omega, all of the files in the `/www` directory will be served up as a website.

## Running the Printer

1. Connect your Omega to your WiFi network, or connect your computer to the Omega's WiFi network.
2. In a web browser, navigate to `omega-ABCD.local/printer.html`, where `ABCD` is the last 4 digits on the sticker on the Omega.
  - On some Android and PC devices, the `omega-ABCD.local` address doesn't always work. Follow our [guide on finding your Omega's IP Address](#) and use the IP address instead of `omega-ABCD.local` when connecting the web interface. It will be something along the lines of `192.168.1.109/printer.html`
3. Type in text in the box in the middle of the webpage.
4. Click print to print it!



The physical output:



## Code Highlight

This project uses the `cgi-bin` method to run scripts on the Omega via a web interface. In the following line, we send the data from the text box to the script in the `/cgi-bin` directory using asynchronous JavaScript (AJAX):

```
$.ajax({
  type: "POST",
  url: "/cgi-bin/print.sh",
  data: $('#printContent').val().split('\n').join('\r'), // <-- We need to replace \n with \r
  contentType: 'text/plain'
})
```

The `print.sh` script works like a simple API endpoint that takes data and does something with it; in this case, sending it to the printer via serial:

```
#!/bin/sh

echo "Content-type: application/json"
echo ""

if [ "$REQUEST_METHOD" = "POST" ]; then
```



```
read -n $CONTENT_LENGTH line
echo $line > /dev/ttyS1
# feed paper
echo '' > /dev/ttyS1
fi

echo '{"success":"ok"}'

exit 0
```

This is just one of many methods to create your own endpoints and services easily and quickly on the Omega!

## Going Further

With a little bit of wire splicing and soldering, we can make this project much more compact! Check out the [next part](#) for more.

## Thermal Printer - A Compact Version

In this project, we'll build on the [Thermal Printer Project](#). While the Expansion Dock definitely suits this purpose well, we make the same thing in a very compact package using the Mini Dock and a little bit of wire splicing and soldering.





## Overview

**Skill Level:** Advanced

**Time Required:** 1 Hour

We'll need a 3D printed plastic base for this project - if you have it printed out already, this will save you some trouble. Additionally, we'll wire up a DC barrel connector for power and wires for serial communication with the Omega instead of powering it from the Expansion dock.

This tutorial will require you to solder a wire to one of the components on the Mini Dock. Please familiarize yourself with proper soldering technique and safety procedures when working with soldering irons, as there is a risk of injury due to the high heat!

If you are not comfortable soldering, try finding a friend or professional who can quickly solder it for you. Or practice soldering wires together and then work your way up to soldering on actual electronics.

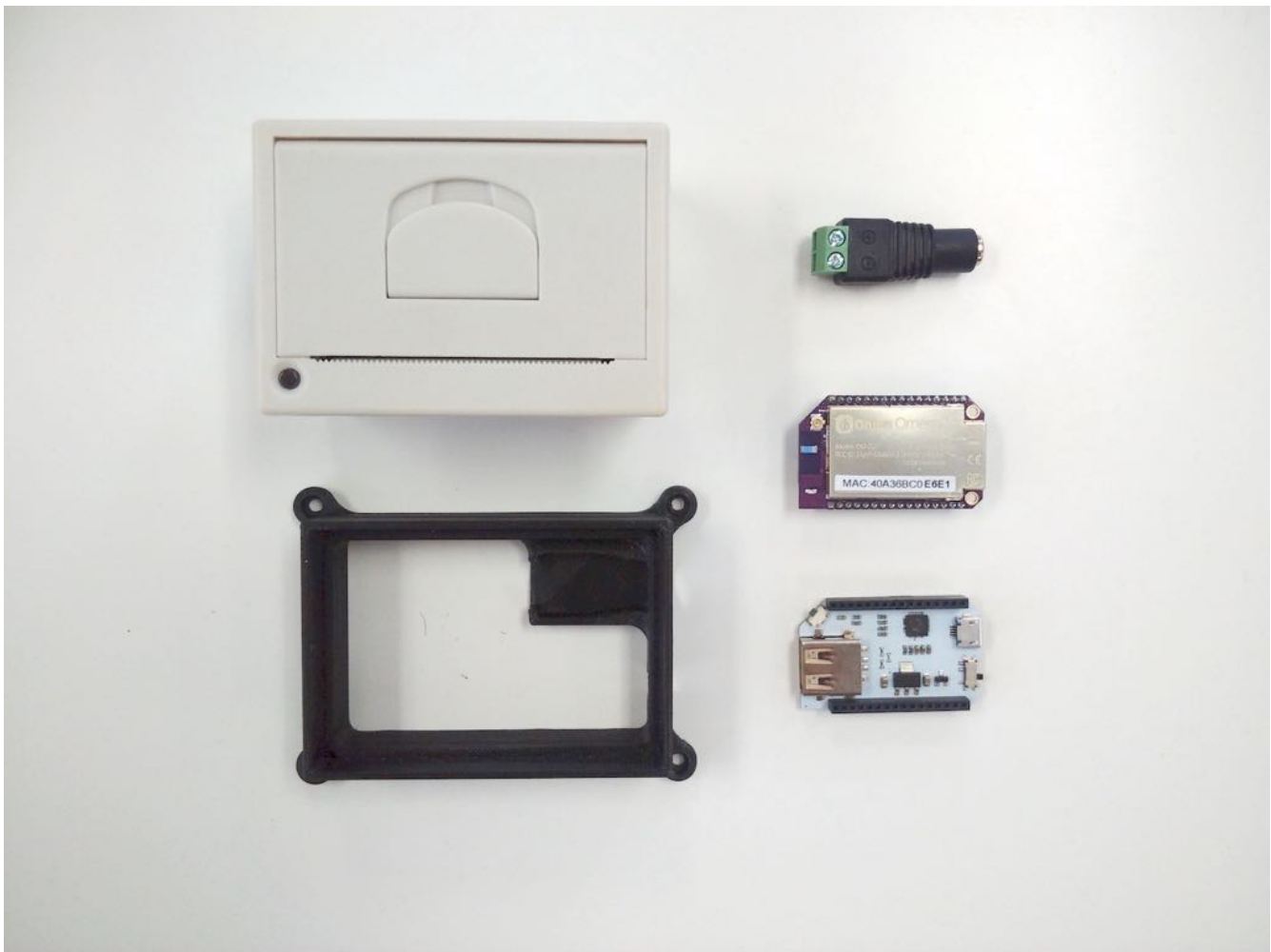
**Warning:** Soldering irons get really hot and can burn skin badly! Be careful and solder only if you know what you're doing and at your own risk, Onion is not responsible for any injury or damage!

## Ingredients

- Onion Omega2 or Omega2+
- Onion Mini Dock
- Thermal Printer
  - comes with a 2-pin JST power cable and a 5-pin TTL cable
- DC barrel jack adapter
- 5V 2A DC Power Supply
- 3D printed base
- Male-to-Male Jumper Wires

## Tools:

1. Soldering iron + solder
2. Wire Strippers
3. Double-sided tape



## Step-by-Step

Follow these steps to turn your Omega into a web-based printer!

### 1. 3D Print the Base

3D print the base to hold our components together. If you do not have a 3D printer available nearby, there are online services available such as [3DHubs](#).

### 2. Install in the Power Jack

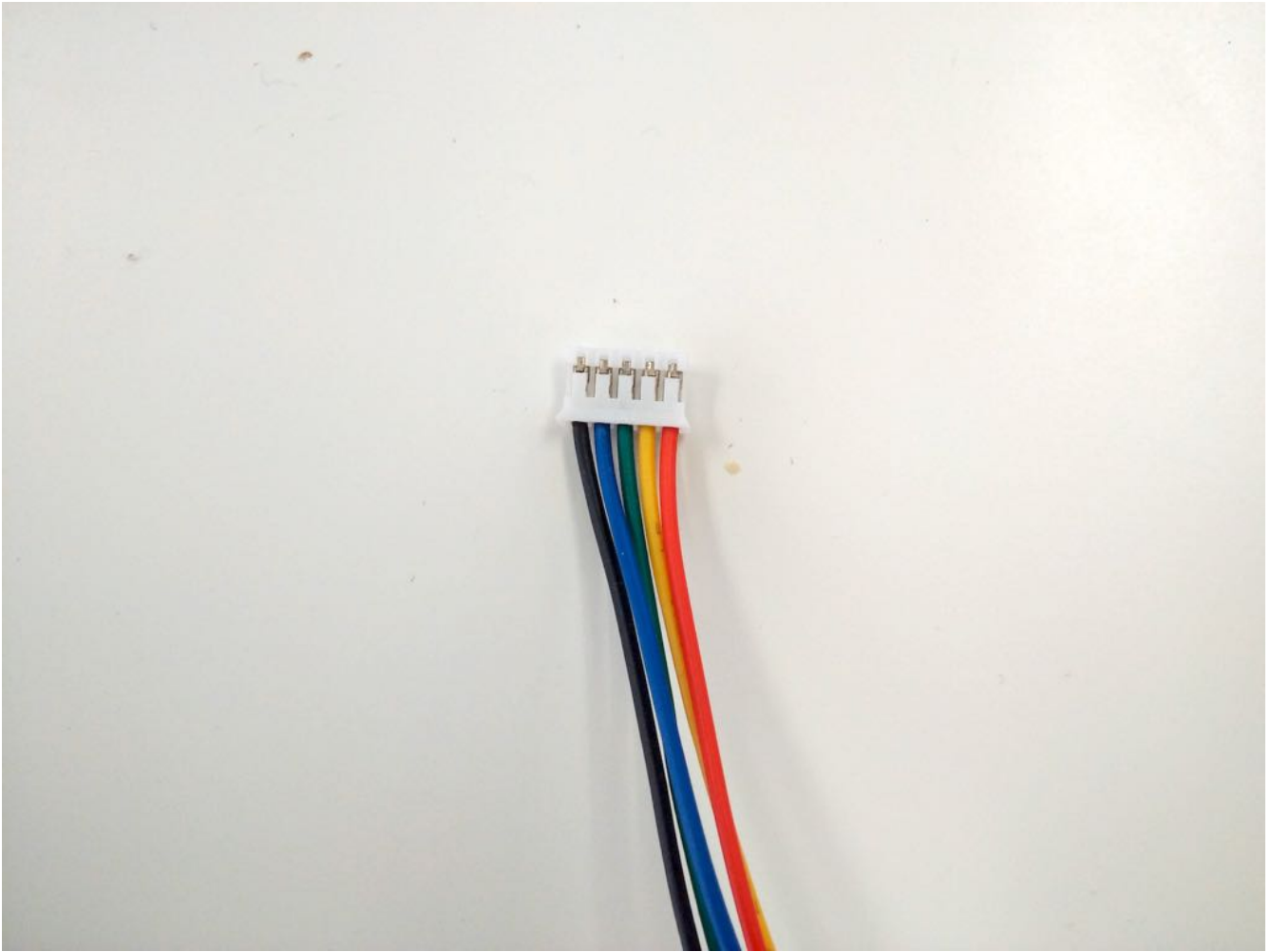
Insert the power jack adapter into the printer base. Do this first, since the other pieces will cover up the base later.



### 3. Trim the Cables

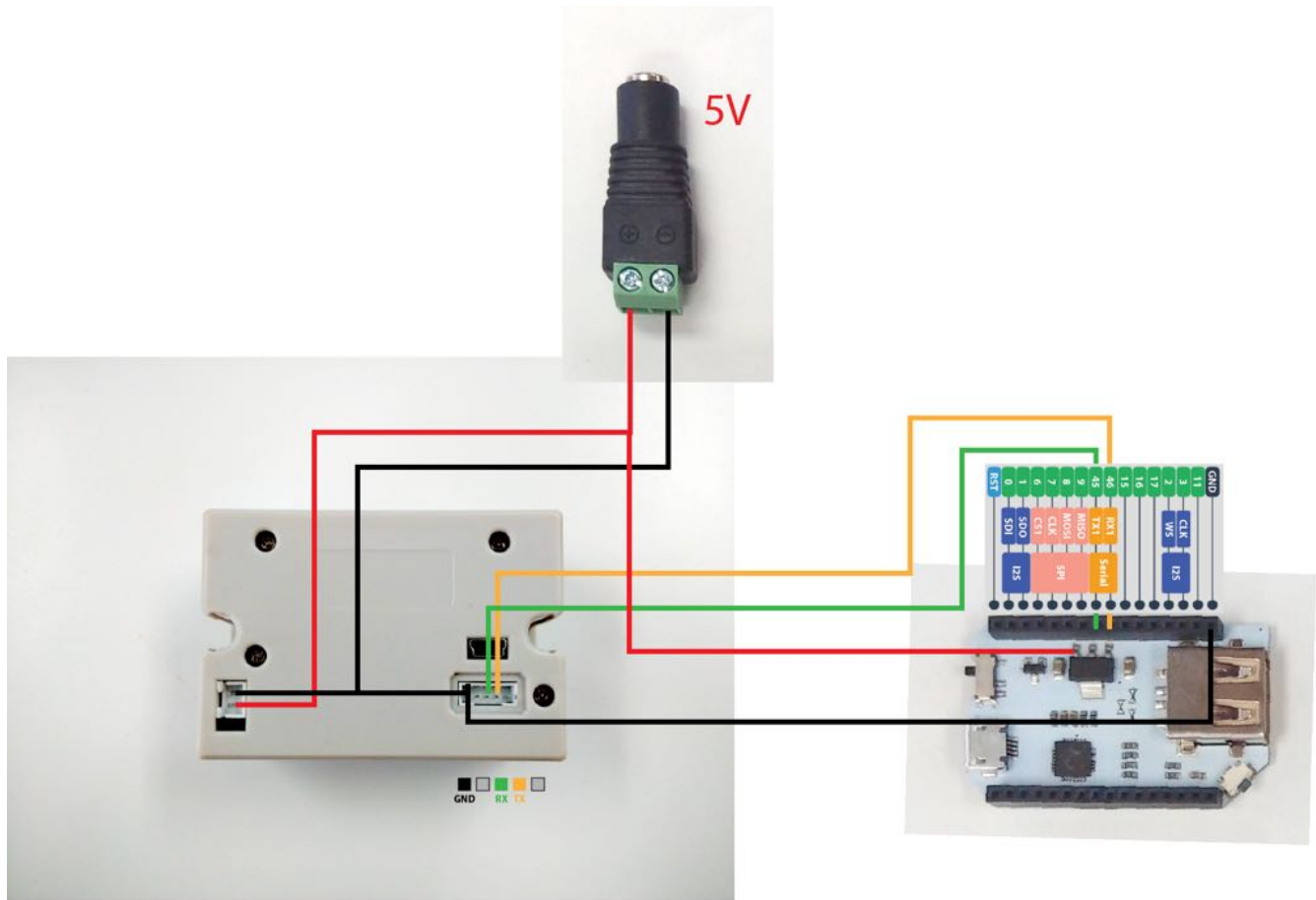
Next we need to cut one end of the 5-pin TTL cable that came with the thermal printer. This is so we can re-route the wires to where they need to go. The other end we'll leave alone, since that goes into the printer.

Cut only **one** of these ends off, leaving bare wire:

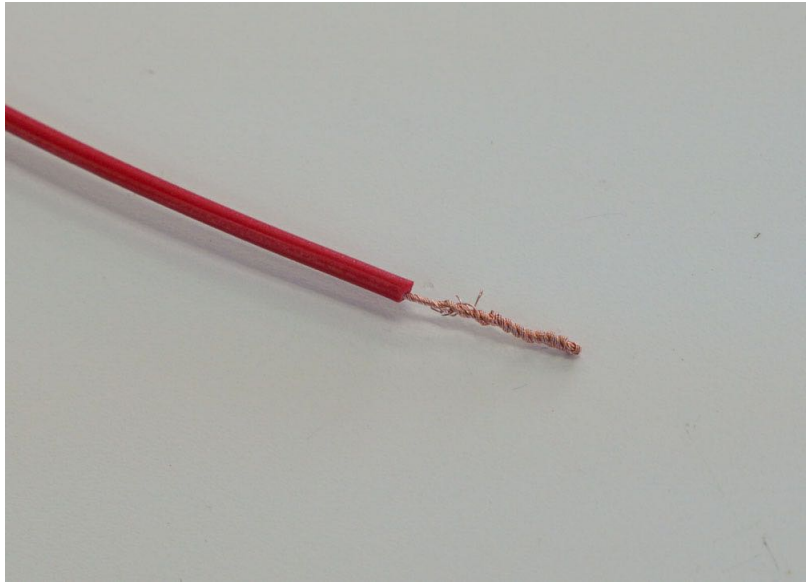


#### 4. Assemble the Circuit

We'll be doing the following to connect the Omega to the printer:

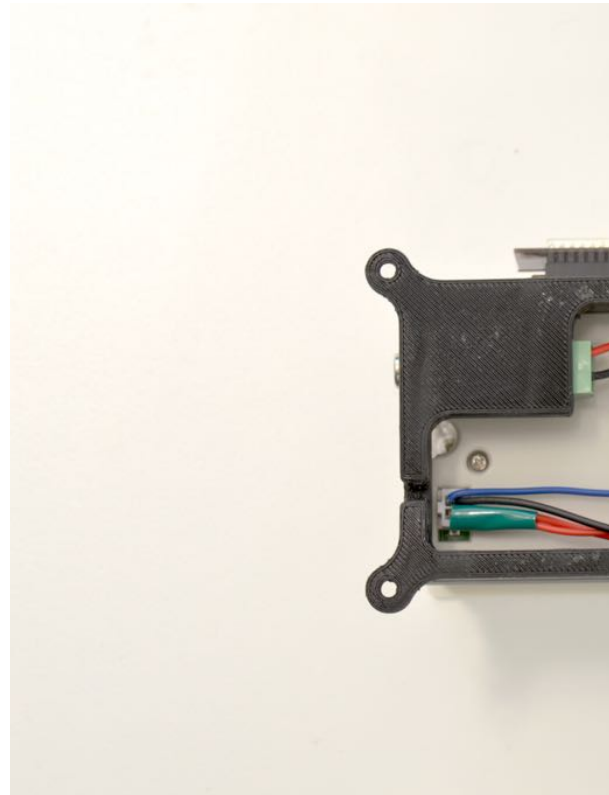


1. Plug in the 2-pin JST power cable into the 2-pin port on the bottom of the printer.
2. Route the red and black wires to the barrel jack; make sure the red wire is connected to the **Positive (+)** terminal and the black to the **Negative (-)** terminal.
3. Then plug the non-cut end of the 5-pin TTL cable into the 5-pin port on the printer. Route the wires through the gap in the printer case on the right side of the USB connector.
4. Route the black, green, and yellow TTL wires to the UART pins on the Mini Dock (highlighted in the diagram above).
  - The middle pin on the printer is the printer's serial RX (Receiving) pin, and it needs to be connected to the Omega's UART1 TX (Transmitting) pin
  - The pin second from the right on the printer is the printer's serial TX pin, connect it to the Omega's UART RX pin
  - To insert the wires into the Mini Dock, you can insert the wires into the headers after you strip



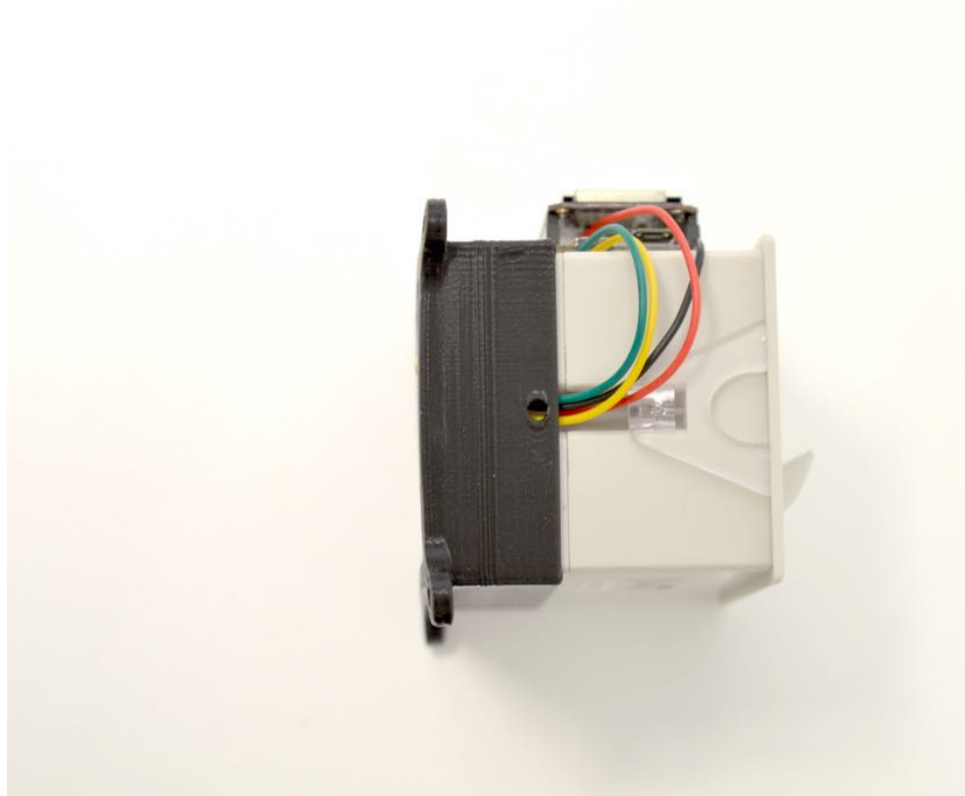
and twist the ends like so:

5. Then solder the red power cable to the 5V pin on the regulator on the Mini Dock as shown above. Take care that you solder to the correct contact or you may damage your board!
  - We do this tricky business because the thermal printer requires 5V to operate and the Mini Dock
6. Insert your printer into the base from the top so that the 5-pin cable is routed through the channel on the side of the printer.



- The wiring on the underside should like something like this:





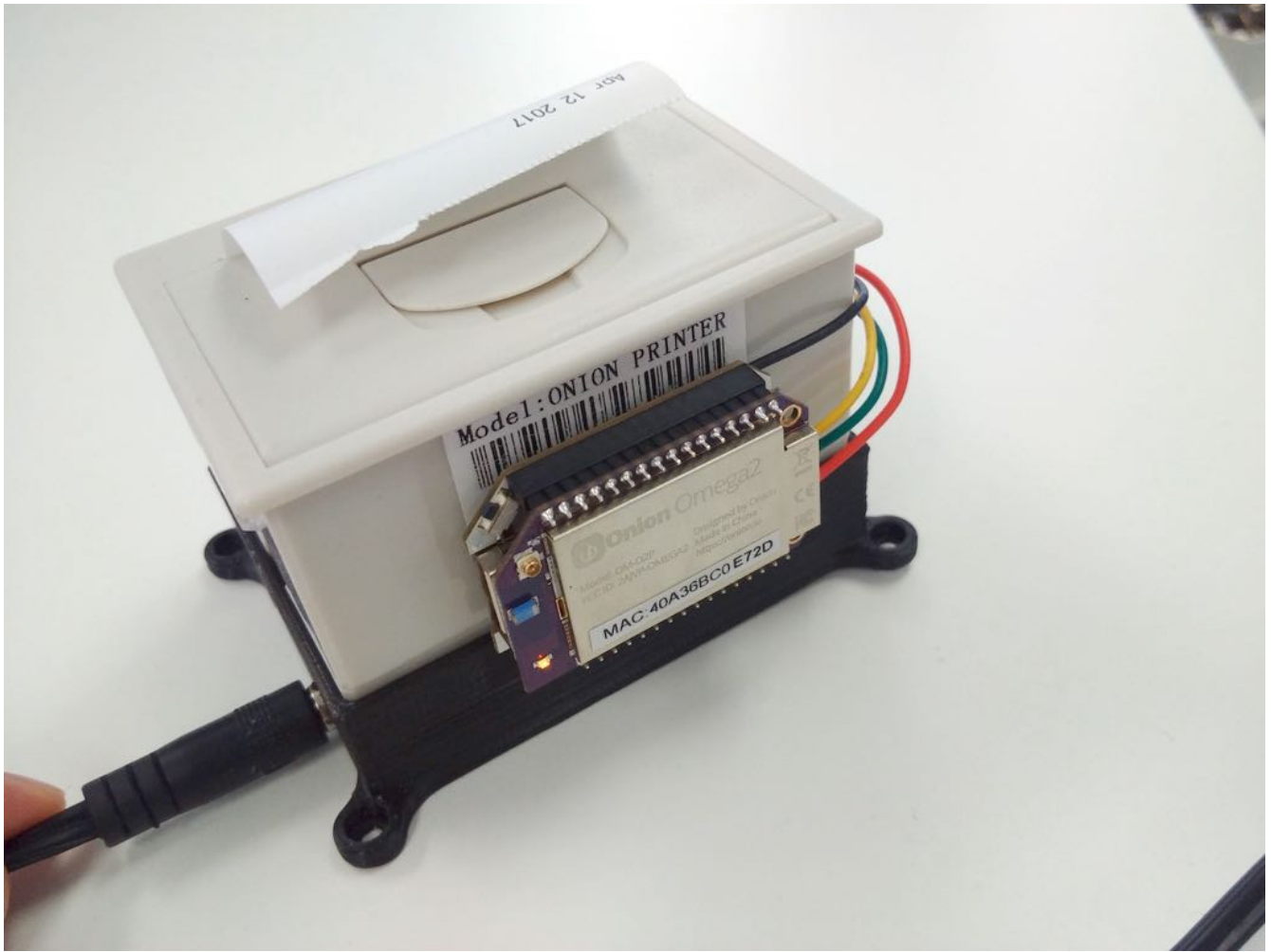
- How the 5-pin cable is routed:
7. If your wires are all connected, you can then flip the printer right side up.

**Do not plug in the power supply just yet**, as we still need to connect and solder some wires to the Omega.

## 5. Connect the Omega

Plug the Omega into the Mini Dock. The Omega's pins will push the stripped wires down into the header. Make sure they don't pop out during the process!

Use double-sided tape or putty to affix the Omega to the rear of the printer like so:





Now plug in the 5V power supply into the barrel jack and turn the switch on the Mini Dock to ON.

## 6. Download the Code

*If you've already downloaded the code in [the first part of the project](#), your printer is ready to go so you can skip this part!*

The code for this project is all done and can be found in Onion's [iot-thermal-printer repo](#) on GitHub.

First, [connect to the Omega's Command line](#) and install git:

```
opkg update
opkg install git git-http ca-bundle
```

And then [use git to download the project code to your Omega](#):

```
cd /root
git clone https://github.com/OnionIoT/iot-thermal-printer.git
```

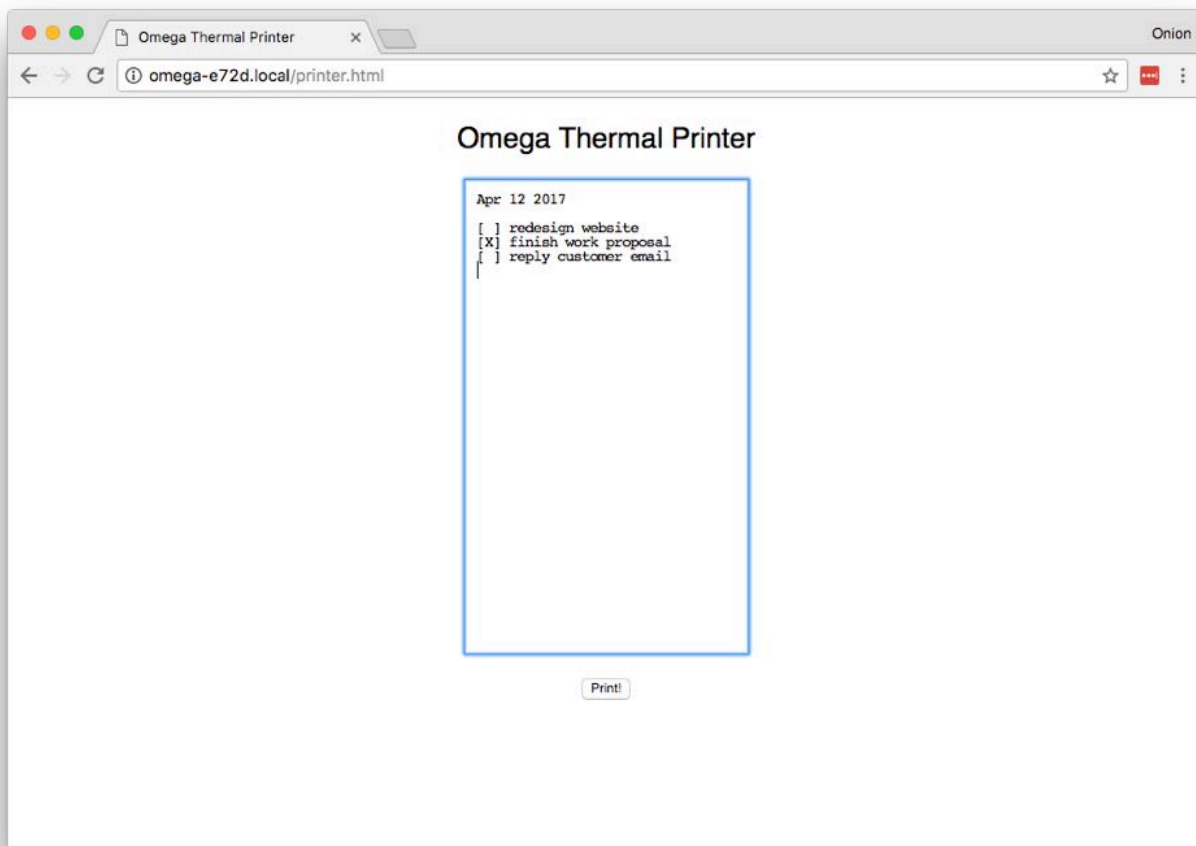
After cloning the repo, enter the repo directory and copy the contents of the `www` to the `/www` directory on the Omega:

```
cd iot-thermal-printer
cp -r www/ /
```

By virtue of `uhttpd`, the HTTP server running on the Omega, all of the files in the `/www` directory will be served up as a website.

## Using the Printer

1. Connect your Omega to your WiFi network, or connect your computer to the Omega's WiFi network.
2. In a web browser, navigate to `omega-ABCD.local/printer.html`, where `ABCD` is the last 4 digits on the sticker on the Omega.
  - On some Android and PC devices, the `omega-ABCD.local` address doesn't always work. Follow our [guide on finding your Omega's IP Address](#) and use the IP address instead of `omega-ABCD.local` when connecting the web interface. It will be something along the lines of `192.168.1.109/printer.html`
3. Type text in the box in the middle of the webpage.
4. Click print to physically print it!

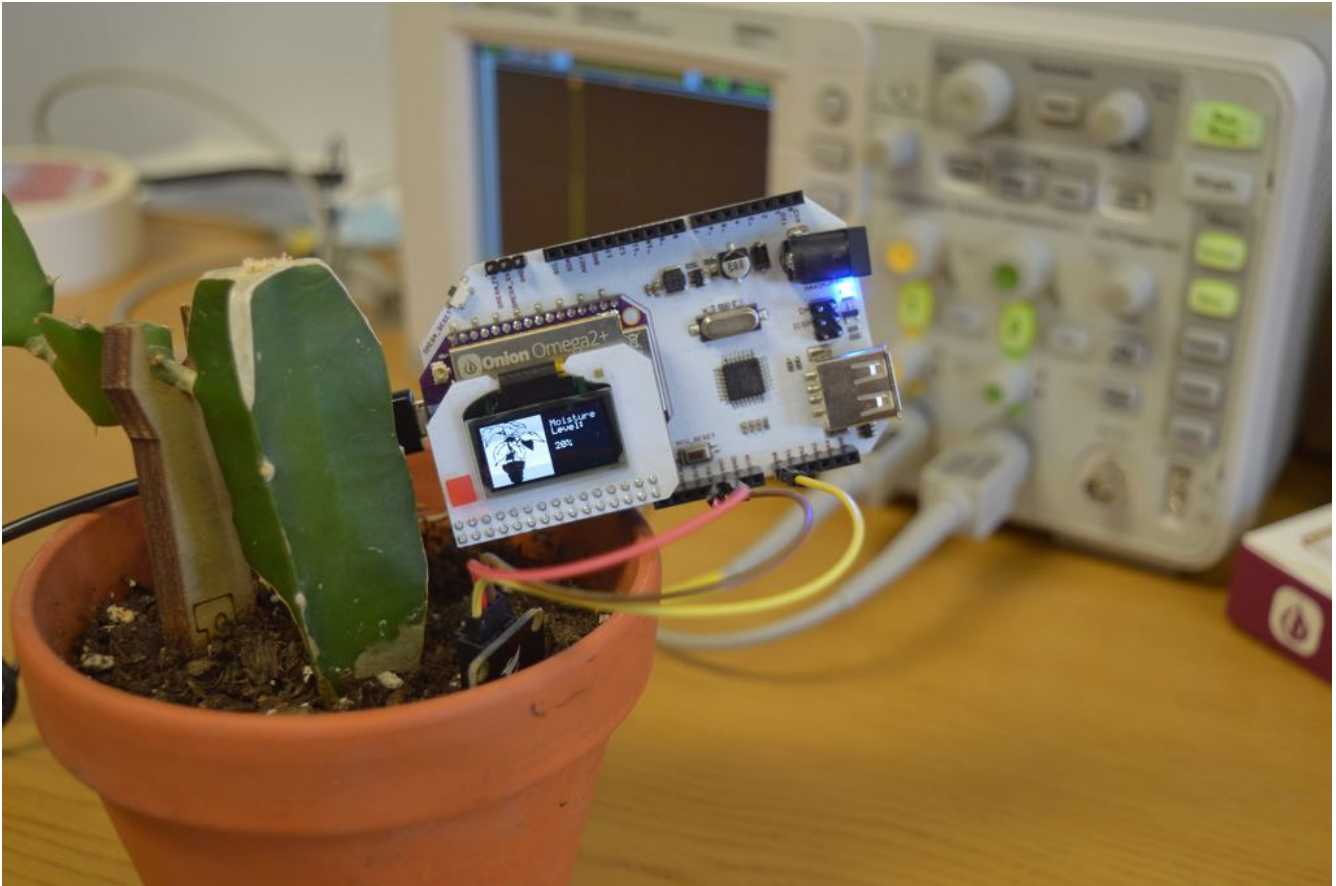


The physical output:



## Smart Plant - Measuring Plant Data

This is the first part of a multi-step project where we'll make one of your plants smart! For now, we'll measure and display the soil moisture level of your plant.



## Overview

**Skill Level:** Beginner-Intermediate

**Time Required:** 45 minutes

We'll be using the Arduino Dock to read the analog measurement from a soil moisture sensor. The code is written in Python and makes use of the UART1 serial port on the Omega to communicate with the Arduino Dock's microcontroller. We're also using [Onion's pyOLED module](#) to provide control of the OLED Expansion.

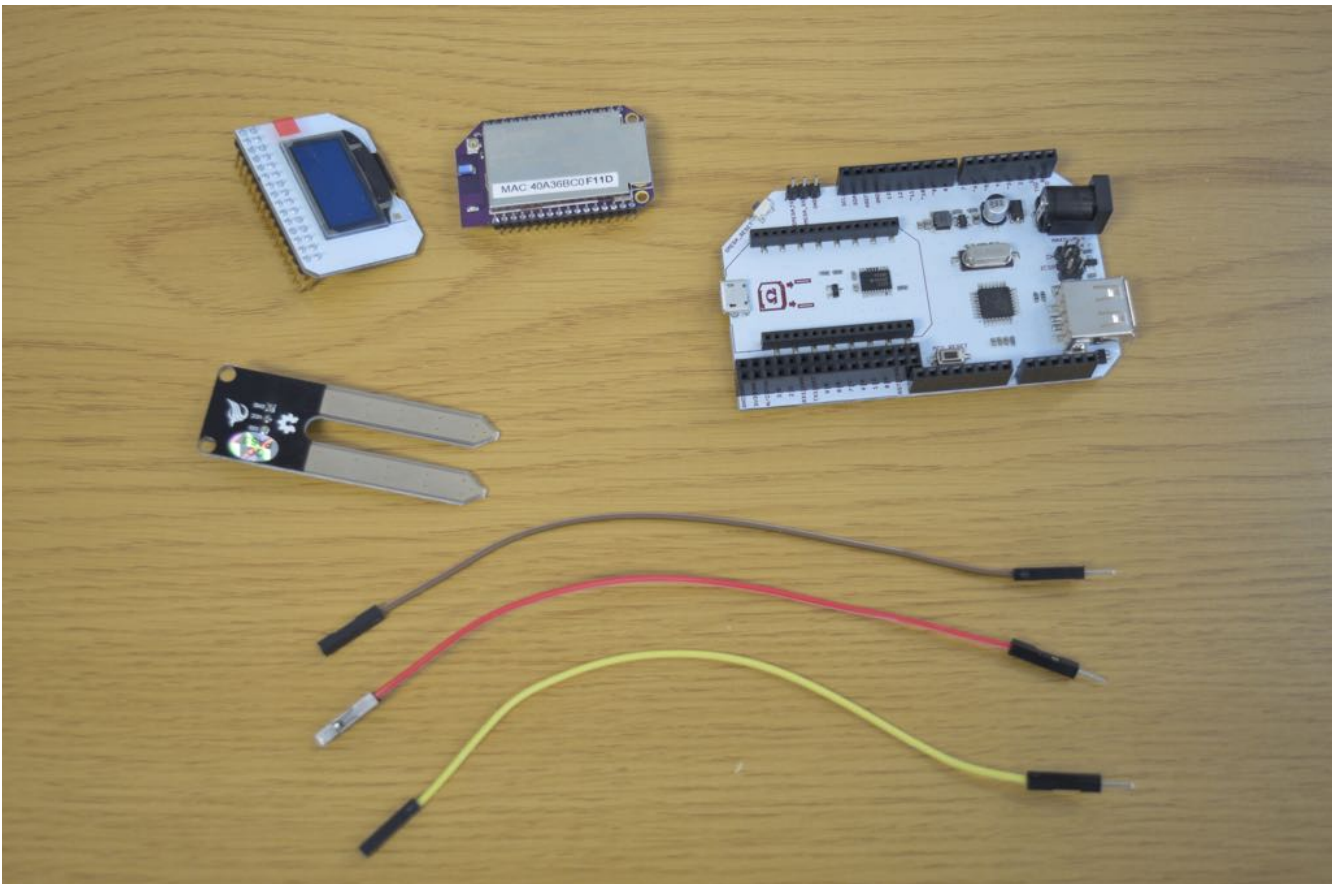
The complete project code can be found in Onion's [smart-plant repo on GitHub](#).

In this project series, we'll be doing the following:

1. Adding smarts to your plant by measuring it's soil moisture level
2. Sending plant data to the Losant IoT Platform and check in on your plant from anywhere by looking at the nicely visualized data
3. Updating the Losant workflow to notify you with a Tweet when your plant needs watering
4. Adding a water pump to your setup and update the Losant workflow to automatically water your plant when it needs watering
5. Updating the smart plant setup so the Omega and pump can be powered with a single supply

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Onion [Arduino Dock 2](#)
- Onion [OLED Expansion](#) (optional but recommended)
- [Soil Moisture Sensor](#)
- 3x [Male-to-Female Jumper Wires](#)

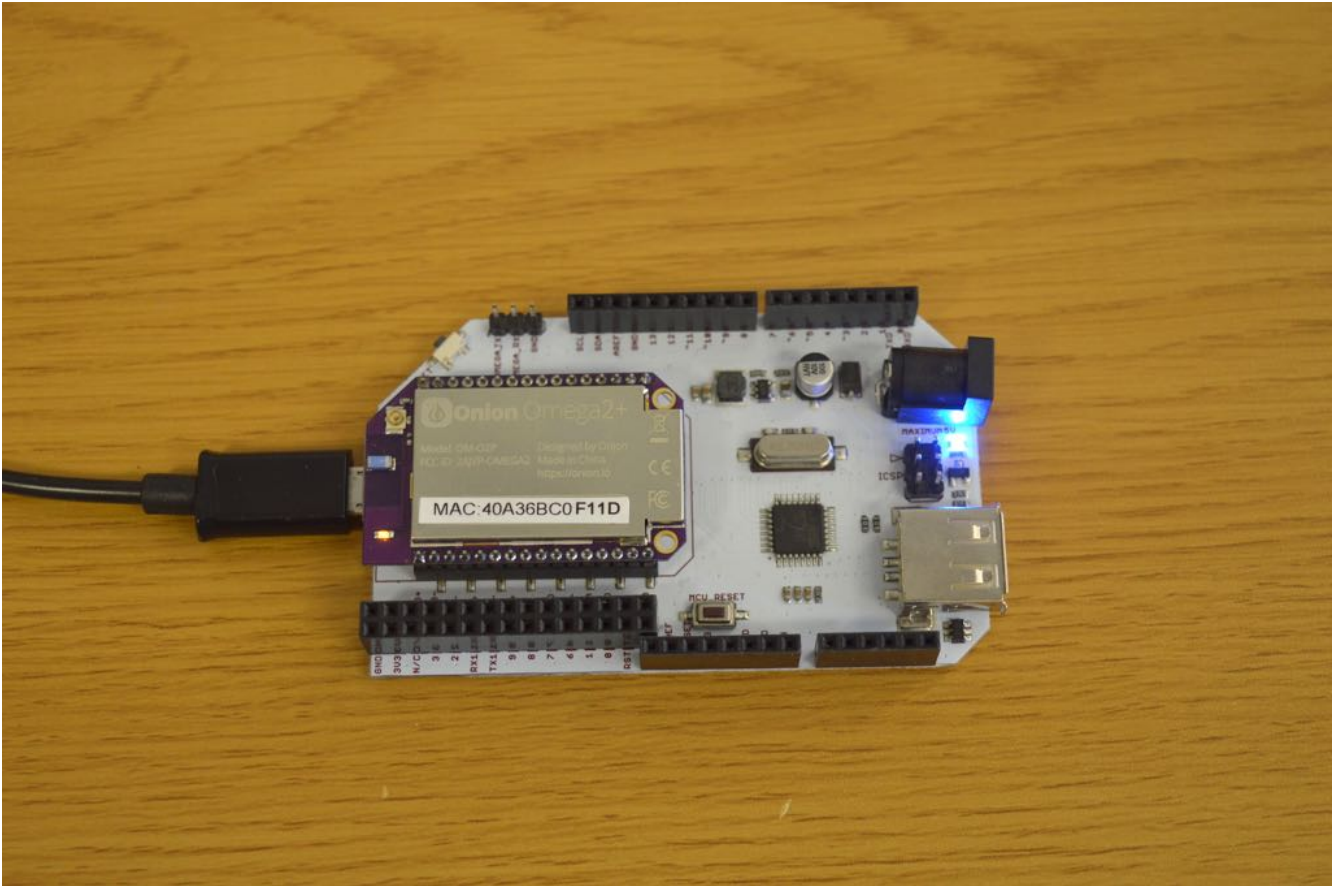


## Step-by-Step

Follow these instructions to setup the Smart Plant project on your very own Omega!

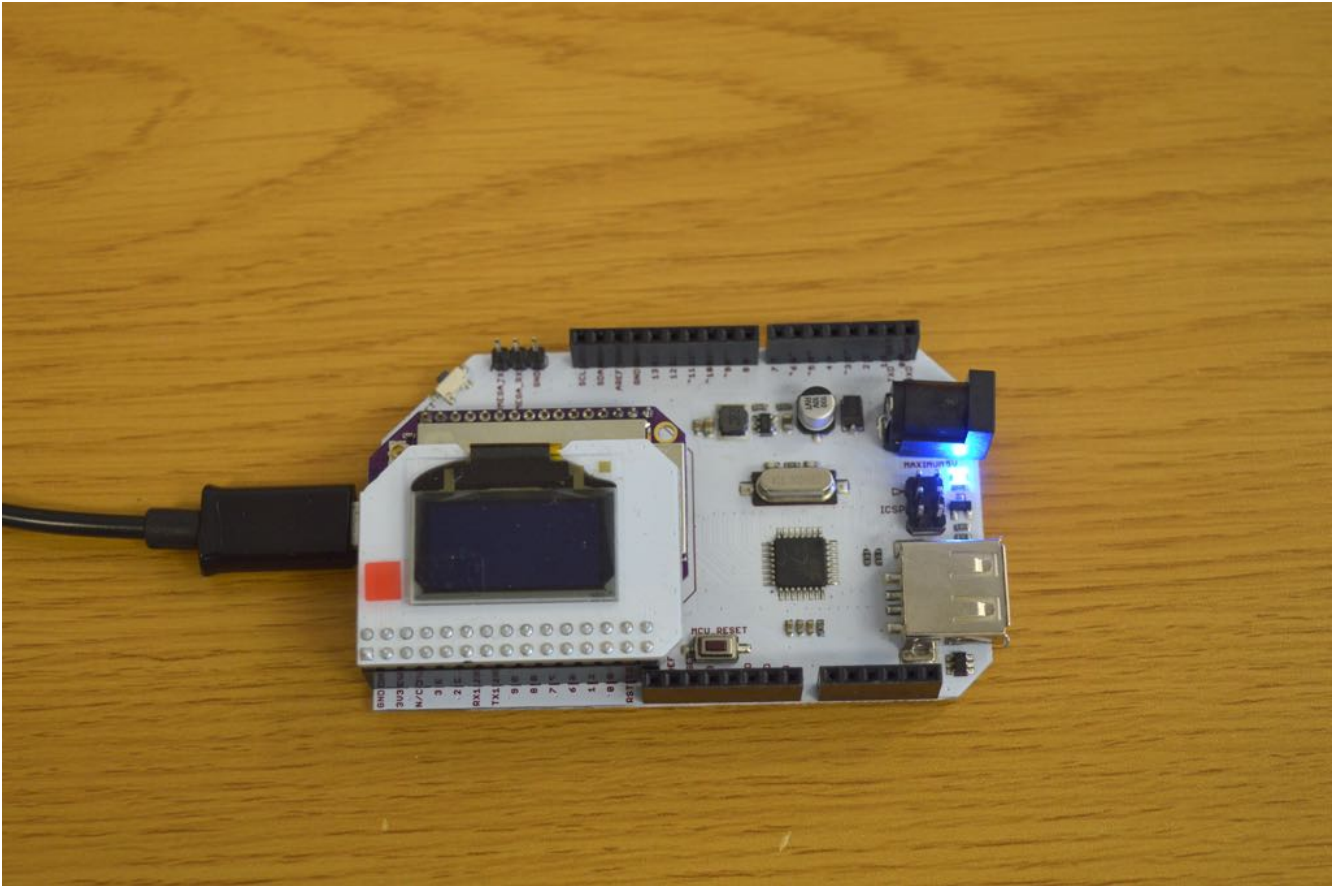
### 1. Prepare

You'll need to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.



Once that's done, plug in your OLED Expansion:





## 2. Install Required Software on the Omega

Connect to the Omega's command line and install Python as well as some of the packages we need:

```
opkg update
opkg install arduino-dock-2 python python-pyserial py0ledExp git git-http ca-bundle
```

The `arduino-dock-2` package installs all the software required to interact with and flash the Arduino Dock. We're also installing the `python` programming language and `python-pyserial`, a module that will allow us to communicate with the microcontroller via the UART1 serial port.

The `git`, `git-http`, and `ca-bundle` packages will allow us to download the project code from GitHub.

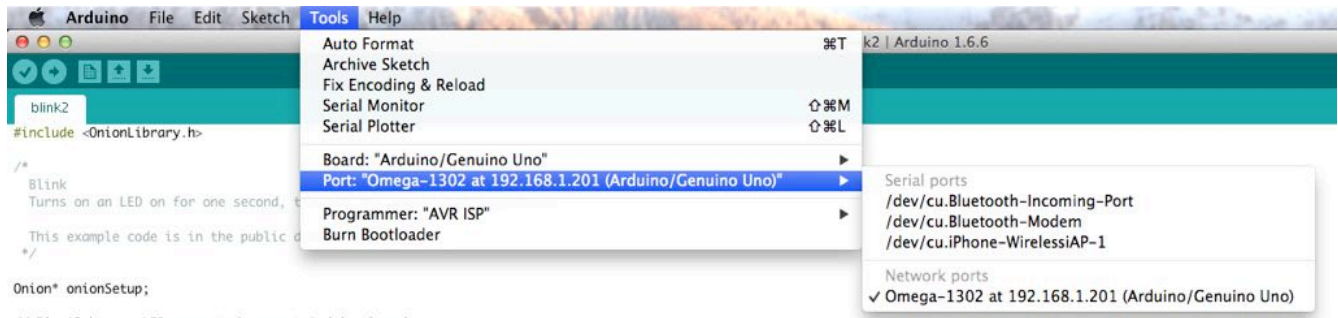
## 3. Arduino IDE Setup

If you don't already have it, install the [Arduino IDE](#) on your computer.

Then you'll need to install the Onion Arduino Library by doing the following:

1. In your web browser, download the [Onion Arduino Library ZIP file](#).
2. Install the ZIP library by following the instructions in the [Arduino Library Installation guide](#).
3. Restart your Arduino IDE to reload the library.

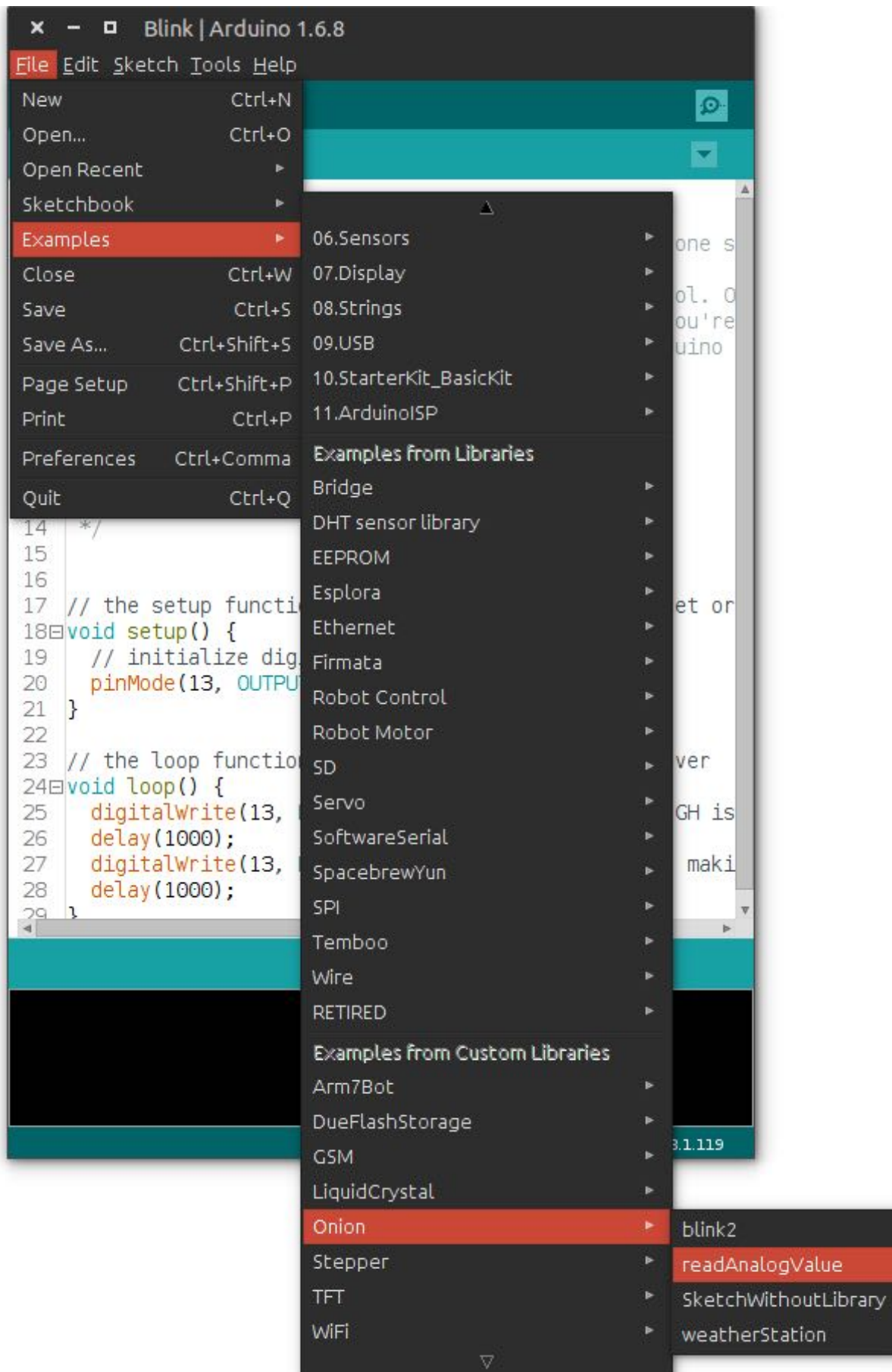
Finally, follow [our Arduino Dock setup instructions](#) to setup the Arduino IDE to wirelessly flash the Arduino Dock 2.



#### 4. Flash the Arduino Dock's Microcontroller

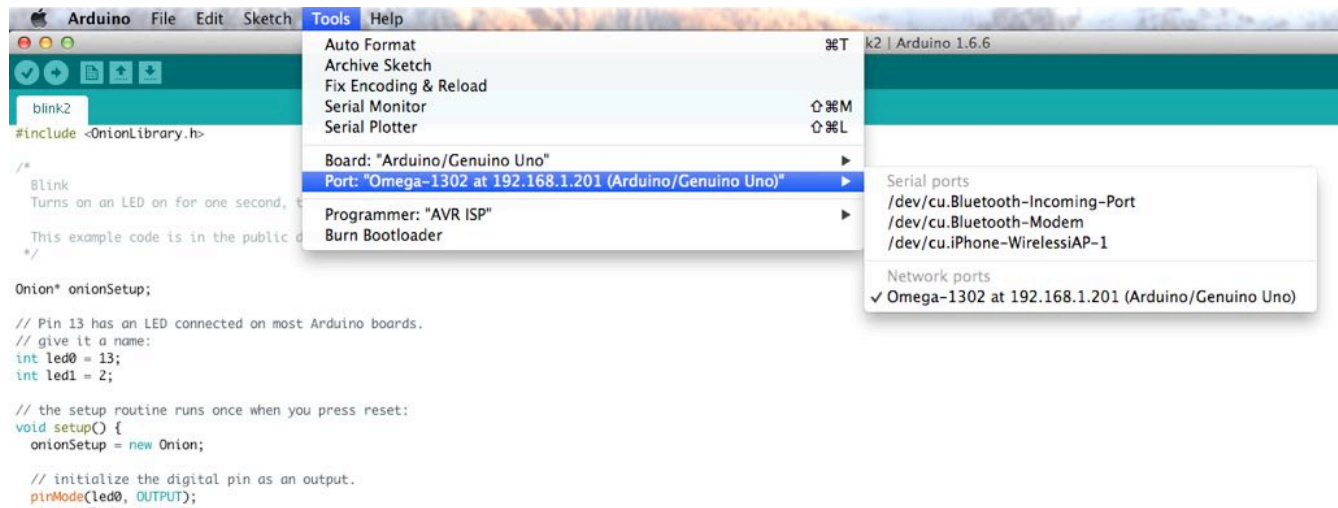
We're going to upload one of the example sketches from the library to the microcontroller on your Arduino Dock.

Go to File -> Examples -> Onion -> readAnalogValue



This sketch will read the signal on Analog pin A0 and will transmit the value via serial if the correct command is received from the other end.

Select your Omega from the listed Network Ports when you open the Tools menu and then Port:



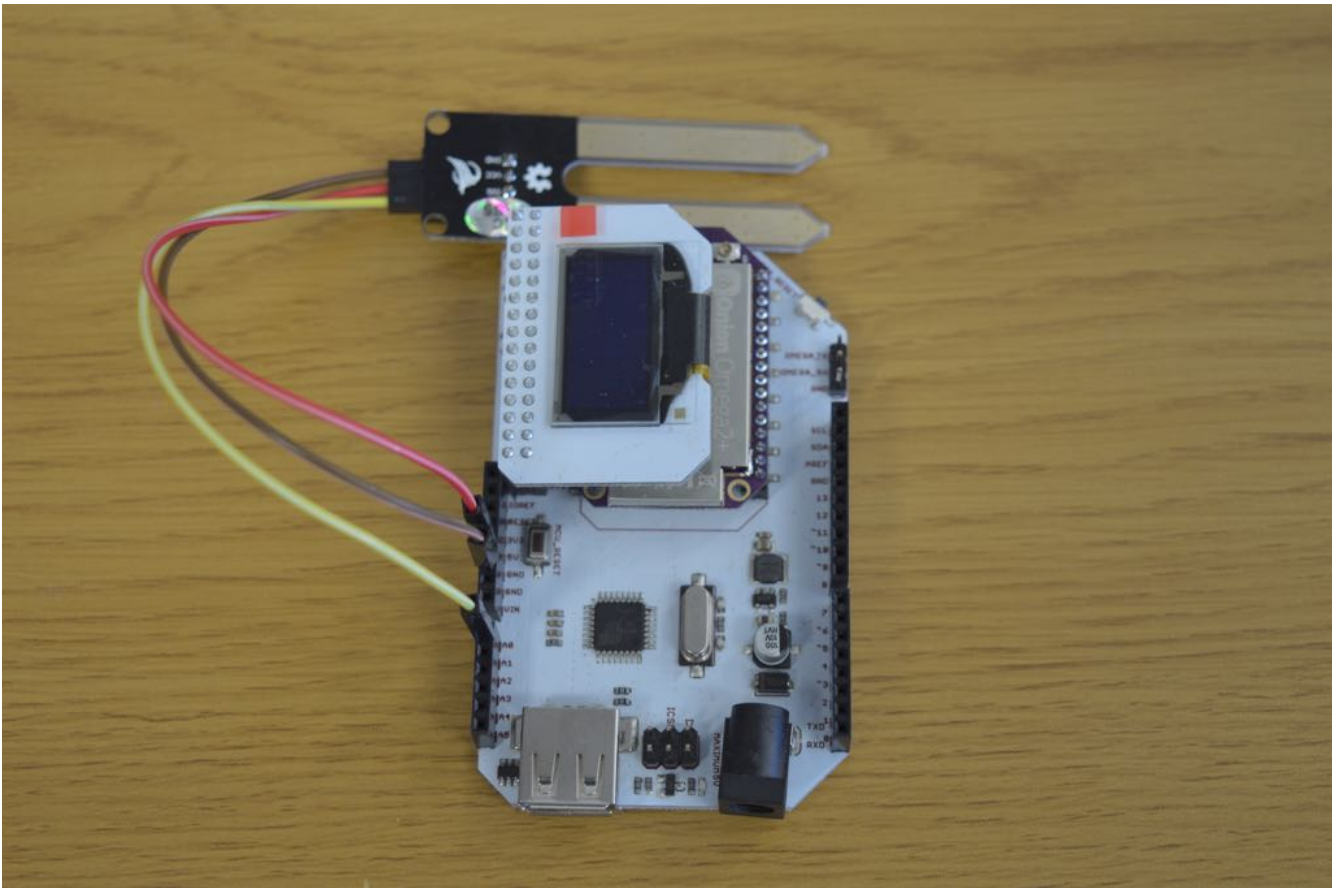
If your Omega doesn't show up in the list of Network Ports, run `/etc/init.d/avahi-daemon restart` and it should show up in about 15 seconds.

Hit the Arrow button to upload your sketch to the Arduino Dock. It will ask for a password during the flashing sequence, this is the Omega's password that it's asking for, by default it is `onioneer`.

See our guide on [using the Arduino Dock](#) for more details on this process.

## 5. Connect the Sensor

There are three connections we'll have to make to wire up the soil moisture sensor to your Arduino:

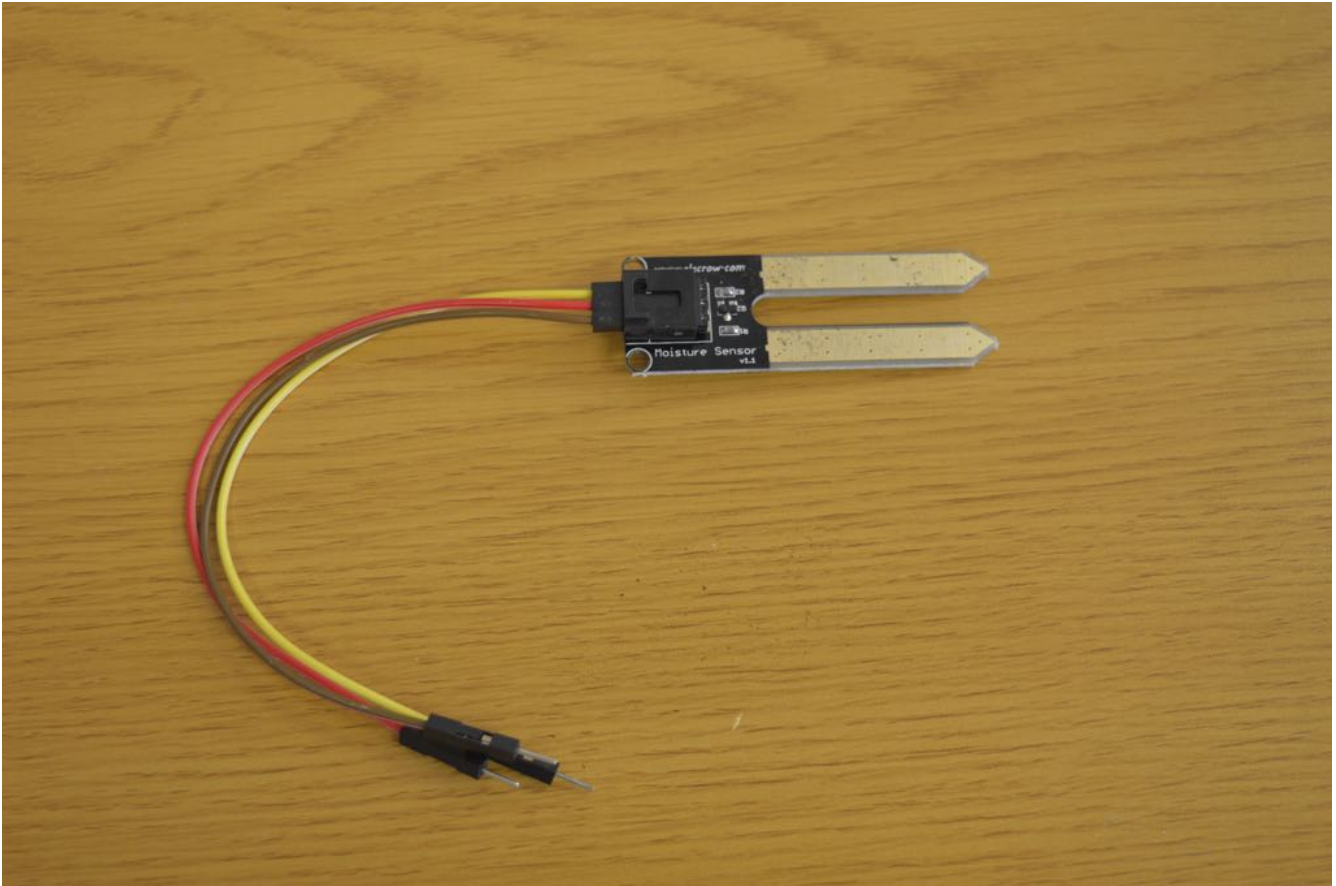


Use jumper wires to make the following connections:

Soil Moisture Sensor Pin	Arduino Dock Pin
GND	GND
VCC	5V
SIG	A0

Depending on where you got your sensor, the labelling might be a little different, but they should all follow the same sort of pattern as above.

Plug in the female ends of the jumper wires into the sensor:



And then the male ends of the jumper wires into the appropriate Arduino Dock pins.

## 6. Sensor -> Plant

To be able to measure the moisture level of the plant's soil, we'll need to put the sensor into the pot!



Insert the sensor into the soil so that the metallic parts are completely covered by the soil.

The sensor isn't super water-proof, so when watering your plant, avoid pouring water directly on the sensor.

## 7. Download the Project Code

The code for the Smart Plant can be found in Onion's [smart-plant repo](#) on Github. We'll use [git to download the code to your Omega](#): navigate to the /root directory on the Omega, and clone the GitHub repo:

```
cd /root
git clone https://github.com/OnionIoT/smart-plant.git
```

Now all of the code will be in the new /root/smart-plant/ directory on your Omega.

## 8. Run the Code

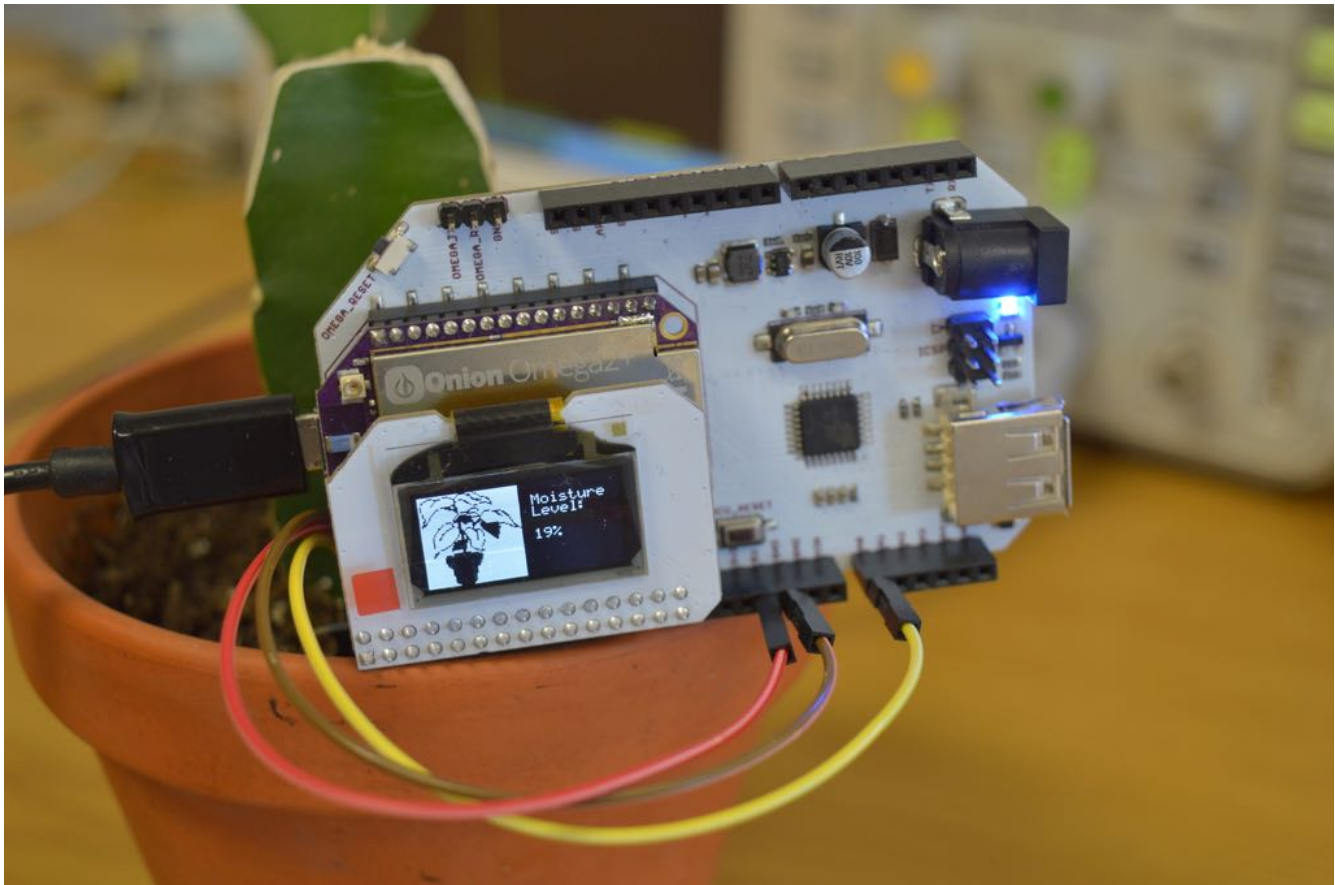
Let's run the code!

```
python /root/smart-plant/smartPlant.py --oled
```

The program will run and collect a moisture level measurement through the Arduino Dock about once a second. The measurement will be displayed on the command line as well as the OLED Expansion:

```
root@Omega-F11D:~# python /root/smart-plant/smartPlant.py --oled
```

```
> Latest measurement: 201
> Measurement List: 19%
>> Average Value: 19%
> Latest measurement: 200
> Measurement List: 19% 19%
>> Average Value: 19%
> Latest measurement: 201
> Measurement List: 19% 19% 19%
>> Average Value: 19%
> Latest measurement: 202
> Measurement List: 19% 19% 19% 19%
>> Average Value: 19%
```



If you don't have an OLED Expansion, leave out the `--oled` part of the command.

The soil moisture sensor is an analog sensor, meaning that the signal it outputs to the Arduino Dock is anywhere from 0V to 5V. The microcontroller sketch will read this as a value between 0 and 1023, where 0 represents 0V and 1023 represents 5V. This number isn't particularly meaningful to us, so on the Omega, we convert it to a percentage. If you take a look at the above example, the first reading is 201,  $201/1023*100\%$  is 19%. And 19% is what we display on the OLED.

### Changing the Program Operation

This part is optional!



To avoid fluctuations, the value from the moisture level sensor is averaged out from the previous 15 readings. This number can be changed with the **command line arguments** that are passed when the program is run.

To have the value averaged out from the 3 latest readings, that is, to make it more reactive to changes, run the following command:

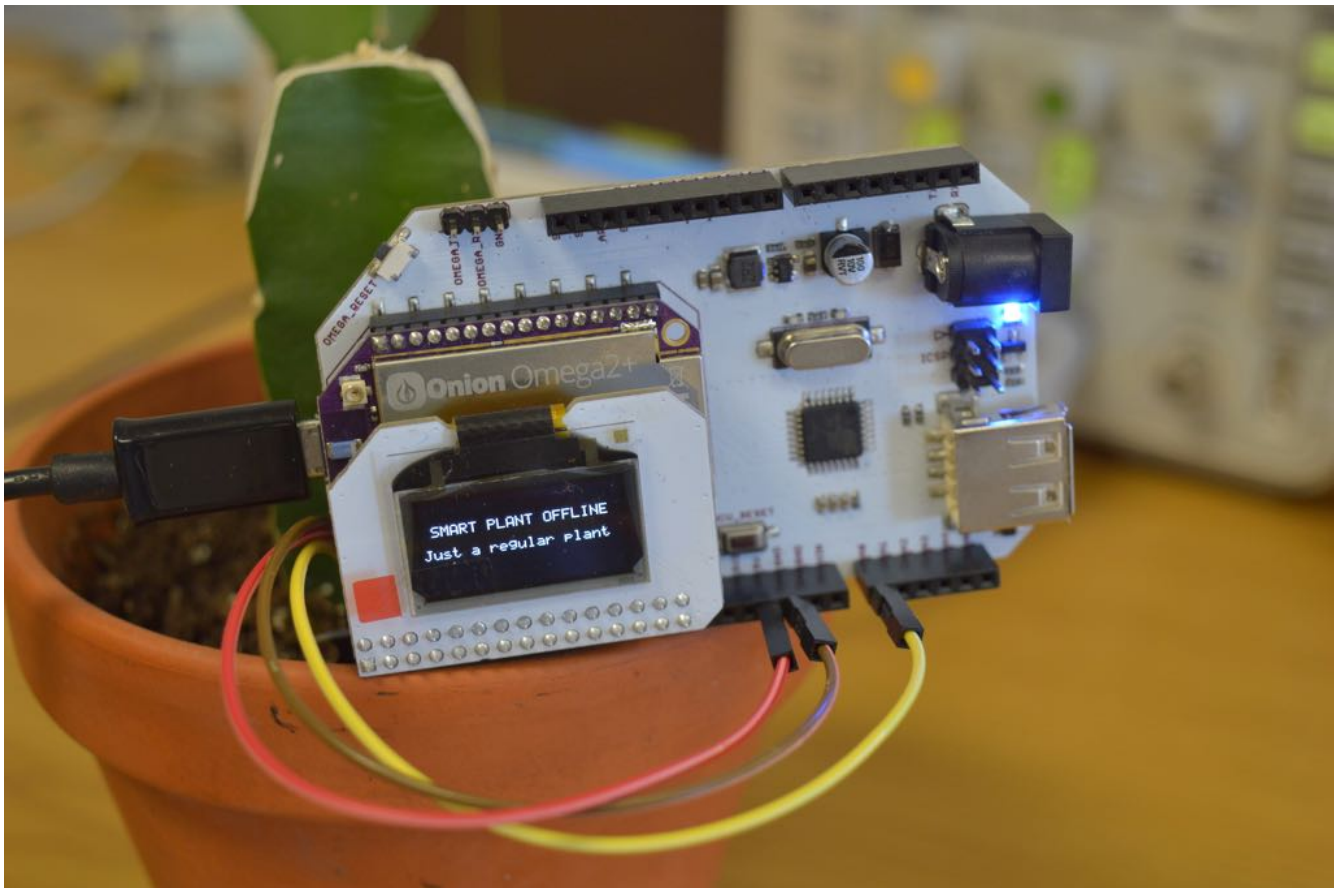
```
python /root/smart-plant/smartPlant.py --oled --number=3
```

If you would like to make the readings less reactive to changes, we can also have the value averaged out from the last 60 readings:

```
python /root/smart-plant/smartPlant.py --oled --number=60
```

### Exiting the Program

Hit 'ctrl+c' to end the program. You'll notice that the OLED will change so that you know your smart plant is now just a regular plant:



## 9. Automate the Program to Run at Boot

To make sure your plant is always smart, we can configure the system so that the smart plant program runs whenever the Omega boots up.

In the project directory, make the `etc/init.d/smart-plant` file executable, copy it into `/etc/init.d`, then enable it by running the following commands:

```
chmod +x etc/init.d/smart-plant
cp etc/init.d/smart-plant /etc/init.d/
/etc/init.d/smart-plant enable
```

The program will now run when the Omega is turned on. Try rebooting your Omega (enter `reboot` in the command line), and you'll see that your program will start up again when the Omega boots up.

## Code Highlight

In this project, the Omega communicates with the Arduino Dock over the serial port. This can be seen in the `measurementHelper.py` module and the Arduino `readAnalogValue` sketch:

```
# read analog value (0-1023) from the microcontroller
# returns None if value is not read successfully
def readMoistureLevel(serialPort):
    # need to write an 'r' character to trigger a measurement response
    serialPort.write('r')

    # read the response
    try:
        value = serialPort.readline()

        if value == "":
            print("Got blank value!")
            value = None
        else:
            value = value.rstrip() #chomp the newline at the end of the response
    except:
        value = None

    return value
```

```
// respond only if correct command is received
    if ((char)inByte == 'r') {
        // respond with analog measurement
        Serial.println(analogValue, DEC);
    }
```

We are also able to easily check and parse command line arguments using Python's `getopt` module in `smartPlant.py`:

```
# read the command line arguments
try:
    opts, args = getopt.getopt(sys.argv[1:], "hvqn:ol:p", ["help", "verbose", "quiet", "number=",
except getopt.GetoptError:
    printUsage()
    sys.exit(2)
for opt, arg, in opts:
    if opt in ("-h", "--help"):
        printUsage()
        sys.exit()
```

```
elif opt in ("-v", "--verbose"):
    VERBOSE = True

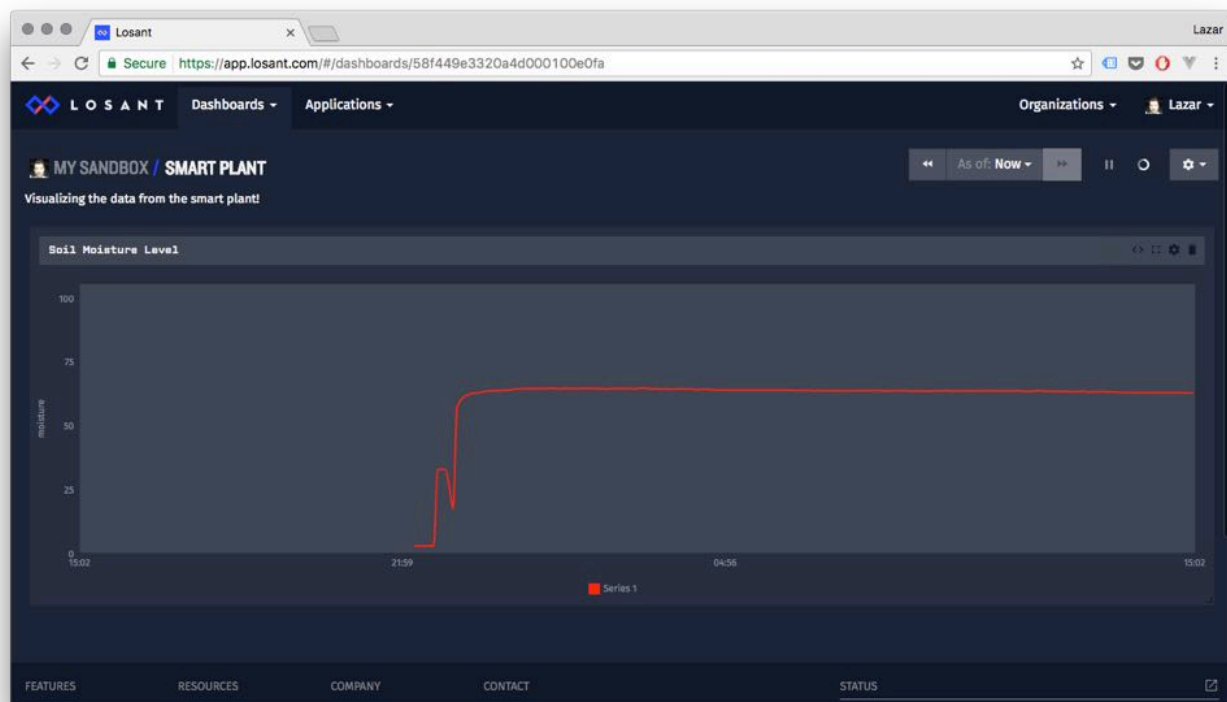
# and so on
```

## Going Further

Next we'll make your plant a little smart by connecting it to a cloud data service so you can remotely monitor it from anywhere in the world!

## Smart Plant - Visualizing Plant Data

This is the second major part of the smart plant project! **Last time**, we setup an Omega to measure the soil moisture level in one of your plants. This part involves sending that data to a cloud service so we can visualize it and open the door to even more possibilities. The cloud service we will be using is the **Losant IoT Platform**.



## Overview

**Skill Level:** Intermediate

**Time Required:** 1 hour

We're keeping the Arduino Dock and analog soil moisture sensor from the first part. Using a command line argument, we will be activating a part of the Python program we didn't use last time. This part of the program will use the MQTT protocol to communicate with Losant.

Losant provides a Python library, `losant-mqtt`, to easily interface devices with their cloud platform. Underneath, the popular `paho-mqtt` module is used to implement the MQTT communication. The same thing can be achieved with just `paho-mqtt` since the Losant module is just a wrapper for `paho-mqtt`, but the Losant module makes the implementation a little easier, so we'll use it.

This project will provide a guide on setting up Losant for our purposes:

- Creating an application
- Creating a device on their platform
- Managing security with Access Keys
- Developing a basic workflow
- Setting up a Dashboard to visualize the smart plant data

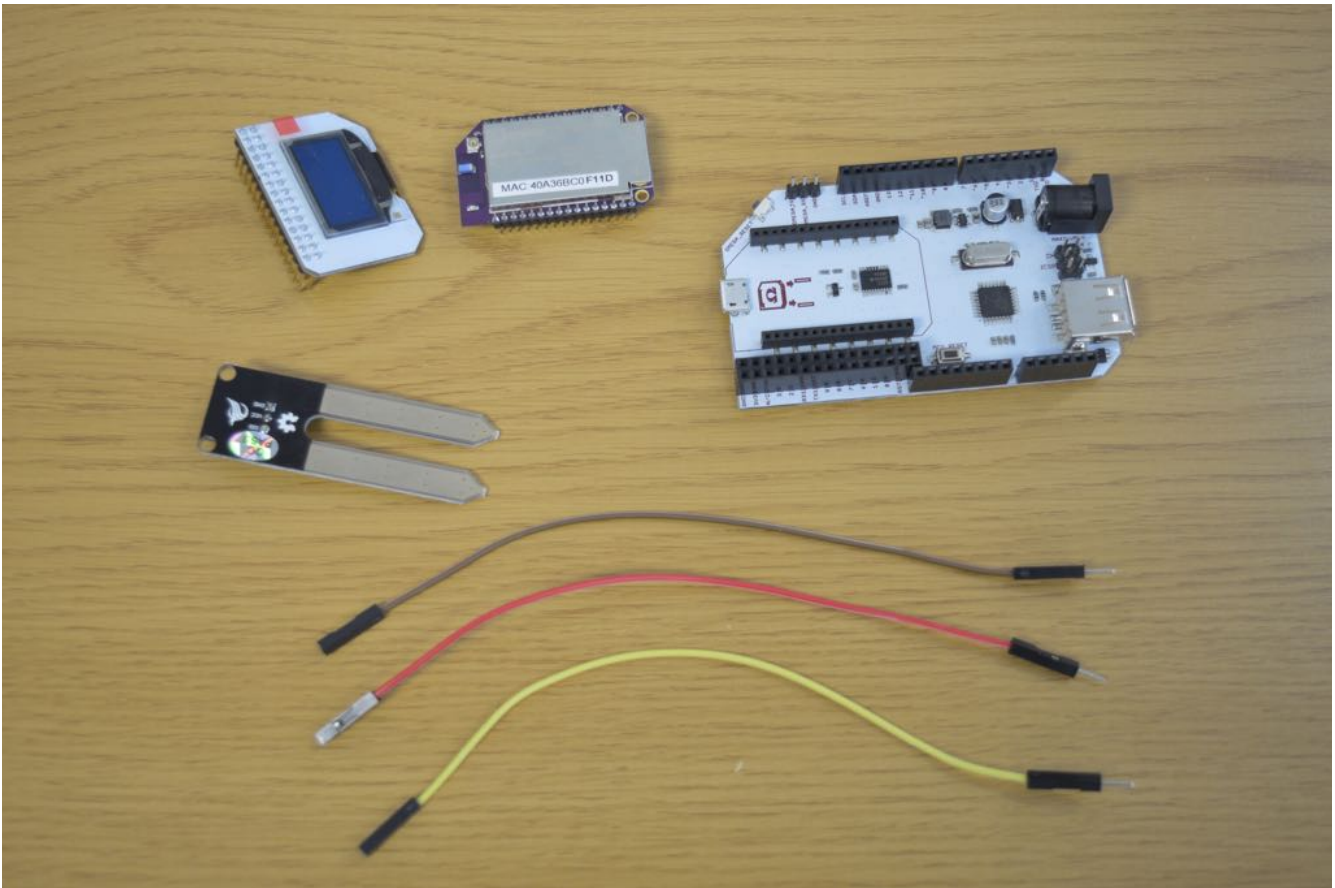
It also shows how to setup an Omega to send data to Losant.

The complete project code can be found in Onion's [smart-plant repo on GitHub](#).

## Ingredients

The same as the first part of the project:

- Onion [Omega2](#) or [Omega2+](#)
- Onion [Arduino Dock 2](#)
- Onion [OLED Expansion](#) (optional but recommended)
- [Soil Moisture Sensor](#)
- 3x [Male-to-Female Jumper Wires](#)



## Step-by-Step

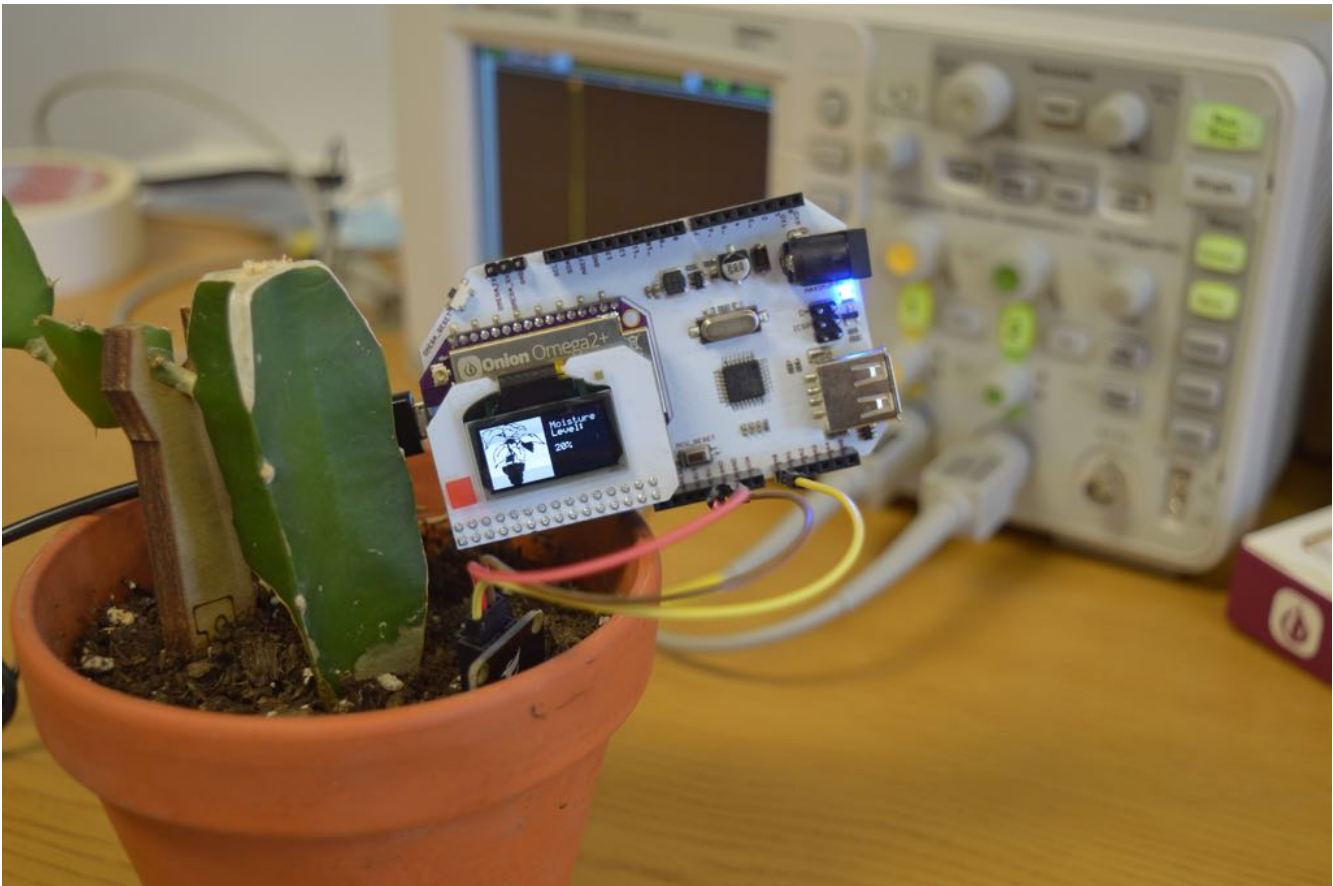
Follow these instructions to set this project up on your very own Omega!

### 1. Prepare

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 2. Complete Part 1 of the Project

This project builds on the first part of the Smart Plant project. If you haven't already completed the [first part](#), go back and do it now!



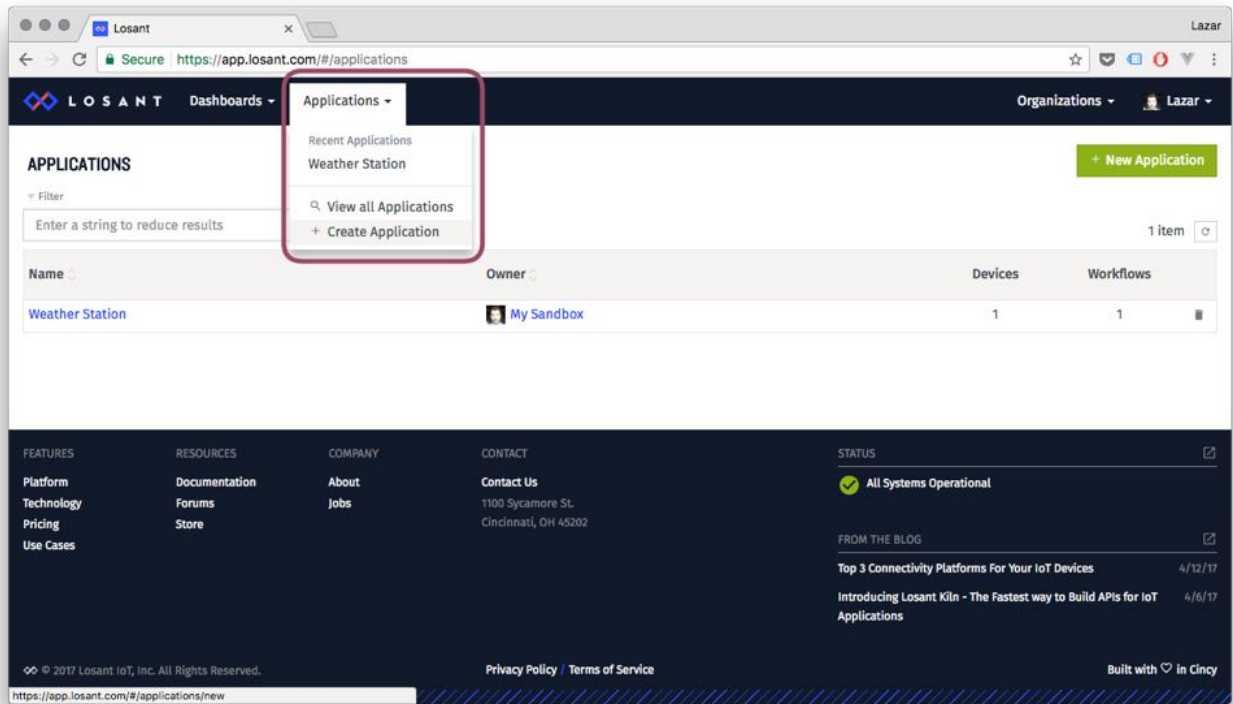
### 3. Register for Losant

Navigate to [Losant.com](https://losant.com) and sign up for their free sandbox tier.

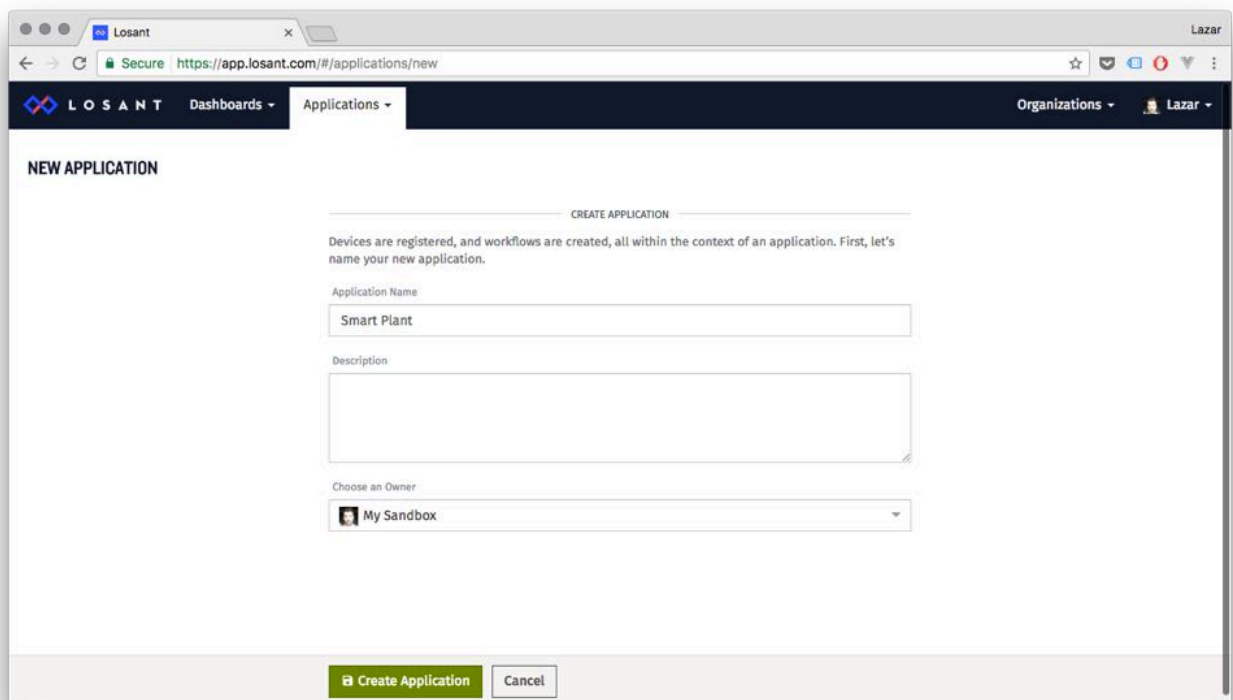
### 4. Create a Losant Application

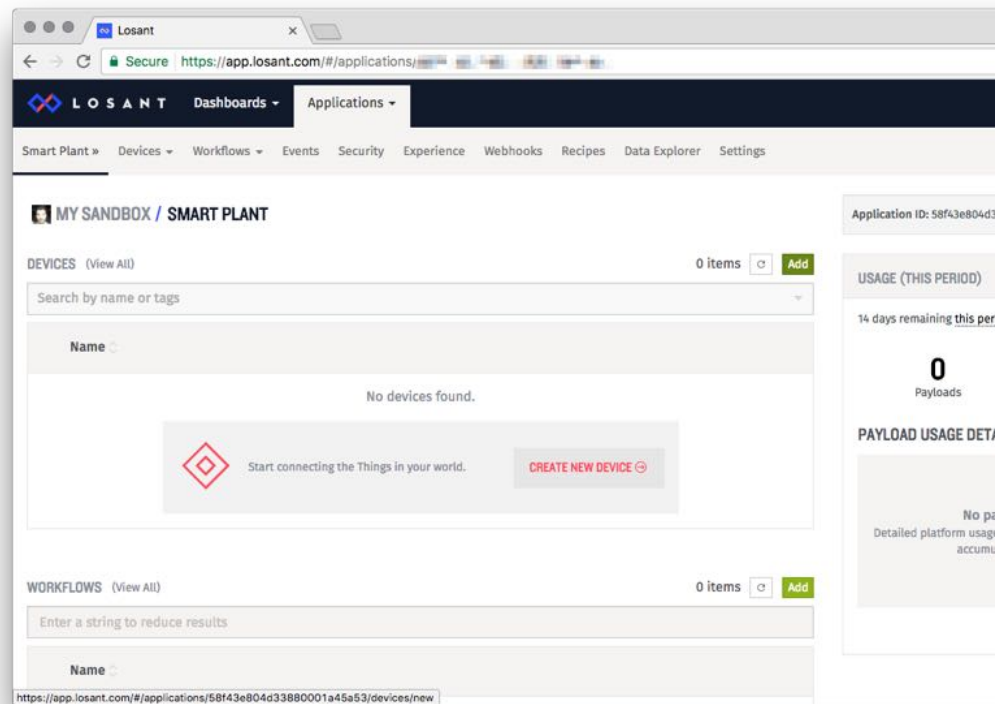
You'll first need to create a Losant **Application** to be able to use their IoT Platform. For more details see the [Losant Application Documentation](#).

Click the Applications Menu, and then Create Application:



Give your Application a name, Smart Plant was our choice:





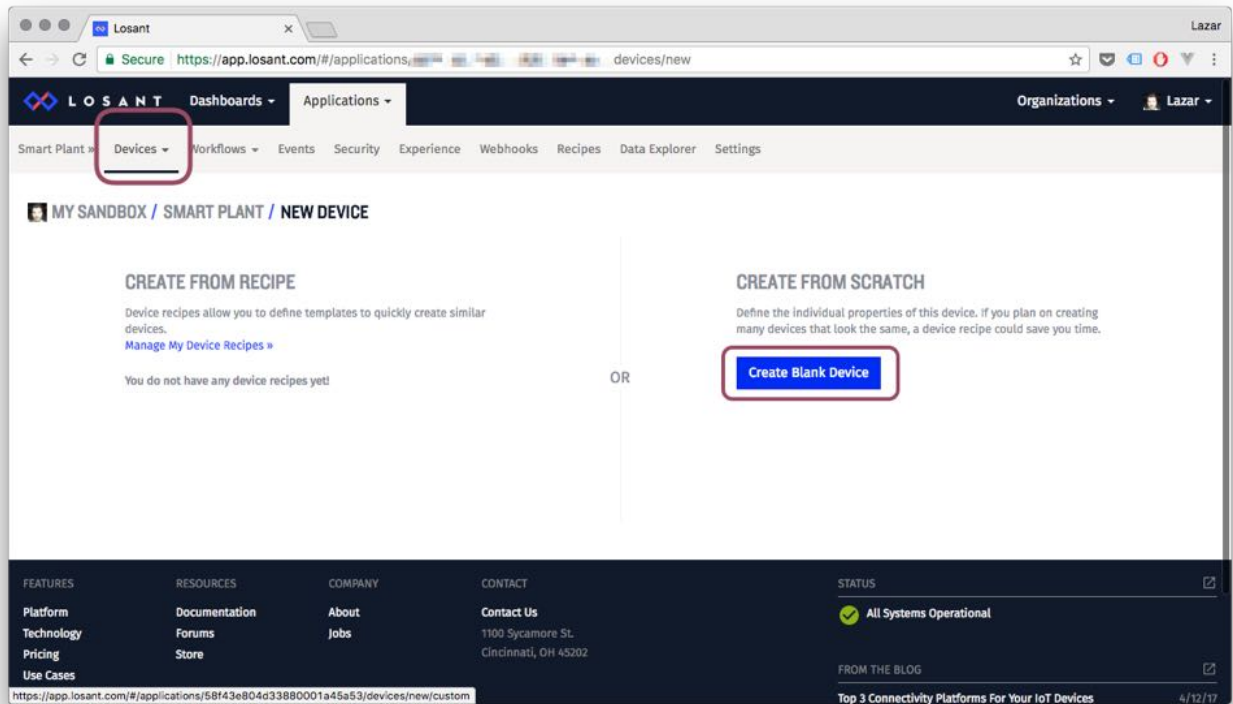
Ok, now your application is created

## 5. Create a Losant Device

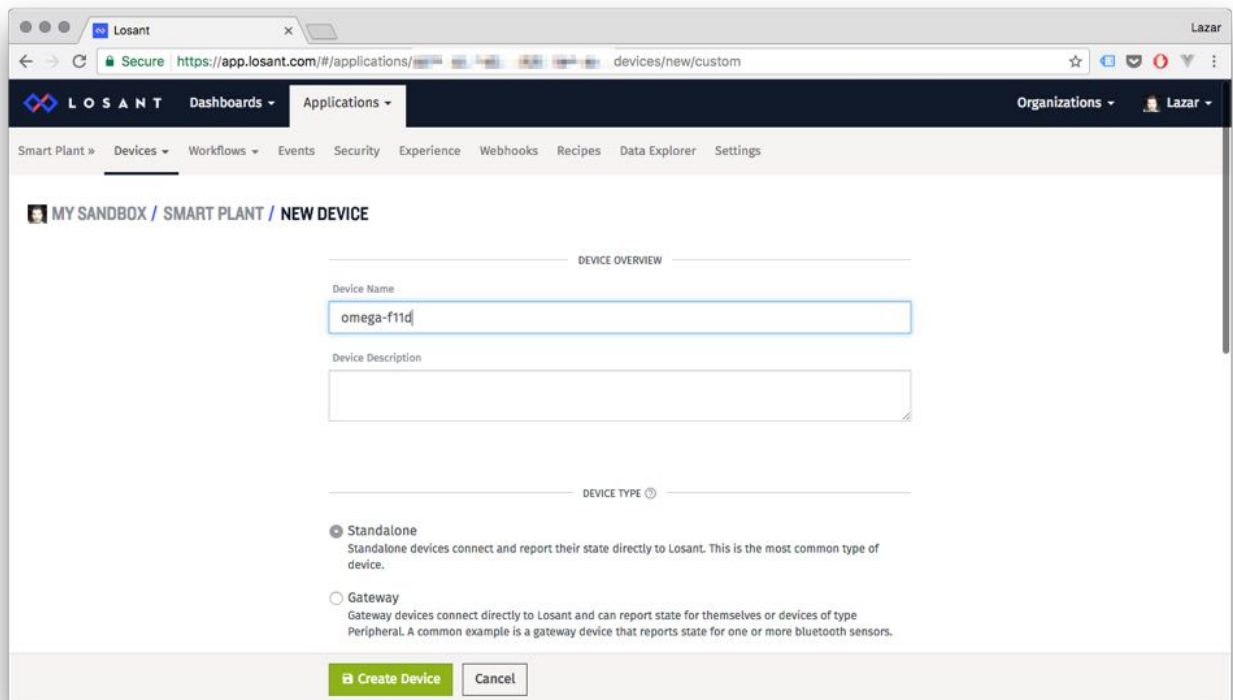
Now that you have an Application, you'll need to create a **Device**. A device on Losant can send data to the cloud and receive commands from the cloud. For more details see the [Losant Device Documentation](#).

Click the **Devices** menu, and then **Create a Blank Device**:



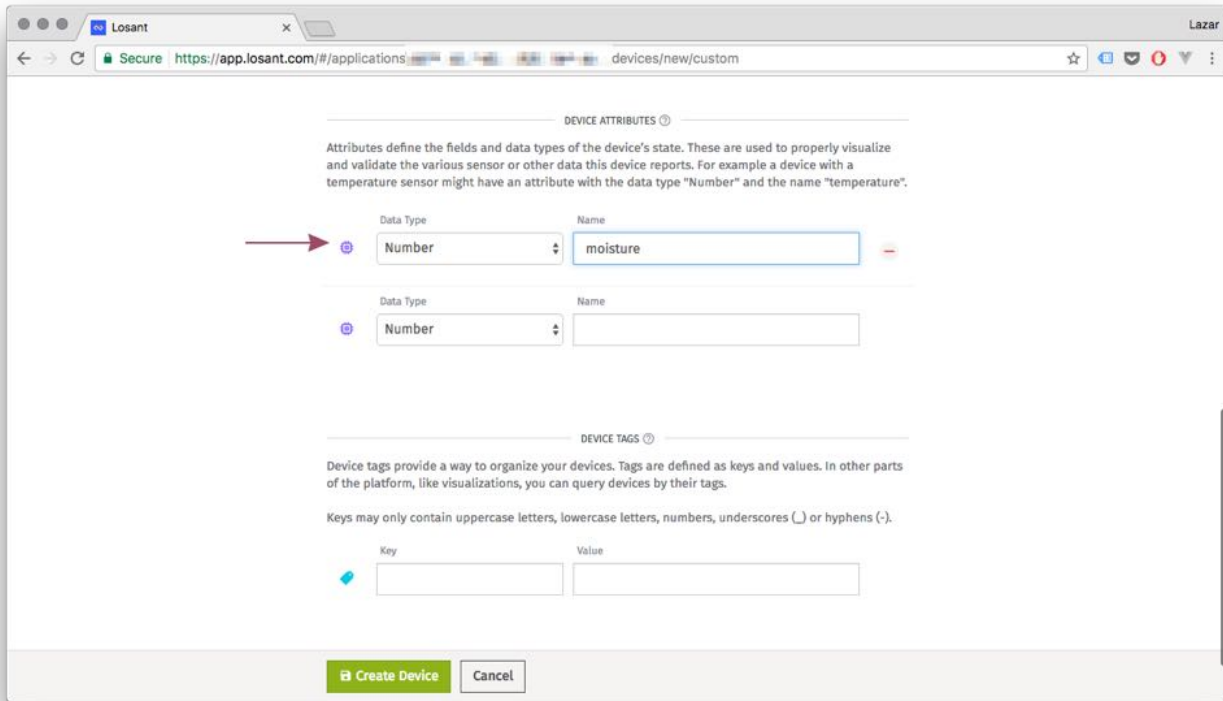


Give your device a name, we found it easiest to use the name of the Omega we're using to measure the soil moisture levels. Make sure the Device Type is set to **Standalone**:

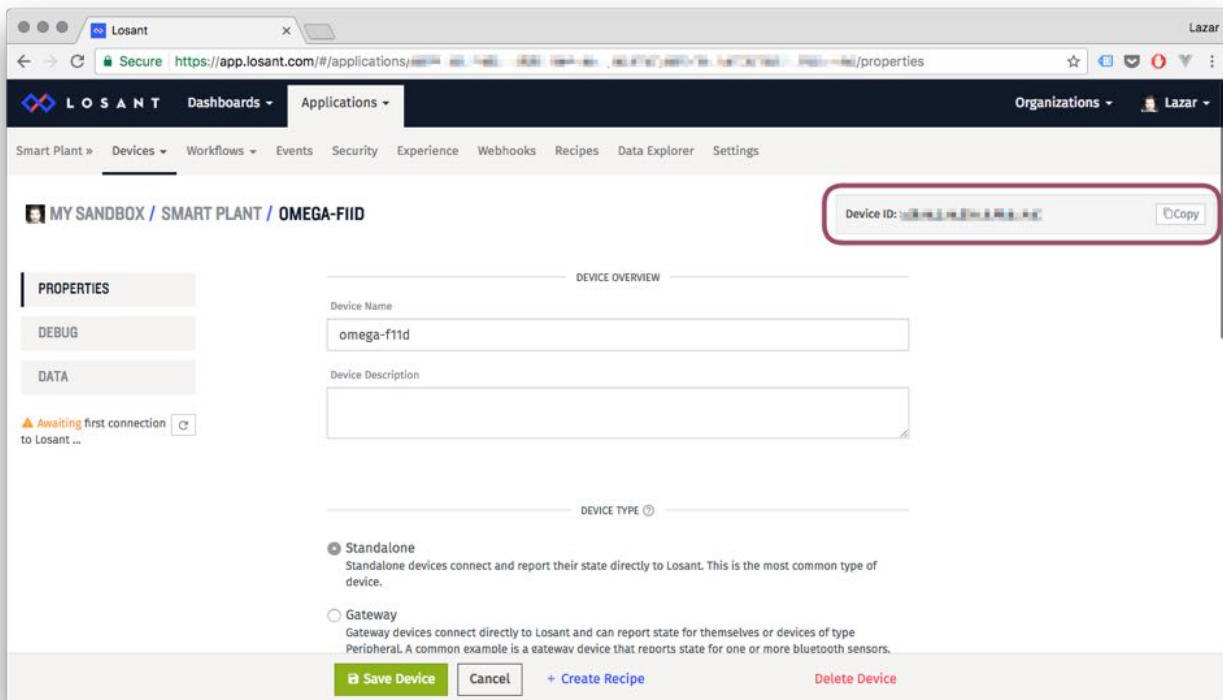


Scroll down and add a **Number** attribute named `moisture` to the device. This attribute will hold the soil

moisture level data coming from your Omega:



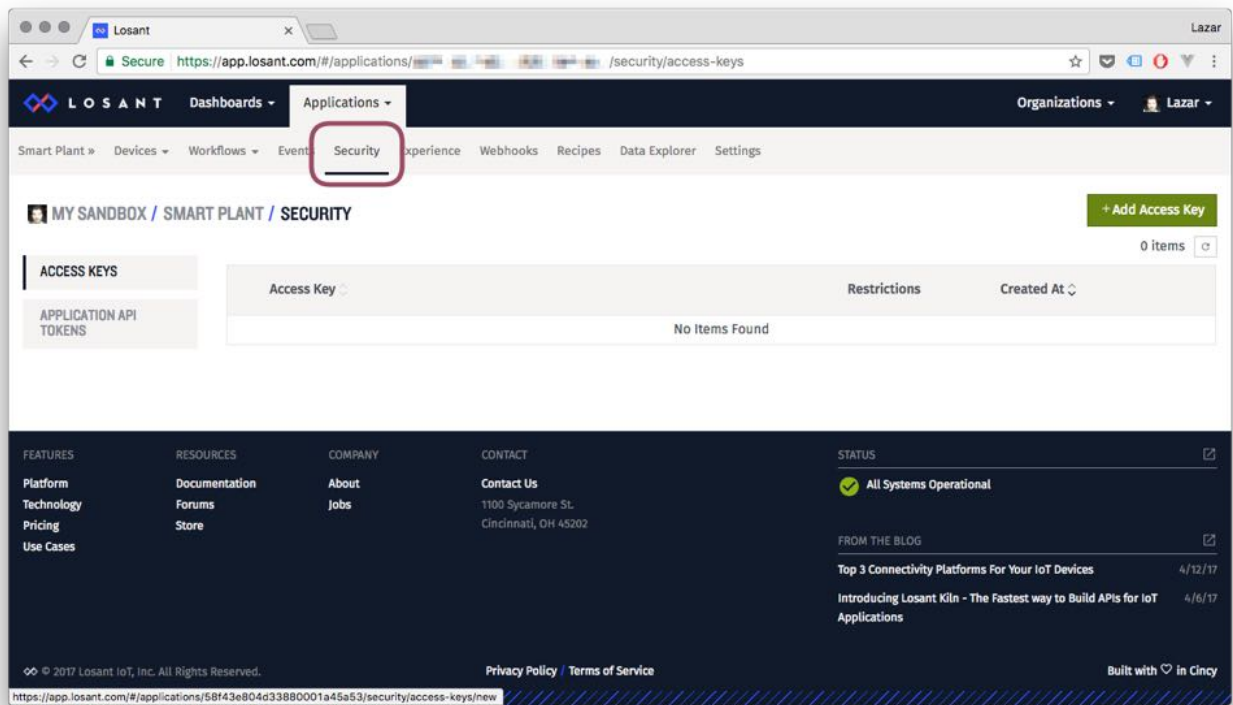
Hit Create Device and your device will be ready to go. Note the Device ID on the upper right hand side:



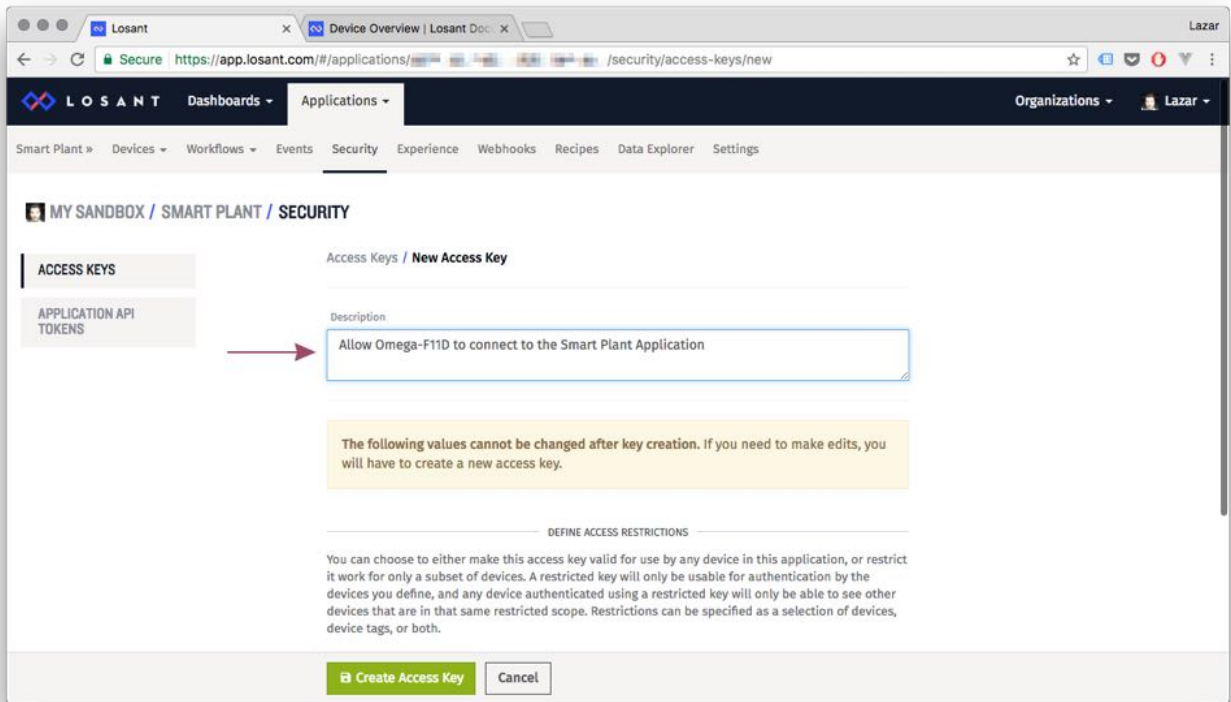
## 6. Create a Losant Access Key

To actually get your Omega to communicate with Losant, it will need to authenticate. To carry out that authentication, we'll use an Access Key. For more details see the [Losant Access Key Documentation](#).

Click the Security Menu and then the Add Access Key button:

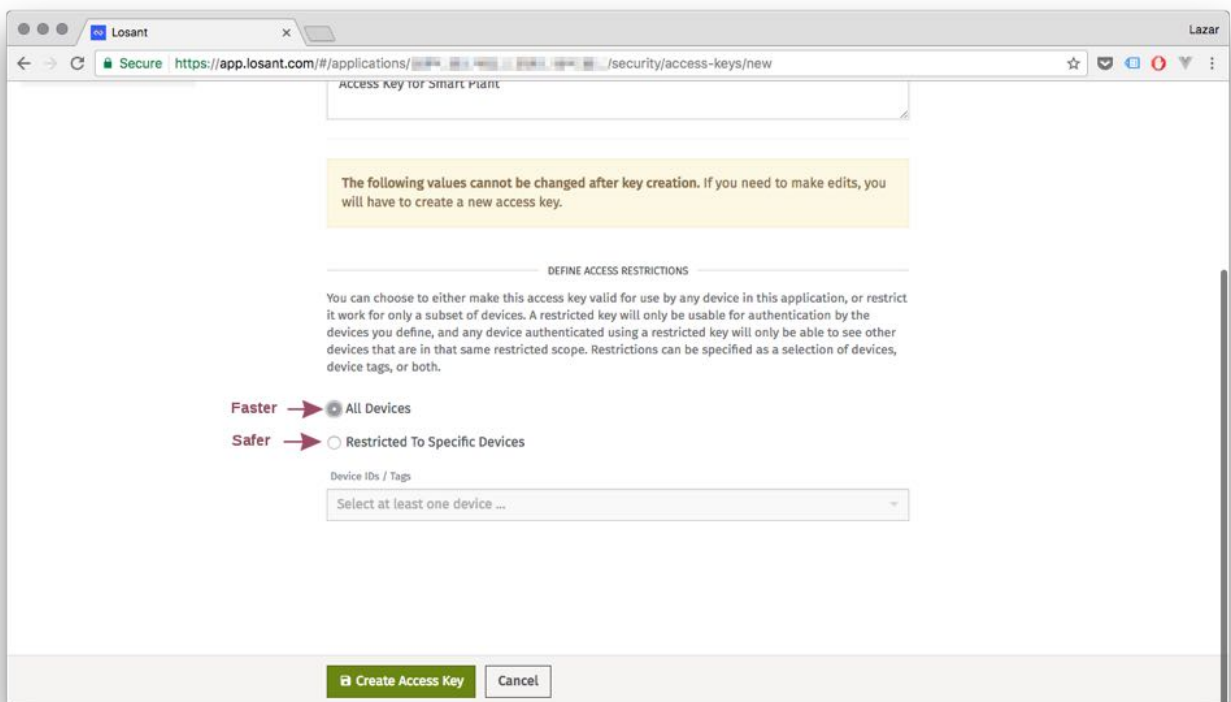


Give the key a description:

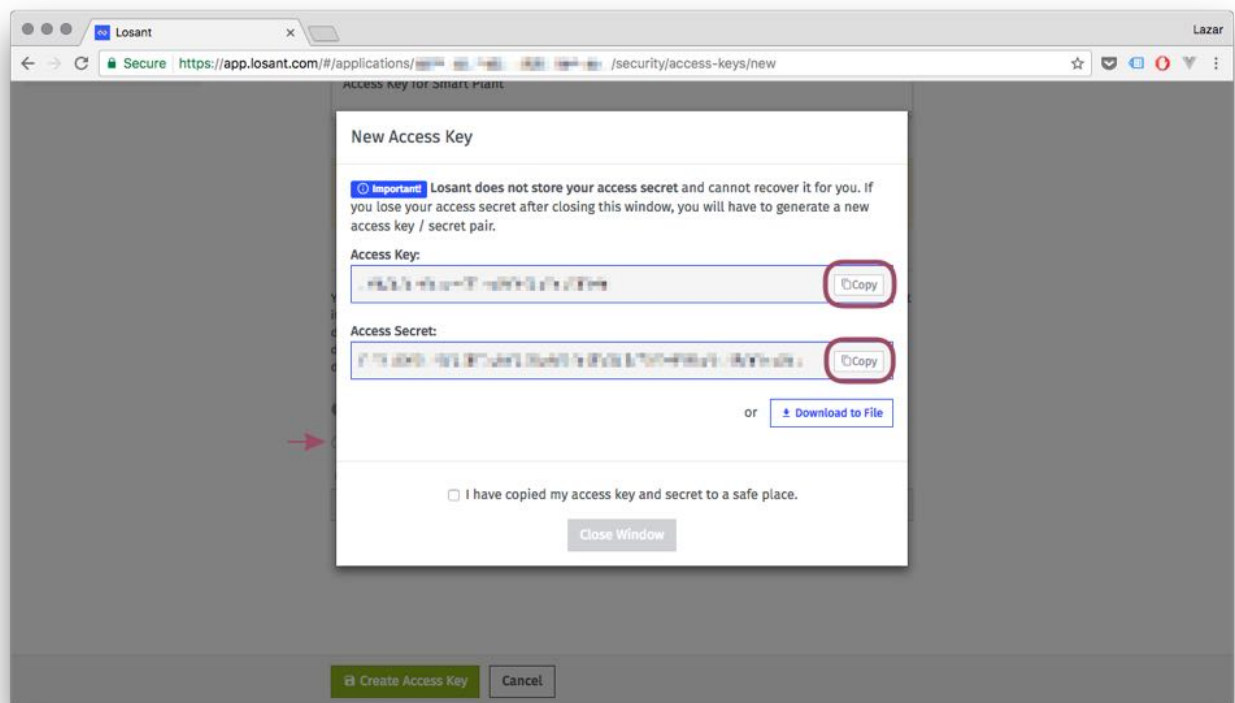


We're setting the Key to authenticate **all** of our devices on Losant. This is slightly risky, if you like, you can change it so that the access key is restricted to just a single device.

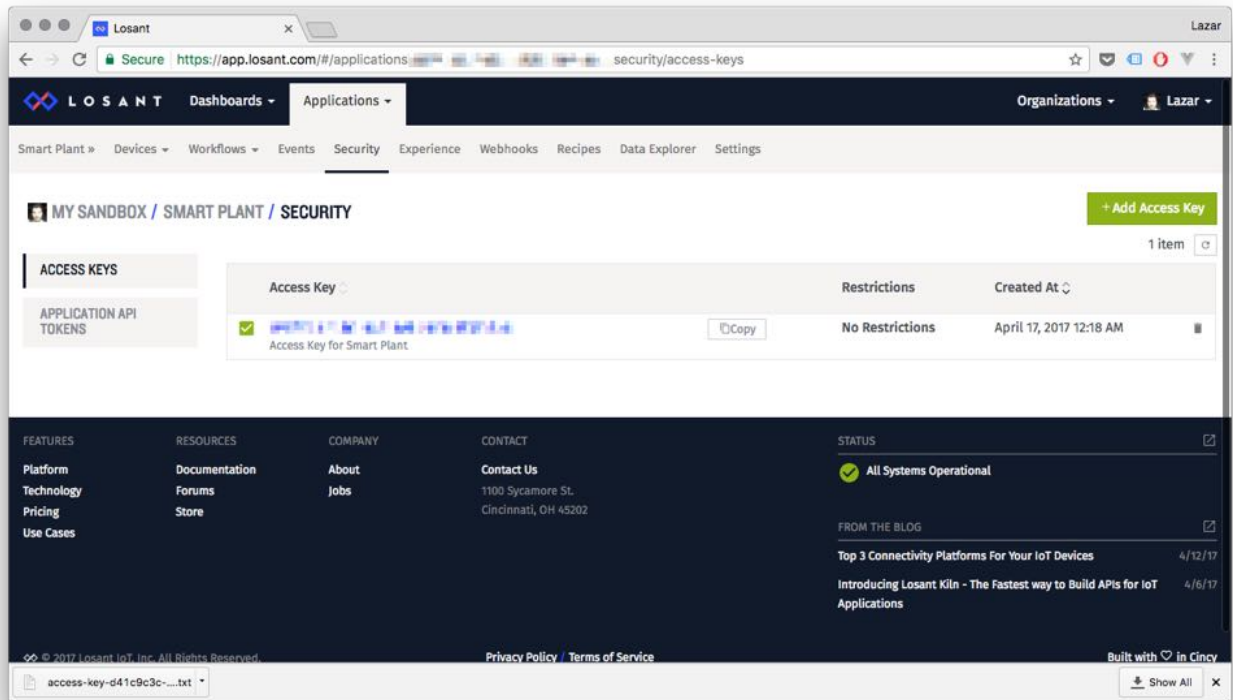
In either case, hit the **Create Access Key** button:



The access key and secret have now been generated! **Make sure to note them both down because this will be the only time you will get to see the Secret!** We recommend downloading to a file.



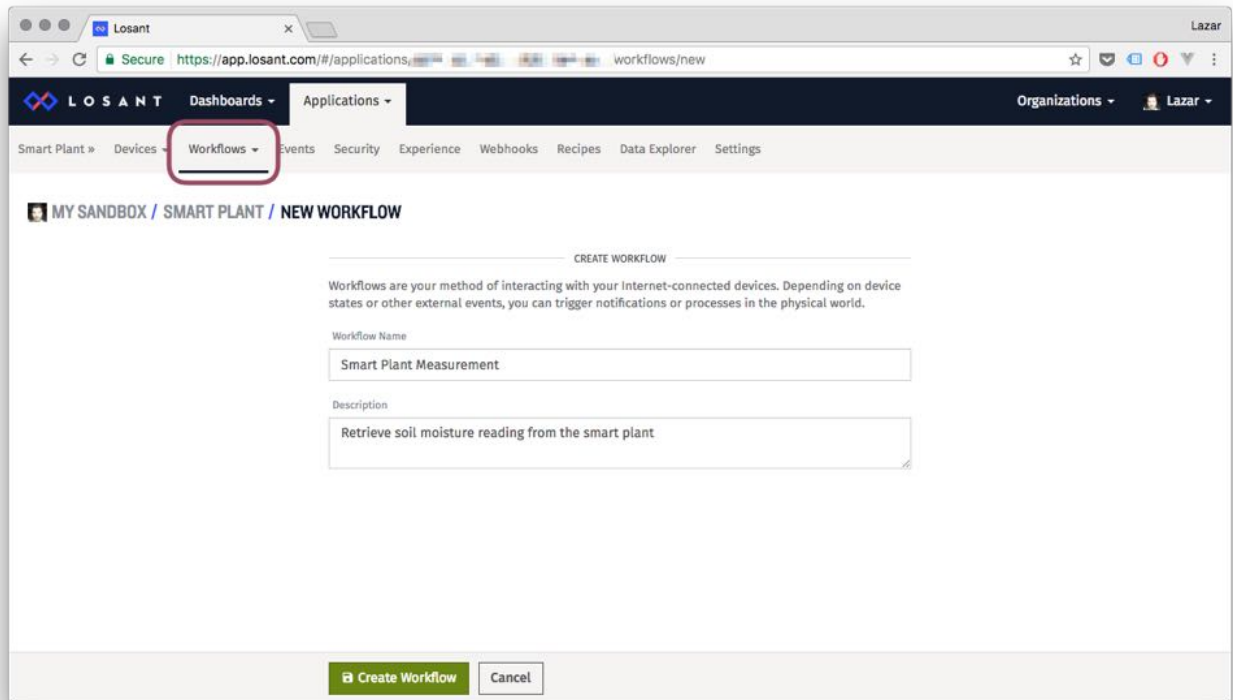
Alright! Your Access Key is ready to go!



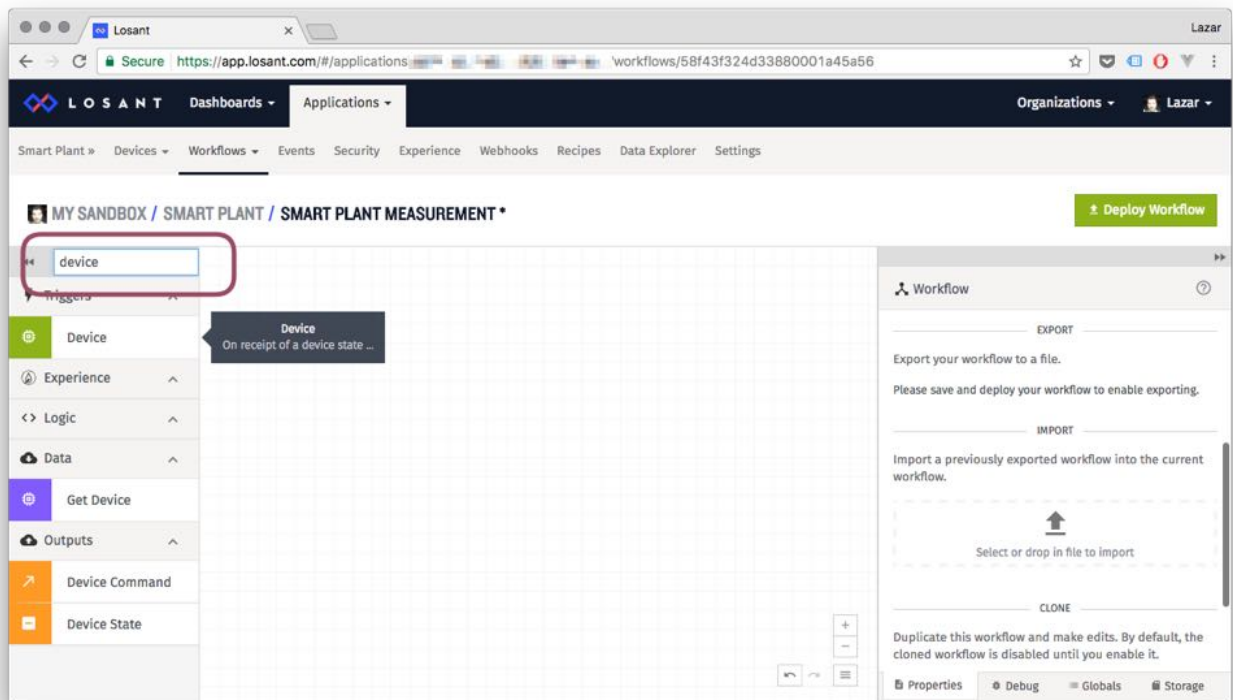
## 7. Create a Losant Workflow

Now we need to make a [Losant Workflow](#) so our device can interact with the rest of Losant. For now, our workflow will be a simple tool for debugging. These workflows are easy to use and very powerful, definitely check out [Losant's workflow documentation](#) to learn more.

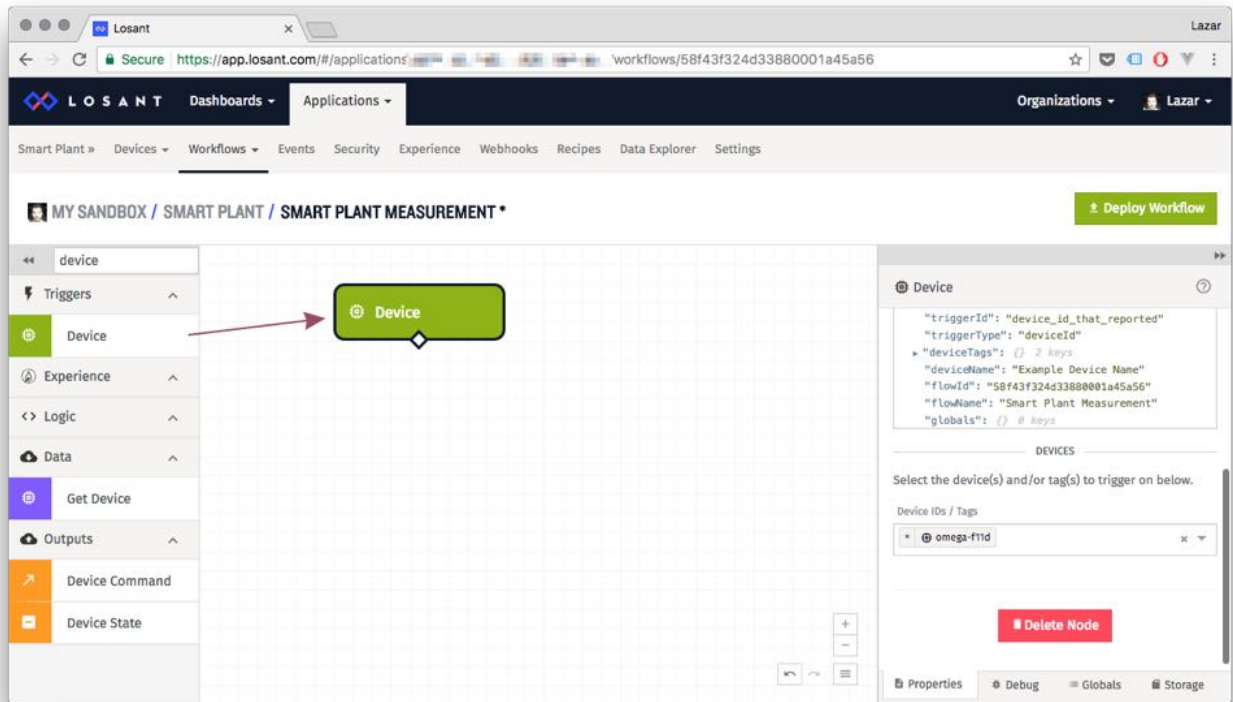
Click on the **Workflows** menu and then **Create Workflow**. Give your workflow a name and a description:



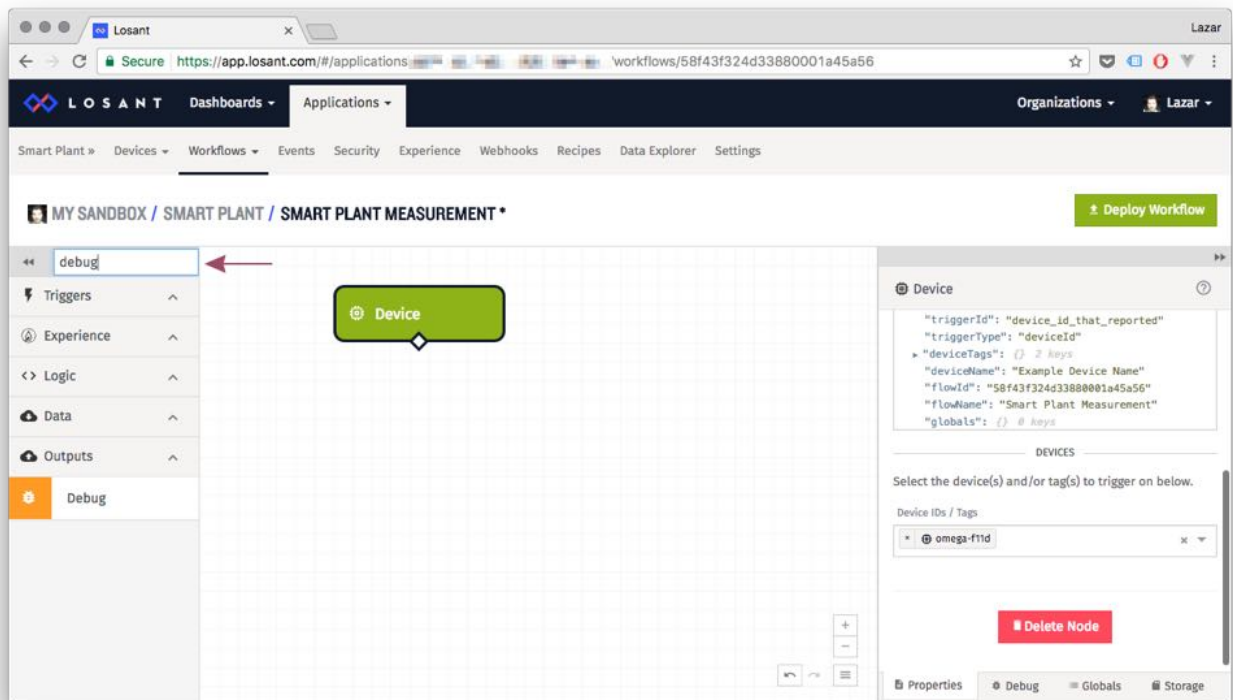
Type device into the search bar, then click and drag the Device block onto the area in the middle:



Click on the new Device block and scroll down in the right-side toolbar to select which of your devices will be associated with this Device block. You'll want to select the device you just created:

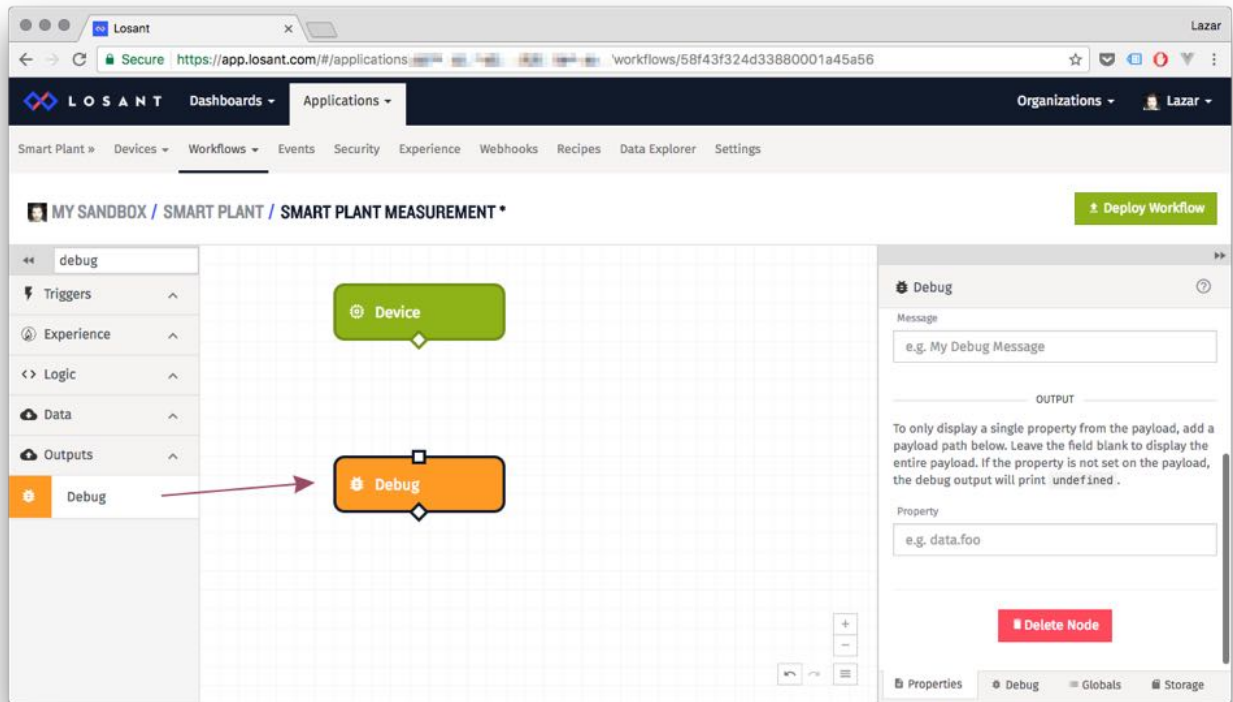


Now, let's add a Debug box so we can view the data coming from our device. Type `debug` into the search box:

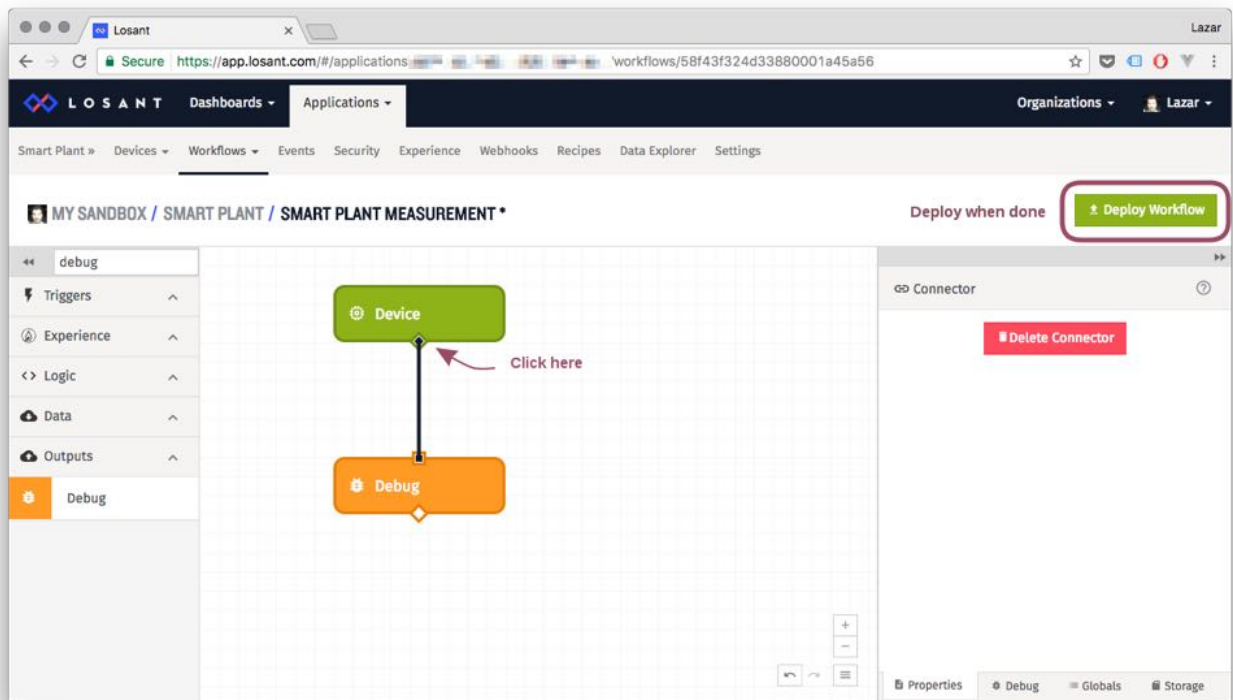


Drag the Debug block into the area in the middle:

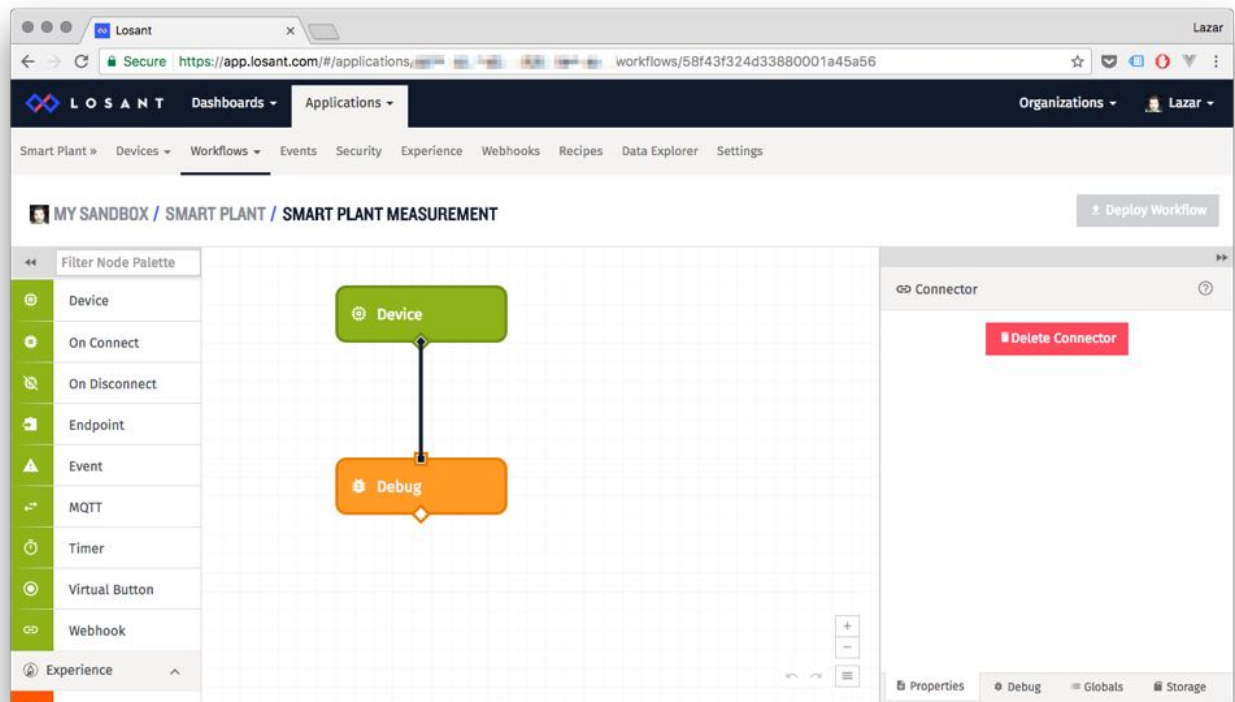




We now need to connect the two blocks. Click on the diamond at the bottom of the Device block and drag it down to the square at the top of the Debug block, and then hit the Deploy Workflow button:



Your workflow is now saved and deployed, meaning that it's running at this very moment on the cloud:



## 8. How does my Omega talk to Losant?

Glad you asked! There's some additional software and configuration we need to do on the Omega to get it to report the soil moisture level to Losant on a regular basis. Behind the scenes, the Python script will be using MQTT to communicate with Losant.

If you haven't completed the [first part of the Smart Plant project](#), you need to go back and do it now before proceeding. The remainder of the steps assume that the `smart-plant` code can be found at `/root/smart-plant` on your Omega.

## 9. Install Required Software on the Omega

To install additional Python packages, we'll need to install the Python package manager, PIP. [Connect to the Omega's Command line](#) and run the following command:

```
opkg update
opkg install python-pip
```

Now that PIP is installed, we'll first fix an issue with the `setuptools` module, and then install two modules.

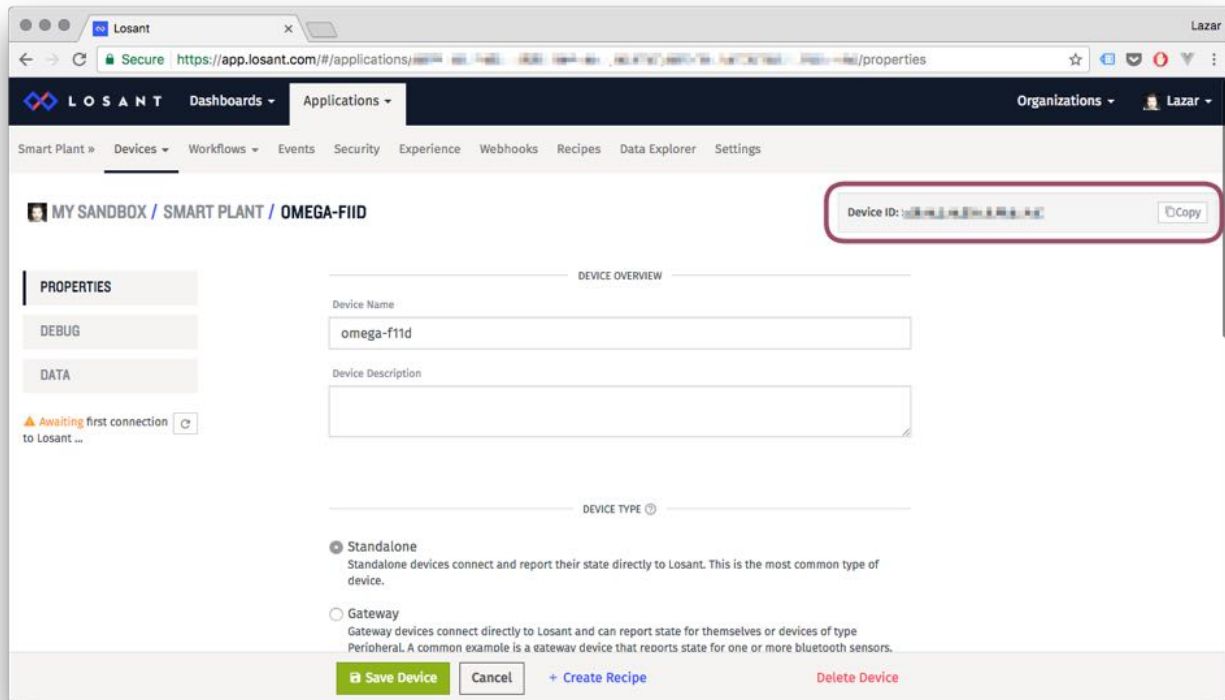
```
pip install --upgrade setuptools
pip install paho-mqtt
pip install losant-mqtt
```

The `paho-mqtt` module provides MQTT functionality, and `losant-mqtt` provides an easy to use interface for connecting to Losant.

## 10. Setup Losant Credentials on Omega

In order to connect and authenticate with Losant, the Smart Plant program will need to know the Device ID of your Losant Device, as well as your Access Key and Secret.

Look at your device on Losant to get the Device ID:



The Access Key and Secret you should have noted down somewhere when you created the Access Key. If you don't have the Access secret noted down somewhere, you'll have to create a new Access Key!

Now on your Omega, in the `/root/smart-plant` directory, there is a configuration file template called `losant.json`. Open it for editing and enter your `deviceId`, `key`, and `secret` to complete the setup.

## 11. Stop the Existing Program

In the first part of the project, we added a script to `/etc/init.d` to automatically run the smart plant program on boot. We'll now need to stop the program before running it again manually.

Run the following command:

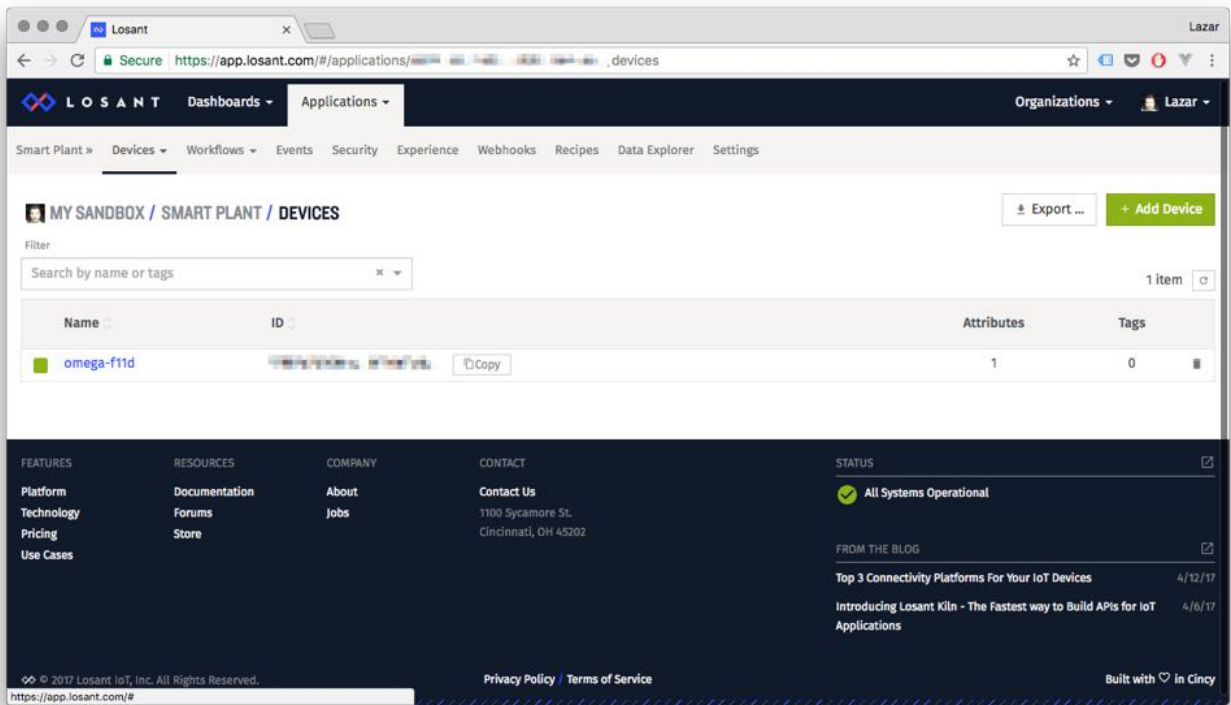
```
/etc/init.d/smart-plant stop
```

## 12. Report the Smart Plant Data to Losant

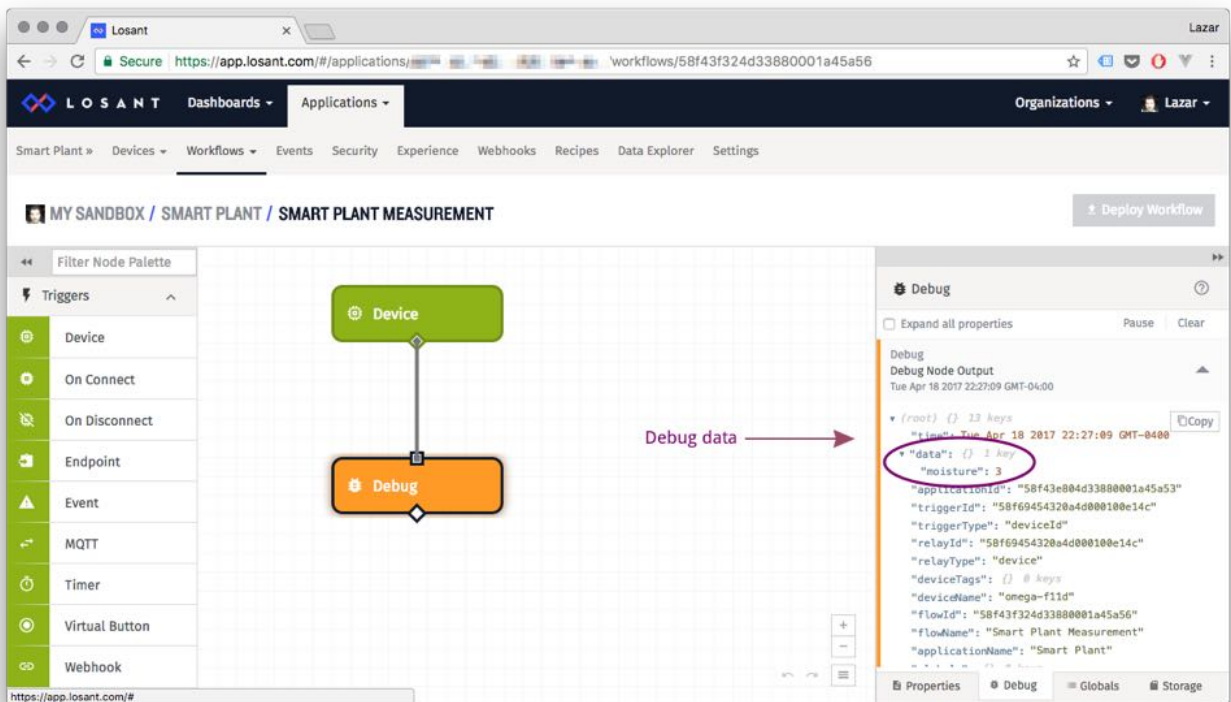
Let's try running the Smart Plant program with reporting to Losant enabled. We're also providing the path to the Losant configuration file:

```
python /root/smart-plant/smartPlant.py --oled --quiet --losant /root/smart-plant/losant.json
```

Assuming your Losant credentials are valid, you should see your device come online on Losant:



Going to the workflow, we can take a look at the Debug block's Debug output to see the data coming from the Omega:



If you look at the Python code, you'll see that what we send from the Omega is showing up in the debug window on Losant:

```
{
  "data" {
    "moisture": "<MOISTURE LEVEL READING>"
  }
}
```

Hit `ctrl+c` to stop the program.

### 13. Update Program Run at Boot

Since we now need to run the Smart Plant program with additional arguments to talk to Losant, we'll need to update the `/etc/init.d/smart-plant` file.

Open the `/etc/init.d/smart-plant` file using the Vi editor: `vi /etc/init.d/smart-plant`. Hit `i` and use the arrow keys to navigate to the line that says `BIN="/usr/...`. After the `--quiet` text, insert a space and type the following:

```
--losant /root/smart-plant/losant.json
```

and insert a space between the `json` and the `>` character.

Your `smart-plant` file should now look like this:

```
#!/bin/sh /etc/rc.common
# Copyright (C) 2016 Union Corporation
START=99

USE_PROCD=1
BIN="/usr/bin/python /root/smart-plant/smartPlant.py --oled --quiet --losant /root/smart-plant/lo

start_service() {
    procd_open_instance
    procd_set_param command $BIN
    procd_set_param respawn
    procd_close_instance
```

Hit `esc` and type `:wq` to save and close the file.

Then re-enable the program:

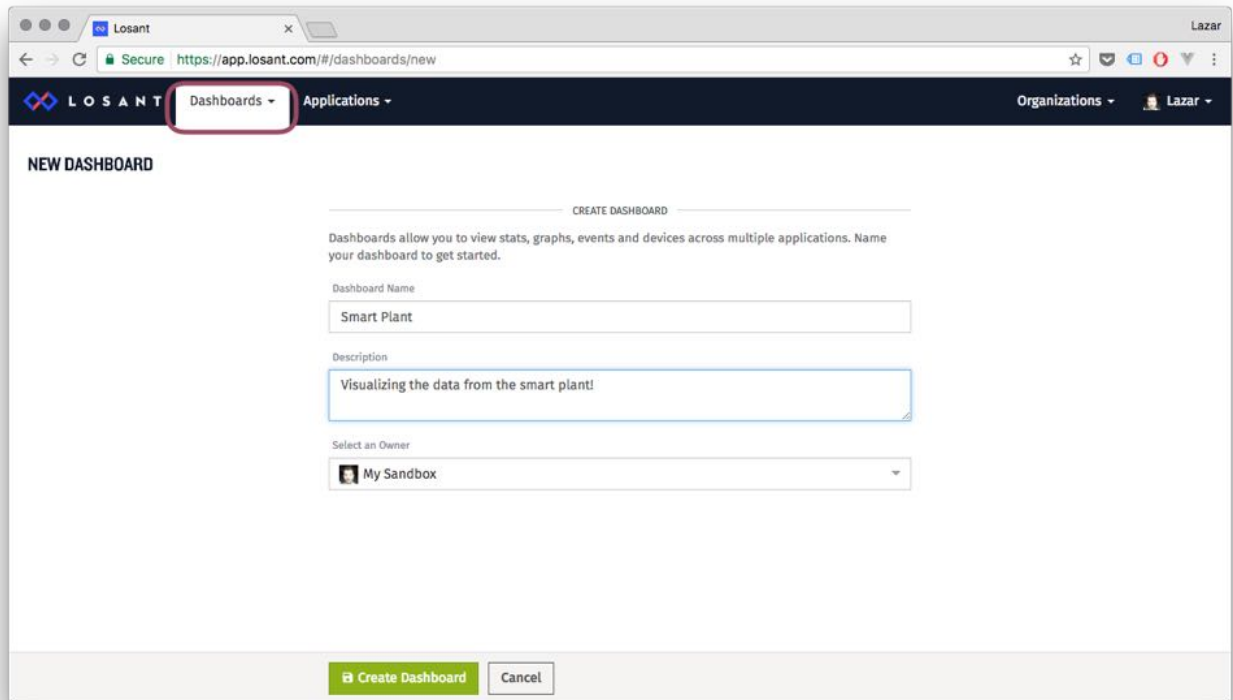
```
/etc/init.d/smart-plant restart
```

Try rebooting your Omega (enter `reboot` in the command line), and you'll see that your program will start up again when the Omega boots up.

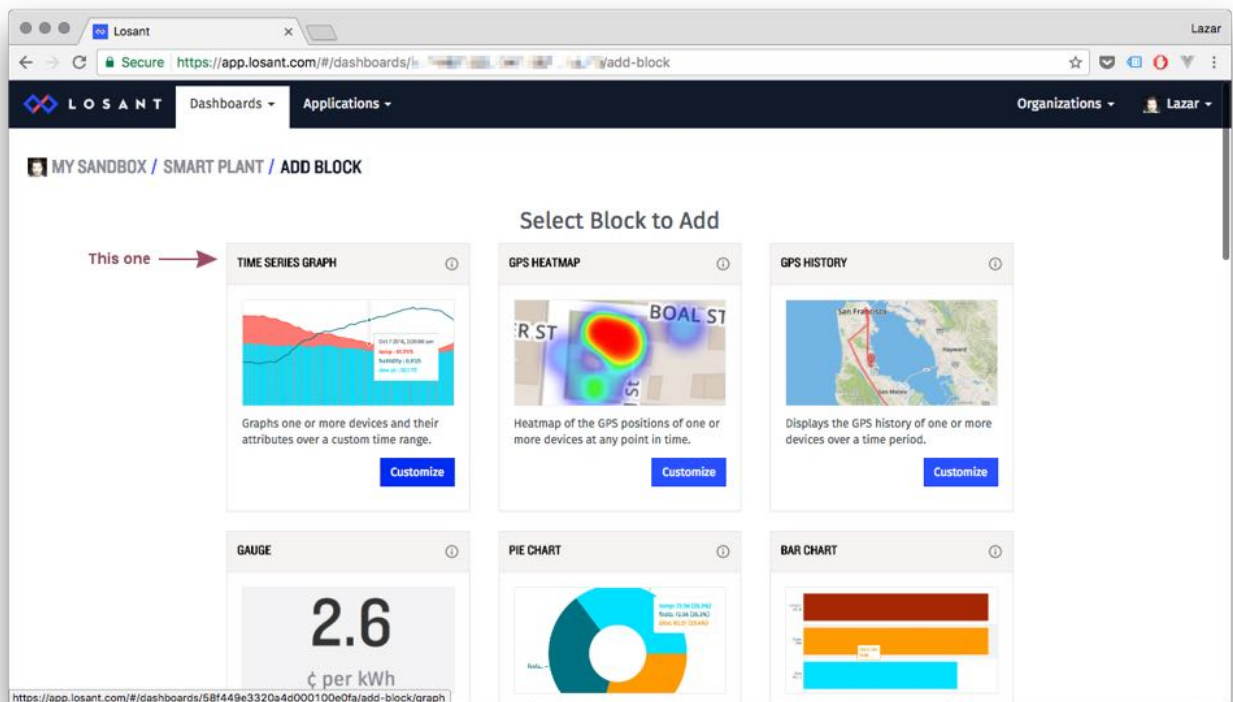
### 14. Create a Losant Dashboard

Now that we've successfully sent data to Losant, let's create a Dashboard so we can easily check in on our plant and see how much moisture there is in the soil.

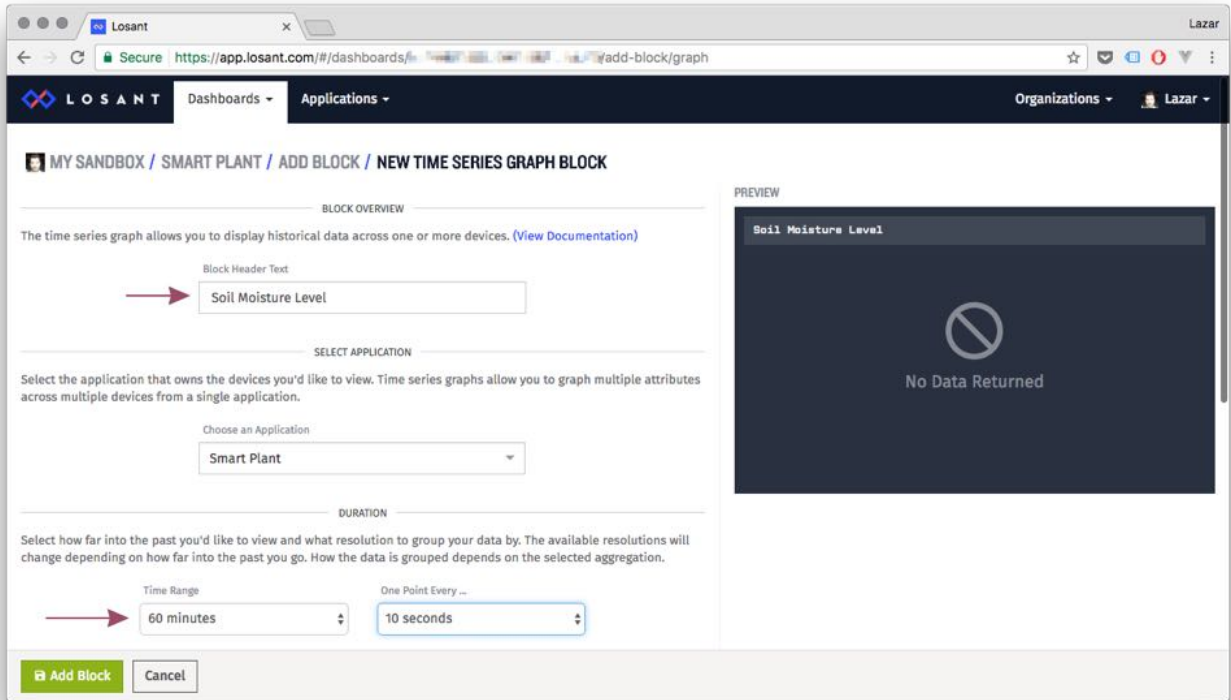
Click the `Dashboards` menu and then `Create Dashboard`, give your dashboard a name and description. Hit `Create Dashboard`:



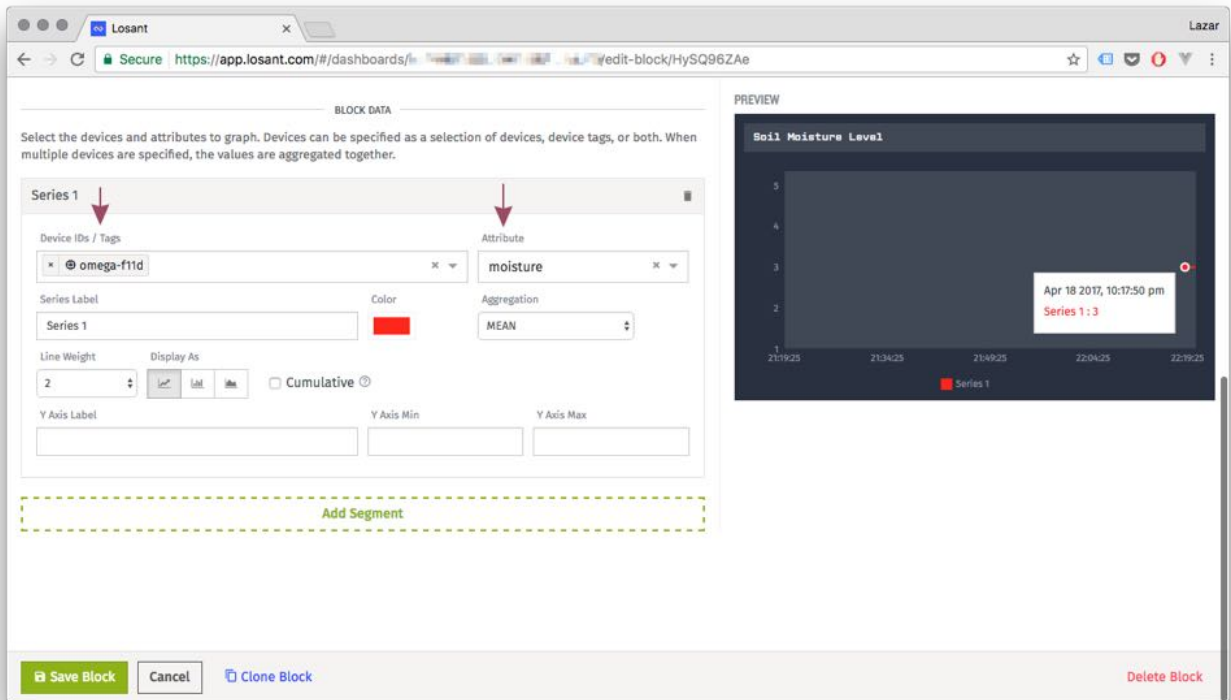
Let's add a Time Series Graph to the dashboard:



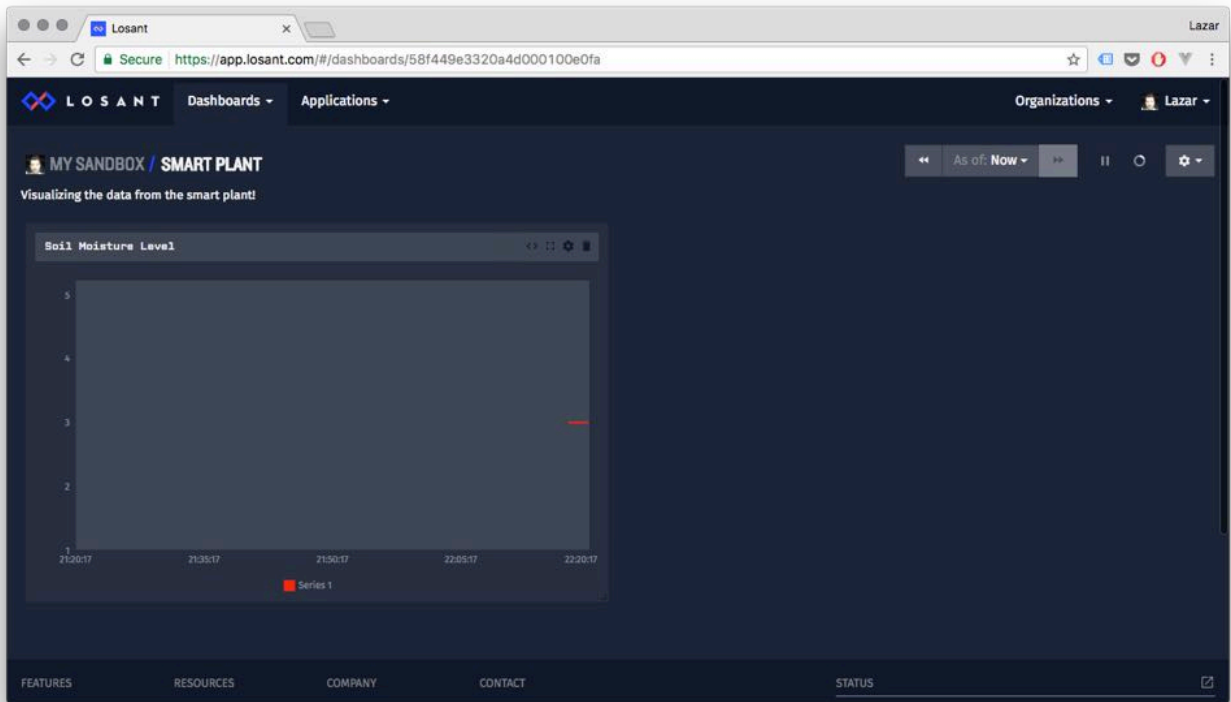
Give the block a Name and set the time range to 60 minutes and one point every 10 seconds:



Scroll down to select your device from the Device IDs dropdown. Then select moisture from the Attribute dropdown:

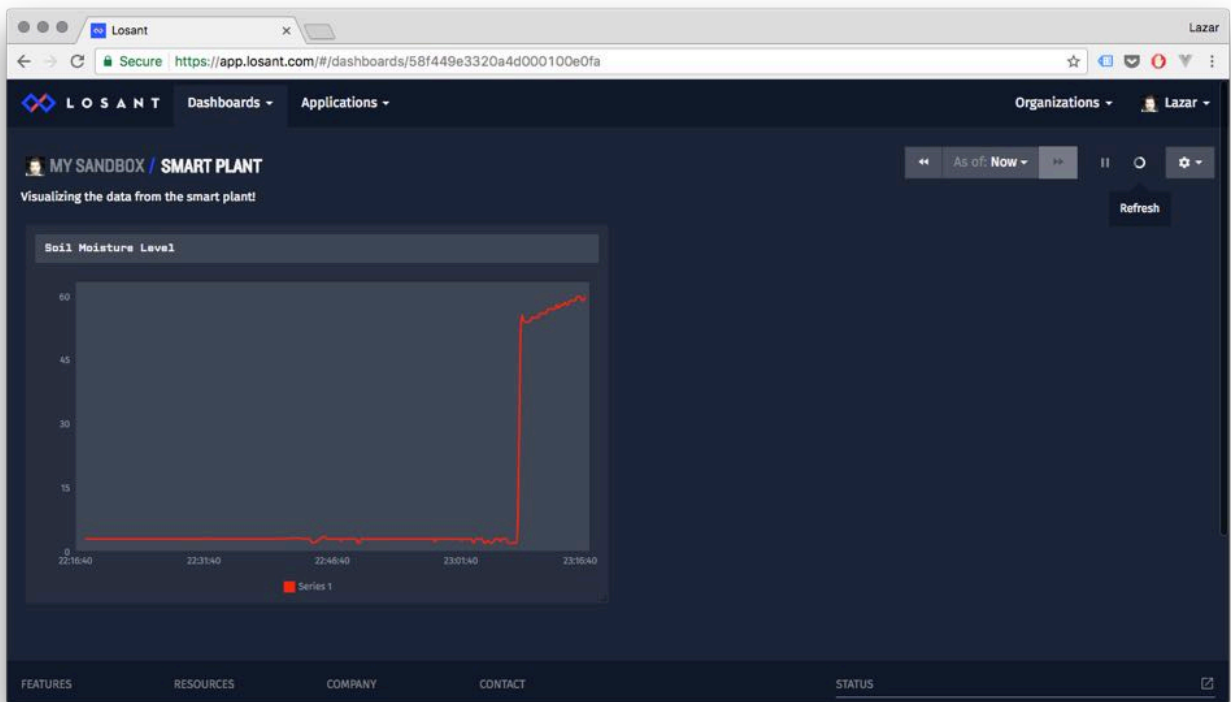


Awesome! Your dashboard is now displaying the soil moisture data being collected by your Omega:



### 15. Playing with the Dashboard

Try watering your plant a little while after you've setup the dashboard:





What a jump!

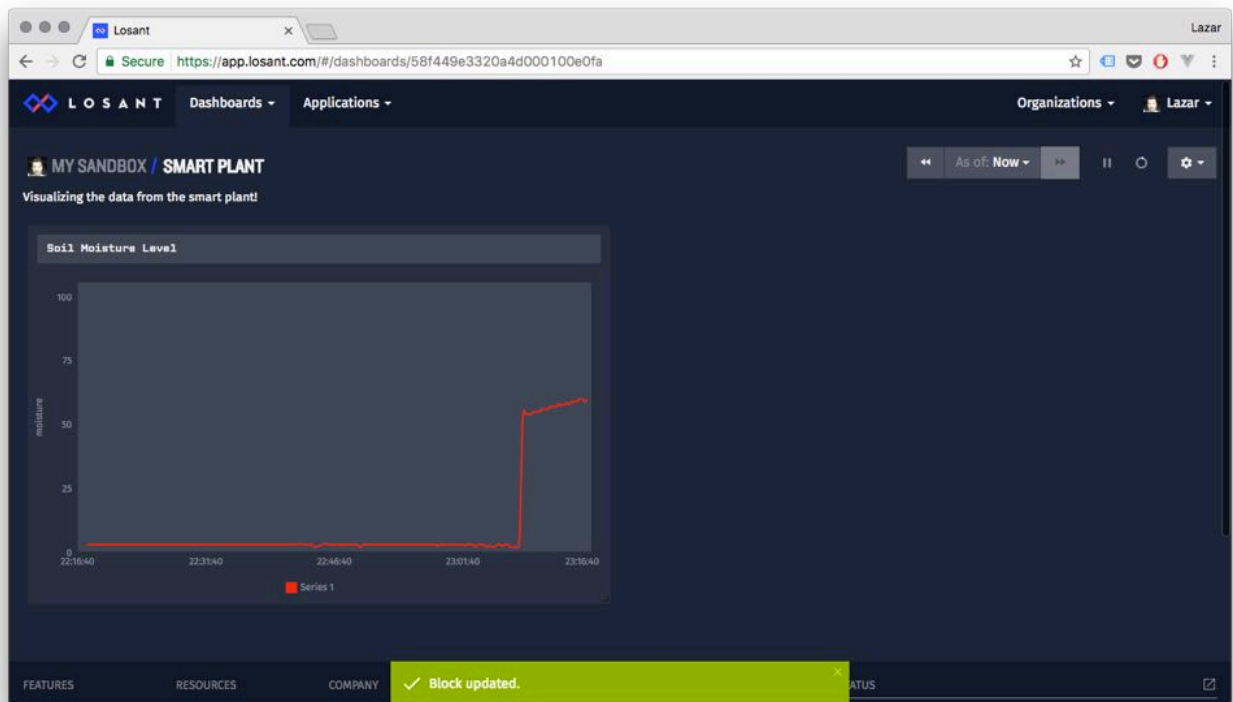
## Adjusting the Y-Axis

You might have noticed that the Y-axis adjusted automatically to fit the data. Since we know our measured value is limited in the 0 to 100 range, we can adjust the graph.

Hover over the graph and click on the Gear icon. Scroll down and adjust the Y Axis Minimum to 0 and the Maximum to 100:

The screenshot displays the Losant dashboard configuration interface. The main configuration area is titled "BLOCK DATA" and includes a "Series 1" configuration panel. The "Device IDs / Tags" field contains "omega-f11d" and the "Attribute" field contains "moisture". The "Series Label" is "Series 1", the "Color" is red, and the "Aggregation" is "MEAN". The "Y Axis Min" is set to 0 and the "Y Axis Max" is set to 100. A "Preview" window on the right shows a line chart titled "Soil Moisture Level" with a red line representing the moisture level over time. The chart shows a sharp increase in moisture level around 23:02:22. The interface also includes a "Save Block" button, a "Cancel" button, a "Clone Block" button, and a "Delete Block" button.

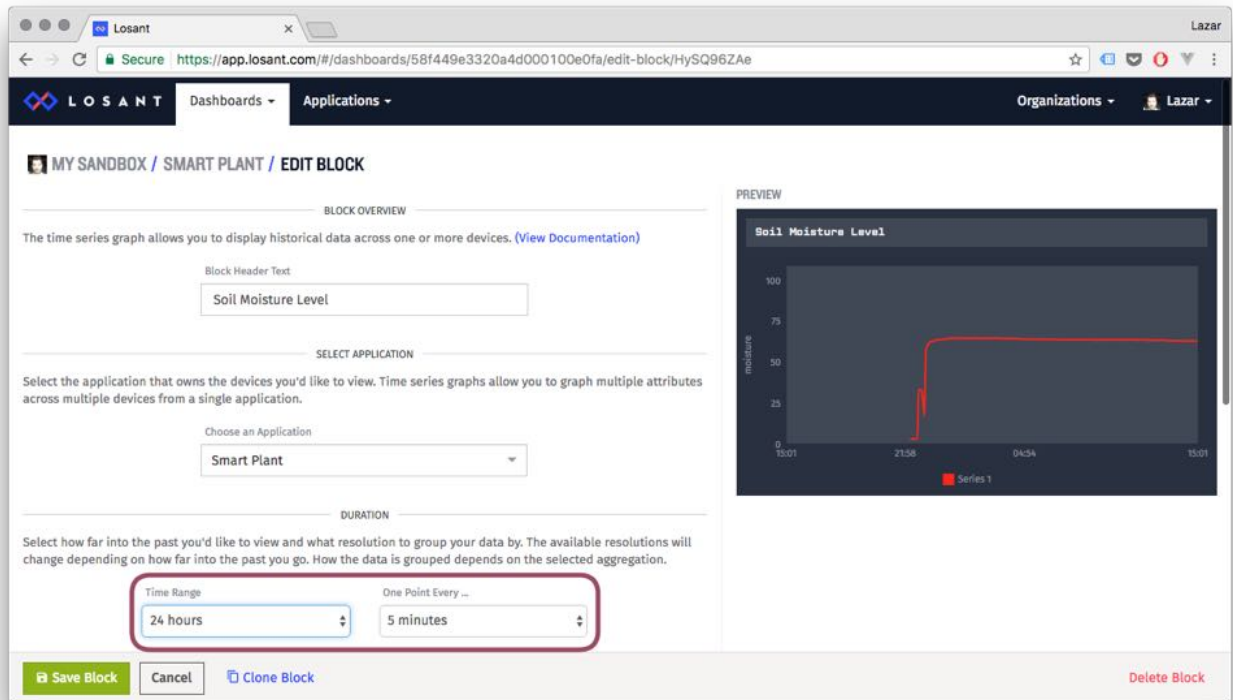
Hit Save Block and check out the chart now:



## Changing the Time Range

Seeing just the last hour of our plant's moisture level isn't too helpful, so let's change it to something more useful!

Hover over the chart and click the Gear icon. Adjust the Graph's Time Range to 24 hours or more.

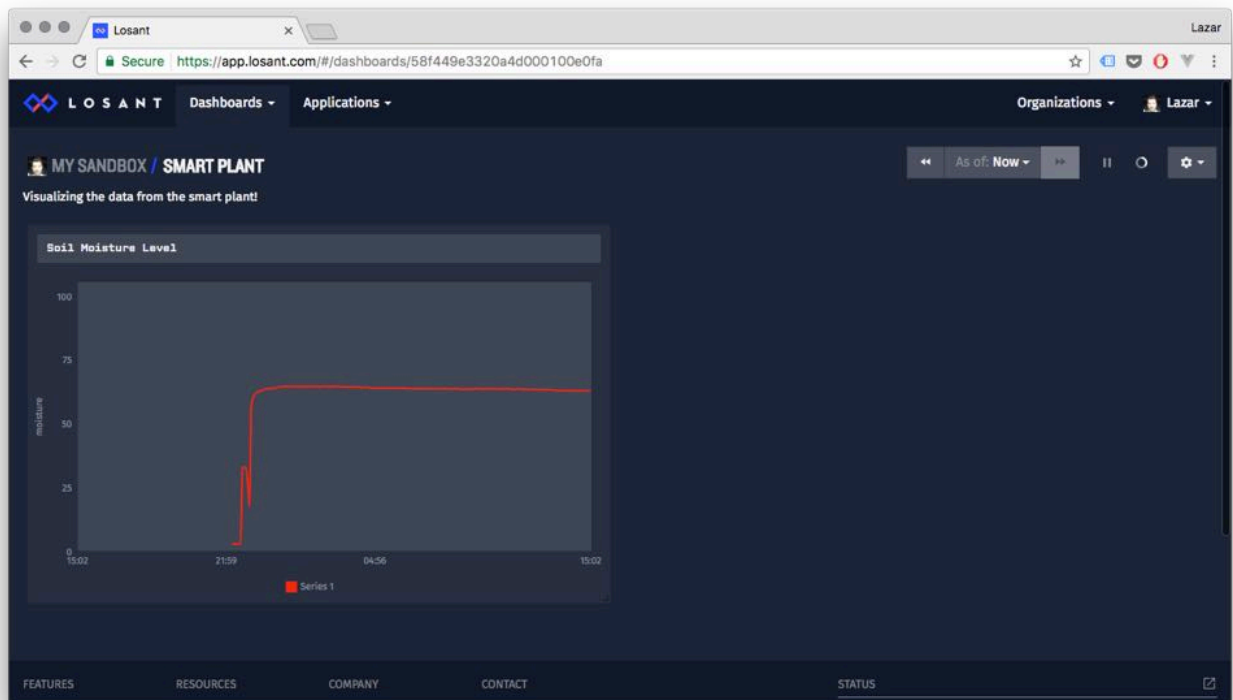


The screenshot shows the Losant 'EDIT BLOCK' interface for a 'Soil Moisture Level' chart. The interface is divided into several sections:

- BLOCK OVERVIEW:** A text area for 'Block Header Text' containing 'Soil Moisture Level'.
- SELECT APPLICATION:** A dropdown menu for 'Choose an Application' set to 'Smart Plant'.
- DURATION:** Two dropdown menus: 'Time Range' set to '24 hours' and 'One Point Every ...' set to '5 minutes'. These two dropdowns are highlighted with a red box.
- PREVIEW:** A small chart showing 'Soil Moisture Level' over time. The y-axis is labeled 'moisture' and ranges from 0 to 100. The x-axis shows time from 15:01 to 15:01. A red line shows a sharp increase in moisture level around 21:58.

At the bottom of the interface, there are buttons for 'Save Block', 'Cancel', 'Clone Block', and 'Delete Block'.

Hit **Save Block** and check out your extended chart. In this case, there was only data for the previous 18 hours, so everything before then is blank. If you have more data, a time range of many days might be more suitable:

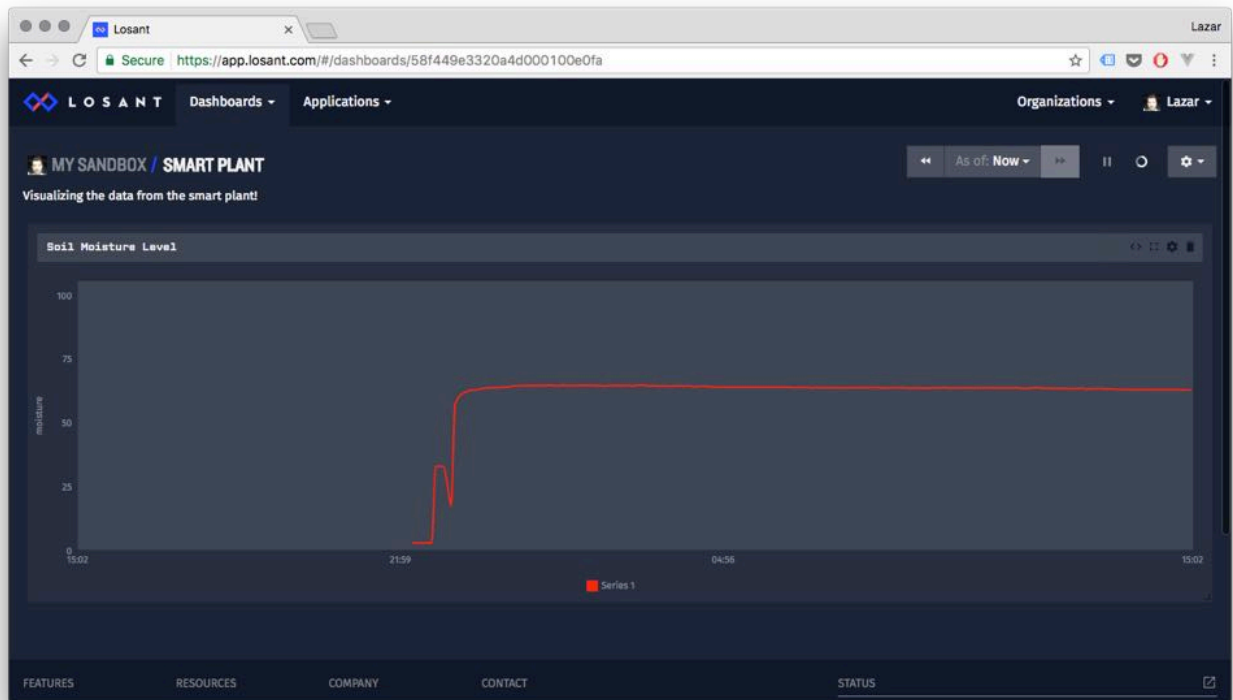


The screenshot shows the Losant 'SMART PLANT' dashboard. The main chart is titled 'Soil Moisture Level' and displays a line graph of moisture levels over time. The y-axis is labeled 'moisture' and ranges from 0 to 100. The x-axis shows time from 15:02 to 15:02. A red line shows a sharp increase in moisture level around 21:59, followed by a steady decline. The chart is labeled 'Series 1'.

At the top right of the dashboard, there are navigation controls: 'As of: Now', a play/pause button, and a settings gear icon. At the bottom of the dashboard, there are links for 'FEATURES', 'RESOURCES', 'COMPANY', 'CONTACT', and 'STATUS'.

If you do want to display more data, it would be useful to have a larger chart! Hover over the chart and

click and drag the icon to change the size:



## Going Further

Next we'll give our plant a voice of its own by connecting it to Twitter and having it broadcast to the world!

## Smart Plant - Twitter Alerts

Give your plant a voice of its own with Twitter! In part three, we'll build on what we've created in part [one](#) and [two](#) to get our plant to Tweet at us based on the moisture data collected.



## Overview

**Skill Level:** Intermediate

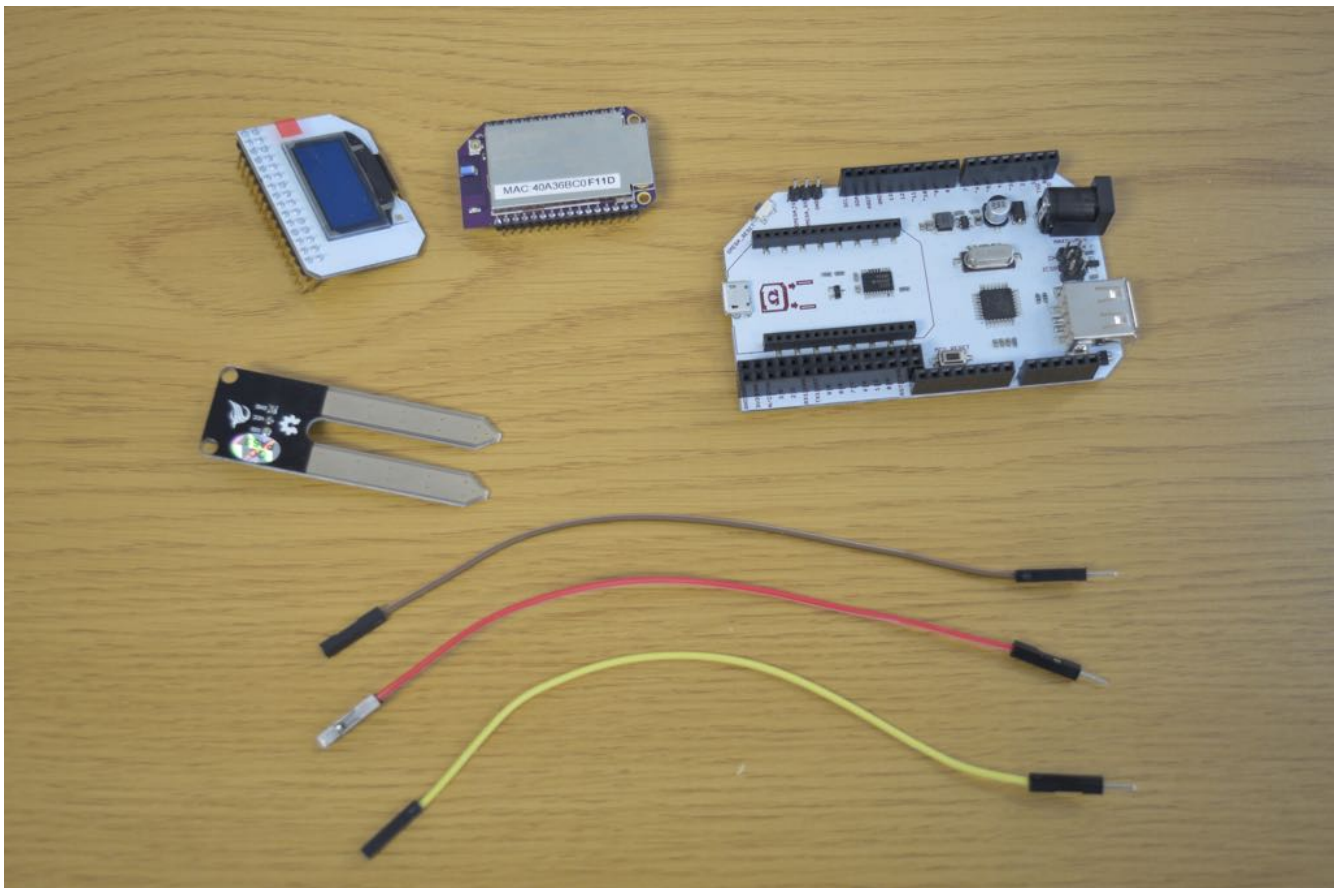
**Time Required:** 25 minutes

To accomplish this, we'll create a Losant workflow to read and check the moisture data from the Omega, then send a Tweet using Losant's Twitter integration. To get there, we'll create an App on Twitter to allow Losant to send Tweets.

## Ingredients

The same as the first part of the project:

- Onion [Omega2](#) or [Omega2+](#)
- Onion [Arduino Dock 2](#)
- Onion [OLED Expansion](#) (optional but recommended)
- [Soil Moisture Sensor](#)
- 3x [Male-to-Female Jumper Wires](#)



## Step-by-Step

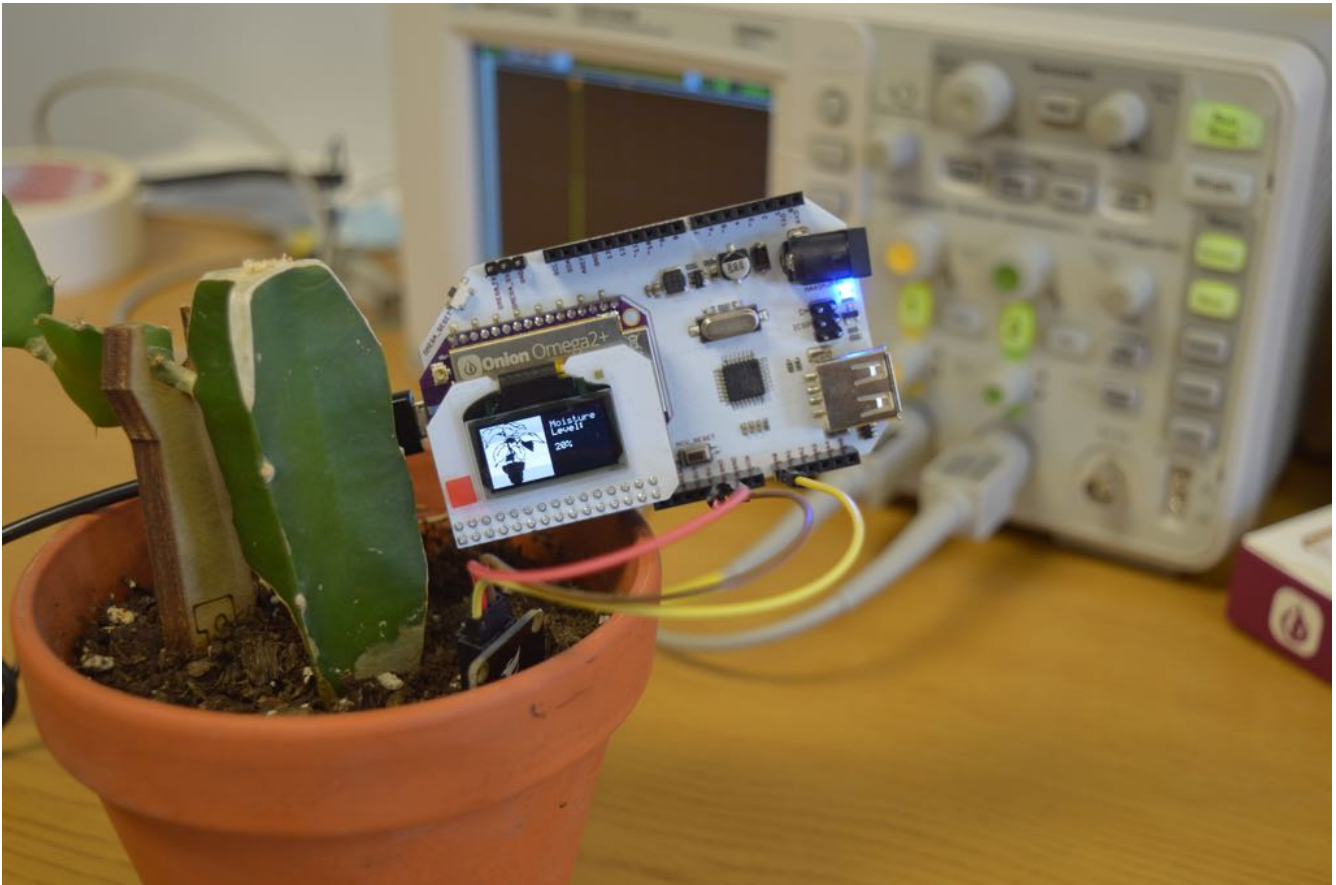
Follow these instructions to set this project up on your very own Omega!

## 1. Prepare

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

## 2. Complete the Previous Parts of the Project

This project builds on the first and second parts of the Smart Plant project. If you haven't already completed the [first part](#) and [second parts](#), go back and do them now!



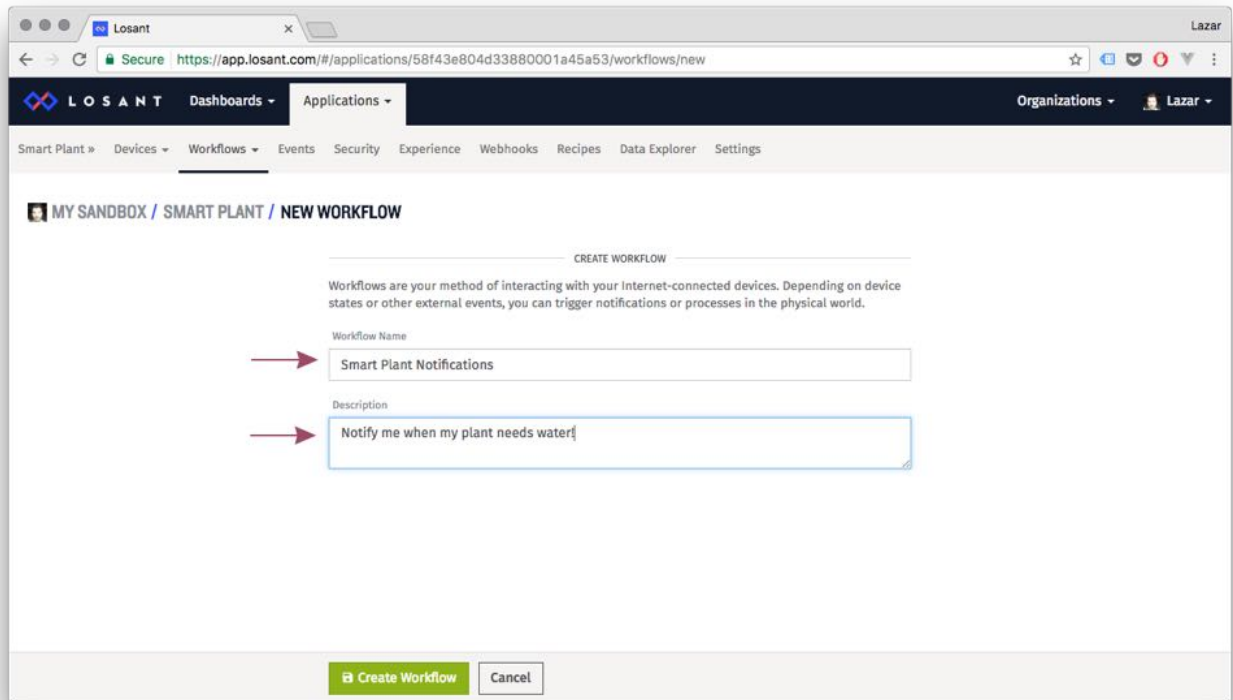
## 3. Login to Losant

Head over to [Losant.com](https://losant.com) and log in.

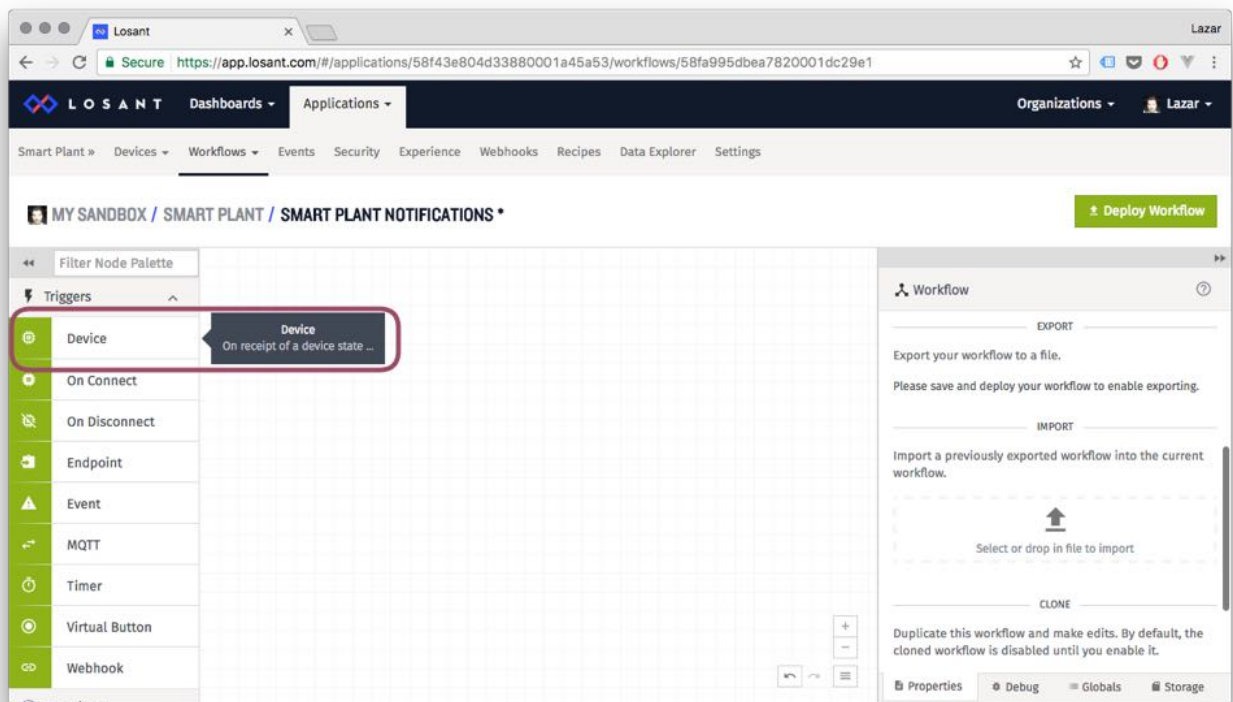
## 4. Losant Workflow: First Things

We'll need to create a new **workflow** to let our plant Tweet at us.

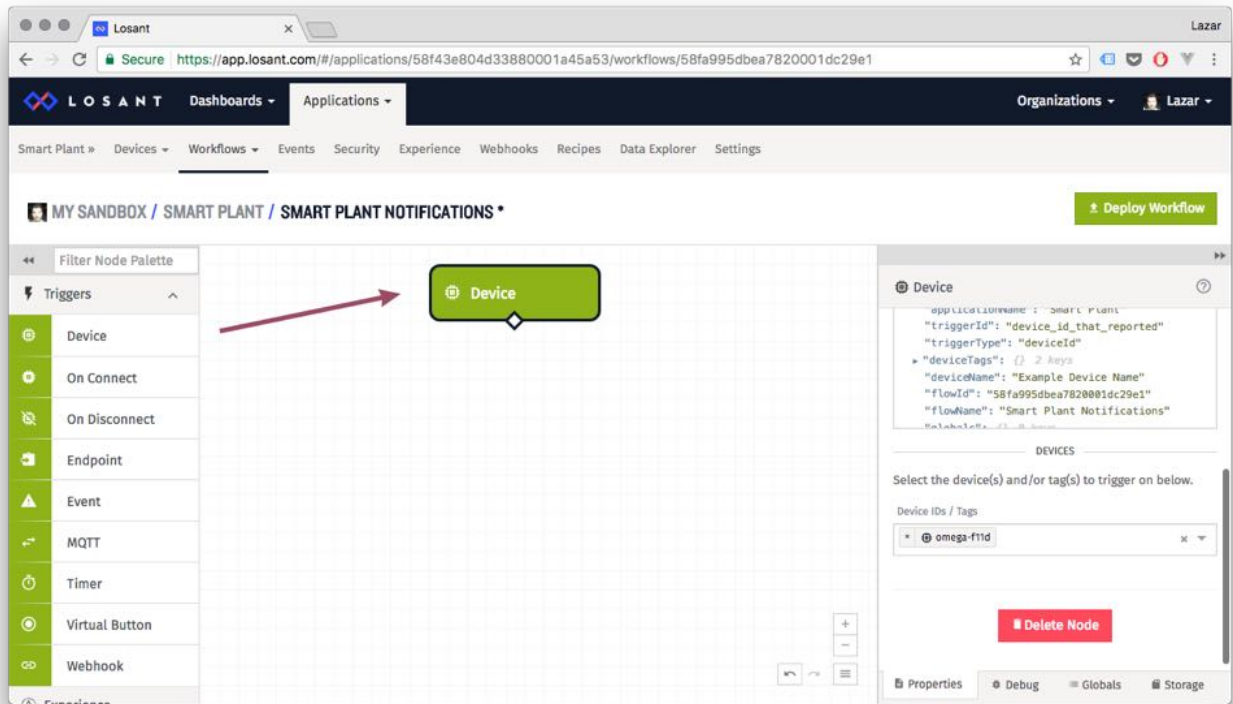
Click on the **Workflows** menu and then **Create Workflow**. Give your workflow a name and a description:



Add a Device block just like before:

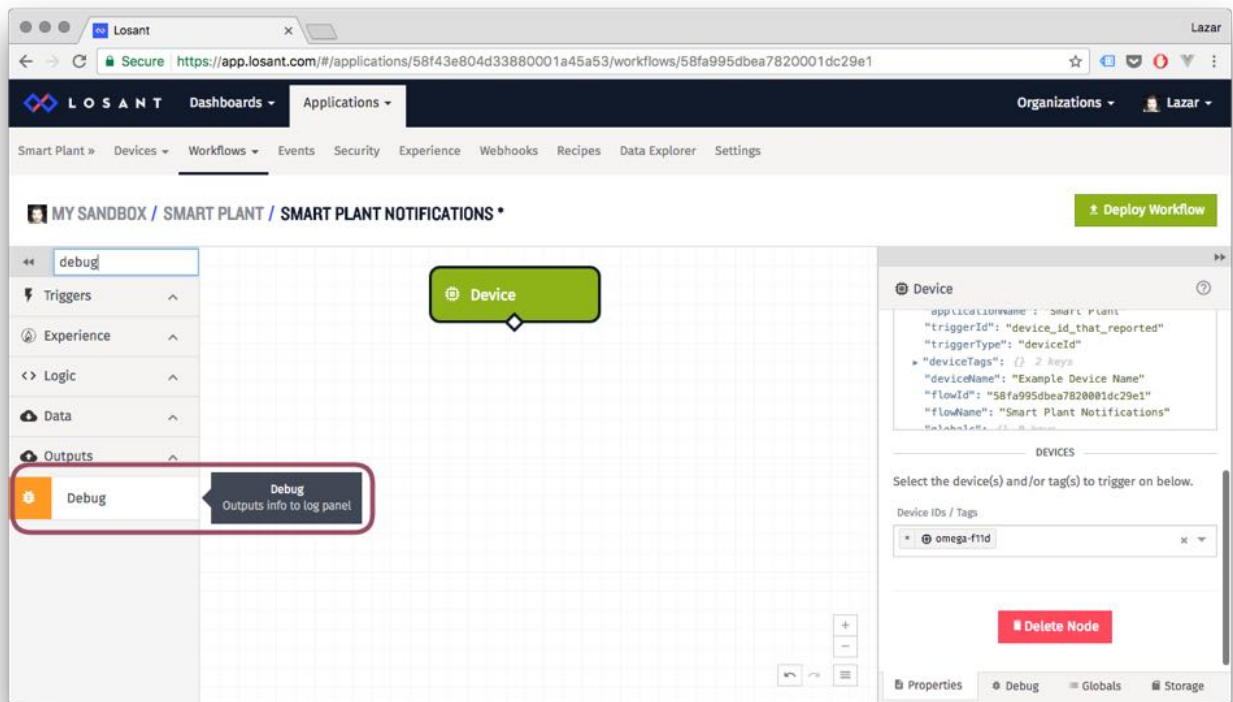


Make sure the device is pointing to the Omega connected to our plant.



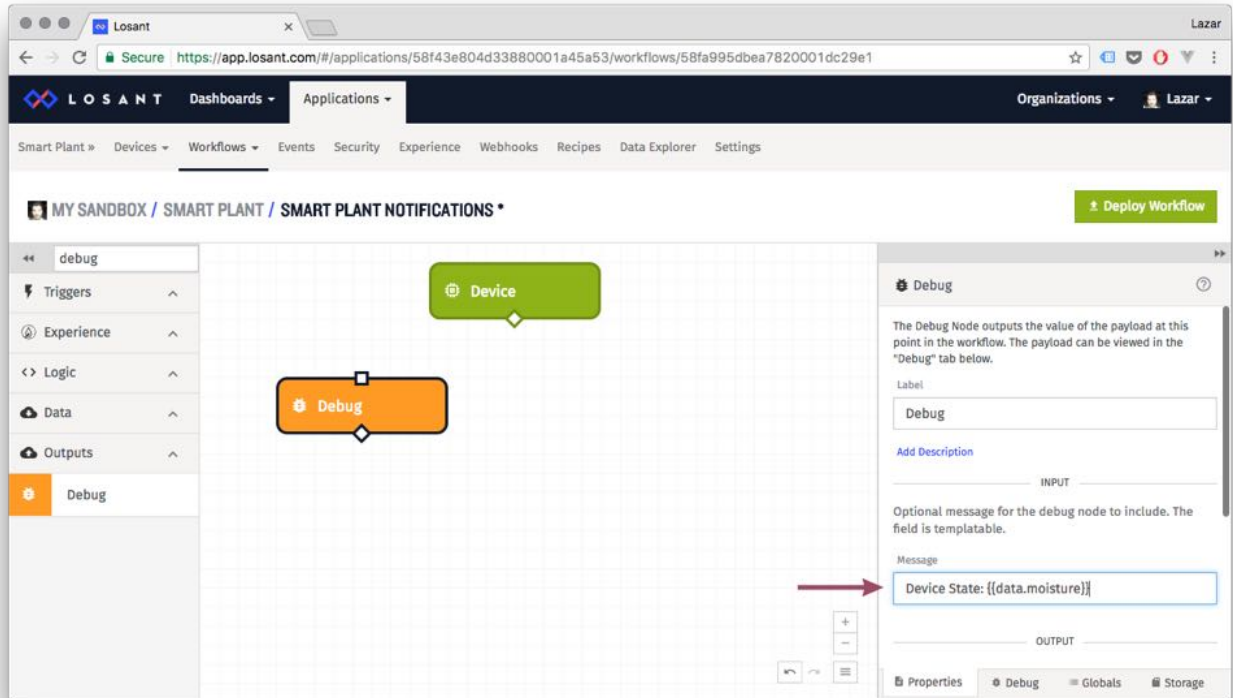
### 5. Losant Workflow: Debugging Node

Let's drop in a Debug block to check our moisture data is being properly received:



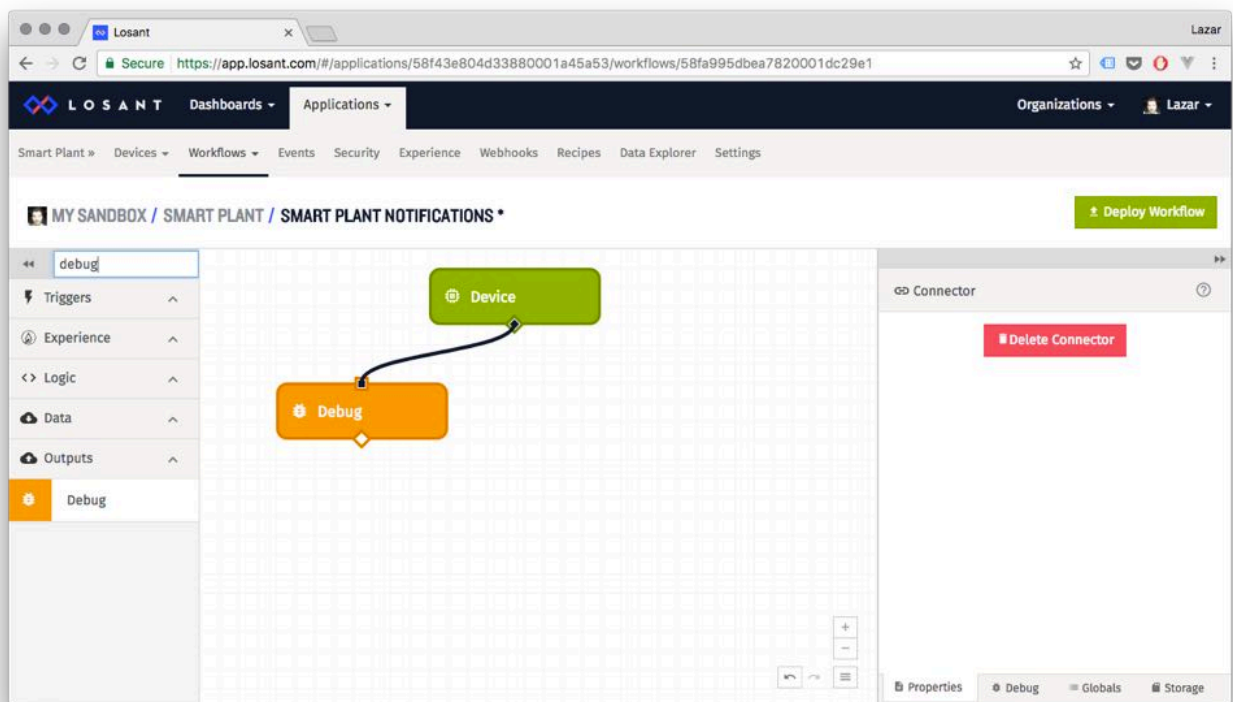


We'll add in a message to print out the moisture level:



The screenshot shows the Losant workflow editor interface. The main workspace contains two nodes: a green 'Device' node at the top and an orange 'Debug' node below it. A red arrow points from the 'Message' field in the 'Debug' node's configuration panel to the text 'Device State: {{data.moisture}}'. The configuration panel on the right includes a 'Label' field with the value 'Debug', an 'Add Description' link, an 'INPUT' section, a 'Message' field containing the templated text, and an 'OUTPUT' section. The left sidebar shows a navigation menu with categories like Triggers, Experience, Logic, Data, Outputs, and Debug.

And make the connection:

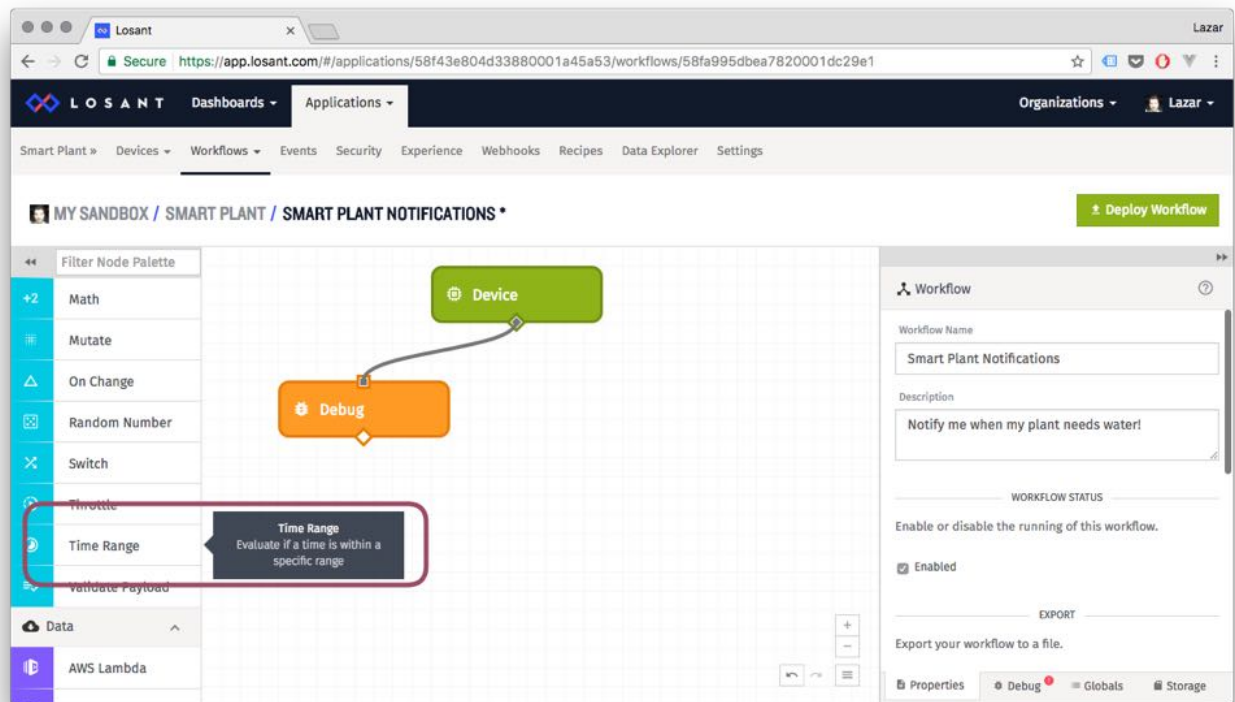


The screenshot shows the same Losant workflow editor interface, but now a black line connects the output of the 'Device' node to the input of the 'Debug' node. The configuration panel on the right is now titled 'Connector' and features a red 'Delete Connector' button. The rest of the interface, including the navigation menu and the 'Debug' node's configuration, remains the same as in the previous screenshot.

## 6. Losant Workflow: Time Window

It would defeat the purpose and be pretty annoying if our plant sent us a notification asking to be watered in the middle of the night. We'll use the **Time Range** node to make sure our notifications go out only during the day. Check out [Losant's Time Range node documentation](#) for more info.

Pull out a Time Range node from the sidebar to get started:



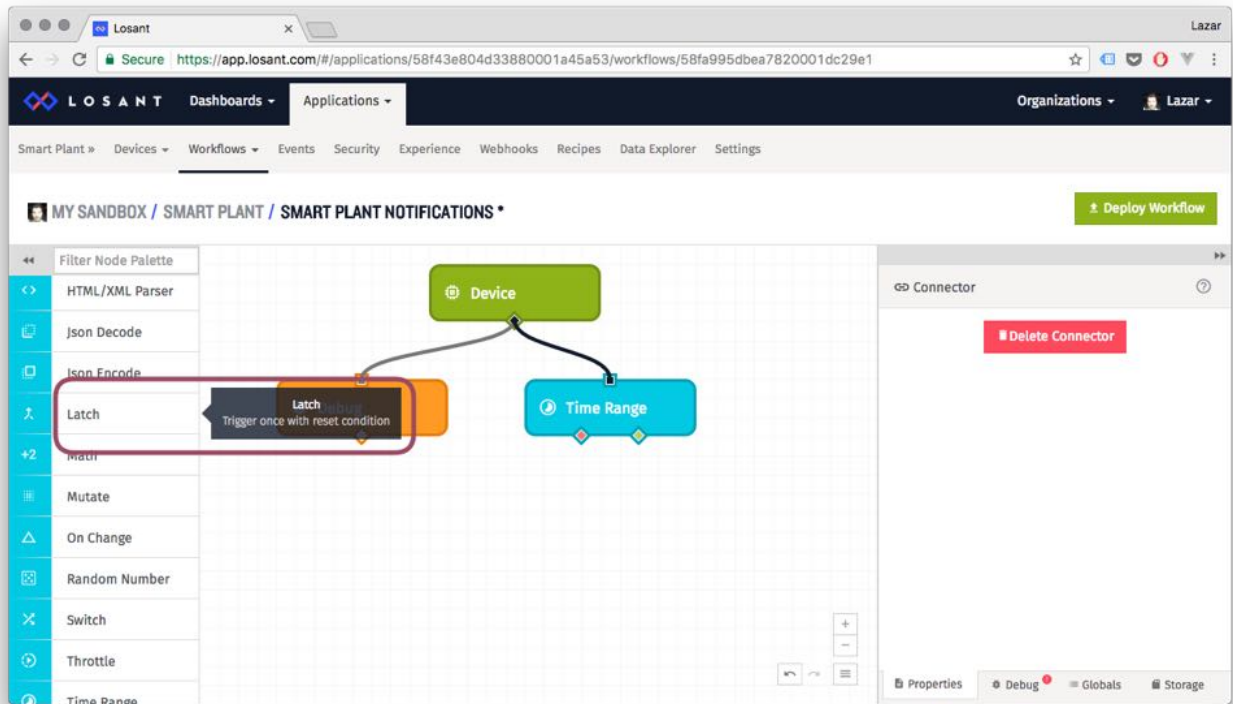
As always, the options provided by the node can be found in the right panel. We've set the node to allow the flow to continue if the time is between 9:00 to 21:00 (9am and 9pm) every day, feel free to decide what times work for your plant. Don't forget to set your Time Zone!

The screenshot displays the Losant workflow editor interface. On the left, a 'Filter Node Palette' lists various nodes including Math, Mutate, On Change, Random Number, Switch, Throttle, Time Range, and Validate Payload. The main workspace shows a workflow with three nodes: a green 'Device' node at the top, an orange 'Debug' node at the bottom left, and a blue 'Time Range' node at the bottom right. A red arrow points to the 'Time Range' node configuration panel on the right. This panel includes a 'TIME RANGE' section with instructions, a 'Time Zone' dropdown set to 'America/Toronto', 'Start Time' and 'End Time' input fields (09:00 and 21:00), and a 'Weekdays' section with checkboxes for S, M, T, W, T, F, S. At the bottom of the panel, there are tabs for 'Properties', 'Debug', 'Globals', and 'Storage'.

## 7. Losant Workflow: Check Moisture

Once we have our time window set up, we'll have to check for moisture!

The **Latch** node is used to perform a task a single time when a condition has been fulfilled. The node will not perform the task again until another condition has been achieved, kind of like a reset switch. It can be used for things such as one-time notifications. Check out [Losant's Latch node documentation](#) for more details.



For example, you can use it to send a single alert when a moisture sensor has dropped below 20%, and not send any more alerts until the level has risen back above 40%.

Each Latch node has two required conditions:

- The 'Latched' condition - when evaluates to `true`, triggers a following node **once and only once** until it has been reset.
- The 'Reset' condition - when evaluates to `true`, resets the 'Latched' condition so that it may trigger a node again.

The screenshot shows the Losant workflow editor interface. The workflow consists of the following nodes:

- Device** (Green): The starting trigger.
- Debug** (Orange): A node for logging.
- Time Range** (Blue): A node for scheduling.
- Latch** (Blue): A node for stateful logic.

The **Latch** node configuration panel is open, showing the following settings:

- LATCH CONDITIONAL**: Specify the latch expression to evaluate. When it evaluates to false, the node will always take the 'false' path. When it evaluates to true, it will only take the 'true' if the node is not in a latched state (and in that case will also 'latch' the node).
- Latch Expression**: `e.g. {{ data.moisture }} < 300`
- RESET CONDITIONAL**: Specify the reset expression to evaluate. When it evaluates to true, the latch state of the node will be reset.
- Reset Expression**: `e.g. {{ data.moisture }} > 500`

Let's create some global variables to dictate the moisture levels we'll use to trigger the latch and to reset it. By using global variables, it's easy for us to later experiment with different moisture levels and then ensure that the values get updated throughout the entire workflow. Create a `LOW_MOISTURE` variable to trigger the latch and a `OK_MOISTURE` variable to reset the latch:

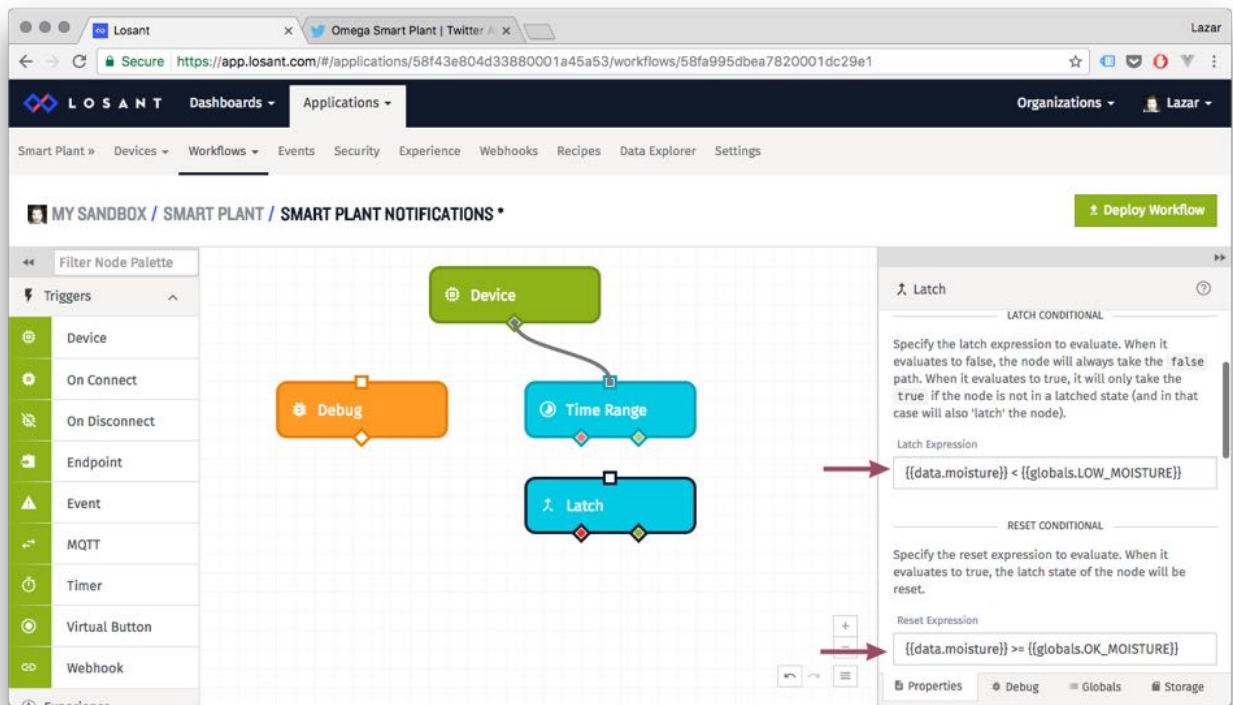
The screenshot shows the Losant workflow editor interface with the **Workflow Globals** panel open. The workflow is the same as in the previous screenshot.

The **Workflow Globals** panel shows the following configuration:

Key	Value	Data Type
<code>LOW_MOISTL</code>	35	Number
<code>OK_MOISTUF</code>	50	Number
		String

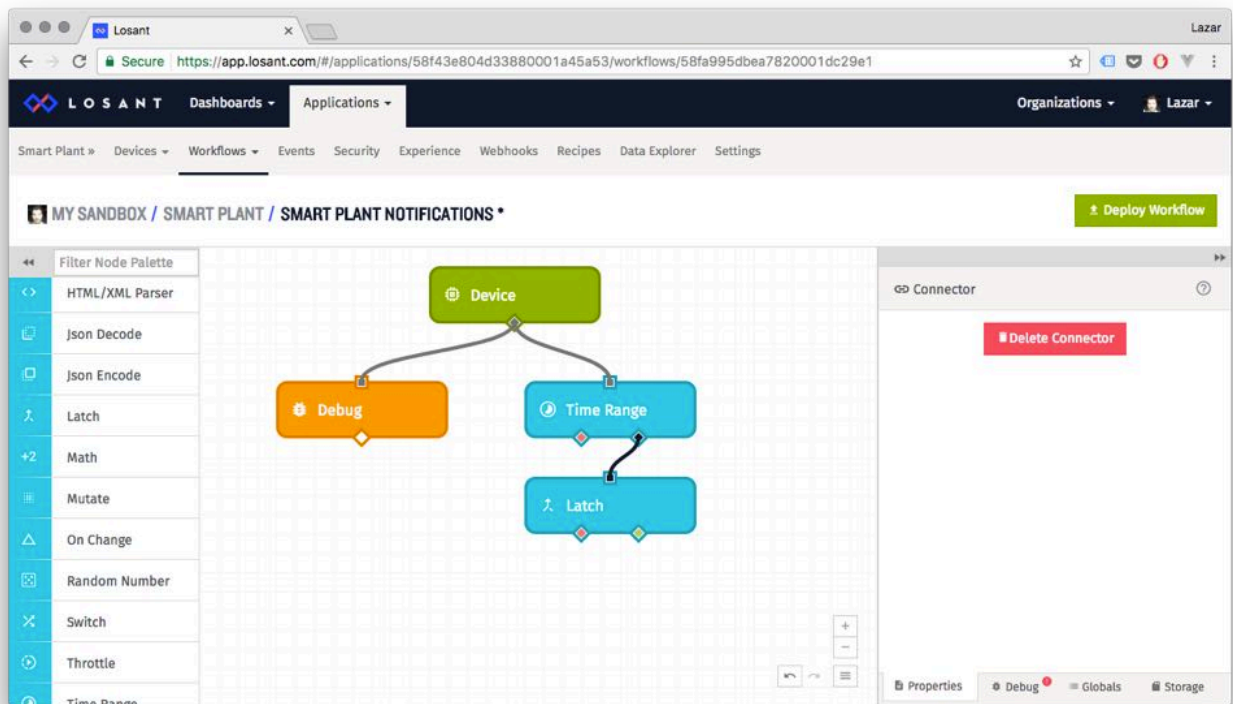
The **Globals** tab is selected in the bottom right corner of the configuration panel.

Once the global variables are setup, they can be used in the Latch node:



The screenshot shows the Losant workflow editor interface. The main workspace displays a workflow diagram with the following nodes: a green 'Device' node at the top, an orange 'Debug' node on the left, a blue 'Time Range' node in the center, and a blue 'Latch' node at the bottom. A line connects the 'Device' node to the 'Time Range' node. The 'Latch' node is connected to the 'Time Range' node. A red arrow points from the 'Latch' node in the diagram to its configuration panel on the right. The configuration panel for the 'Latch' node is titled 'LATCH CONDITIONAL' and contains the following text: 'Specify the latch expression to evaluate. When it evaluates to false, the node will always take the false path. When it evaluates to true, it will only take the true if the node is not in a latched state (and in that case will also latch the node)'. Below this text is a field for 'Latch Expression' containing the code `{{data.moisture}} < {{globals.LOW_MOISTURE}}`. Further down, there is a section for 'RESET CONDITIONAL' with the text: 'Specify the reset expression to evaluate. When it evaluates to true, the latch state of the node will be reset.' Below this is a field for 'Reset Expression' containing the code `{{data.moisture}} >= {{globals.OK_MOISTURE}}`. At the bottom of the configuration panel, there are tabs for 'Properties', 'Debug', 'Globals', and 'Storage'.

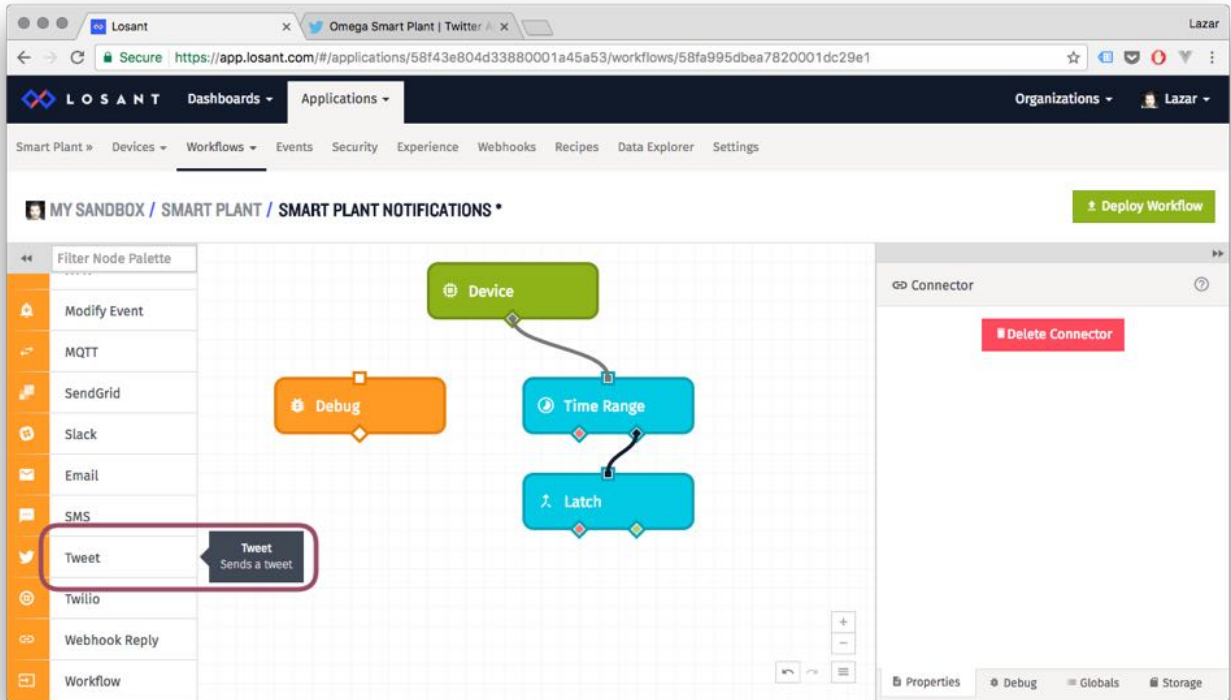
Now that it's set up, we'll connect the **Latch** to the **Time Range** node. Make sure to connect to the **Time Range** node's **true** path. This is the path that will be active if the current time is within our previously defined time range:



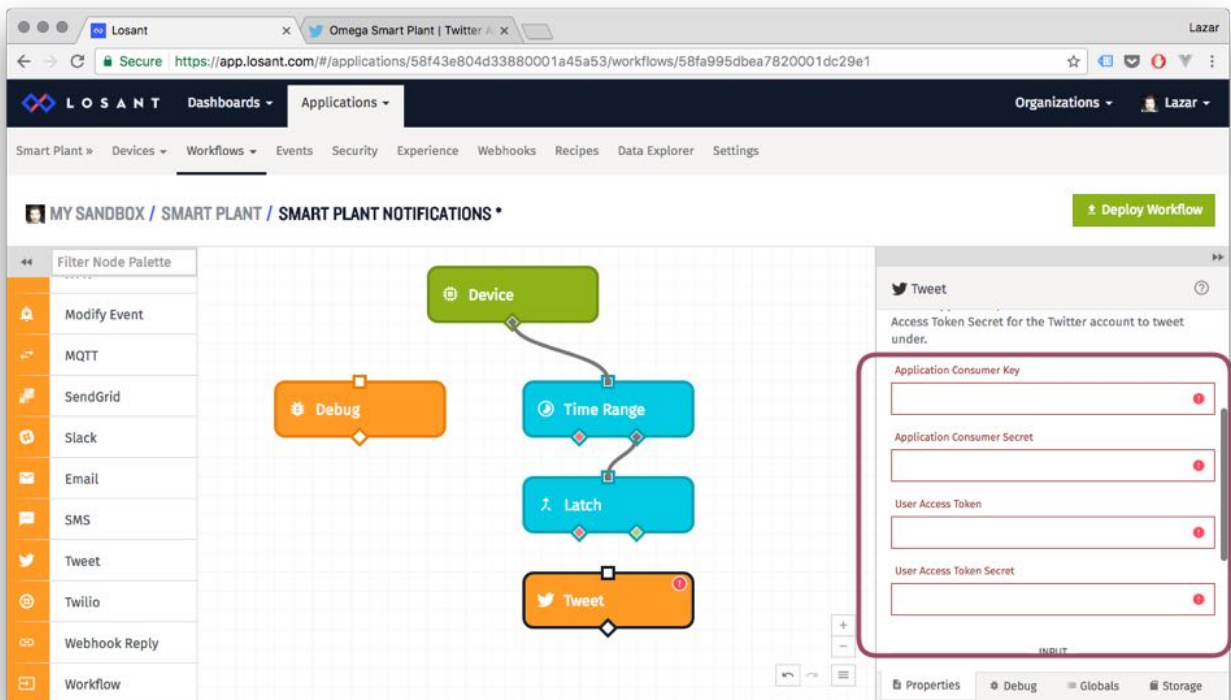
## 7. Losant Workflow: Twitter Event

The goal of this project is to get our plant to Tweet us. So when the Latch triggers, we definitely want it to send off a Twitter event.

Luckily for us, Losant provides a Tweet node! Check out [Losant's Tweet node documentation](#) for more info.



Taking a look at the properties, it looks like we'll need to register an App with Twitter so we can obtain an API key and User Access Token to send Tweets:



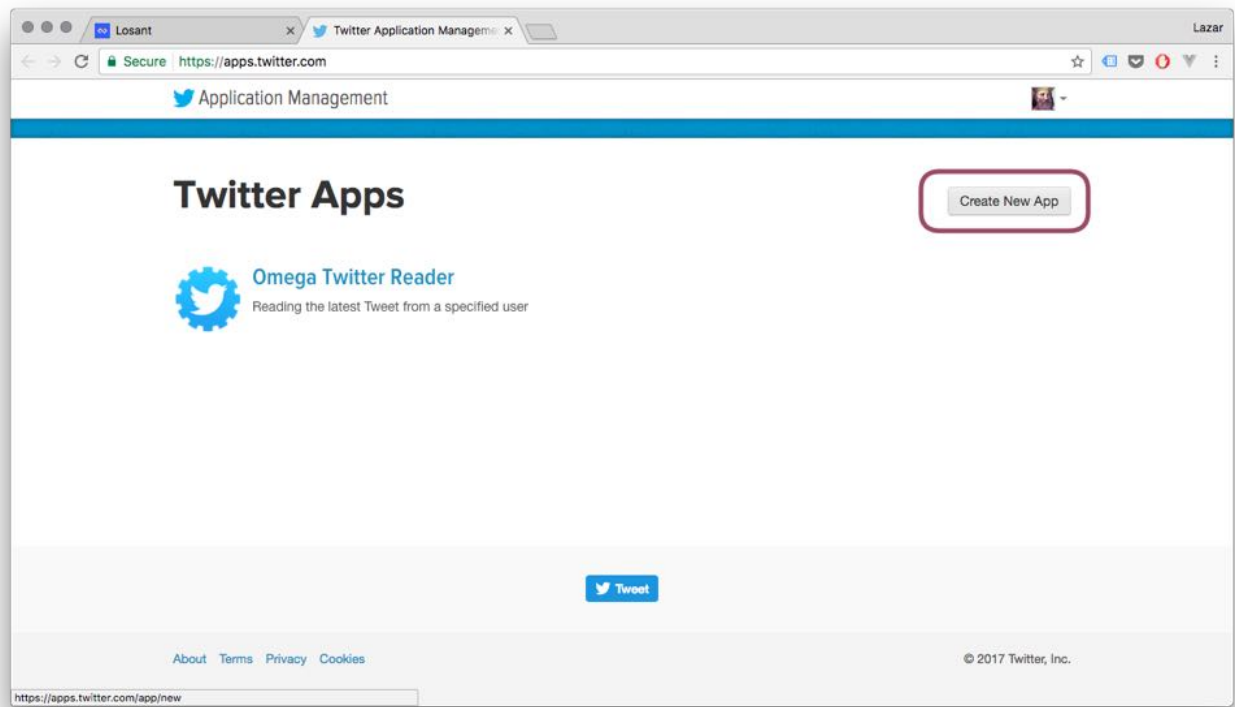


## 8. Create a Twitter Application

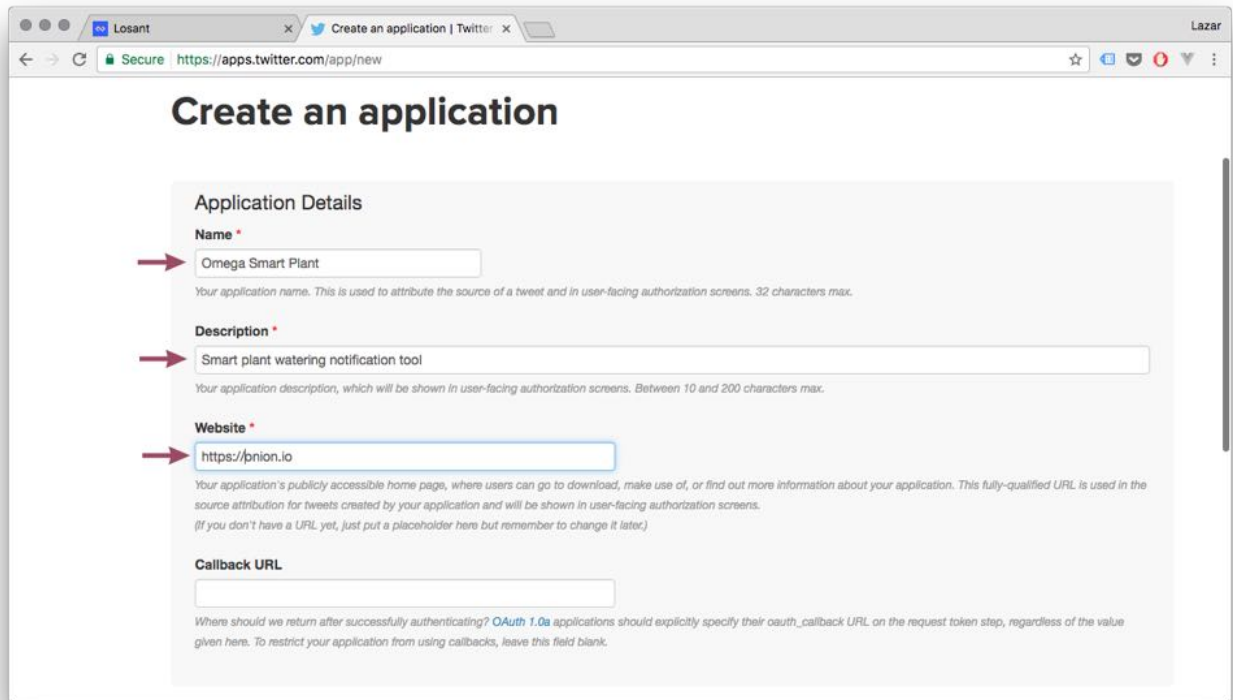
It's time to pay Twitter a visit!

Login to Twitter with the account of your choice. Feel free to create a new one - your plant is special, after all!

When you're in, visit <https://apps.twitter.com> where we'll be able to create a new App to access Twitter's APIs:



Give it a name, a description, and a website. The website can really be anything you wish - it will be used by Twitter to give credit and send users for more information.



The screenshot shows the 'Create an application' page on Twitter. The browser address bar shows 'https://apps.twitter.com/app/new'. The page title is 'Create an application'. The 'Application Details' section contains the following fields:

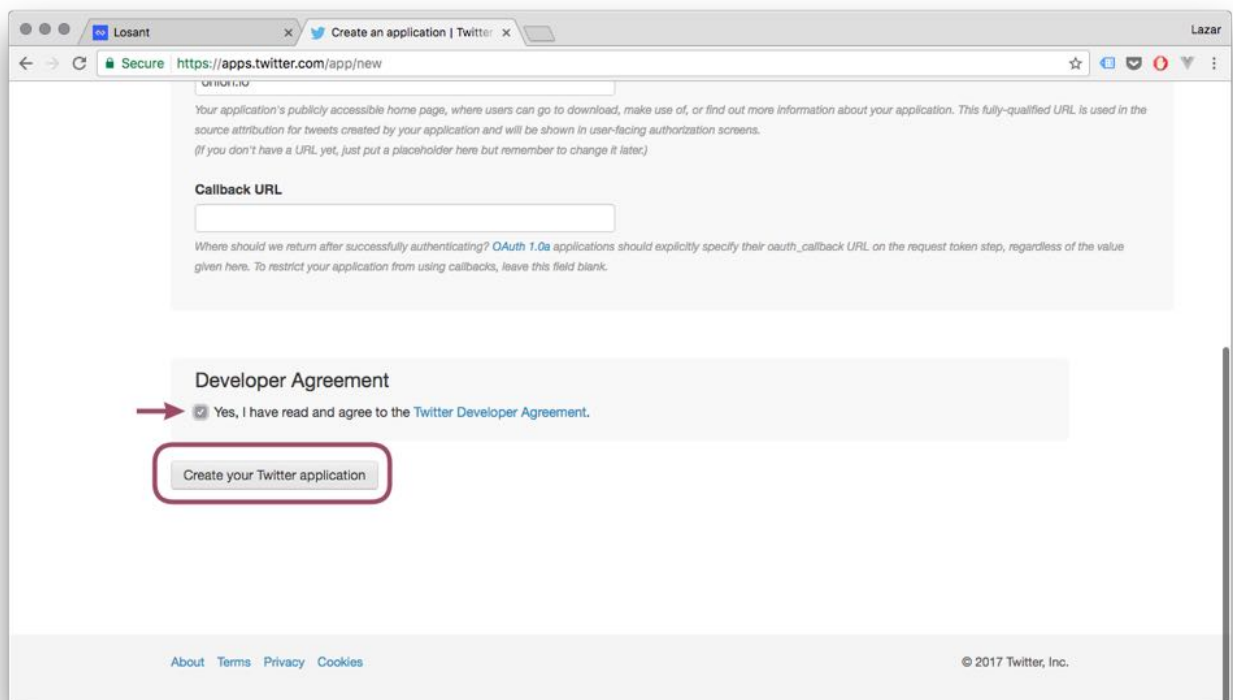
- Name \***: A text input field containing 'Omega Smart Plant'. A red arrow points to this field.
- Description \***: A text input field containing 'Smart plant watering notification tool'. A red arrow points to this field.
- Website \***: A text input field containing 'https://pnion.io'. A red arrow points to this field.
- Callback URL**: An empty text input field.

Below each field is a small explanatory text:

- Name**: Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.
- Description**: Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.
- Website**: Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)
- Callback URL**: Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth\_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Feel free to link it to this project!

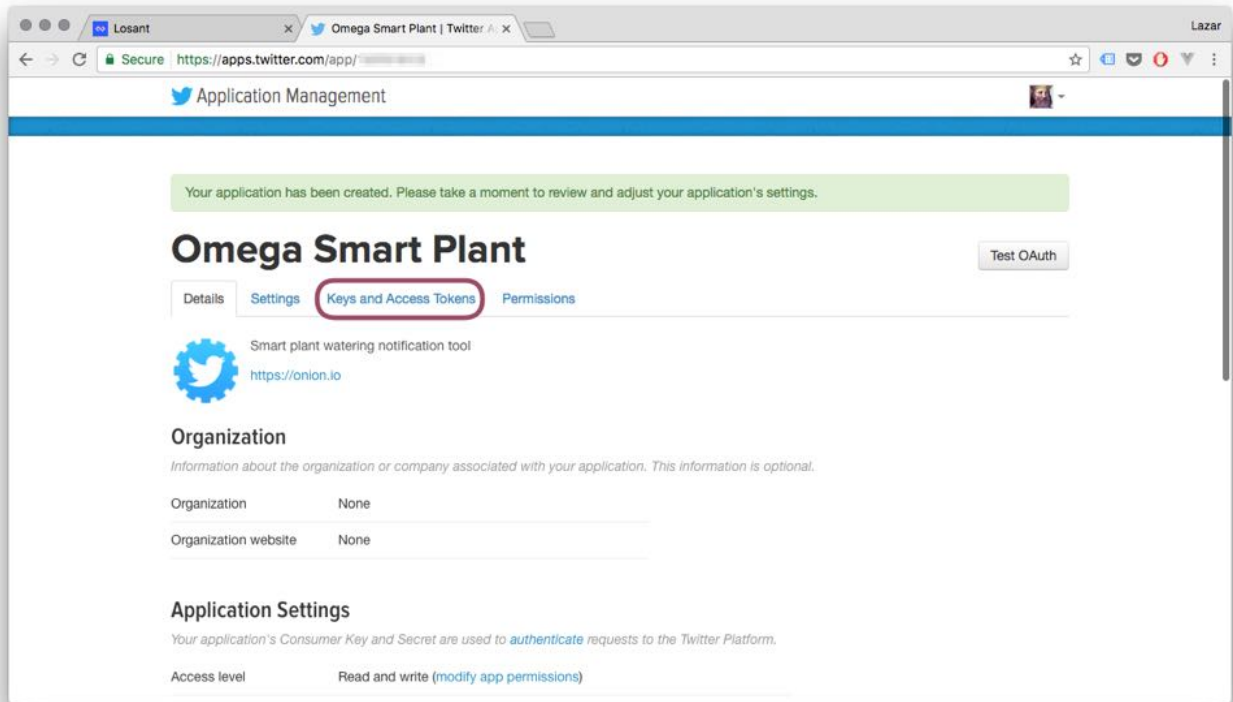
Agree to the Twitter Developer Agreement - read it over if you can - and hit the Button to create your App!



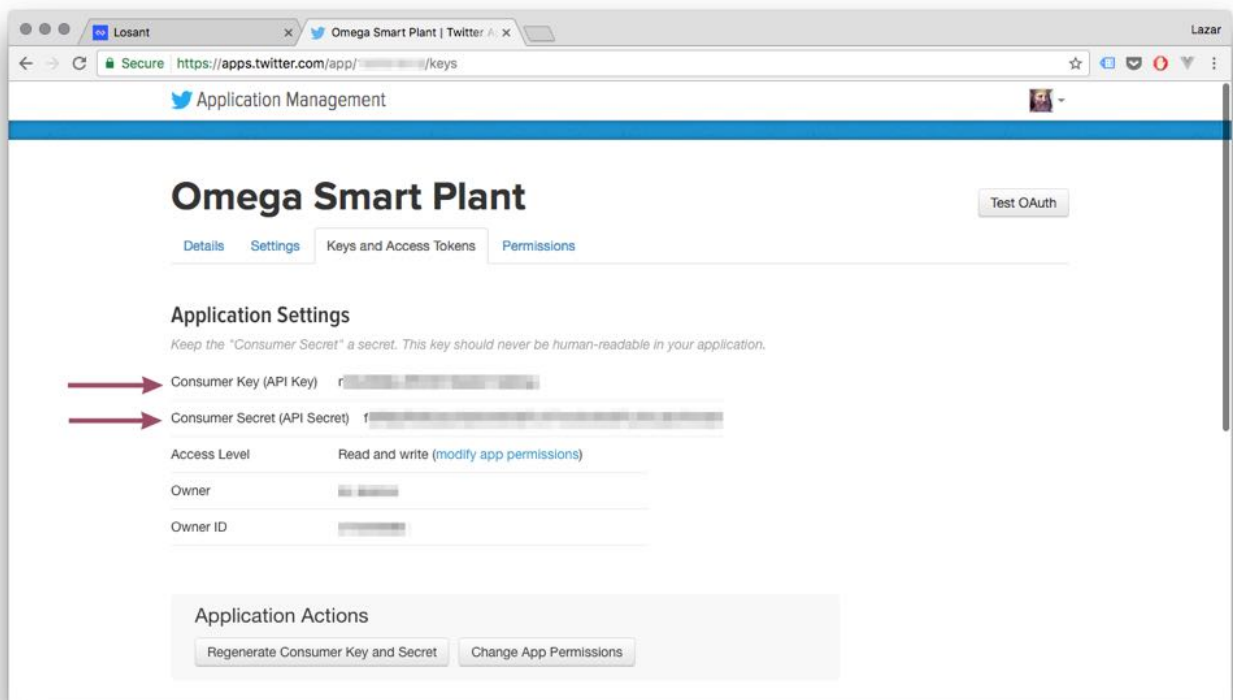
The screenshot shows the 'Developer Agreement' section of the 'Create an application' page. The 'Callback URL' field is visible above this section. The 'Developer Agreement' section contains a checkbox that is checked, with the text 'Yes, I have read and agree to the Twitter Developer Agreement.' A red arrow points to this checkbox. Below the checkbox is a button labeled 'Create your Twitter application', which is circled in red. At the bottom of the page, there are links for 'About', 'Terms', 'Privacy', and 'Cookies', and a copyright notice for '© 2017 Twitter, Inc.'

Welcome to your Twitter App!

Now let's go and get what we came for: the API keys. Navigate to the 'Keys and Access Tokens' tab:



And you'll be greeted with your Consumer Key and Consumer Secret:



If you think your keys have fallen into the hands of **evil**, you can always regenerate a new pair here. The old ones will no longer be useable at all when you do this, so take care with the regenerate button.

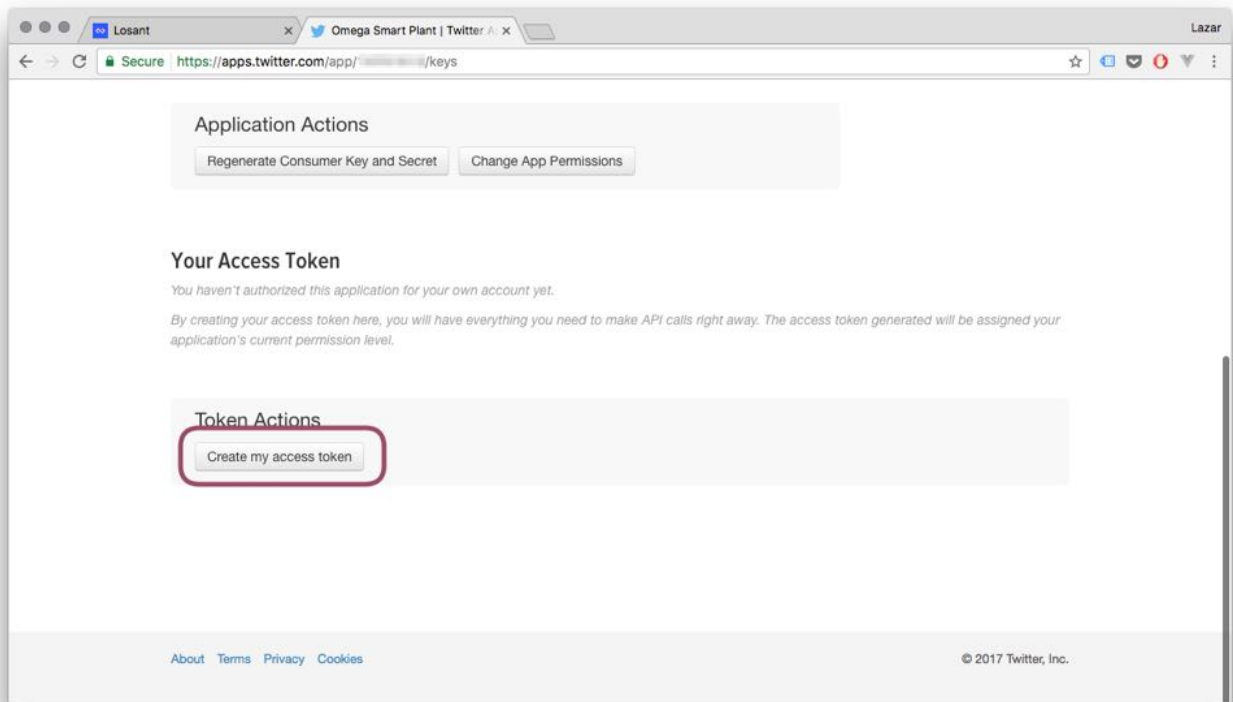
Copy the keys and head back to your workflow. Create `CONSUMER_KEY` and `CONSUMER_SECRET` global variables to hold the values:

The screenshot shows the Losant workflow editor interface. The workflow is titled "MY SANDBOX / SMART PLANT / SMART PLANT NOTIFICATIONS \*". The workflow consists of the following nodes: a green "Device" node, an orange "Debug" node, a blue "Time Range" node, a blue "Latch" node, and an orange "Tweet" node. The "Workflow Globals" panel is open on the right, showing a table of global variables:

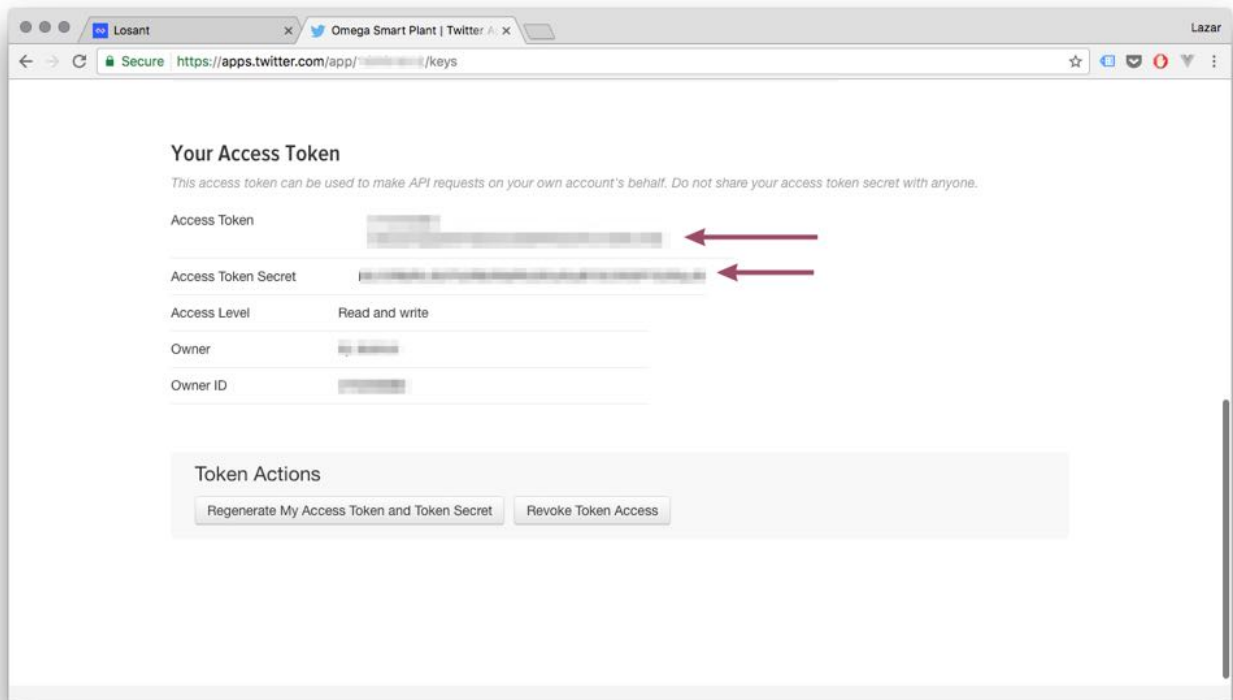
Key	Value	Data Type
LOW_MOISTL	35	Number
OK_MOISTUF	50	Number
CONSUMER_KEY	[Redacted]	String
CONSUMER_SECRET	[Redacted]	String
		String

The "CONSUMER\_KEY" and "CONSUMER\_SECRET" rows are highlighted with red boxes, and red arrows point from these boxes to the corresponding nodes in the workflow. The "Globals" tab in the bottom right corner is also highlighted with a red circle.

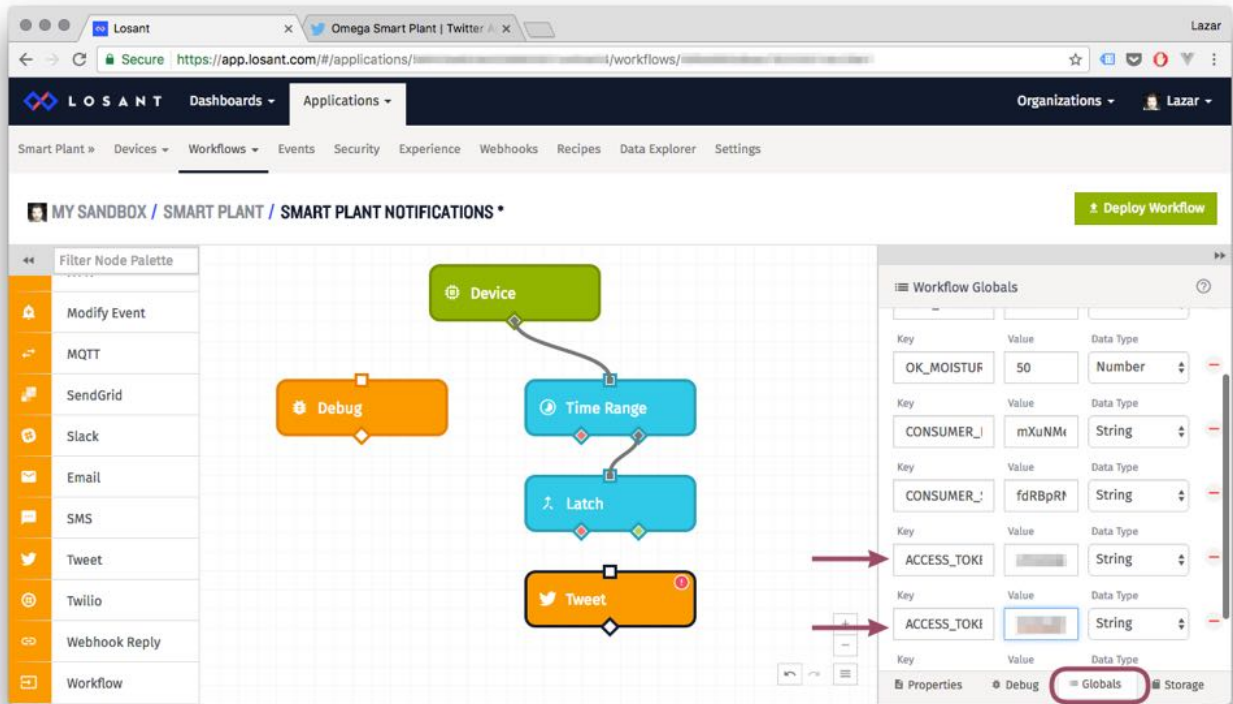
Once the keys are in, it's time to generate Access Tokens. Head back to Twitter and create a new access token:



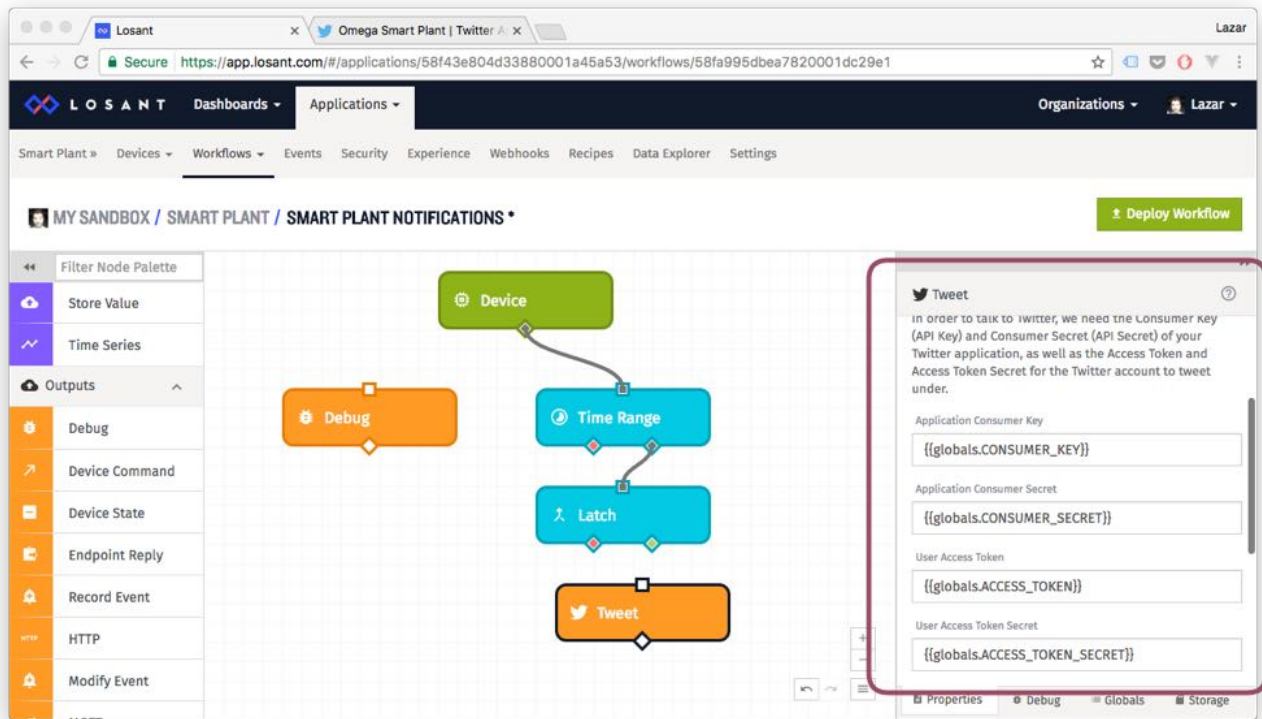
Note the token values:



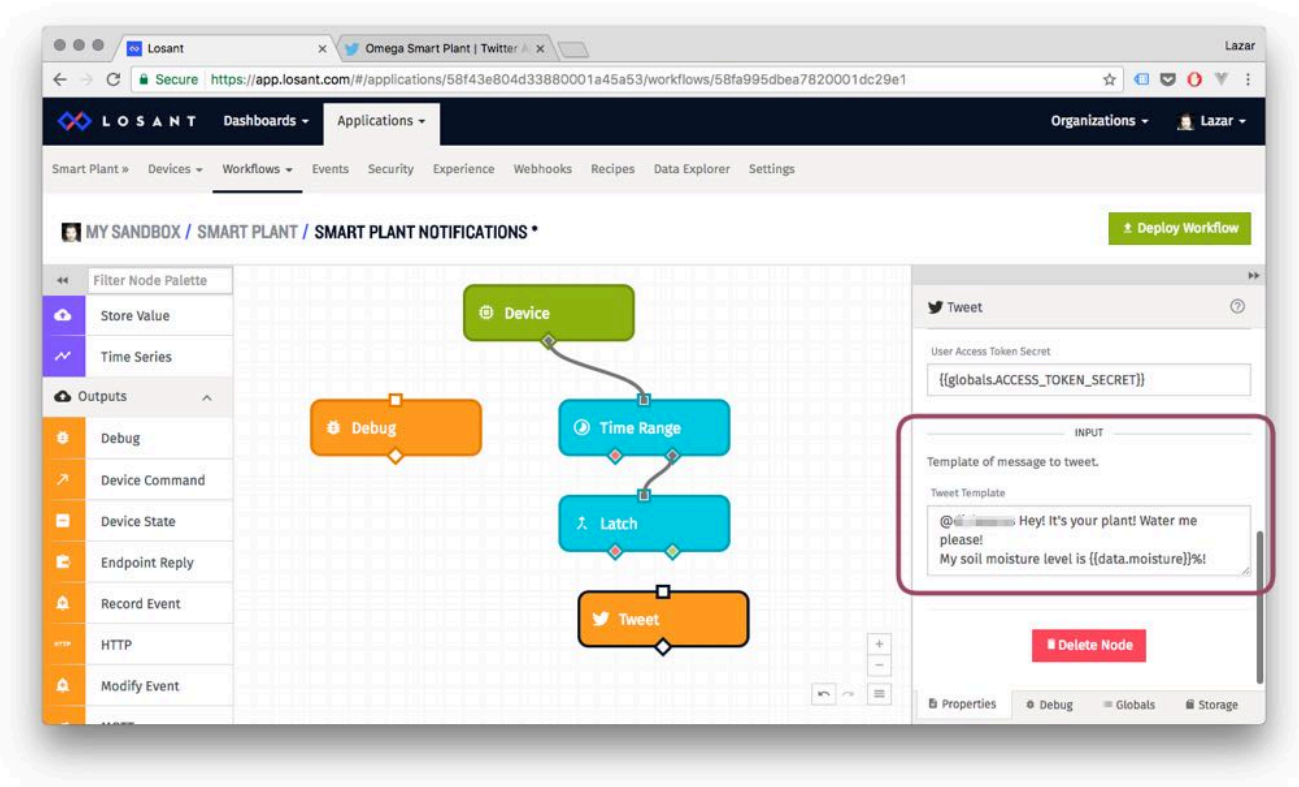
Just like with the consumer key and secret, create `ACCESS_TOKEN` and `ACCESS_TOKEN_SECRET` global variables in the Losant Workflow to hold the Access Token values:



Now we can put the keys into the Twitter node by referencing the global variables:



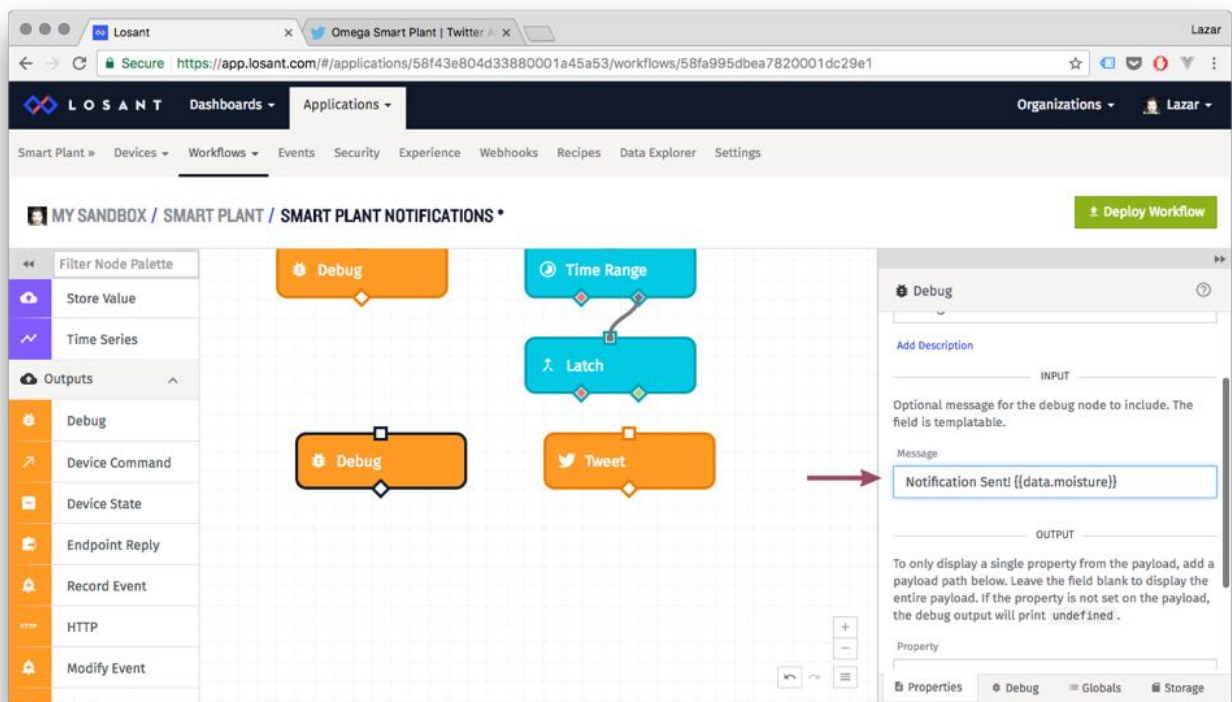
Come up with something you think your plant would say:



The screenshot shows the Losant workflow editor interface. The workflow is titled "SMART PLANT NOTIFICATIONS" and is located in the "MY SANDBOX" environment. The workflow consists of the following nodes: a green "Device" node, an orange "Debug" node, a blue "Time Range" node, a blue "Latch" node, and an orange "Tweet" node. The right-hand panel shows the configuration for the "Tweet" node. The "User Access Token Secret" is set to `{{globals.ACCESS_TOKEN_SECRET}}`. The "Template of message to tweet" is a text area containing the following text: `@{{data.device_username}} Hey! It's your plant! Water me please! My soil moisture level is {{data.moisture}}%!` . A red box highlights the "Template of message to tweet" field. The "Delete Node" button is visible below the template field.

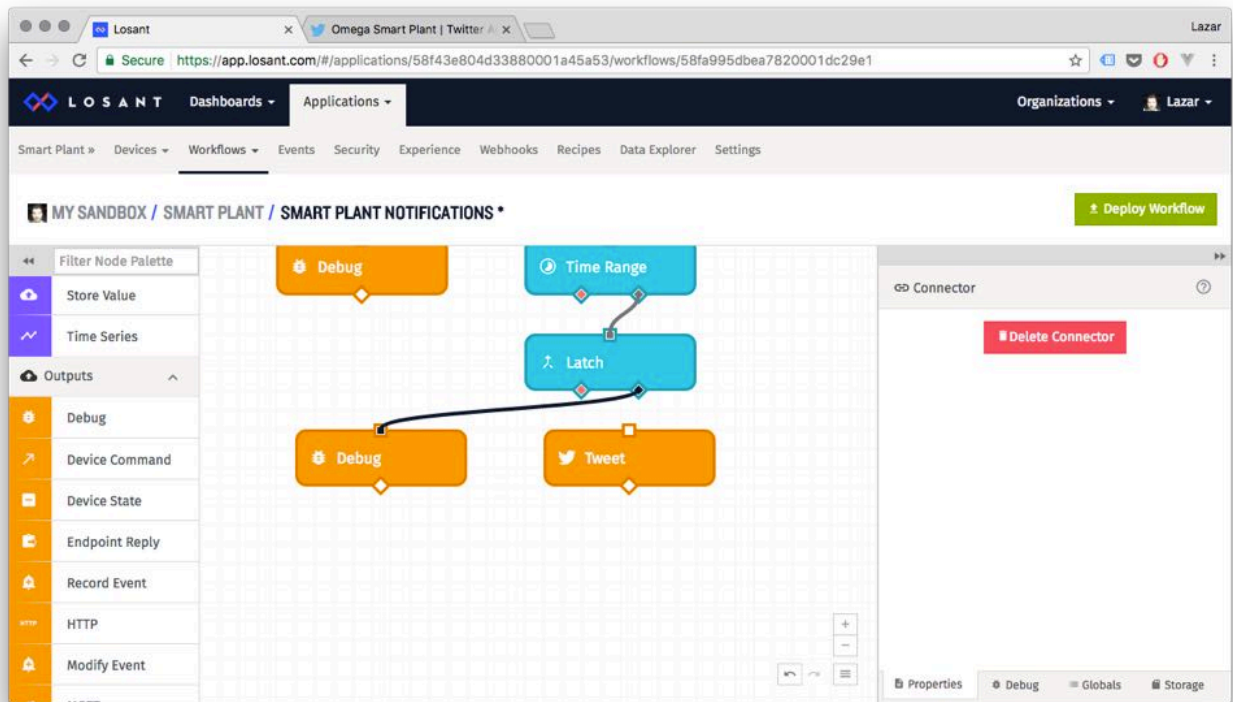
And the Twitter Node is ready for action!

As good practice, let's put in a debug message so we can easily see in the Debug Log when a Tweet should have been sent out:



The screenshot shows the Losant workflow editor interface. The workflow is titled "SMART PLANT NOTIFICATIONS" and is located in the "MY SANDBOX" environment. The workflow consists of the following nodes: a green "Device" node, an orange "Debug" node, a blue "Time Range" node, a blue "Latch" node, an orange "Debug" node, and an orange "Tweet" node. The right-hand panel shows the configuration for the "Debug" node. The "Message" field is set to `Notification Sent! {{data.moisture}}`. A red arrow points from the "Message" field to the "Debug" node in the workflow.

And connect it to the same trigger that will fire the Tweet - the moisture level Latch:

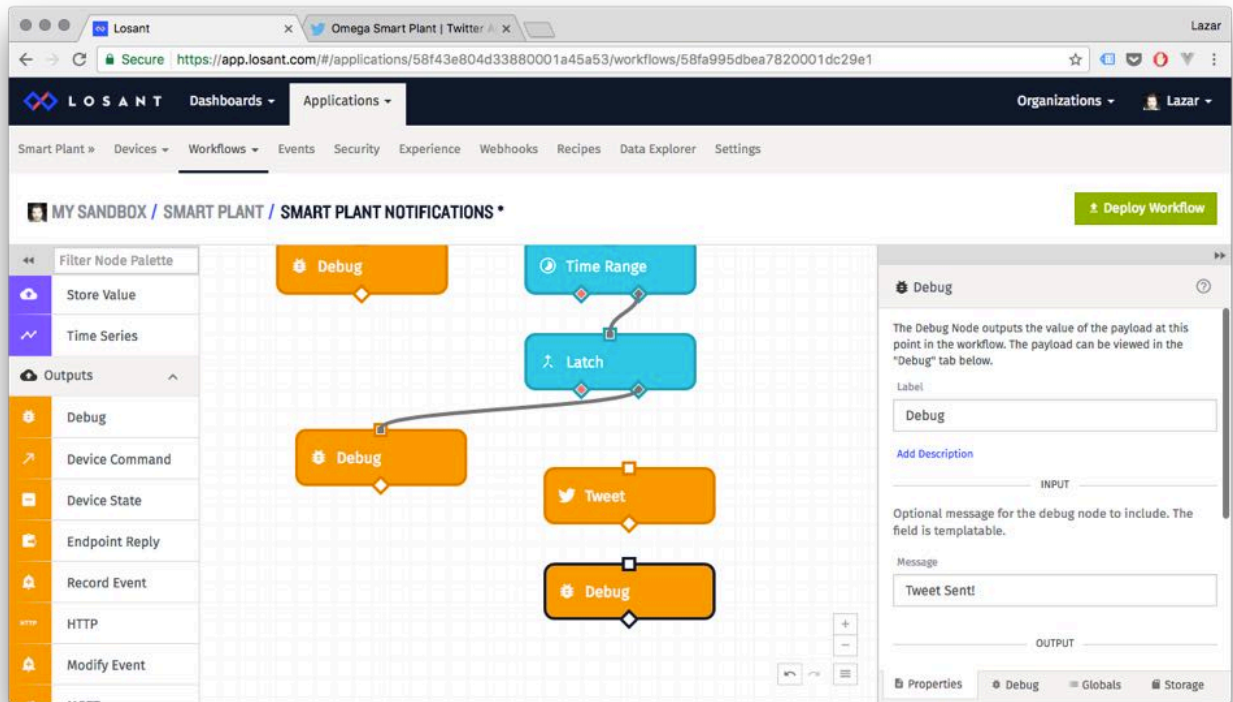


## 8. Test Tweeting

It's a good idea to test out smaller pieces first. So let's make sure that our **Tweet** node works as intended.

First, set up a debug message to follow up on the Tweet event:

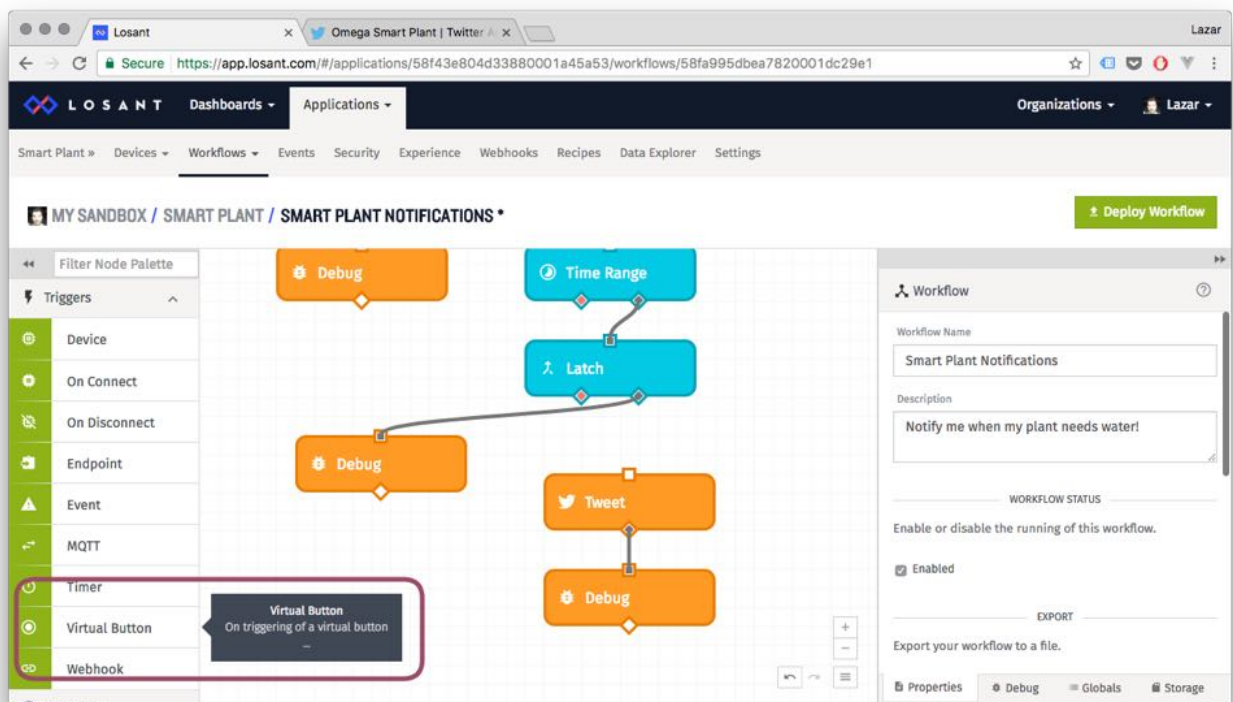




The screenshot shows the Losant workflow editor interface. The main workspace displays a workflow with the following nodes: a 'Debug' node (orange), a 'Time Range' node (blue), a 'Latch' node (blue), a 'Tweet' node (orange), and another 'Debug' node (orange). The workflow is connected as follows: 'Time Range' connects to 'Latch', 'Latch' connects to the first 'Debug' node, and the first 'Debug' node connects to the 'Tweet' node. The 'Tweet' node connects to the second 'Debug' node. On the left, the 'Filter Node Palette' is open, showing various node categories like 'Outputs' and 'Debug'. On the right, a configuration panel for the selected 'Debug' node is visible, showing fields for 'Label' (set to 'Debug'), 'Add Description', 'Message' (set to 'Tweet Sent!'), and 'OUTPUT'.

Connect it to the Tweet node, this will let us know that a Tweet was attempted by Losant - successfully or not.

Now let's add a Button node so we can trigger the Tweet event on demand for testing:



This screenshot shows the same Losant workflow editor interface as the previous one, but with a 'Virtual Button' node added to the 'Triggers' section of the 'Filter Node Palette' on the left. The 'Virtual Button' node is highlighted with a red box and has a tooltip that reads 'Virtual Button On triggering of a virtual button'. The workflow diagram remains the same as in the previous screenshot.

The button needs a payload to send to the node it triggers.

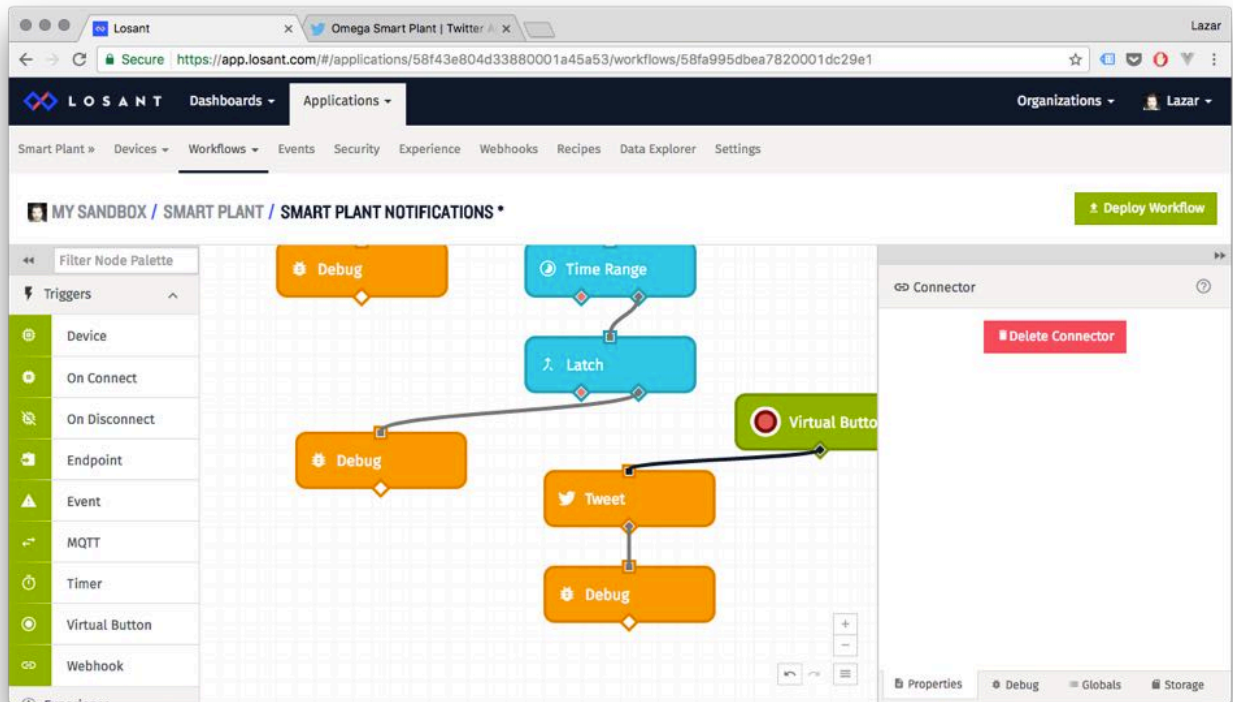
The payload we used is this json string:

```
{"moisture":100}
```

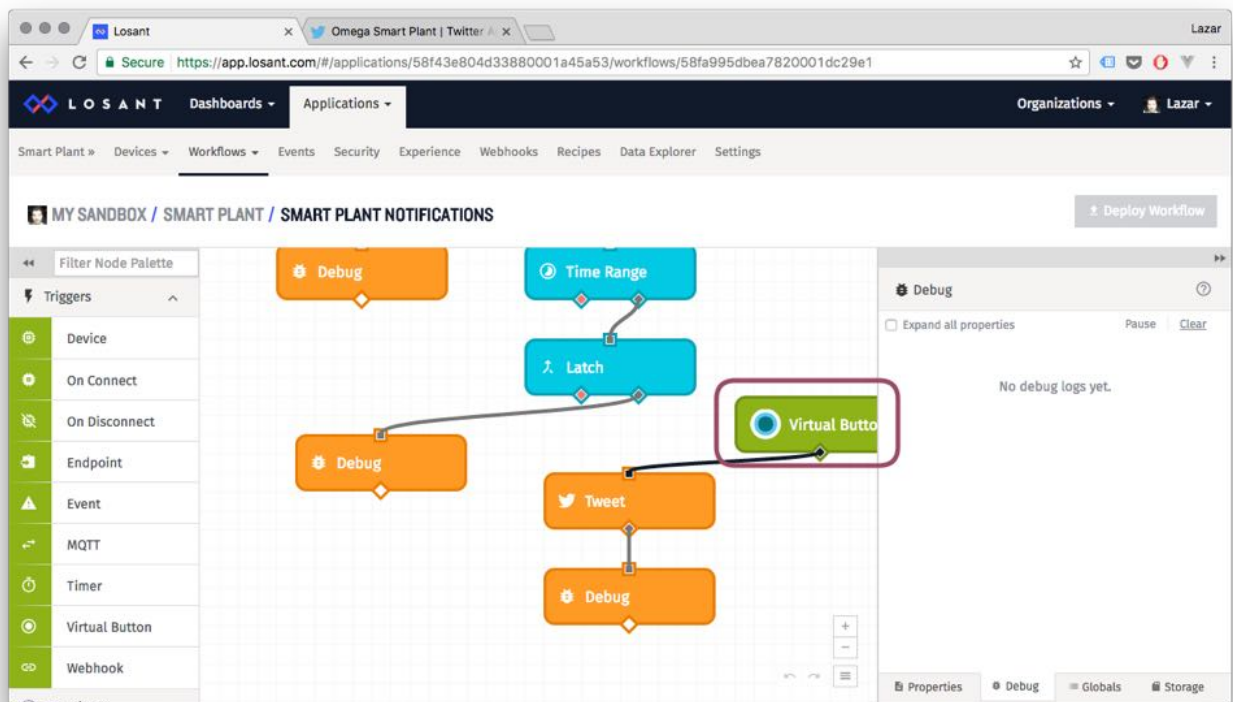
Looking something like this:

The screenshot displays the Losant IoT platform interface. The main workspace shows a workflow diagram for 'SMART PLANT NOTIFICATIONS'. The workflow starts with a 'Virtual Button' node, which triggers a 'Latch' node. The 'Latch' node is connected to a 'Time Range' node, which then triggers a 'Debug' node. Another 'Debug' node is connected to the 'Latch' node. The 'Latch' node also triggers a 'Tweet' node, which is connected to a final 'Debug' node. The 'Virtual Button' node configuration panel is open, showing an example payload and a custom payload field containing the JSON string: {"moisture":100}. The interface also shows a 'Filter Node Palette' on the left and a 'Deploy Workflow' button on the right.

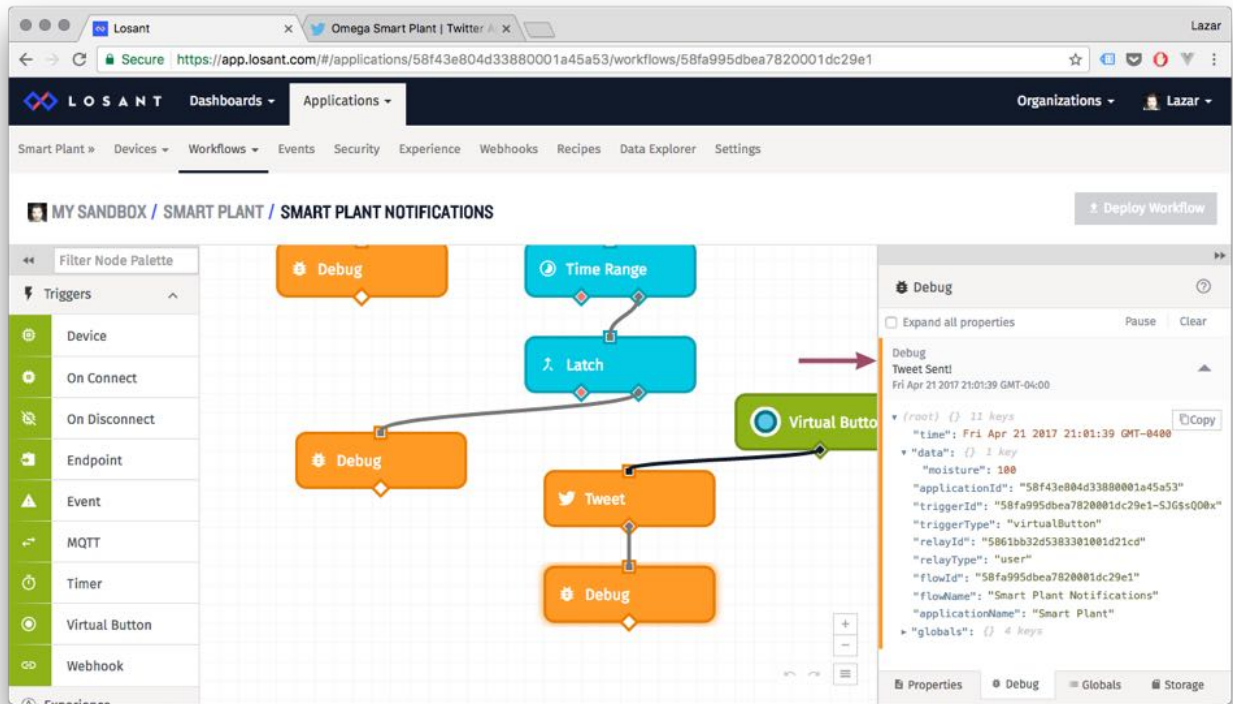
Connect the button to the Tweet node and Deploy the Workflow:



And hit that sucker!



We can see the 'Tweet Sent!' Message in the debug log:



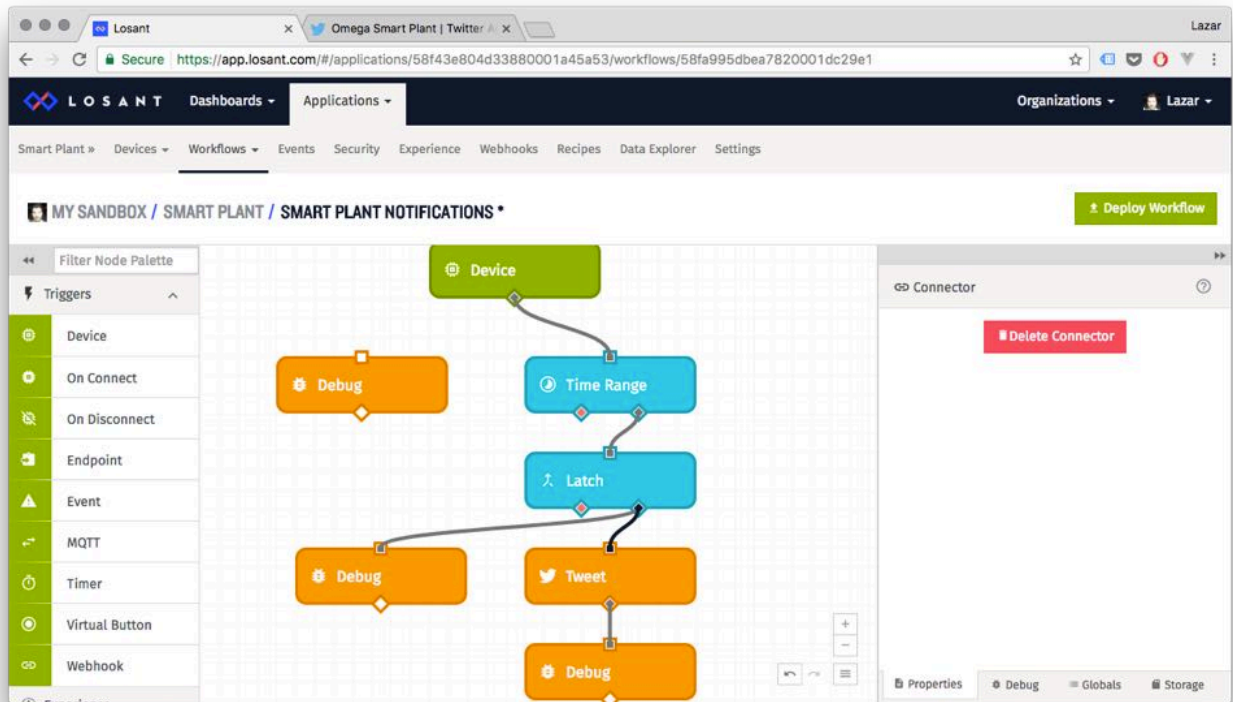
And then check Twitter for the actual tweet:



Looks like the Twitter node is working as expected!

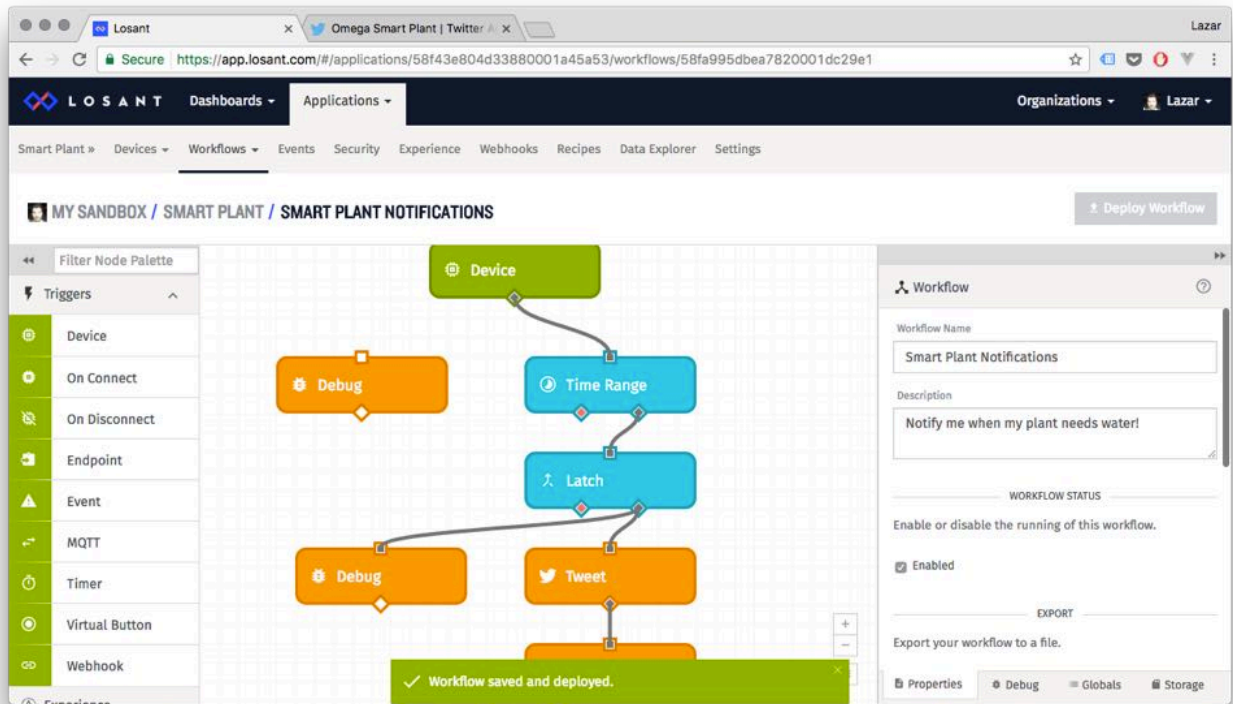
## 9. Complete the Workflow

Now that we're done testing the Tweet node, let's delete the Button block and finish the Workflow. Connect the Twitter Node to the true path of the Latch node:



This ensures the **Twitter** node will be activated (just once) when the **Latch** condition is true, that is, when the soil moisture level drops below the value we set for the `LOW_MOISTURE` global variable. Note that because of the **Latch**, the **Twitter** node will not be activated again until the plant is watered enough so that the soil moisture level rises above the value set for the `OK_MOISTURE` global variable.

Now Deploy the workflow and we're done!

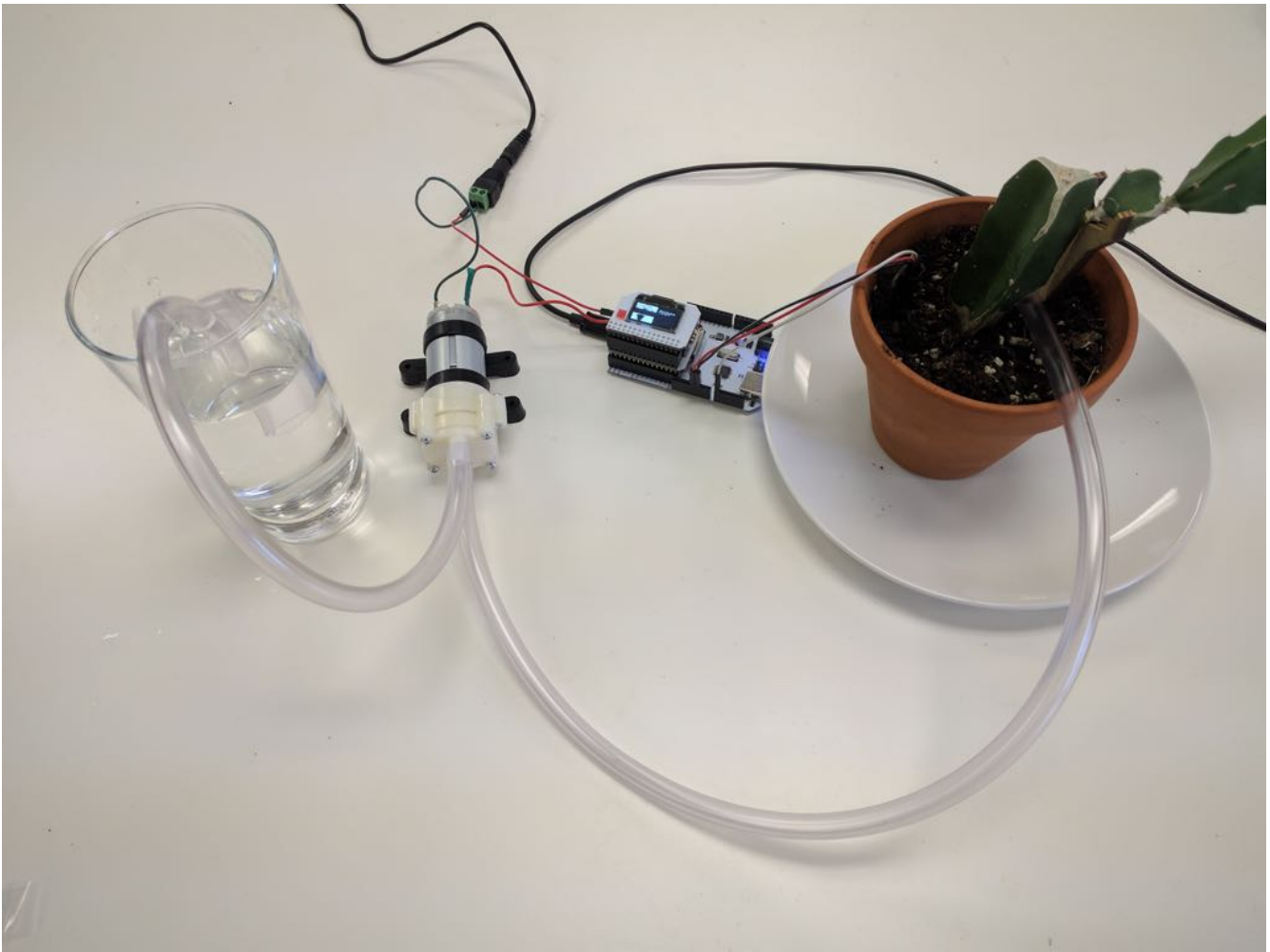


## Going Further

You can extend the Losant workflow to send more types of notifications, such as an SMS text message, email, or even a command to another device. But wouldn't it be nice if we could tell the Omega to water the plant for us?

## Smart Plant - Automatic Plant Watering

Now that our plant is smart enough to Tweet us when it needs water, let's see if we can make it even smarter and have it water itself! For this project, we'll add a water pump to our work in [Smart Plant Part 3](#) so we can automate the watering process.



## Overview

**Skill Level:** Intermediate-Advanced

**Time Required:** 1 Hour

For this project, we'll be using the Relay Expansion to switch a water pump on and off, enabling our smart plant to water itself! To do that, we'll build a circuit to power the water pump, and use the [Relay Expansion Python Module](#) to control the pump with our script.

Once our pump works as expected, we'll build a new Losant **workflow** to test it out. Finally, we'll add it to the Losant workflow we've built in [Smart Plant Part 3](#) to get our plant to water itself!

The complete project code can be found in Onion's [smart-plant repo on GitHub](#).

## Ingredients

We'll need all of the same materials as in the previous parts:

- Onion [Omega2](#) or [Omega2+](#)
- Onion [Arduino Dock 2](#)
- Onion [OLED Expansion](#) (optional but recommended)

- Soil Moisture Sensor
- 3x Male-to-Female Jumper Wires

And some new ingredients:

- Onion Relay Expansion
- DC Barrel Jack Adapter
- 12V 1A DC Power Supply
- 3x Male-to-Male Jumper Wires
- Water Pump (12V DC)
- Flexible Plastic Tubing
  - Make sure to match the tubing's **inner** diameter (ID) is slightly less than the pump's ports' **outer** diameter (OD). This is so the tubing will stretch and grip the ports, preventing any leaks!
- A piece of paper the size of your hand to test the pump's polarity
- A plate or bowl to hold your plant and collect excess water
- A glass or bowl of water you can use as a reservoir

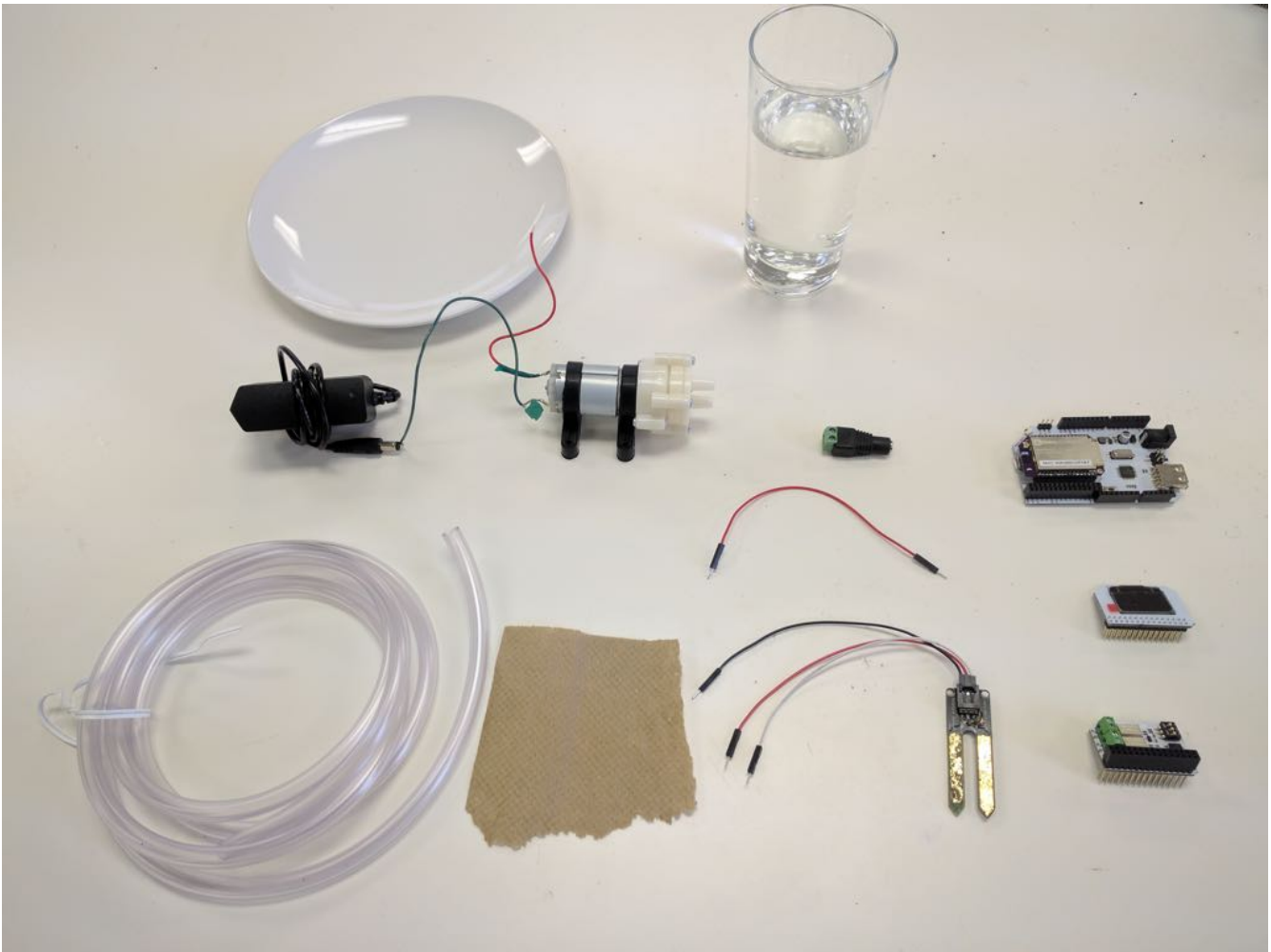
Tools:

- Flat-head screwdriver
- Philips-head screwdriver

If your pump does not come with wires attached, then you will need:

- Electrical Tape
- Wire Cutters
- Wire Strippers





## Step-by-Step

Follow these instructions to set this project up on your very own Omega!

### 1. Prepare

You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 2. Complete the Previous Parts of the Project

This project builds on the previous parts of the Smart Plant project. If you haven't already completed the [first](#), [second](#), and [third parts](#), go back and do them now!

### 3. Install Required Software on the Omega

To control the Relay Expansion from a Python program, you'll need to install the [Onion Relay Expansion Python Module](#):

```
opkg update
opkg install pyRelayExp
```

#### 4. Prepare the Pump

Let's prepare the pump so that we can connect it to our circuit. We'll be doing a few sub-steps here:

1. Preparing the Power leads
2. Connecting to the Barrel Jack Adapter
3. Locating the Inlet and Outlet
4. Determining the Polarity

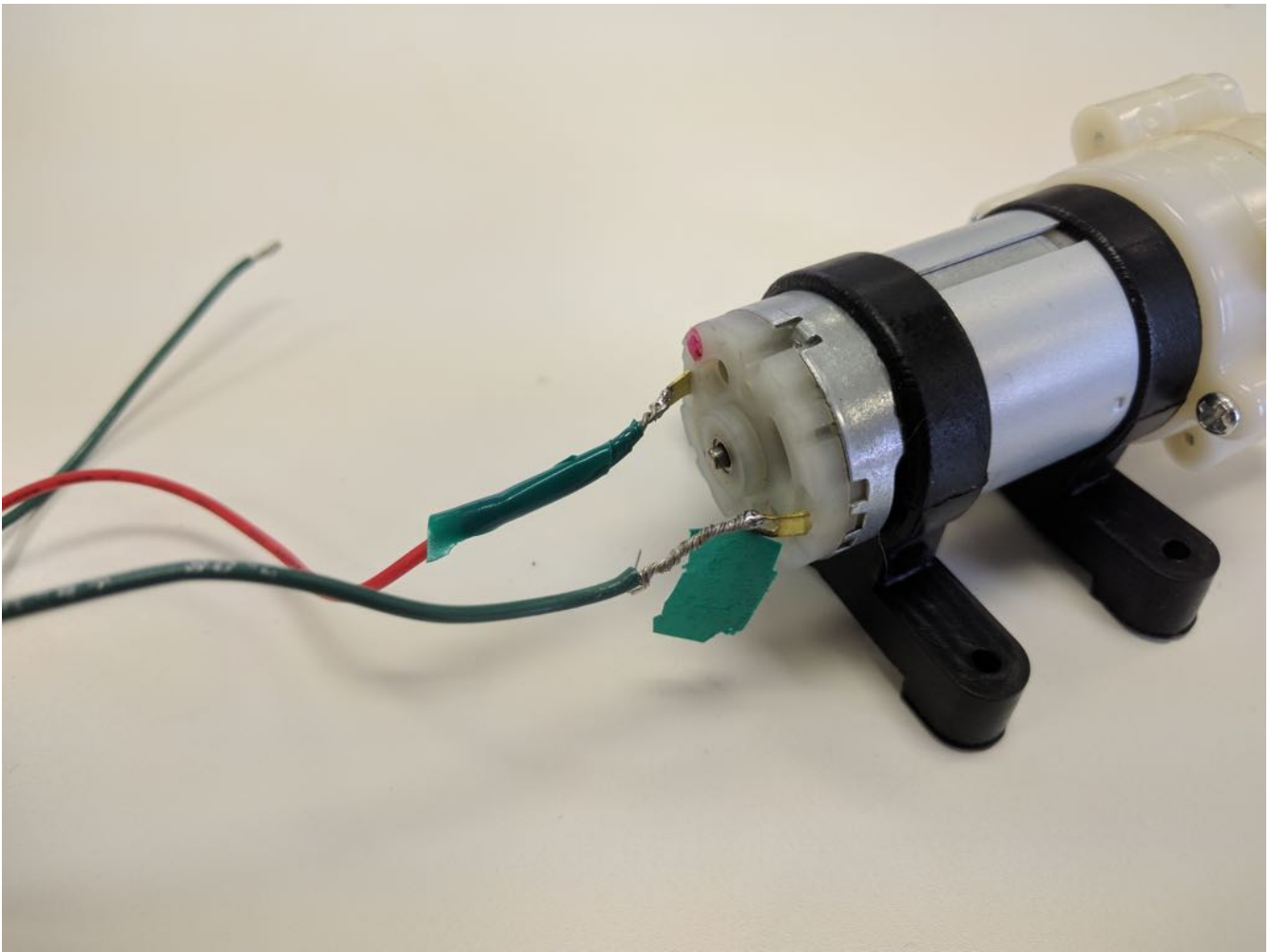
##### Preparing the Power Leads

Not all water pumps come with wires attached to the 2 power leads on the back/top. If yours does, skip to the "Wire the Pump" step.

If yours does not, you will need to put the wires together and determine the pump motor's polarity. This is extremely important because connecting it wrong will, at best, reverse the in and out ports and, at worst, break the pump!

Take two pieces of jumper wire, one red and one black, and strip about 1" from the ends. Some pumps without wires attached may have a marking for where to attach the red wire; examine your pump thoroughly for any hints. If you can find a marking, loop the bare end of the red wire through the hook. Just twisting it around and then covering it with electrical tape is enough. Repeat for the black wire.

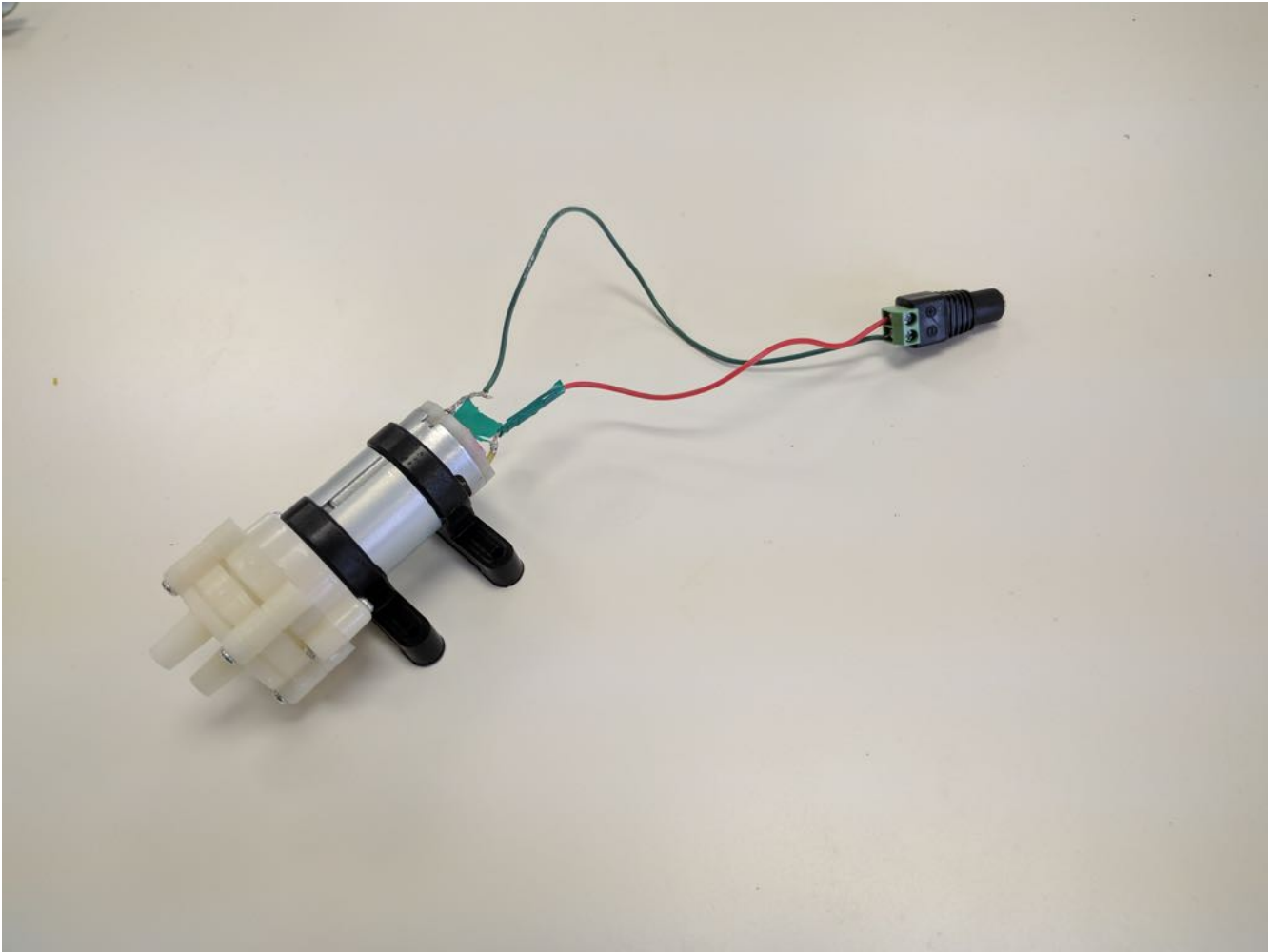
If there are no markings, connect them to the leads whichever way. If the order happens to be wrong, you can switch them later.



### Connecting to the Barrel Jack Connector

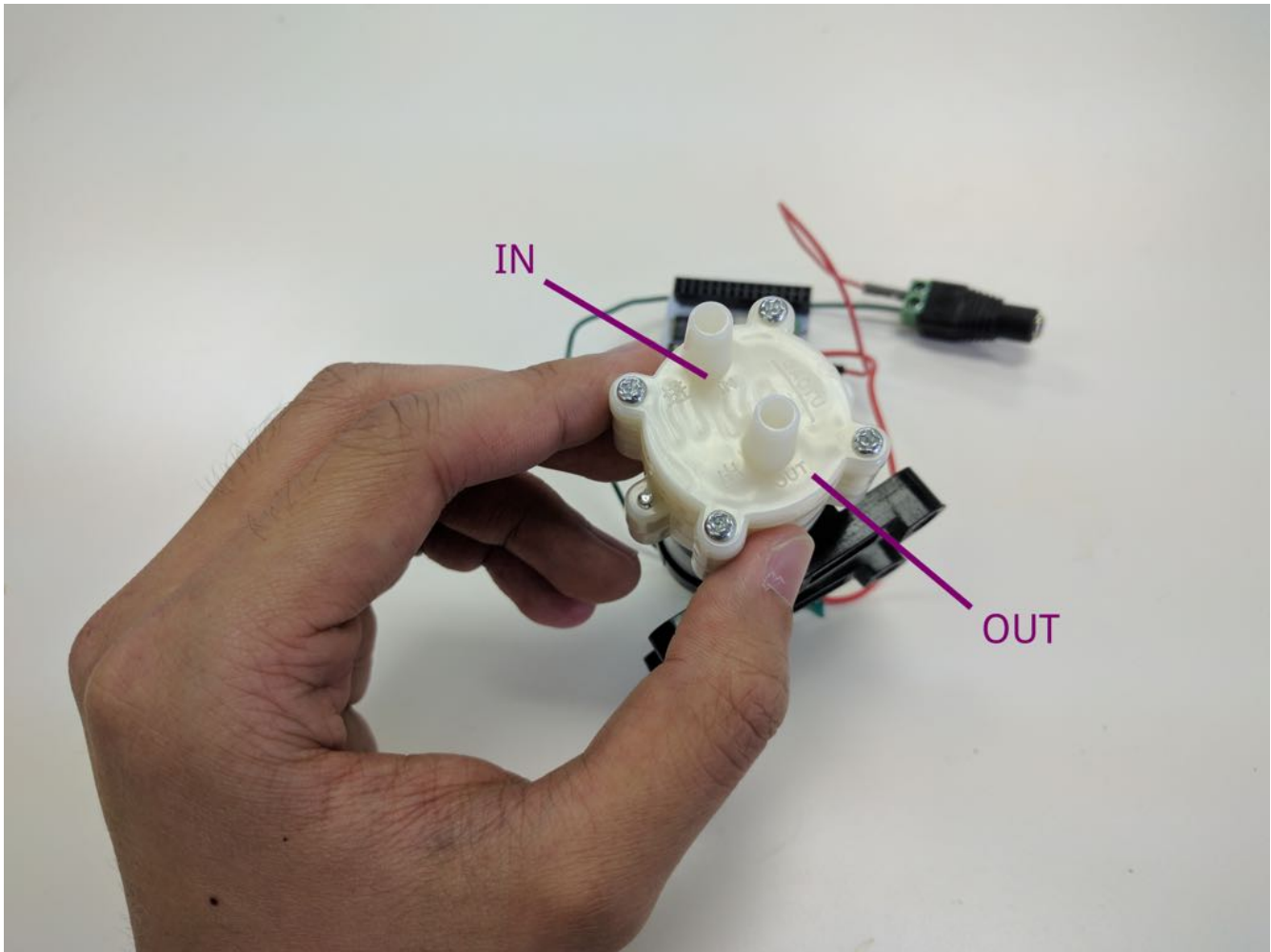
With the 12V power supply **not** plugged in yet, connect the other end of the red wire to the (+) terminal on the barrel jack adapter, and the other end of the black to the (-) terminal.

The screw terminal on the barrel jack adapter will rise and sink depending on the clamp position. When the screw is roughly flush with the top, it is open. To attach a wire, insert it into the terminal and turn the screw clockwise until it sinks to about halfway, or until it becomes difficult to continue turning.



### Locating the Inlet and Outlet

Examine your pump's instruction manual (if there is one) or the ports to see if there are any markings or labels for "IN" and "OUT". Our pump had them in raised letters on the plastic housing:



### Determining Polarity

Now we will **briefly** run the pump to make sure the polarity of the motor is correct.

There is a risk of it overheating when running the pump “dry” without water for extended periods of time. This is because some pumps rely on water for cooling. Try not to leave it on for more than 10 seconds to avoid the risk of damage to your pump.

Prepare a small piece of paper about the size of your hand. Then plug in the 12V power supply and hear your pump come to life!

Move the piece of paper towards the outlet.

- If it gets blown away from it, the polarity is correct.
- If it gets sucked towards it, the wiring is backwards.

Unplug the power supply and remove the two wires from the barrel jack adapter. If the polarity is backwards, switch where the red and black wires are connected to the pump.

- The terminal on the pump where the red wire should be connected is known as the **positive (+)** terminal.
- Likewise, the place where the black wire should be connected is the **negative (-)** terminal.

## Optional - Solder the Terminals

If you want, you can solder the wires to the pump terminals to make the connections more secure. You'll need to remove the electrical tape, solder the terminals, then replace the tape again.

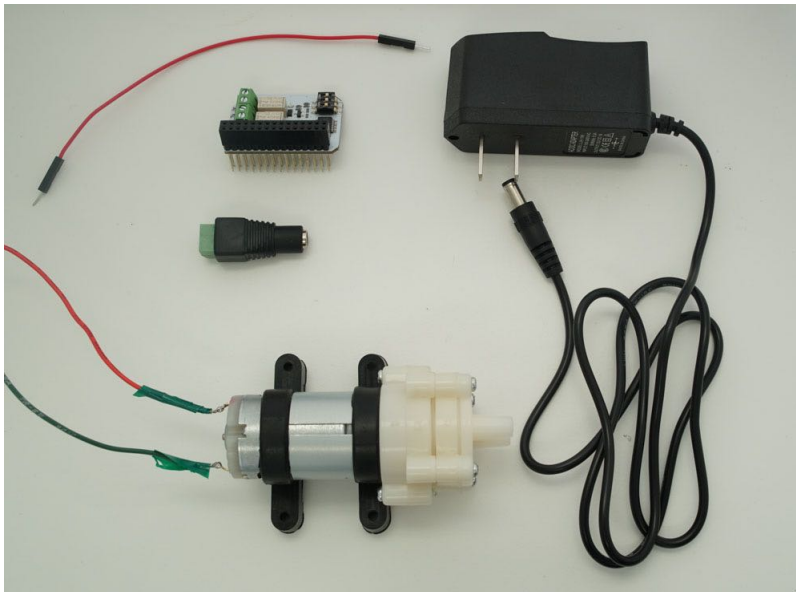
Please familiarize yourself with proper soldering technique and safety procedures when working with soldering irons, as there is a risk of injury due to the high heat!

If you are not comfortable soldering, try finding a friend or professional who can quickly solder it for you. Or practice soldering wires together and then work your way up to soldering on actual electronics.

**Note:** Solder at your own risk, Onion is not responsible for any injury or damage!

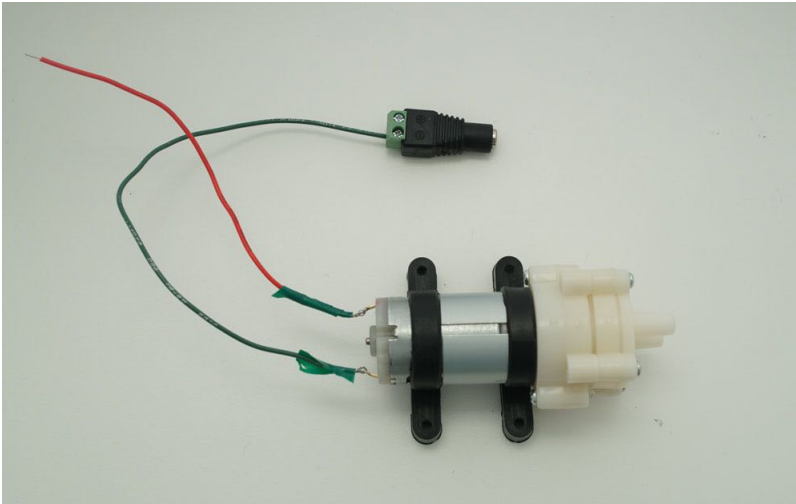
## 5. Connect the Pump to the Omega

We'll wire up the Water Pump with the Relay Expansion before connecting the Relay Expansion to the Dock.

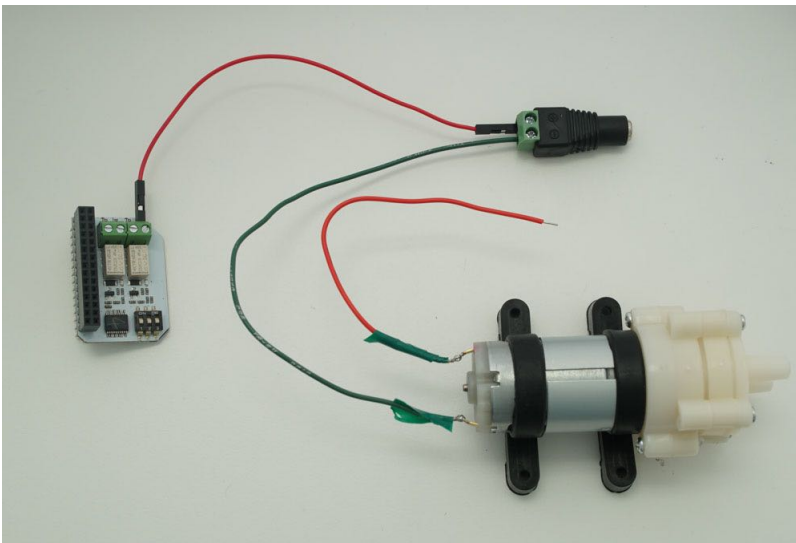


To set up the terminals on the Relay Expansion, turn the screw on the terminal counterclockwise until the metal clamp inside is sitting a bit less than halfway in the bottom of the housing, not too much or the screw might pop out.

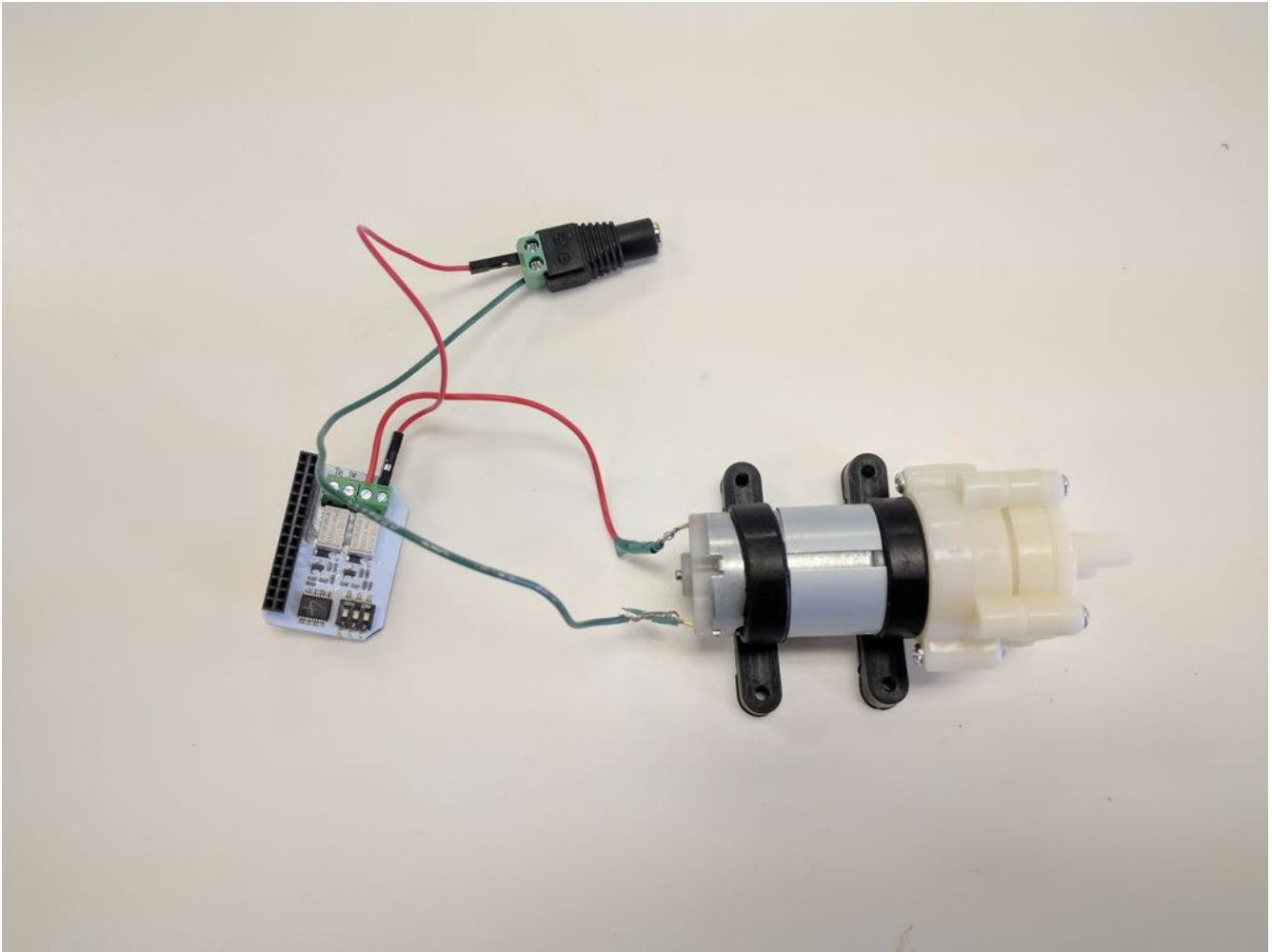
1. Run a jumper wire from the **negative terminal** of the DC Barrel Jack Adapter to the **negative terminal** of the water pump.



1. Run a jumper wire from the **positive terminal** of the DC Barrel Jack Adapter to the **IN** screw terminal on Channel 0 of the Relay Expansion.



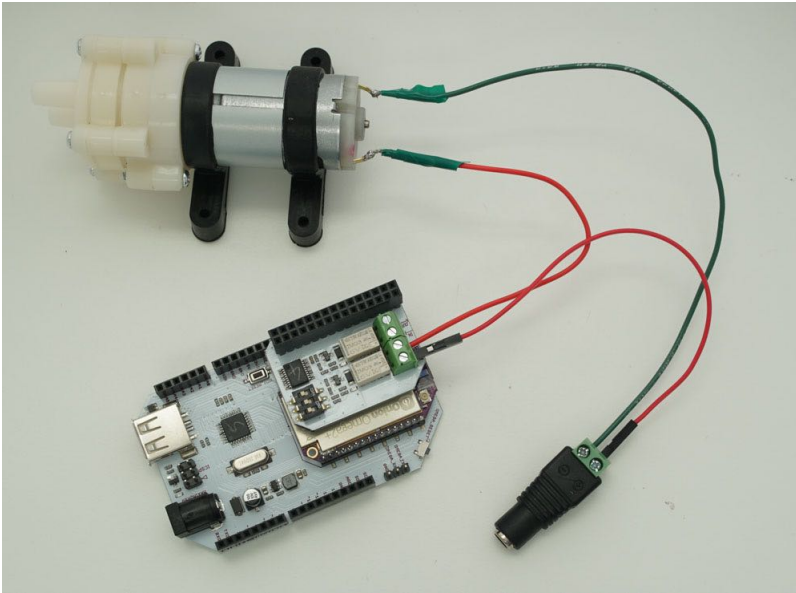
1. Run a jumper wire from the **OUT** screw terminal on Channel 0 of the Relay Expansion to the **positive terminal** of the water pump.



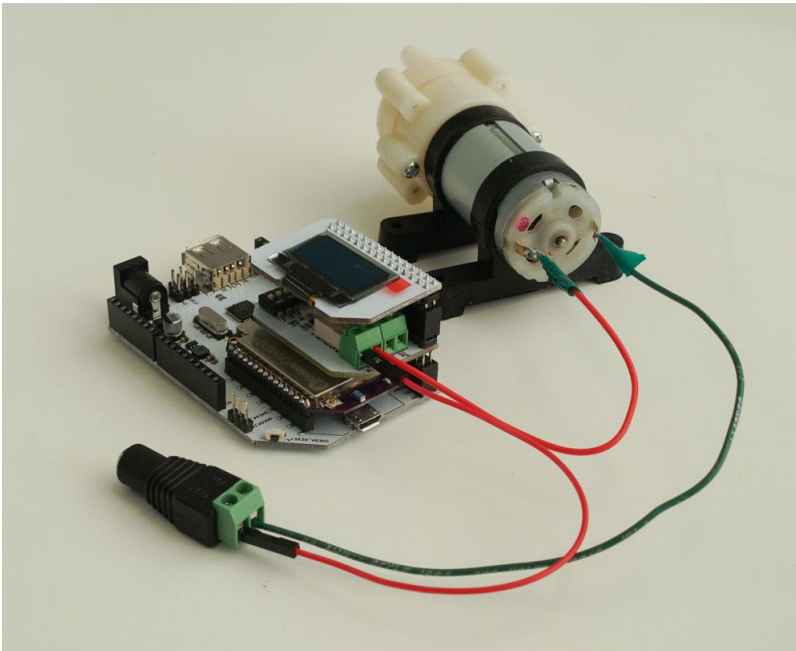
## 6. Connect the Relay Expansion and Provide Power

Grab your Smart Plant Omega and Arduino Dock and unplug it from power. Take off the OLED Expansion and plug in your freshly wired Relay Expansion.

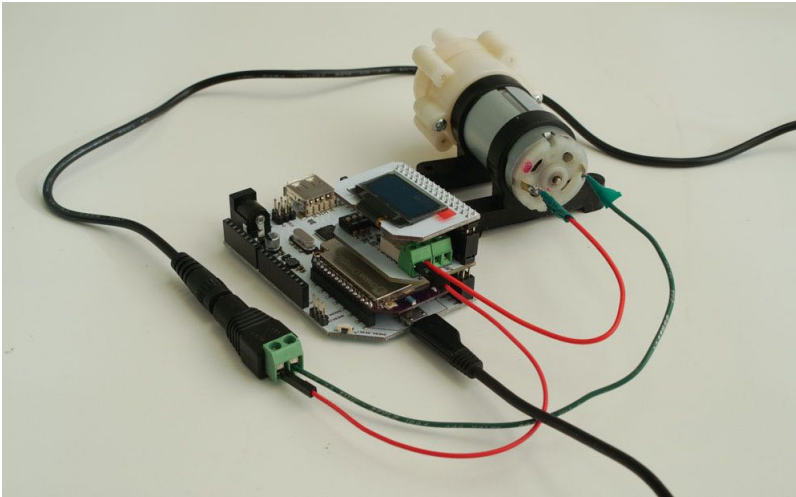




You can then plug the OLED Expansion into the Relay Expansion.



Power the Omega and Arduino Dock through the Micro-USB port and connect the 12V power supply to the DC Barrel Jack Adapter:



## 7. Test your Setup

When your Omega boots up again, login to the Omega's command line and run the following command to turn on the relay connected to the water pump:

```
relay-exp 0 on
```

Your pump should now come to life!

Turn off the pump for now by running the following:

```
relay-exp 0 off
```

For more info on the `relay-exp` command, see our [Relay Expansion documentation](#).

## 8. Tubing and Sensor Setup

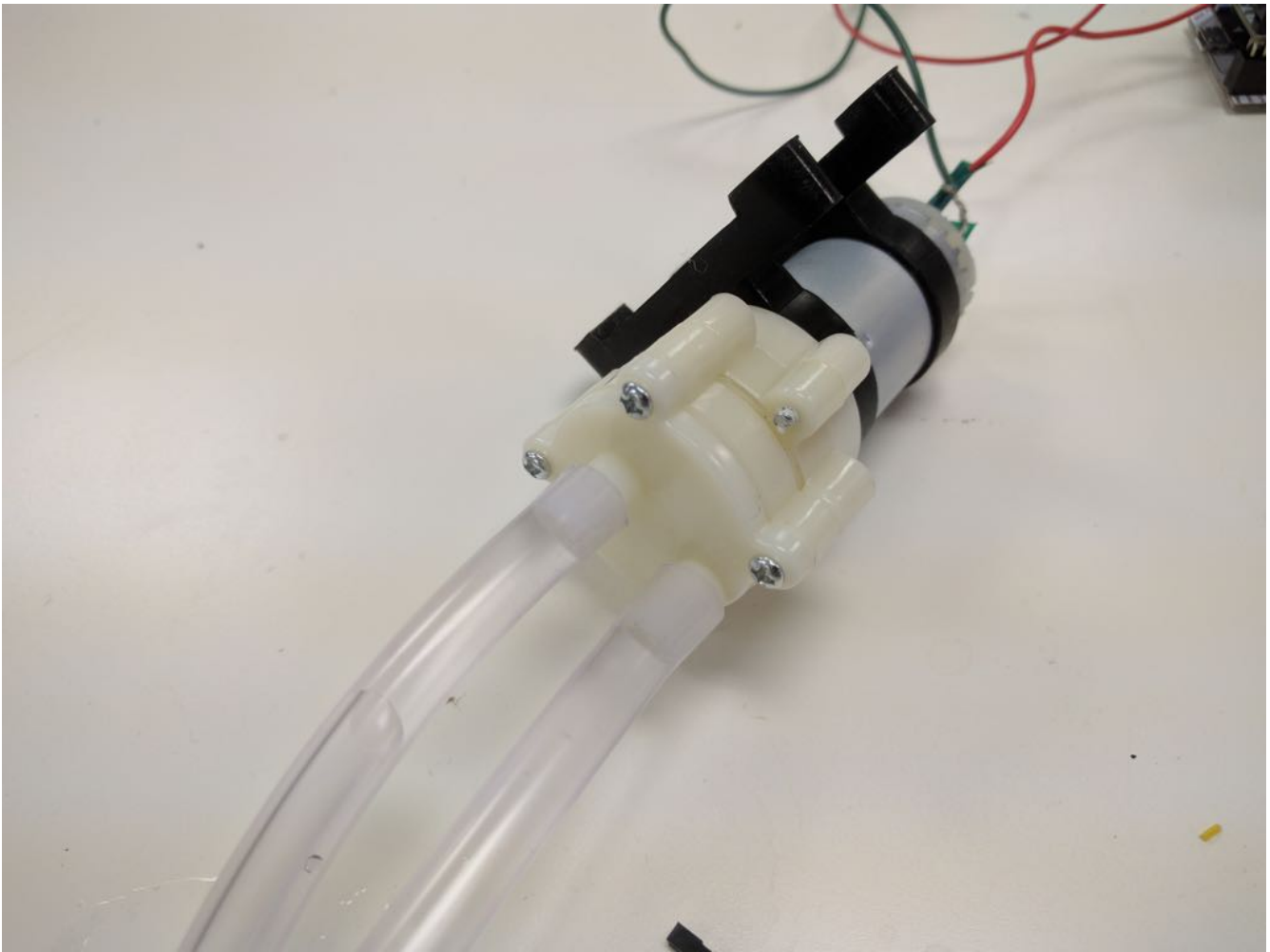
Before we connect the tubing, disconnect the motor from the circuit. This is so you can more easily work with the pump and avoid spilling water on your components.

Place your plant in the plate to catch any excess water. Then prepare a water reservoir; it can be as simple as a big drinking glass. Then measure a length of tubing that would go from the bottom of your reservoir to the inlet of the pump. Cut off the tubing, then first fit one end to the pump's inlet. Make sure it's snug and tight to avoid leaks!



Then insert the other end into your reservoir. We recommend securing it to the reservoir using some tape so that the tubing stays safe.

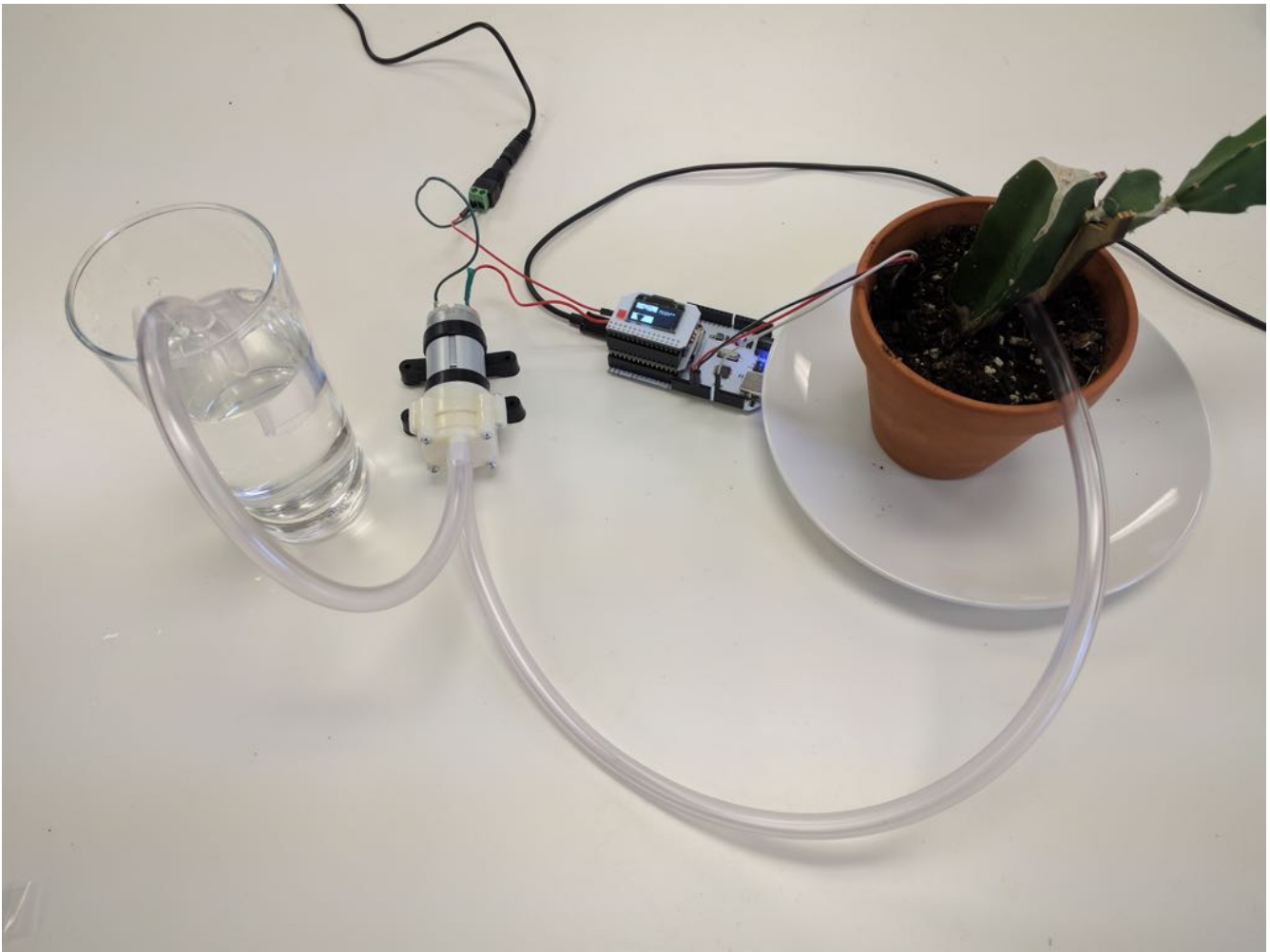
Repeat this process for another piece of tubing that will go from the pump outlet to your plant.



Now connect the motor back to the circuit. Then reconnect the moisture sensor:

1. Connect the Arduino Dock's **5V pin** to the sensor's **Vcc** pin.
2. Connect the Arduino Dock's **GND** pin to the sensor's **GND** pin.
3. Connect the Arduino Dock's **A0** pin to the sensor's **SIG** pin.

and plug in the power supplies for the Omega and pump:



The Omega should now be booting.

## 9. Stop the Existing Program

In the first part of the project, we added a script to `/etc/init.d` to automatically run the smart plant program on boot. We'll now need to stop the program before running it again manually.

Run the following command:

```
/etc/init.d/smart-plant stop
```

## 10. Pump Calibration

We'll need to play with the pump a little to see how long it should be enabled in order to properly water the plant.

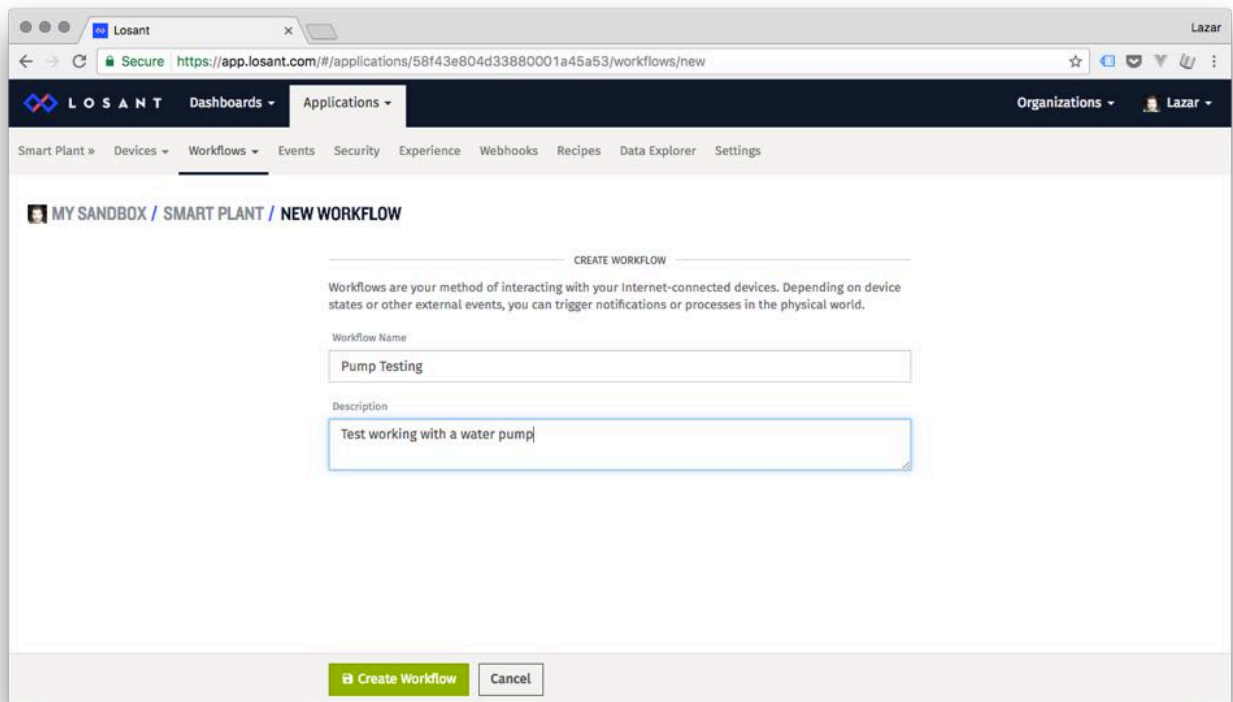
Let's first run the Smart Plant program with the pump option enabled:

```
python /root/smart-plant/smartPlant.py --oled --quiet --losant /root/smart-plant/losant.json --pu
```

The `--pump` option enables receiving and reacting to commands from Losant. It calls the `pumpWater()` function that accepts a duration in seconds for how long to turn the pump on.

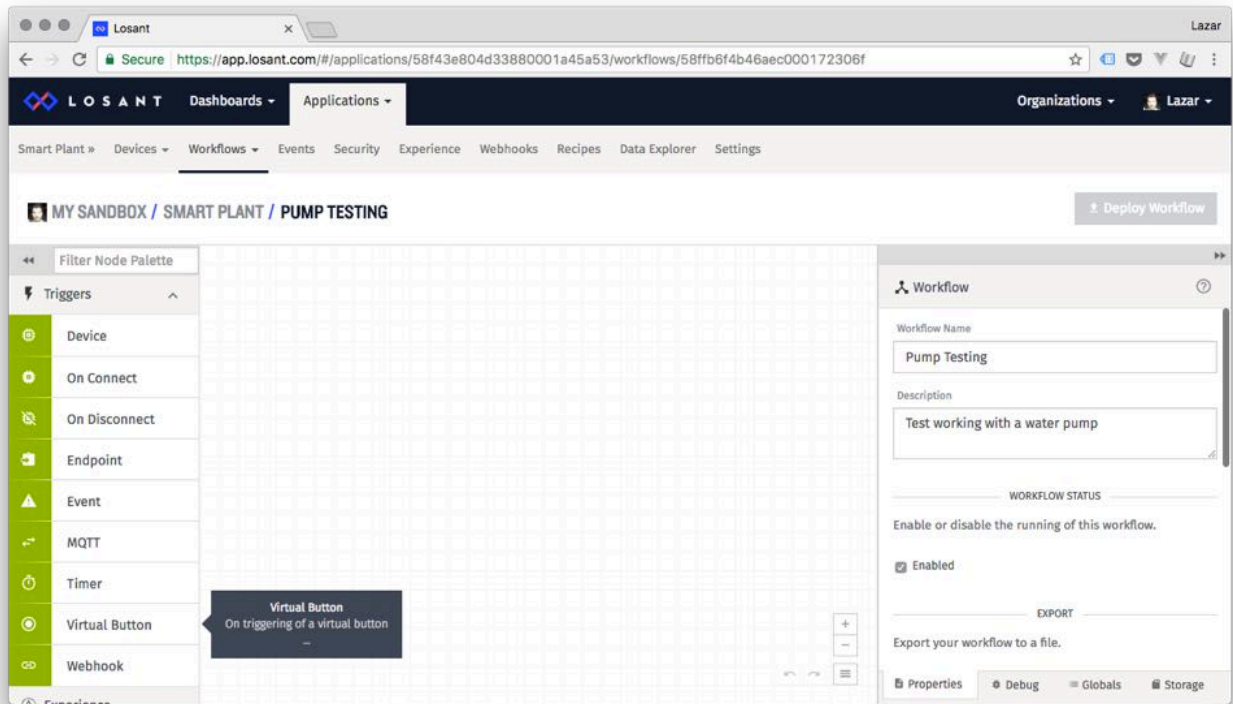
Now, let's create a Losant workflow where we can use a virtual button to turn on our pump. We'll use this workflow now to find the optimal watering duration for your plant, and then afterwards you can use it to water your plant from anywhere in the world!

Head over to [Losant.com](https://losant.com) and log in. Select your Smart Plant Application, click on the Workflows menu and then Create Workflow. Give your workflow a name and a description:

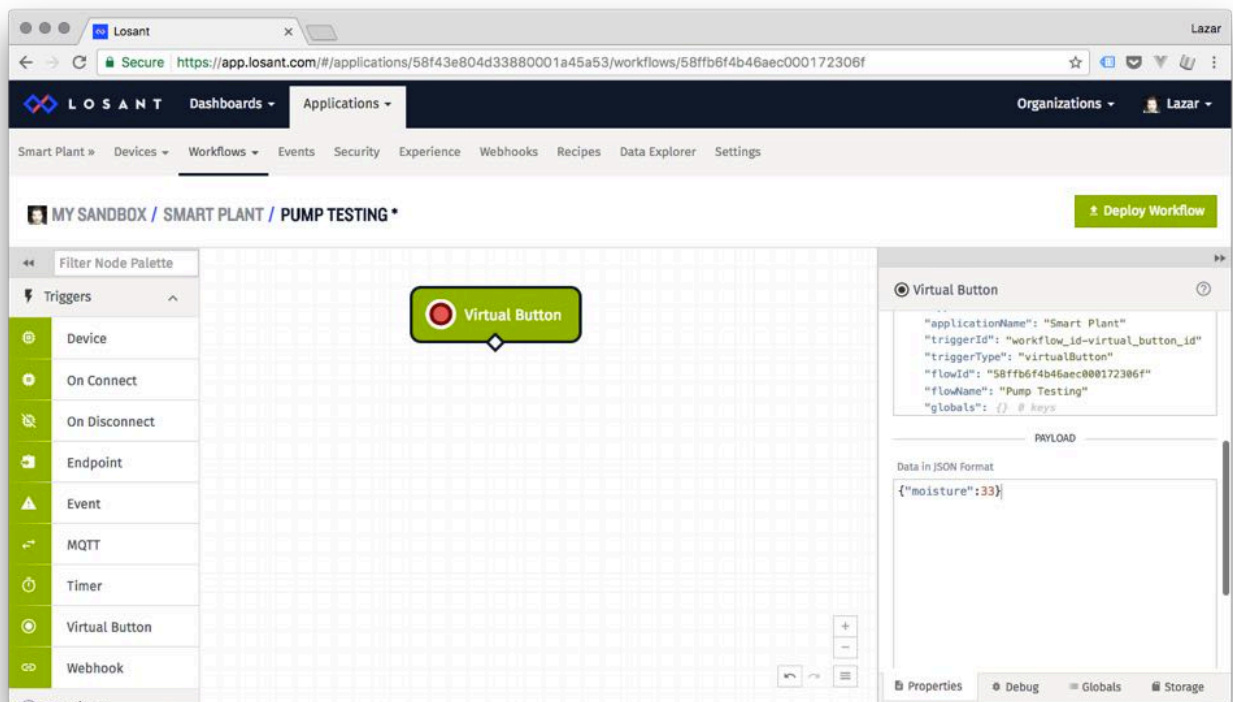


The screenshot shows a web browser window with the Losant interface. The URL is <https://app.losant.com/#/applications/58f43e804d33880001a45a53/workflows/new>. The page title is "MY SANDBOX / SMART PLANT / NEW WORKFLOW". The main heading is "CREATE WORKFLOW". Below this, there is a paragraph explaining that workflows are used to interact with internet-connected devices. The form contains two fields: "Workflow Name" with the value "Pump Testing" and "Description" with the value "Test working with a water pump". At the bottom of the form, there are two buttons: "Create Workflow" (highlighted in green) and "Cancel".

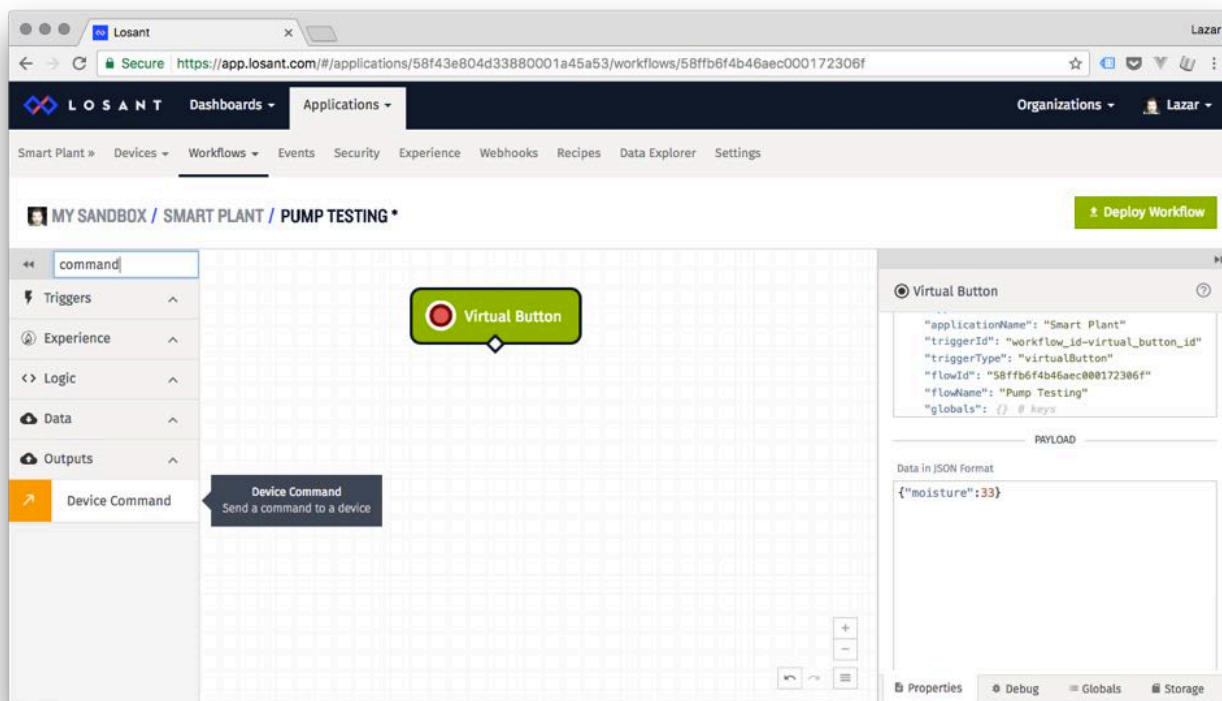
Add a Virtual Button block to your workflow:



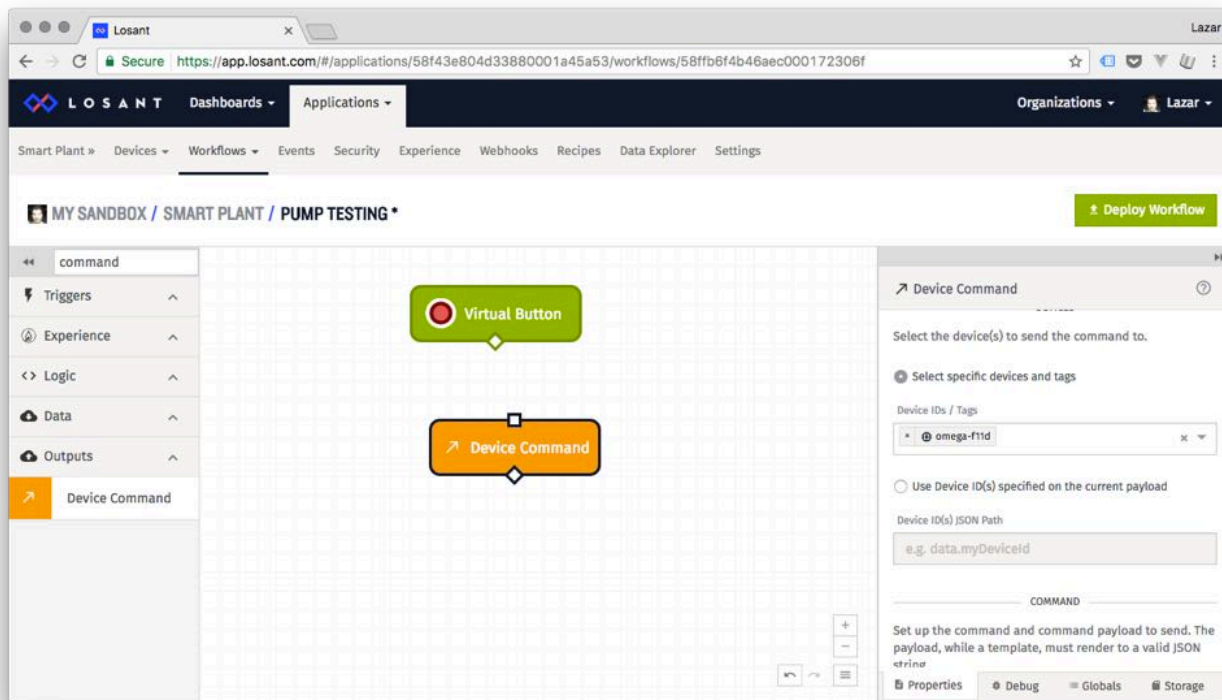
For completeness, have the button send a payload:



Now add a Device Command block:



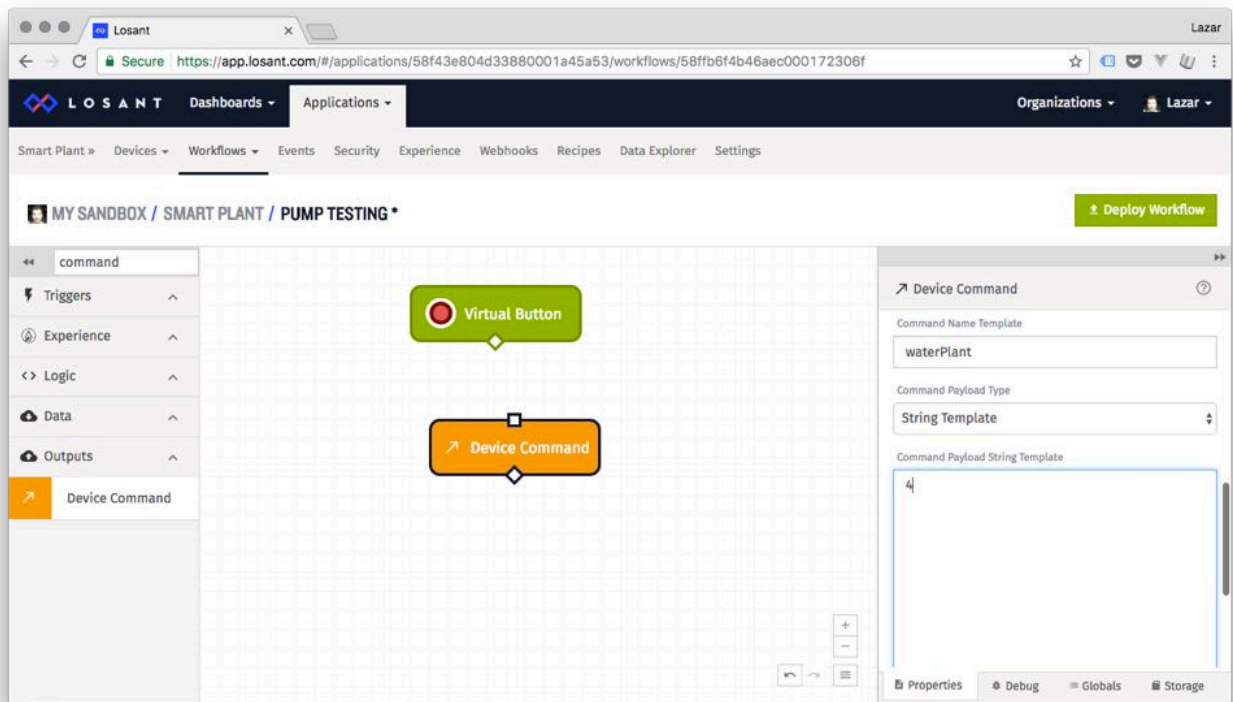
Set it up to use the device associated with your Smart Plant Omega, in our case, that was `omega-f11d`:



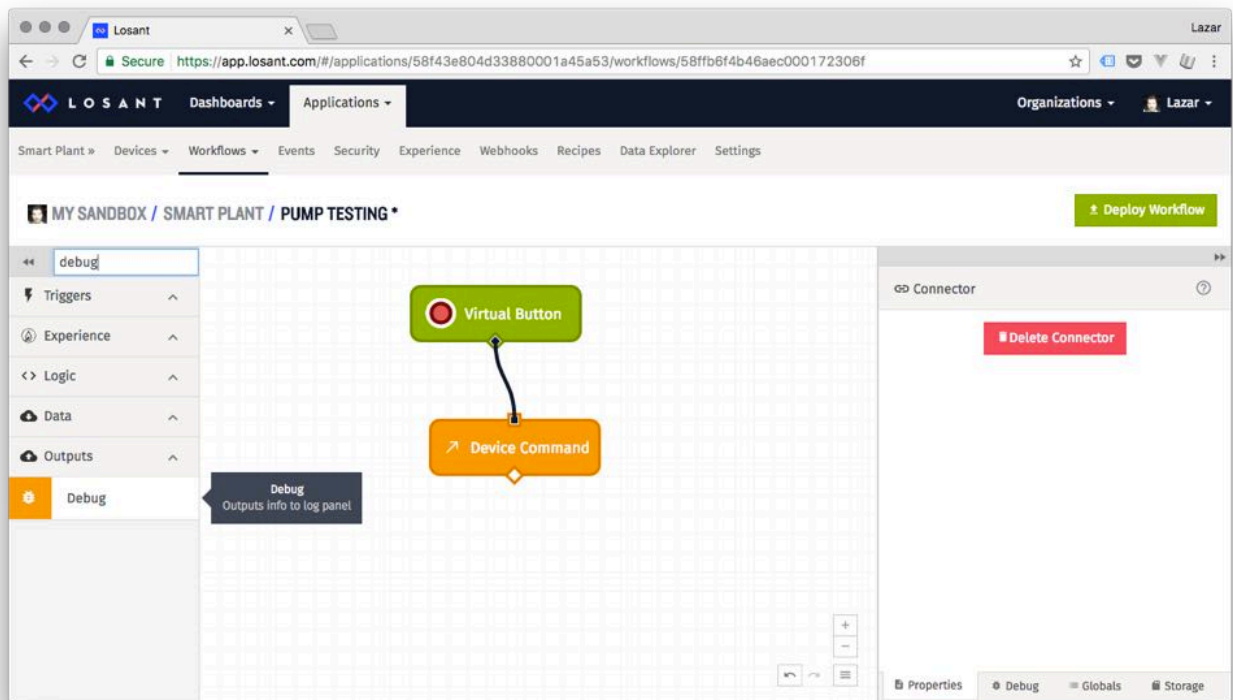
Now we need to setup the command to send to the device. The name of command the program on the Omega is expecting is `waterPlant`, the payload is a string that is the number of seconds to keep the



pump enabled. In our case, we started with 4 seconds:



Let's also add a Debug block:



Set the debug message to something simple so we know our button click has gone through, and Deploy

the workflow:

The screenshot shows the Losant workflow editor interface. The workflow is titled "MY SANDBOX / SMART PLANT / PUMP TESTING". It consists of three nodes connected in a vertical sequence: a green "Virtual Button" node at the top, an orange "Device Command" node in the middle, and an orange "Debug" node at the bottom. The left sidebar shows a list of components including Triggers, Experience, Logic, Data, Outputs, and Debug. The right sidebar shows the configuration for the selected "Debug" node, with the "Message" field set to "command sent!".

Try pressing the button and seeing how much water actually makes it to your plant:

This screenshot shows the same workflow editor after the "Virtual Button" node has been triggered. The "Virtual Button" node is now highlighted with a blue circle. The "Debug" node's output is visible in the right sidebar, showing a JSON payload:

```

{
  "time": "Tue Apr 25 2017 16:55:40 GMT-0400",
  "data": {
    "moisture": 33
  },
  "applicationId": "58f43e804d33880001a45a53",
  "triggerId": "58ffb6f4b46aec000172306f-MyXuvEaCx",
  "triggerType": "virtualButton",
  "relayId": "5861bb32d538301001d21cd",
  "relayType": "user",
  "flowId": "58ffb6f4b46aec000172306f",
  "flowName": "Pump Testing",
  "applicationName": "Smart Plant",
  "globals": {}
}

```

The "Debug" node's output is displayed in a scrollable area, and the "Copy" button is visible next to the JSON payload.

Experiment with the payload of the Device Command block to see how much water suits your plant. Also,

keep in mind the `LOW_MOISTURE` and `OK_MOISTURE` levels we set in the [previous part of the project](#), when your plant is watered at the `LOW_MOISTURE` level, the amount of water added should take it back up above `OK_MOISTURE` level.

We recommend starting at **1 second** and adjusting from there to avoid accidentally overflowing. In our lab, we found a duration of **3 seconds** to work well, but this depends on both your pump and plant.

## 11. Update Program Run at Boot

Since we now need to run the Smart Plant program with additional arguments to use the pump, we'll need to update the `/etc/init.d/smart-plant` file.

Open the `/etc/init.d/smart-plant` file using the Vi editor: `vi /etc/init.d/smart-plant`. Hit `i` and use the arrow keys to navigate to the line that says `BIN="/usr/...`. After the `...losant.json` text, insert a space and type the following:

```
--pump
```

and insert a space between `--pump` and the `>` character.

Your `smart-plant` file should now look like this:

```
#!/bin/sh /etc/rc.common
# Copyright (C) 2016 Union Corporation
START=99

USE_PROCD=1
BIN="/usr/bin/python /root/smart-plant/smartPlant.py --oled --quiet --losant /root/smart-plant/lo

start_service() {
    procd_open_instance
    procd_set_param command $BIN
    procd_set_param respawn
    procd_close_instance
```

Hit `esc` and type `:wq` to save and close the file.

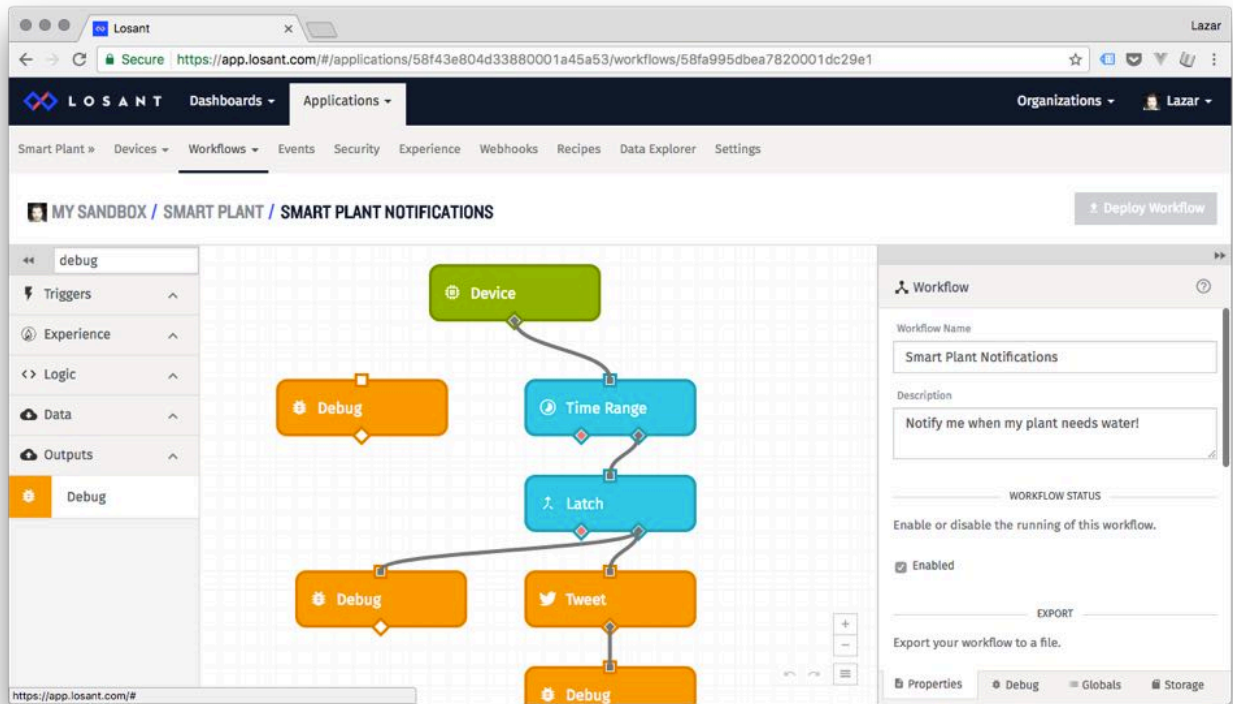
Then re-enable the program:

```
/etc/init.d/smart-plant restart
```

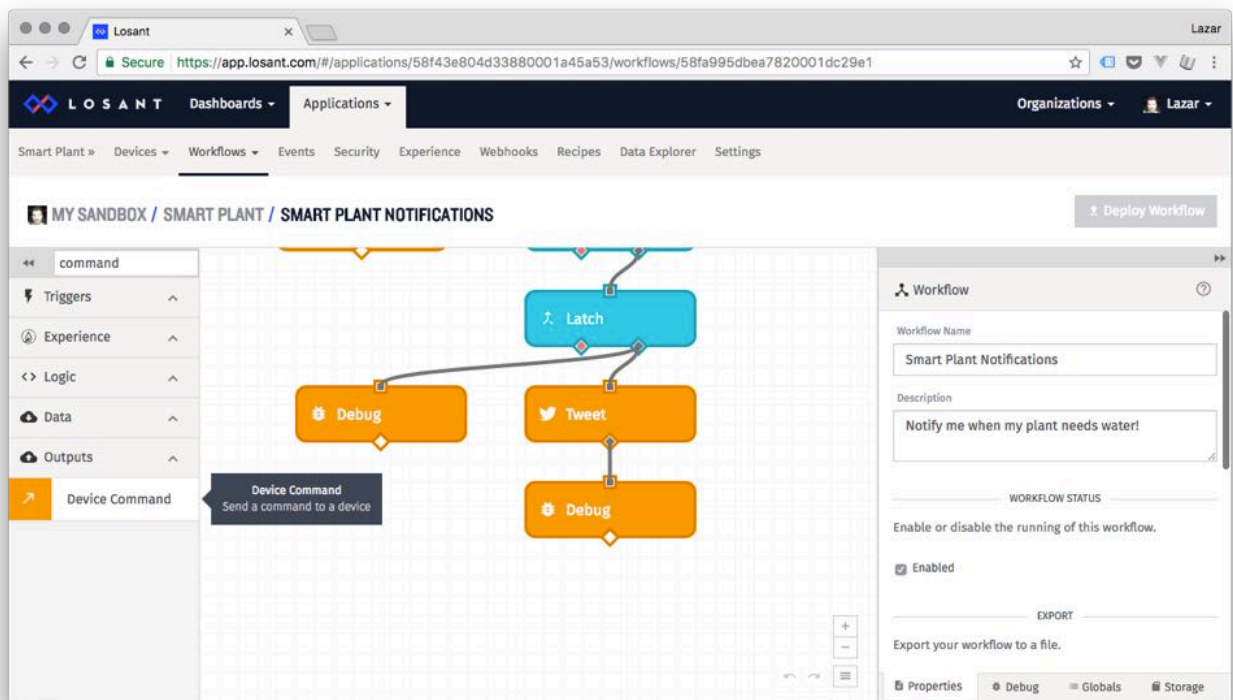
Try rebooting your Omega (enter `reboot` in the command line), and you'll see that your program will start up again when the Omega boots up.

## 12. Update the Existing Workflow

To make our smart plant truly automated, we need to add sending the `waterPlant` command to the notification workflow made in the [previous part of the project](#):



Add a Device Command Command Block to the bottom:



Like before, we need to set it up to use the device associated with your Smart Plant Omega:

The screenshot shows the Losant workflow editor interface. The workflow is titled "MY SANDBOX / SMART PLANT / SMART PLANT NOTIFICATIONS". The workflow diagram includes a "Latch" block, a "Debug" block, a "Tweet" block, and a "Device Command" block. The "Device Command" block is selected, and its configuration panel is visible on the right. The configuration panel shows the "DEVICES" section with "Select specific devices and tags" selected, and the "COMMAND" section with a text input field containing "e.g. data.myDeviceId".

Create a global variable, PUMP\_DURATION to store the duration for which the pump will be active:

The screenshot shows the Losant workflow editor interface. The workflow is titled "MY SANDBOX / SMART PLANT / SMART PLANT NOTIFICATIONS". The workflow diagram includes a "Time Range" block, a "Latch" block, a "Debug" block, a "Tweet" block, and a "Device Command" block. The "Workflow Globals" configuration panel is visible on the right, showing a table of global variables.

Key	Value	Data Type
CONSUMER_	mXuNMx	String
CONSUMER_	fdRBpRt	String
ACCESS_TOKI	2753556i	String
ACCESS_TOKI	iALYZf8b	String
PUMP_DURA	7	String

Now, go back to the Device Command block to setup the command name and payload. It will be the same as in the testing workflow used above, just this time, use `{{globals.PUMP_DURATION}}`, the global

variable we just created:

The screenshot shows the Losant workflow editor interface. The workflow is titled "MY SANDBOX / SMART PLANT / SMART PLANT NOTIFICATIONS". The workflow diagram consists of the following blocks:
 

- Time Range** (blue trigger block)
- Latch** (blue logic block)
- Debug** (orange logic block)
- Tweet** (orange logic block)
- Device Command** (orange logic block)
- Debug** (orange logic block)

 The connections are: Time Range to Latch, Latch to Debug, Latch to Tweet, and Tweet to Debug. The Device Command block is currently disconnected from the Tweet block. The right-hand panel shows the configuration for the "Device Command" block:
 

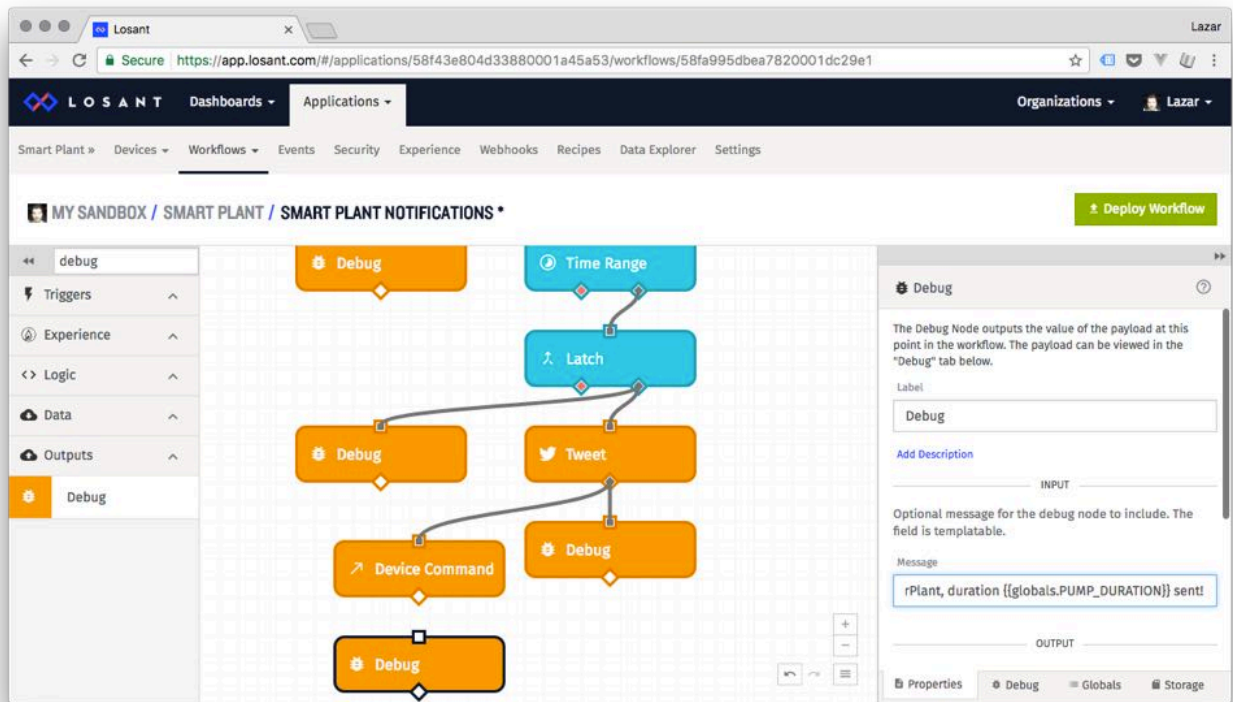
- COMMAND**: Set up the command and command payload to send. The payload, while a template, must render to a valid JSON string.
- Command Name Template**: waterPlant
- Command Payload Type**: String Template
- Command Payload String Template**: `{{globals.PUMP_DURATION}}`

Connect the Device Command block to the output of the Tweet block so that the plant gets watered right after the Tweet is sent out:

The screenshot shows the Losant workflow editor interface with the same workflow as the previous screenshot. The workflow diagram is identical, but the connection between the **Tweet** block and the **Device Command** block has been established. The right-hand panel now shows the configuration for the "Connector" block:
 

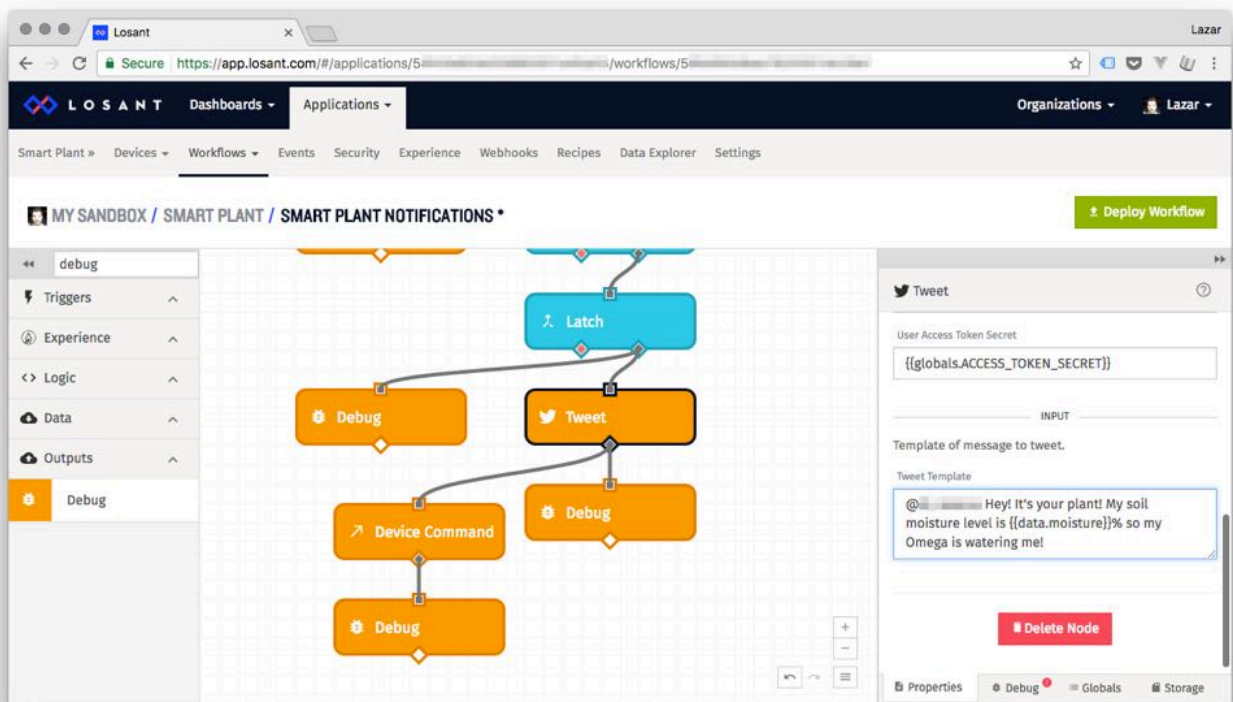
- Connector**: A red button labeled "Delete Connector" is visible.

Let's also add a Debug block that will output that the command indeed got sent out:



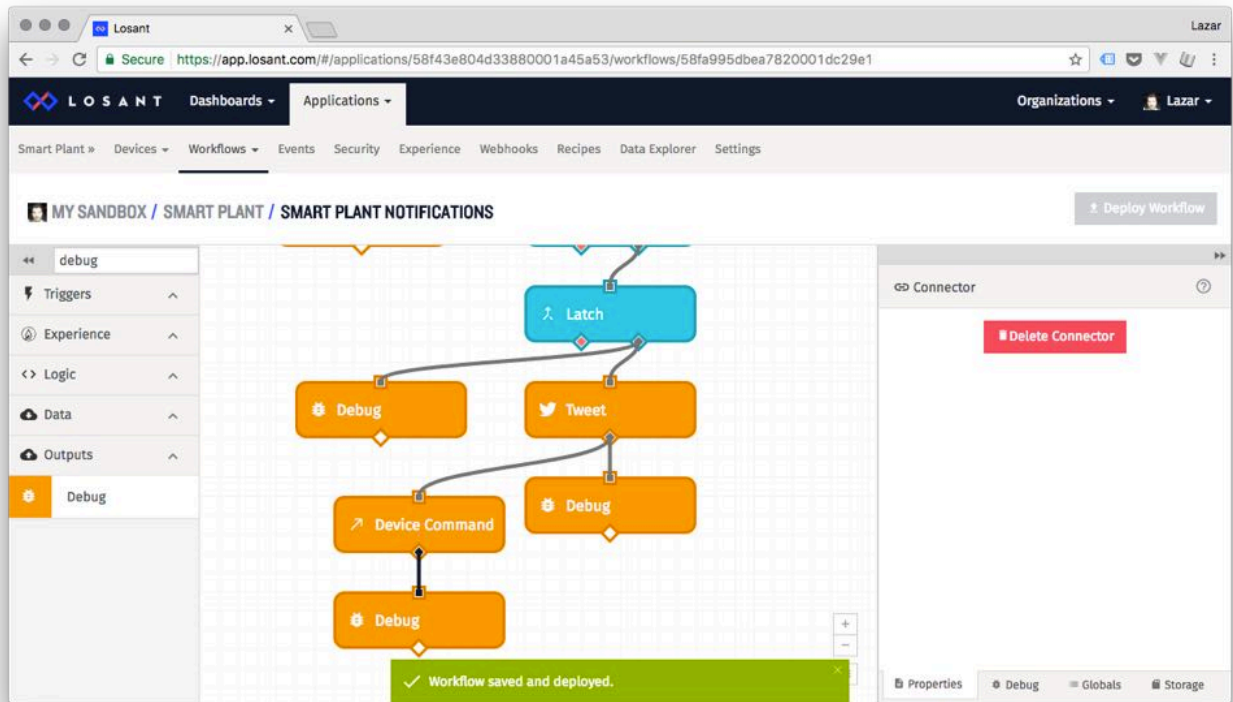
The screenshot shows the Losant workflow editor interface. The workflow is titled "SMART PLANT NOTIFICATIONS" and is located in the "MY SANDBOX / SMART PLANT" environment. The workflow consists of several blocks: a "Time Range" trigger, a "Latch" block, a "Debug" block, a "Tweet" block, a "Device Command" block, and another "Debug" block. The "Device Command" block is connected to the "Debug" block. The right-hand panel shows the configuration for the selected "Debug" block, with the "Message" field containing the text: "rPlant, duration {{globals.PUMP\_DURATION}} sent!".

It might also be a good idea to change the contents of the Tweet:



The screenshot shows the Losant workflow editor interface. The workflow is titled "SMART PLANT NOTIFICATIONS" and is located in the "MY SANDBOX / SMART PLANT" environment. The workflow consists of several blocks: a "Time Range" trigger, a "Latch" block, a "Debug" block, a "Tweet" block, a "Device Command" block, and another "Debug" block. The "Device Command" block is connected to the "Tweet" block. The right-hand panel shows the configuration for the selected "Tweet" block, with the "Template of message to tweet" field containing the text: "@Hey! It's your plant! My soil moisture level is {{data.moisture}}% so my Omega is watering me!".

Connect the Debug block to the Device Command block and hit Deploy Workflow:



### 13. Sit Back and Relax

Now whenever your plant's soil moisture level falls below the level in the `LOW_MOISTURE` global variable, your plant will water itself and alert you with a Tweet!



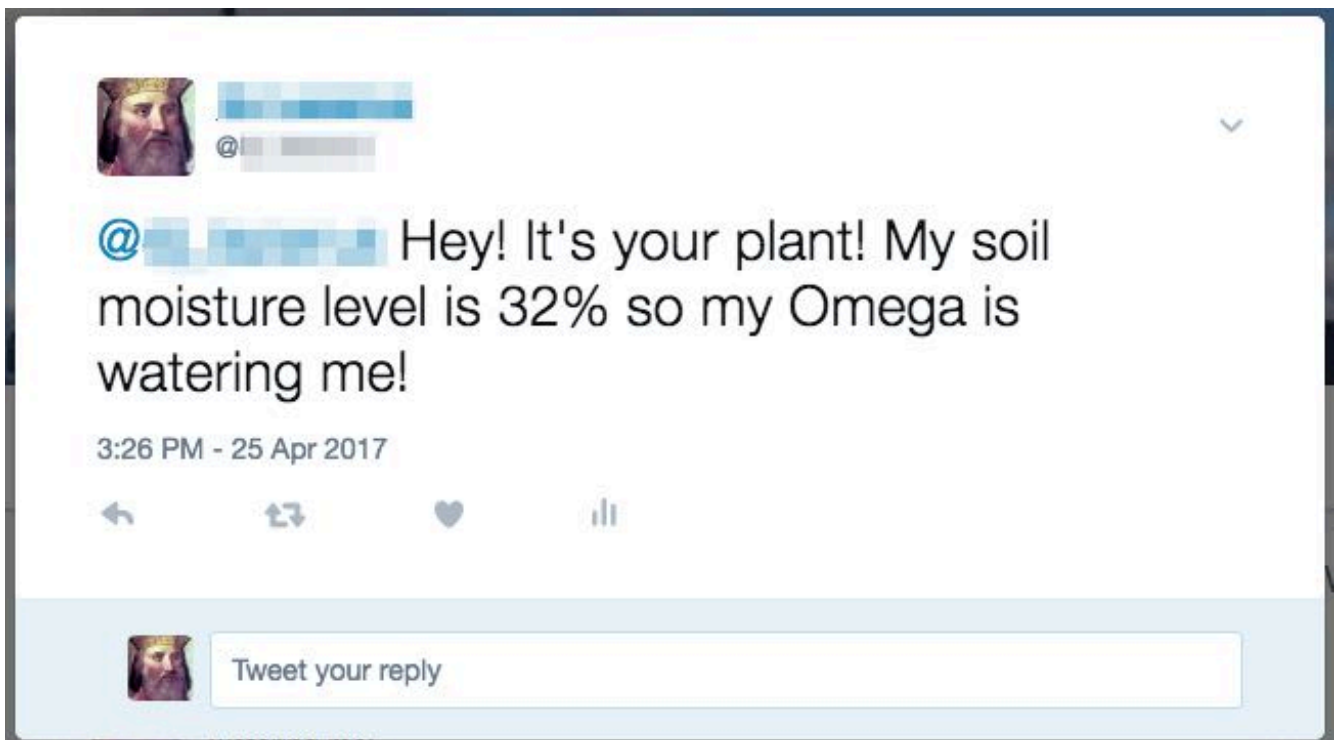
The screenshot shows the Losant workflow editor interface. The workflow is titled "SMART PLANT NOTIFICATIONS" and is located in the "MY SANDBOX / SMART PLANT" environment. The workflow consists of several nodes: a "Debug" node, a "Tweet" node, a "Device Command" node, and another "Debug" node. The "Debug" panel on the right displays the following output:

```

Debug
waterPlant, duration 7 sent!
Tue Apr 25 2017 17:11:56 GMT-04:00

{
  "time": "Tue Apr 25 2017 17:11:53 GMT-0400",
  "data": {
    "moisture": 33,
    "applicationId": "58f43e804d33880001a45a53",
    "triggerId": "58fd713e8ea7820001dc2a1e",
    "triggerType": "deviceId",
    "relayId": "58fd713e8ea7820001dc2a1e",
    "relayType": "device",
    "deviceTags": {
      "deviceName": "omega-fla7",
      "flowId": "58fa995d8ea7820001dc29e1",
      "flowName": "Smart Plant Notifications",
      "applicationName": "Smart Plant"
    }
  }
}

```



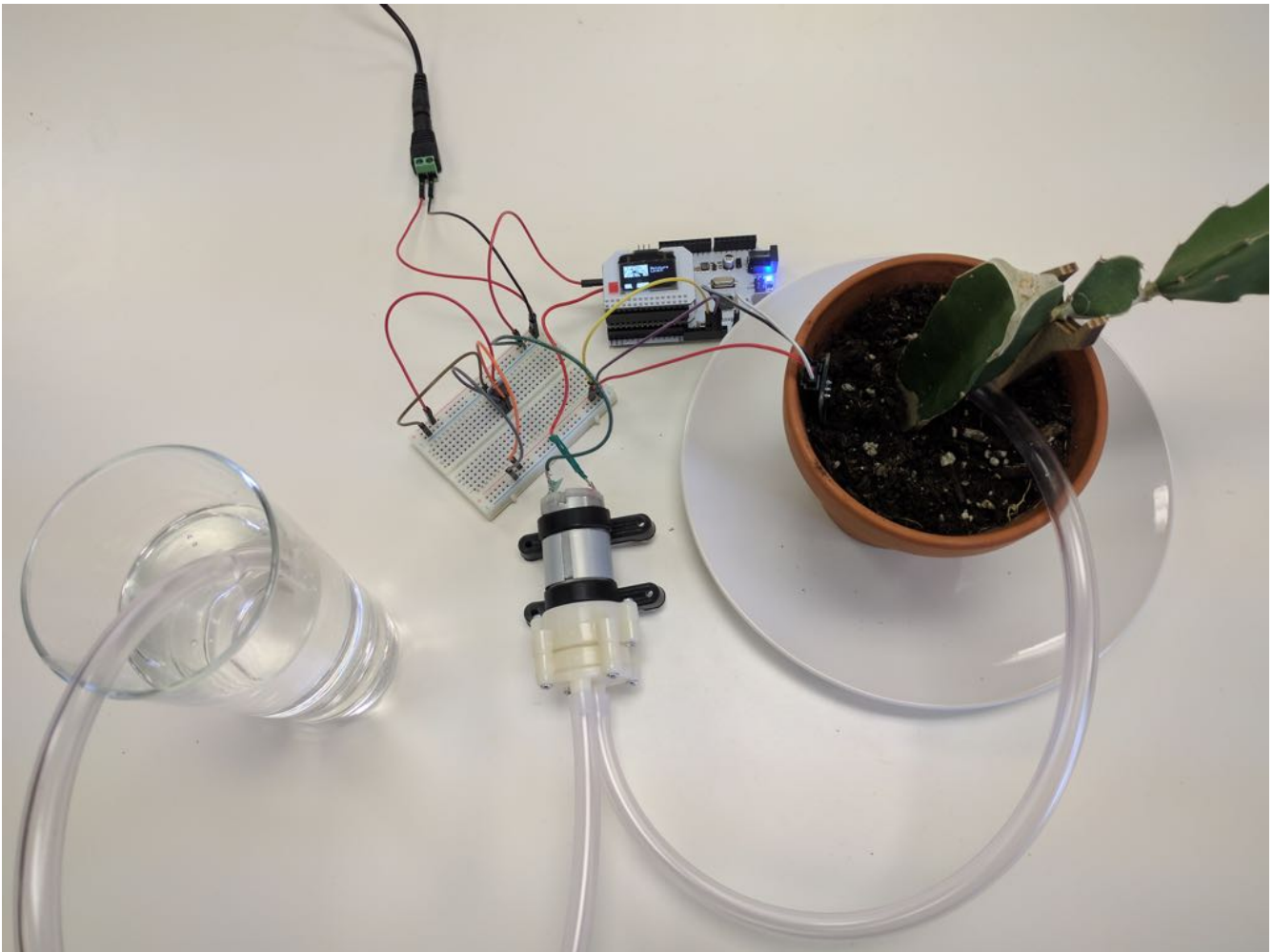
You'll have to experiment with the `LOW_MOISTURE`, `OK_MOISTURE`, and `PUMP_DURATION` variables to see what is best for your plant.

Wow, you've made your plant so smart that all it needs is for you to refill the water reservoir!

Now if only there were some way to make this project truly plug 'n' play...

## Smart Plant - A Single Power Supply

We've built out some really cool features on our smart plant in the previous parts. Now to top it all off, let's upgrade the power circuitry so we can run the entire project on a single 12V power supply!



### Overview

**Skill Level:** Intermediate

**Time Required:** 20 minutes

When you were building the smart plant, you had to power the pump and Arduino separately. Let's change that by replacing the USB power supply with a regulator that can convert the motor's 12V into 5V. This way, when you plug in the 12V supply, the entire project comes to life!

### Ingredients

We'll need all of the same materials as in the previous part:

- Onion [Omega2](#) or [Omega2+](#)
- Onion [Arduino Dock 2](#)

- Onion [OLED Expansion](#) (optional but recommended)
- [Soil Moisture Sensor](#)
- 3x [Male-to-Female Jumper Wires](#)
- Onion [Relay Expansion](#)
- [DC Barrel Jack Adapter](#)
- [12V 1A DC Power Supply](#)
- 3x [Male-to-Male Jumper Wires](#)
- [Water Pump \(12V DC\)](#)
- [Flexible Plastic Tubing](#)
- A piece of paper the size of your hand to test the pump's polarity
- A plate or bowl to hold your plant and collect excess water
- A glass or bowl of water you can use as a reservoir

We'll need some new ingredients:

- [MC33269T 5V Linear Voltage Regulator](#)
- A [Breadboard](#)
- 10x [Male-to-Male Jumper Wires](#)

Tools:

- [Flat-head screwdriver](#)
- [Philips-head screwdriver](#)

## Step-by-Step

Follow these instructions to set this project up on your very own Omega!

### 1. Prepare

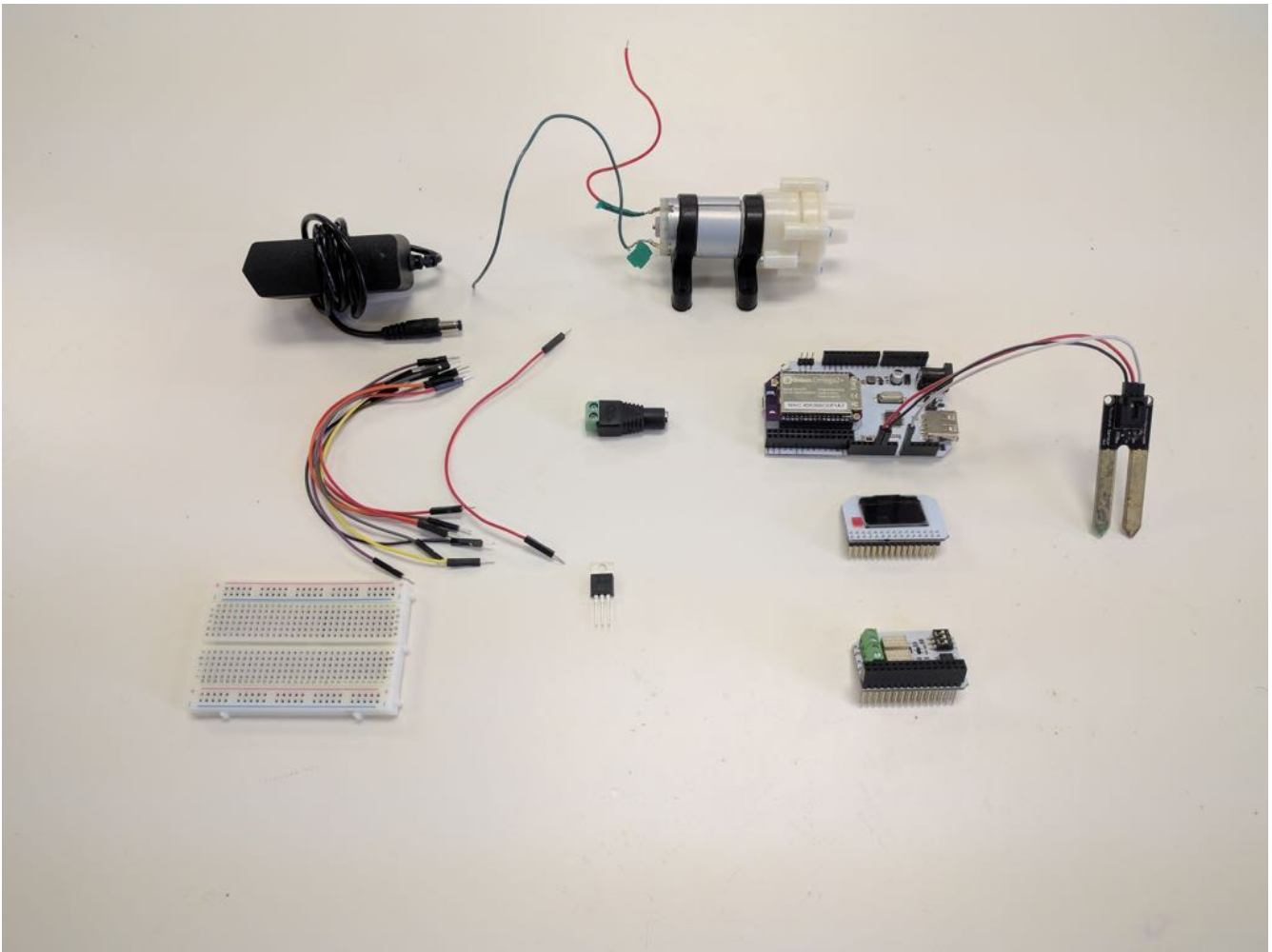
You'll have to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 2. Complete the Previous Parts of the Project

This project builds on the previous parts of the Smart Plant project. If you haven't already completed the [first](#), [second](#), [third](#), and [fourth parts](#), go back and do them now!

### 3. Prep

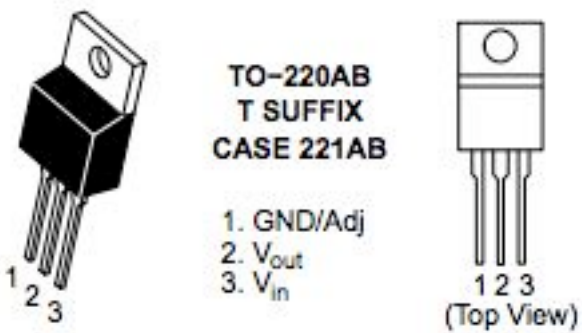
Before you start, take apart the wiring and tubing we did in the [previous part of the project](#) and unplug your Arduino Dock. Now you should have:



**IMPORTANT:** Make sure your Power Supply is no longer connected to the DC Barrel Jack Adapter!

#### 4. Wiring the Circuit

This is the pinout diagram for the MC33269T. We'll be referring back to this when wiring it up.



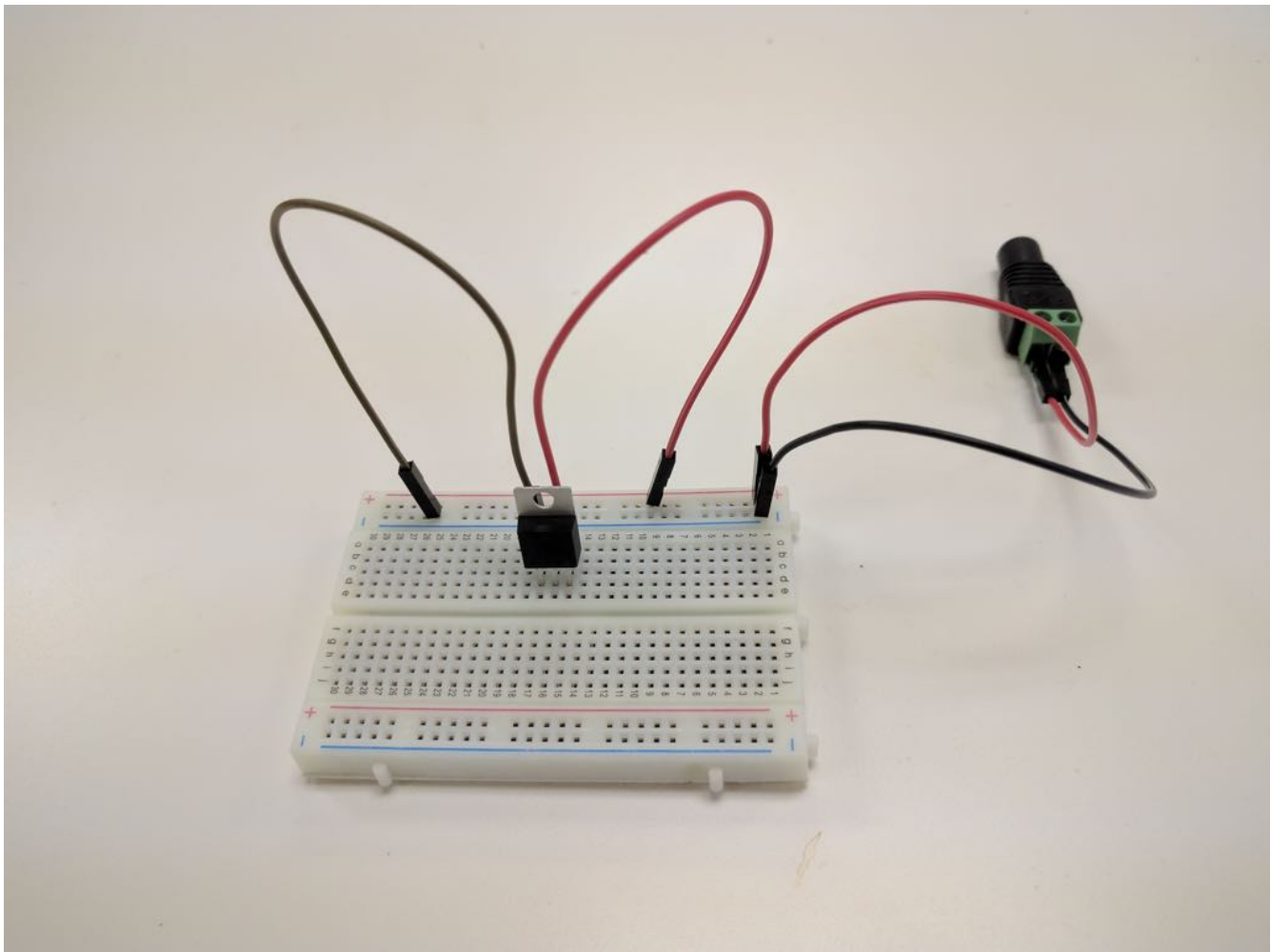
We'll assemble the circuit in a few sub-steps:

1. Regulator input
2. Regulator output
3. Water pump

4. Arduino Dock
5. Moisture sensor

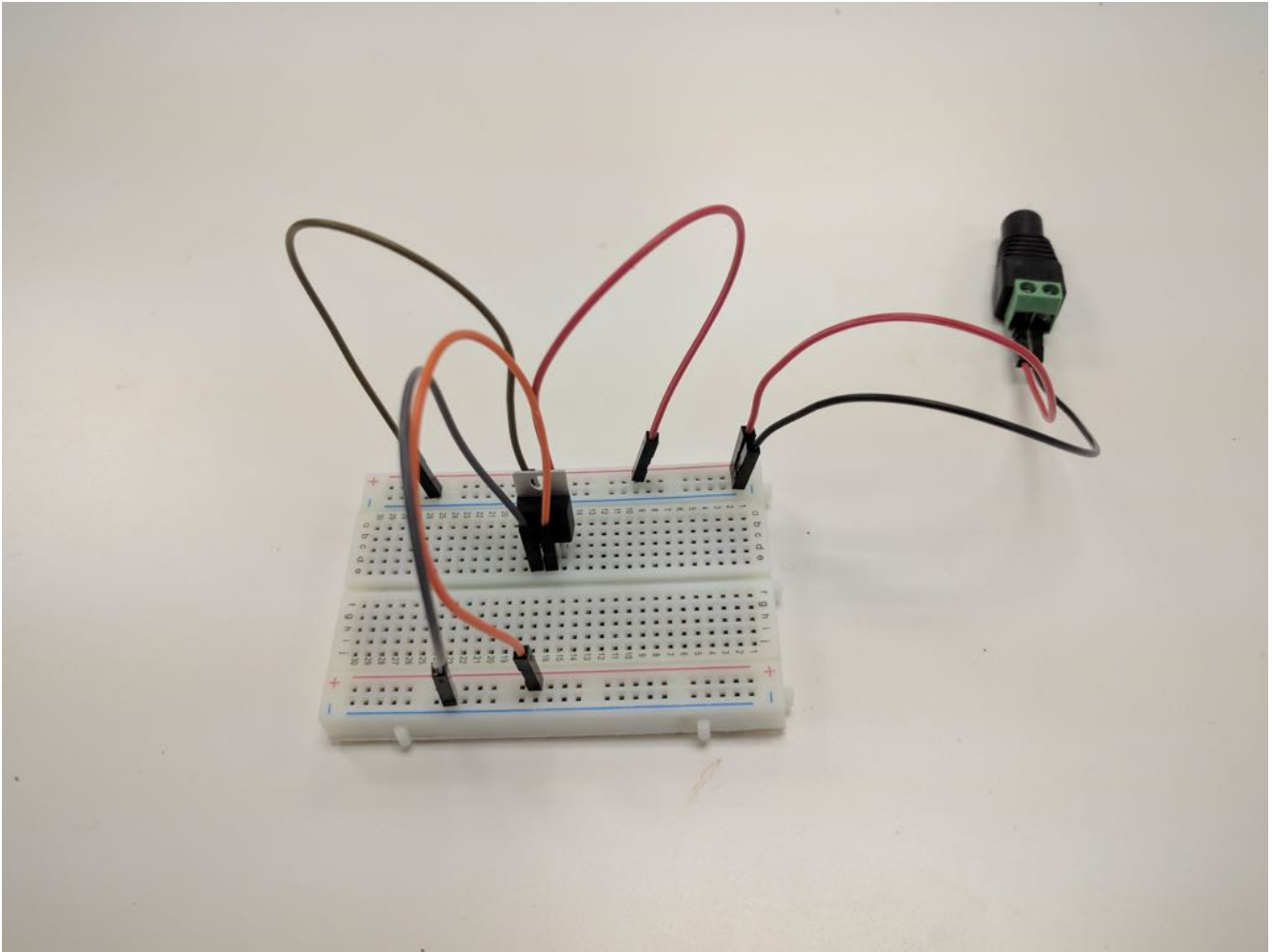
### Regulator Input

1. Connect jumper wires to both DC Barrel Jack Adapter terminals.
2. Connect the DC Barrel Jack to one pair of the + and - rails on the breadboard.
  - We'll call this the **12V rail**.
3. Plug the MC33269T regulator into the Breadboard across three empty rows.
4. Connect the 12V - rail to the GND pin of the regulator with a jumper wire.
  - The left most pin when looking from the front.
5. Connect the 12V + rail to the Vin pin of the regulator with a jumper wire.
  - The right most pin when looking form the front.



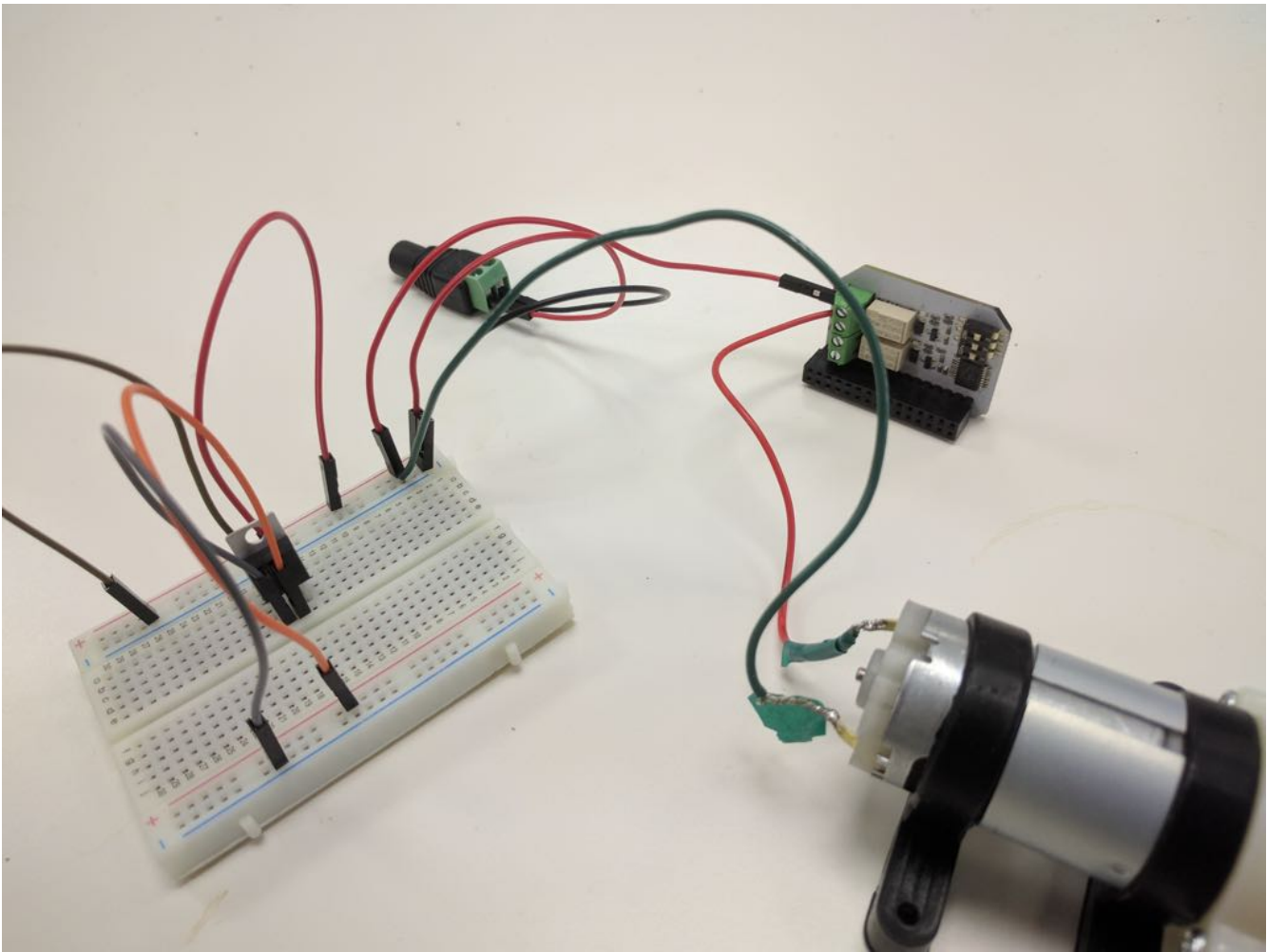
### Regulator Output

1. We'll use the other pair of + and - rails on the breadboard for our **5V rail**
2. Connect the GND pin of the regulator (left most pin when looking from the front) to the 5V - rail with a jumper wire
3. Connect the Vout pin of the regulator (middle pin) to the 5V + rail with a jumper wire. Now this rail can be used to power the Arduino Dock and Omega



## Water Pump

1. Run a jumper wire from the 12V -rail to the **negative terminal** of the Water Pump
2. Connect a jumper wire from the 12V +rail to the **IN** screw terminal on Channel 0 of the Relay Expansion
3. Connect a jumper wire from the **OUT** screw terminal on Channel 0 of the Relay Expansion to the **positive terminal** of the Water Pump

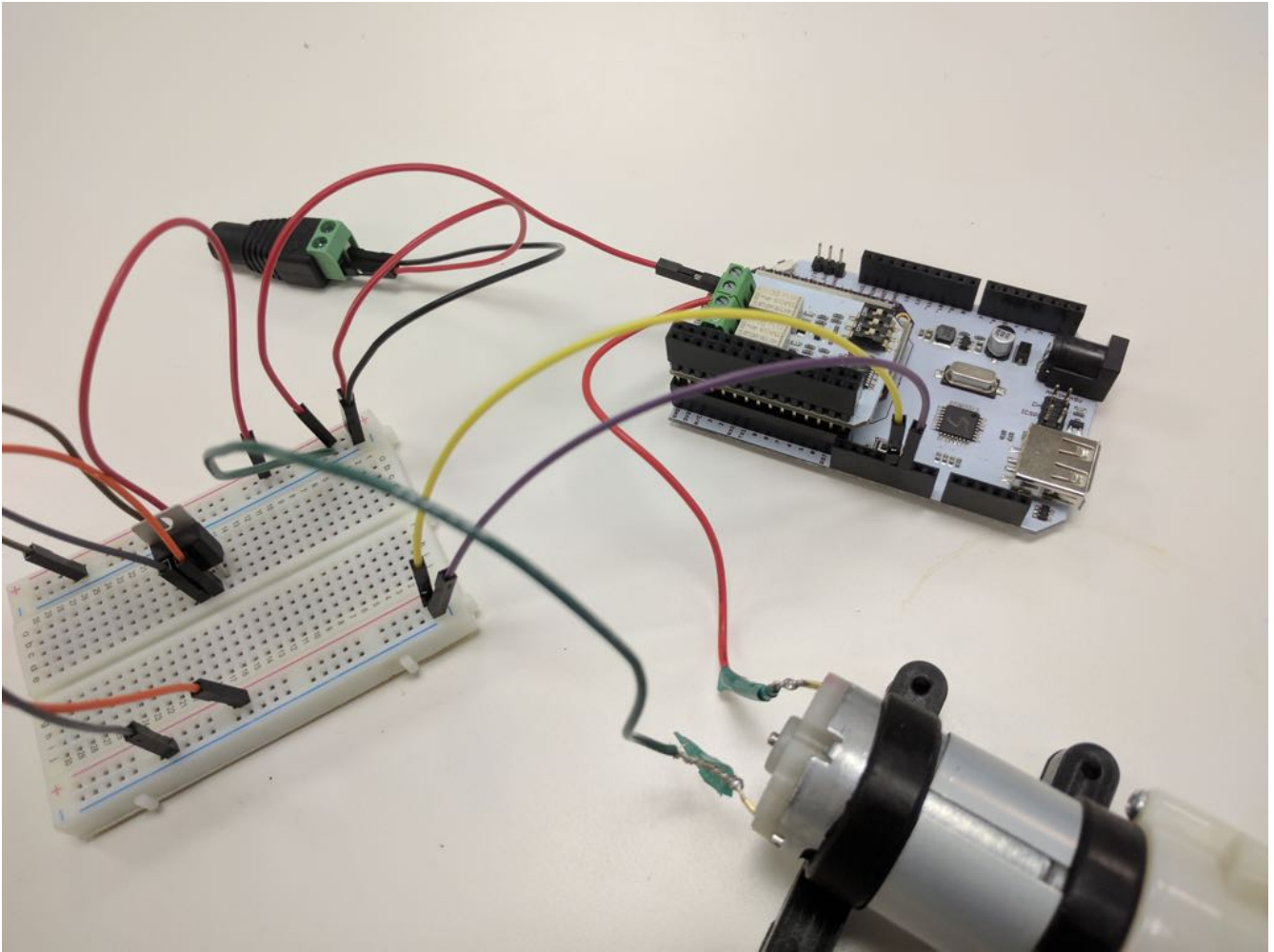


Once you've done that, plug the Relay Expansion back into the Arduino Dock.

### Arduino Dock

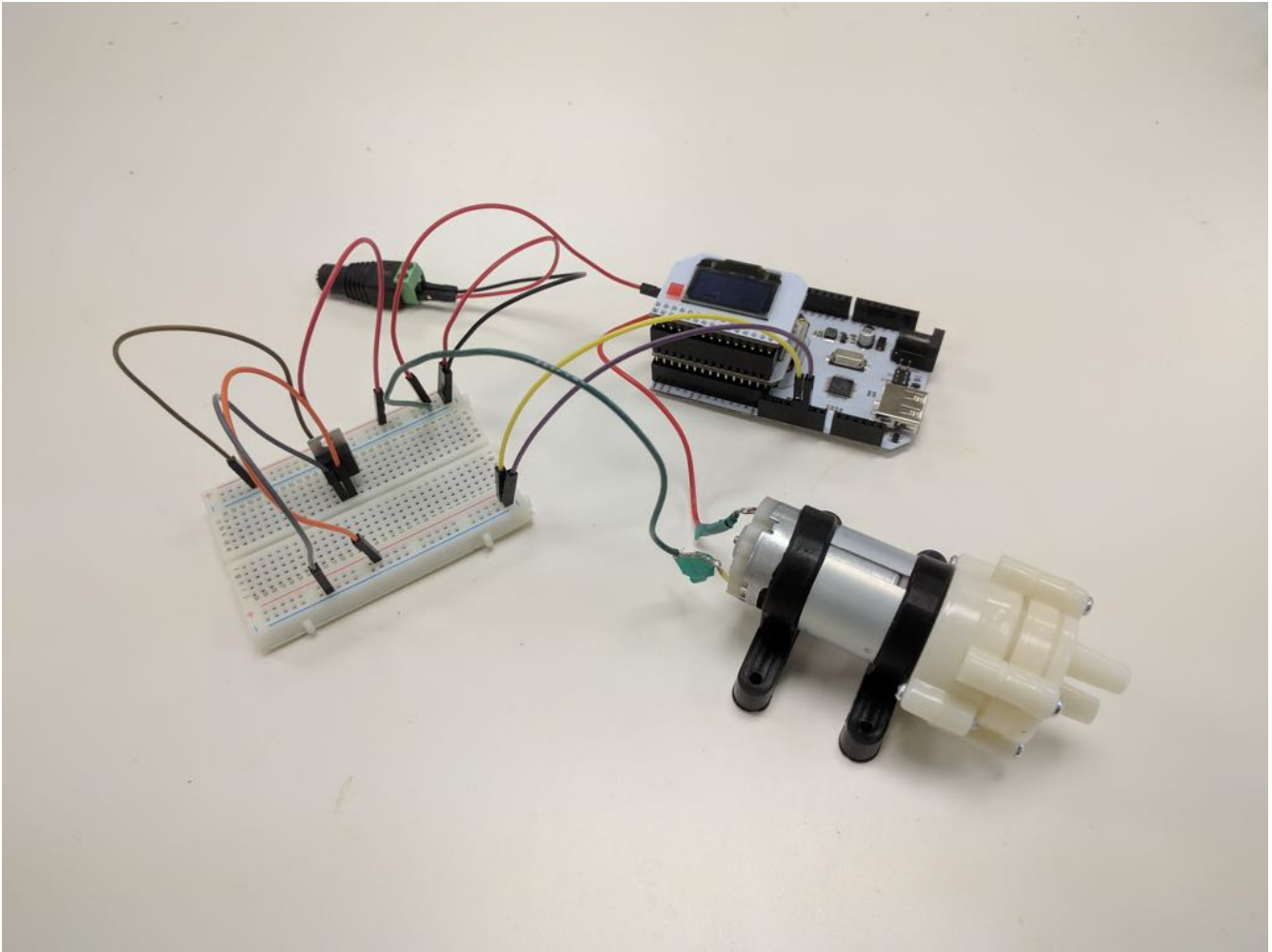
**WARNING:** It is very important that you connect the 5V rail to your Arduino Dock. Accidentally using the 12V rail will for sure damage your Arduino Dock and Omega. Proceed at your own risk, but don't worry, if you follow the instructions, you'll be fine.

1. Connect the **5V - rail** to one of the Arduino Dock's GND pins
2. Connect the **5V + rail** to the Arduino Dock 5V pin



Then plug in the OLED Expansion.

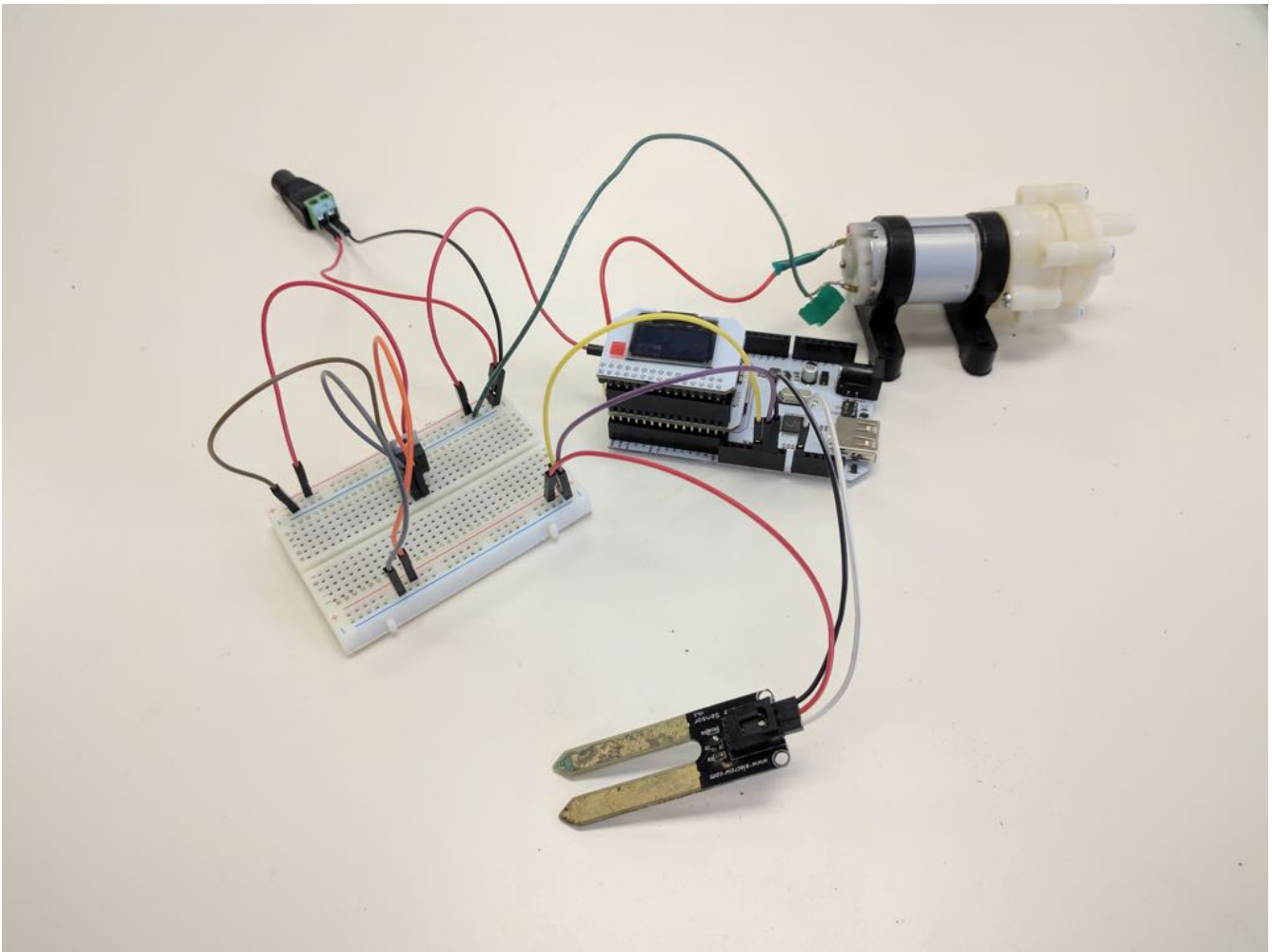




## Moisture Sensor

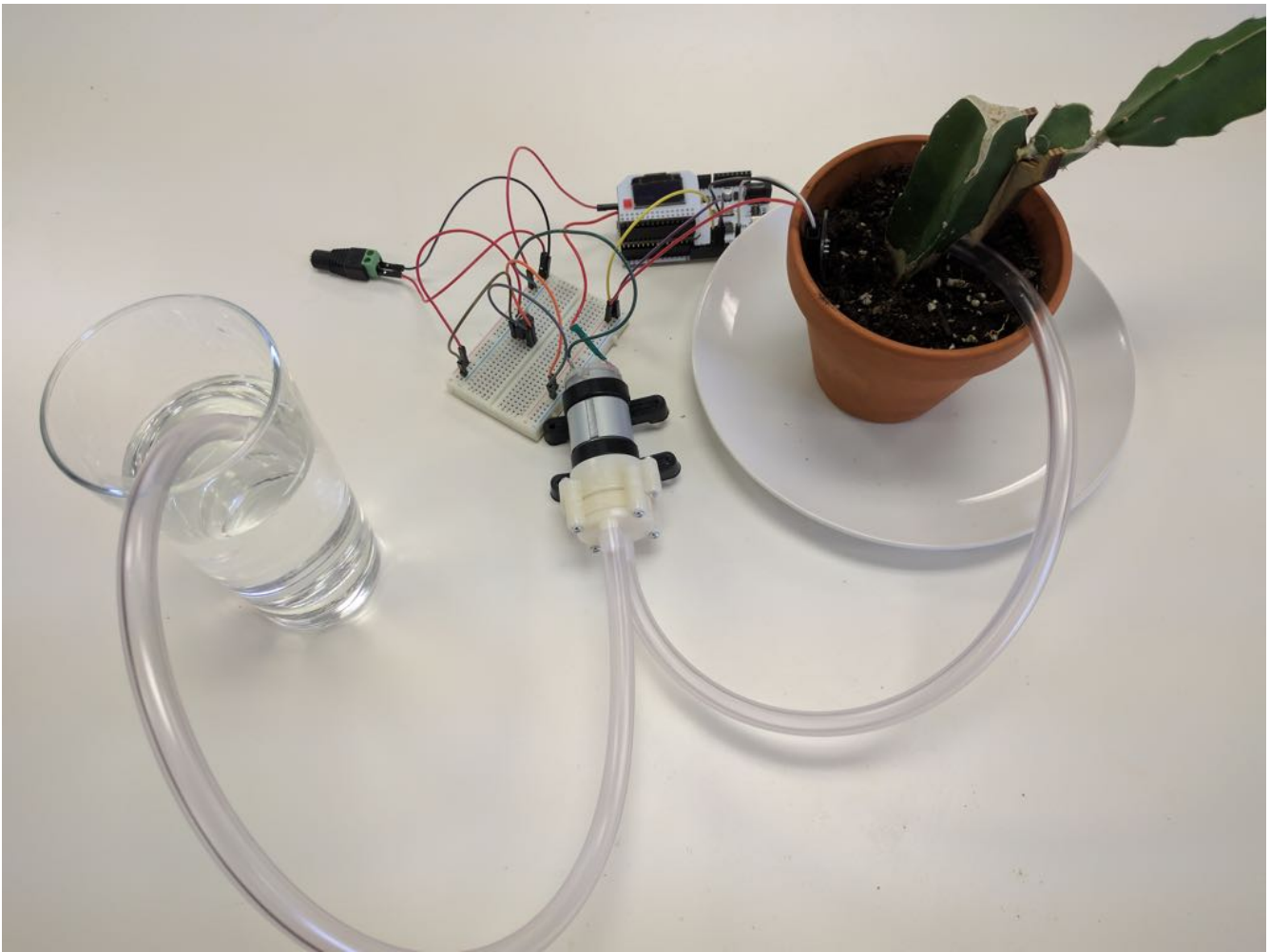
Using the wires from the moisture sensor:

1. Connect the **5V + rail** to the sensor's **Vcc** pin.
2. Connect the Arduino Dock's other **GND** pin to the sensor's **GND** pin.
3. Connect the Arduino Dock's **A0** pin to the sensor's **SIG** pin.



## 5. Provide Power

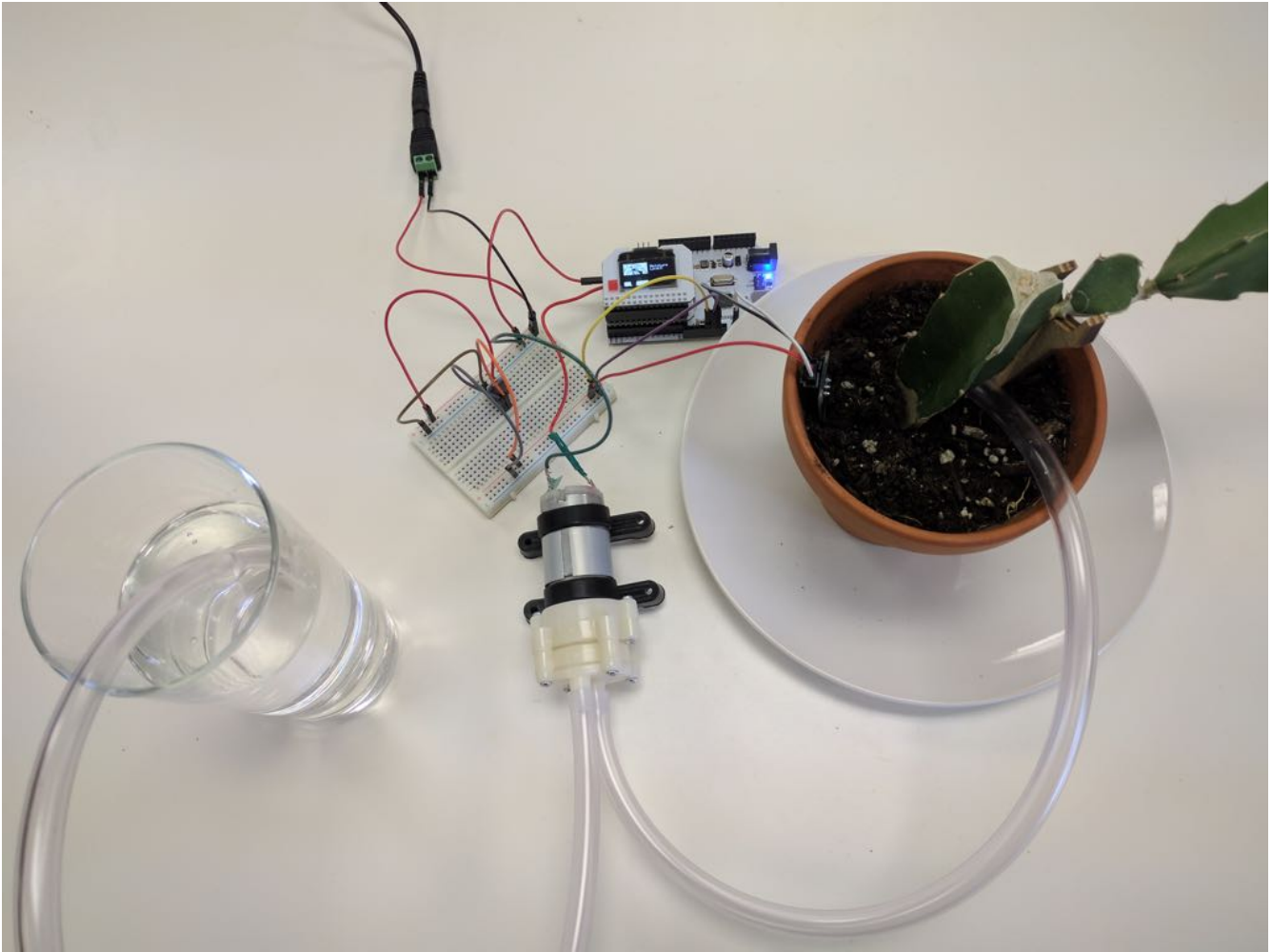
Now reassemble your pump tubing, reservoir, sensor, and plant:



Then provide power by connecting the 12V power supply to the DC Barrel Jack Adapter. Your Omega should now be booting.

**Congratulations, You Made It!**

Revel in the fact that you've created a regulator circuit that can power your Omega as well as the 12V pump!

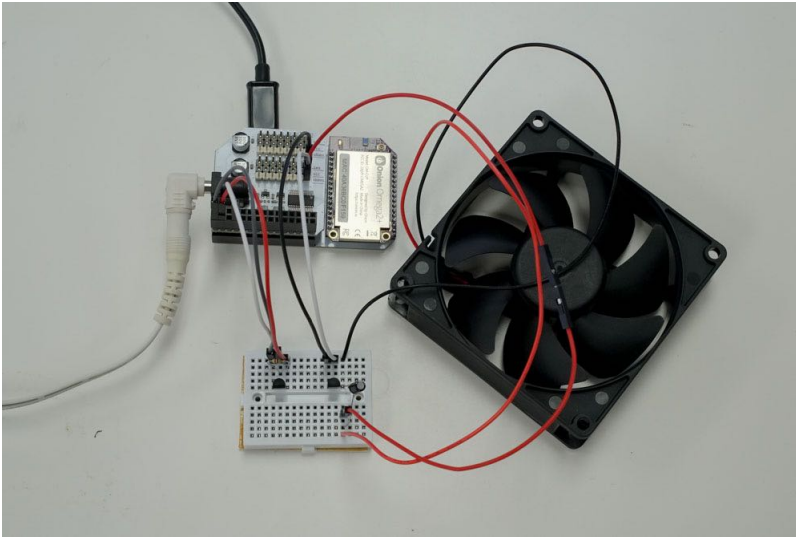


Show off your amazing smart-plant setup and Dashboards on our [Community Forum](#), and let us know how you liked these tutorials!

## Temperature-Based Smart Fan

Mornings too cold, but gets too hot by noon? By hooking up a temperature sensor to the Omega, we can use the data it provides to modulate the speed of a fan - cooling us down only when we need it!

This kind of setup is used in many places: the cooling fans in your laptop or desktop computer operate in the same way. Other applications include home-brewing beer or wine kegs and anywhere temperature control is required.



## Overview

**Skill Level:** Intermediate ~ Advanced

**Time Required:** 40 Minutes

There's a lot of implementation details in this project that will change depend on the exact hardware you have access to. We used a D18B20 1Wire temperature sensor, for example. For the fan, we recommend a computer case fan, since those are quite easy to come by and works decently well.

We also cooked up a DC motor with a 3D printed rotor setup just like in the [Omega2 Maker Kit](#) since we had all of those handy.

To control the fan, we'll be using a python script and the [Onion PWM Expansion Python Module](#) to control the fan speed. We also use a library to operate the 1Wire sensor, but your methods may vary depending on your exact sensor.

All the code we used is written for a case fan with a transistor switching it. It can be found in Onion's [iot-smart-fan repository](#) on GitHub.

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
- Onion [Servo \(PWM\) Expansion](#)
- [Breadboard](#) (optional, but it helps a lot)
- [Computer case fan](#)
  - Note that we used a fan that was roughly double the size!
- [D18B20 1-Wire Temperature Sensor](#)
  - The Omega accepts I2C, 1Wire, and SPI, among other protocols, so other digital sensors will work as well.
- [12V DC power supply](#)
  - Must be capable of supplying at least 0.5A
- [Resistors](#)
  - 1x 5.1k $\Omega$

- 1x 1k $\Omega$
- 1x 47 F Capacitor
- NPN Transistor rated for 12V at 0.5A
- Jumpers
  - 2x M-F
  - 4x M-M

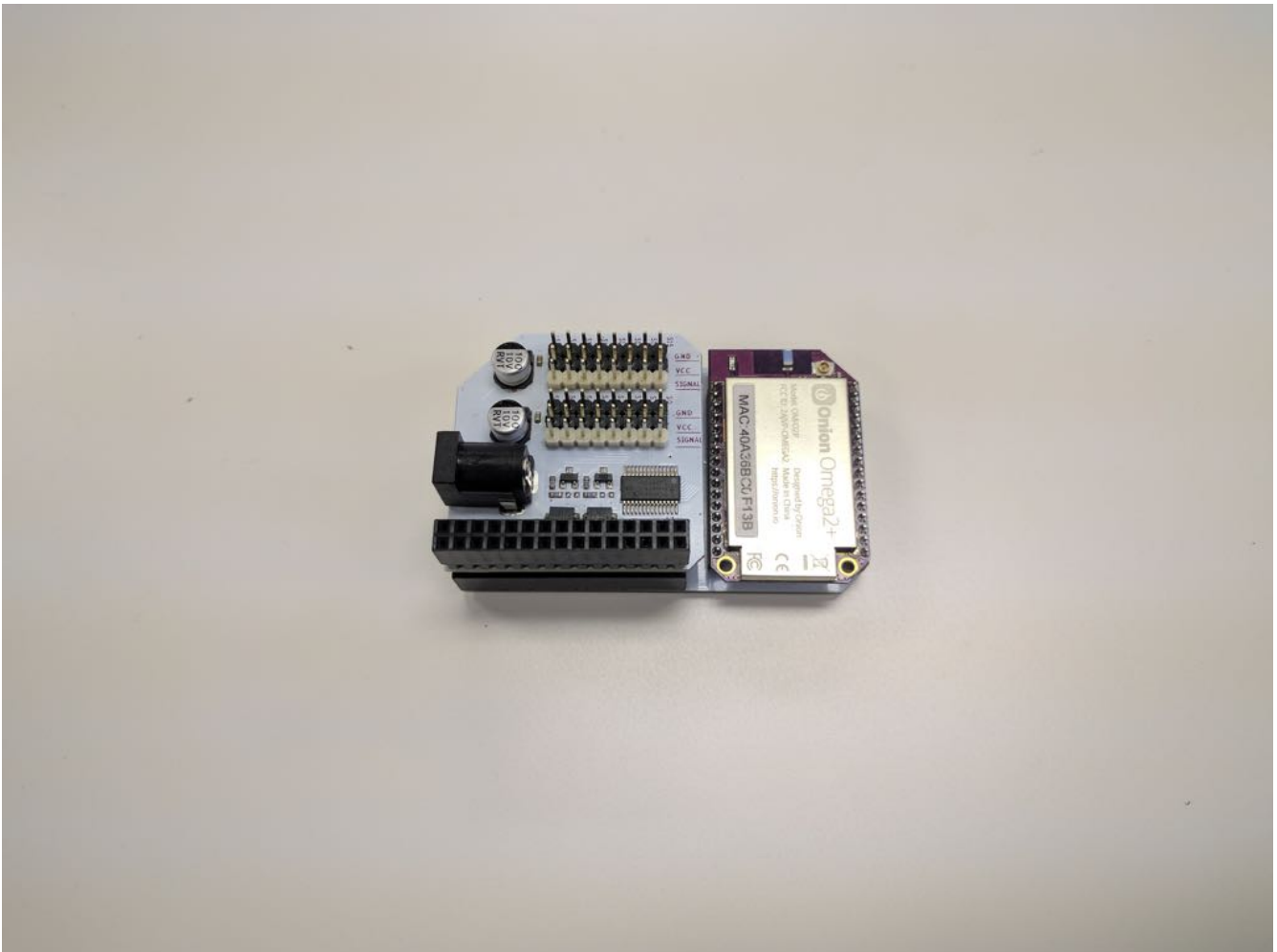
## Step-by-Step

Follow these instructions to set this project up on your very own Omega!

### 1. Prepare

First let's get the Omega ready to go. if you haven't already, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

Plug in the PWM Expansion to the Dock and grab all the components:



## 2. Install the Required Software

We need Python and the [Onion PWM Expansion Python Module](#) to make this work:

```
opkg update
opkg install python-light pyPwmExp
```

Everything else will be included in the GitHub repo.

## 3. Connect the Fan

Computer case fans are voltage driven, but we can cheat by using PWM with a transistor to switch the supply voltage.

If you have jumpers handy, we recommend using them as a bridge between the header of the fan and the PWM expansion.

First, we'll have to set up the transistor. For our lab setup, we used an S9014 NPN transistor with a 2-wire PC case fan. If you use a different model, make sure to note which pin is the base/collector/emitter.

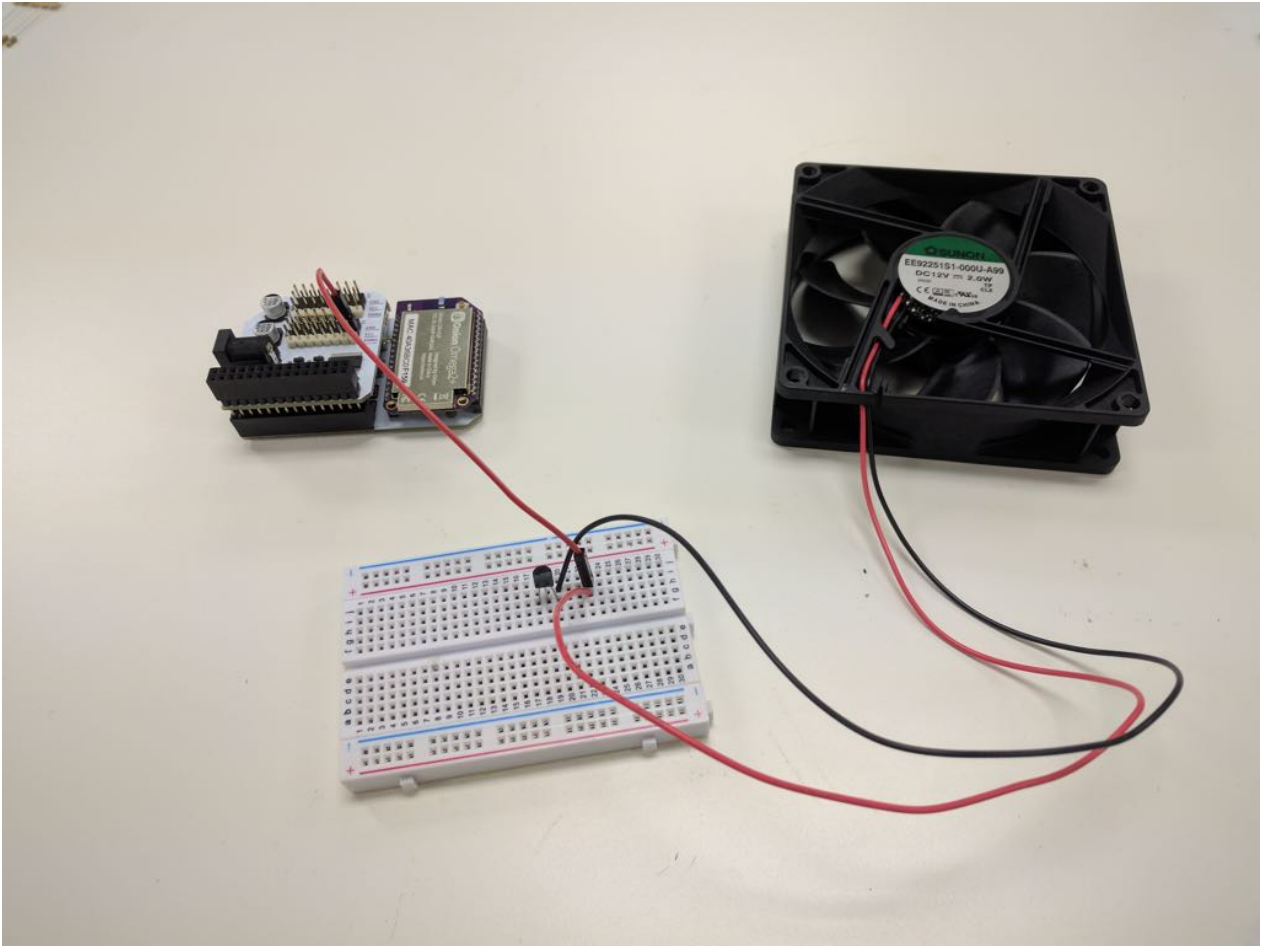
If you use a PNP transistor, your fan will automatically turn on unless you set the PWM output to 100%. This is because PNP transistors turn 'on' when the base draws current, when the PWM channel is at 0% duty, it draws a tiny bit of current - enough to turn on the transistor!

Most commonly, case fans have three pins/wires - one of which is a tachometer output. If you're using one of these, make sure there's no power being supplied to the output pin, this will cause damage to the fan.

The output pin sends the current speed of the fan, it can be used in your code to check if the fan is working as a bonus!

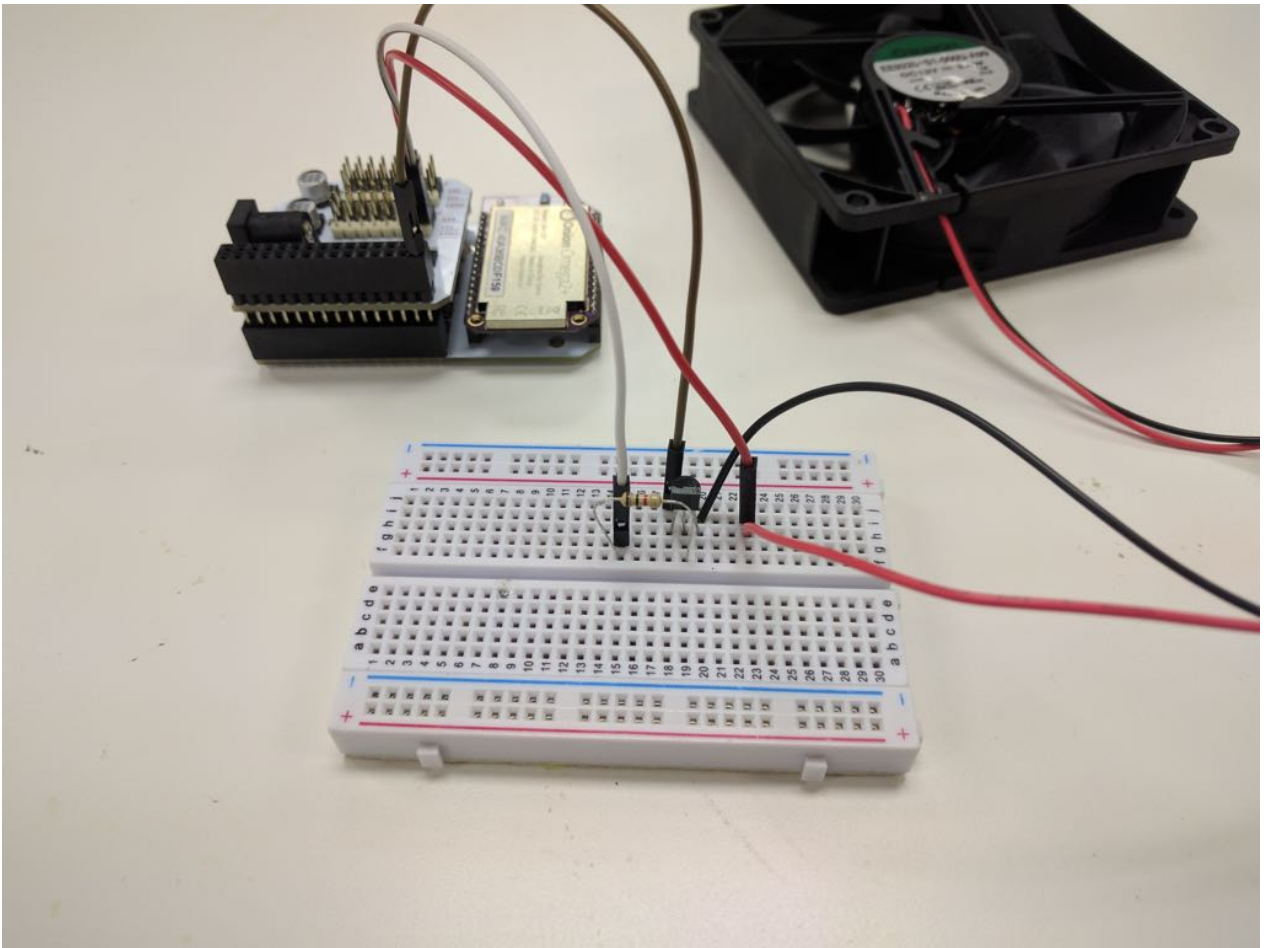
We connected the power supply to the PWM expansion for cleaner wiring.

1. Connect the transistor to the breadboard across 3 empty rows.
2. Connect the (-) (usually black) wire of the fan to the transistor's collector pin (right pin when looking at the flat front).
3. Connect the (+) (usually red) wire of the fan to an empty row a few spaces away.
4. Connect the Vcc pin on the PWM Expansion's S0 channel to the (+) pin of the fan using a M-F jumper wire.



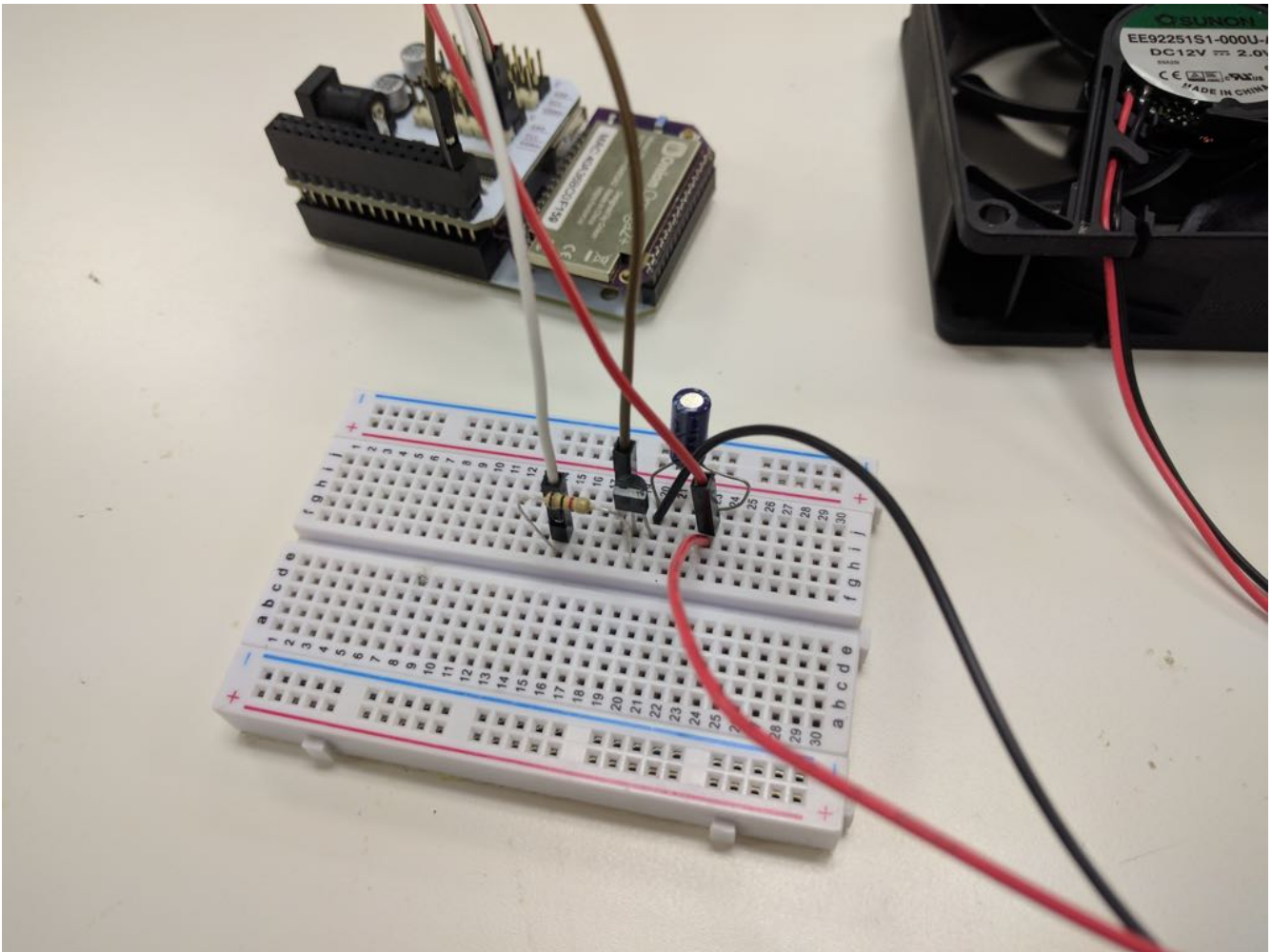
5. Connect one end of the  $1k\Omega$  resistor to the transistor's base pin (middle).
6. Connect the other end of the resistor to the SIGNAL pin on the PWM Expansion's S0 channel using a M-F jumper.
7. Connect the transistor's emitter pin (left pin when looking at the flat front) to one of the Expansion Dock's GND pins using a M-M jumper.





8. Connect the capacitor across the fan's (+) and (-) wires where they are connected to the breadboard.

- If you have a polarized capacitor with the (-) or (+) side clearly marked, make sure to match the terminals with the fan's ((-) to (-), (+) to (+))!



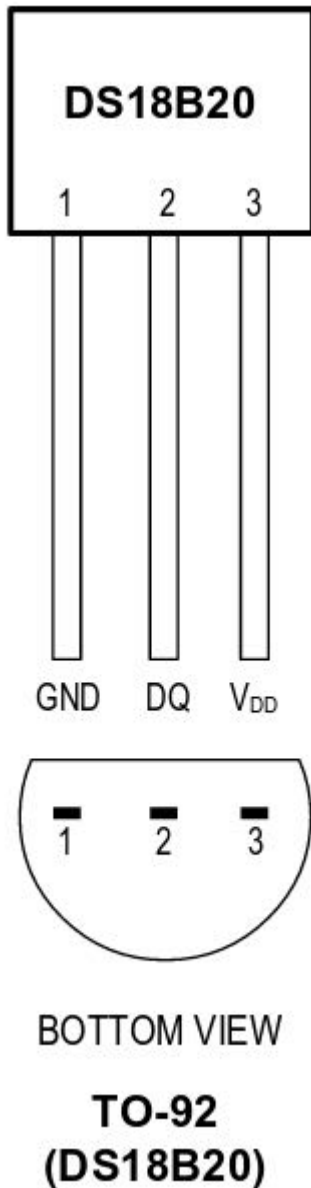
This circuit will now switch the Fan's voltage based on the PWM signal from channel 0!

The capacitor acts as a simple low-pass filter to supply the fan with a smooth analog voltage.

#### 4. Wire up the Temperature Sensor

This part is written assuming you're working with the D18B20, if your sensor is different, you may have to find a guide elsewhere on wiring it properly.

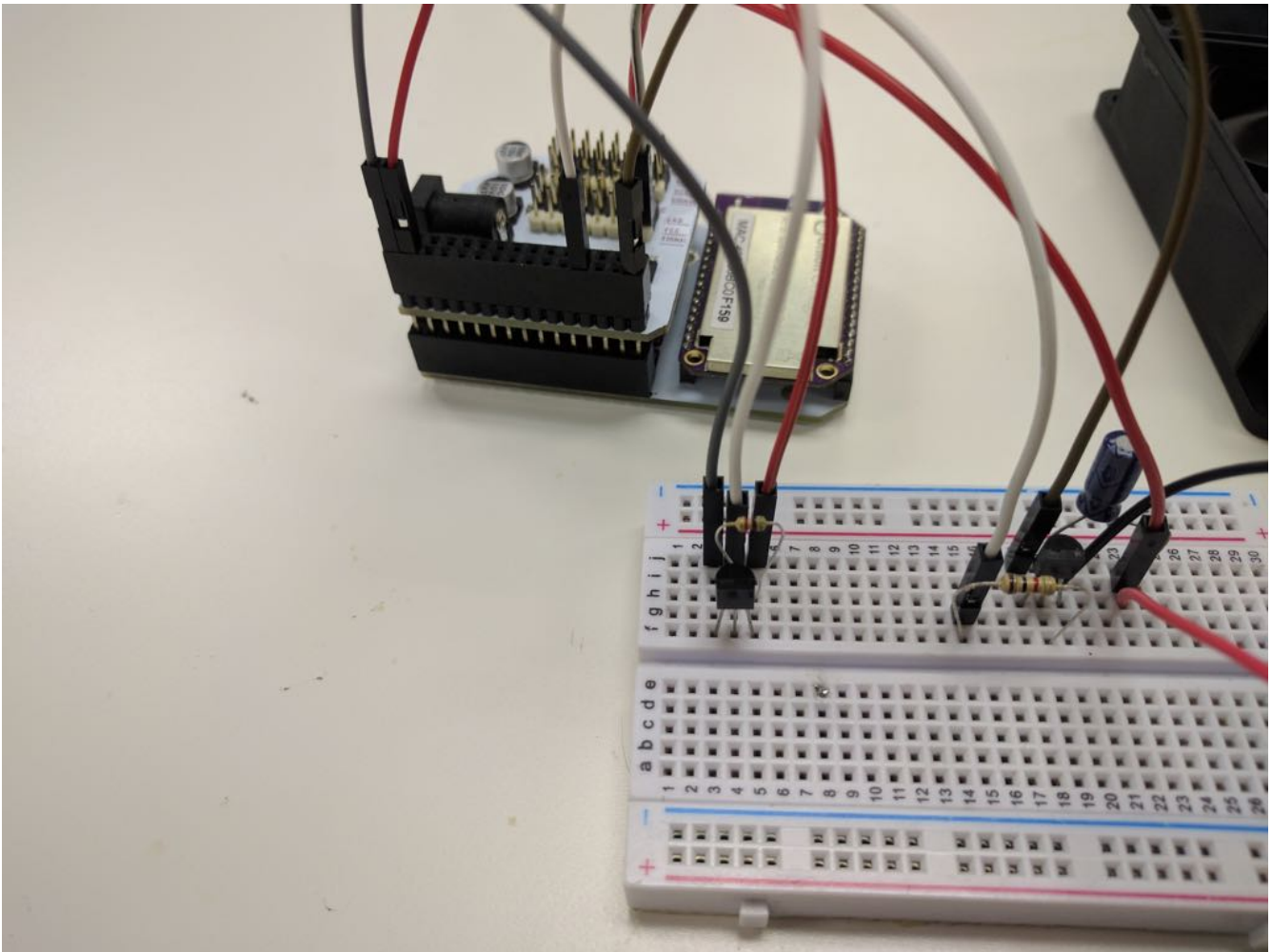
The D18B20 has a pinout that looks like this:



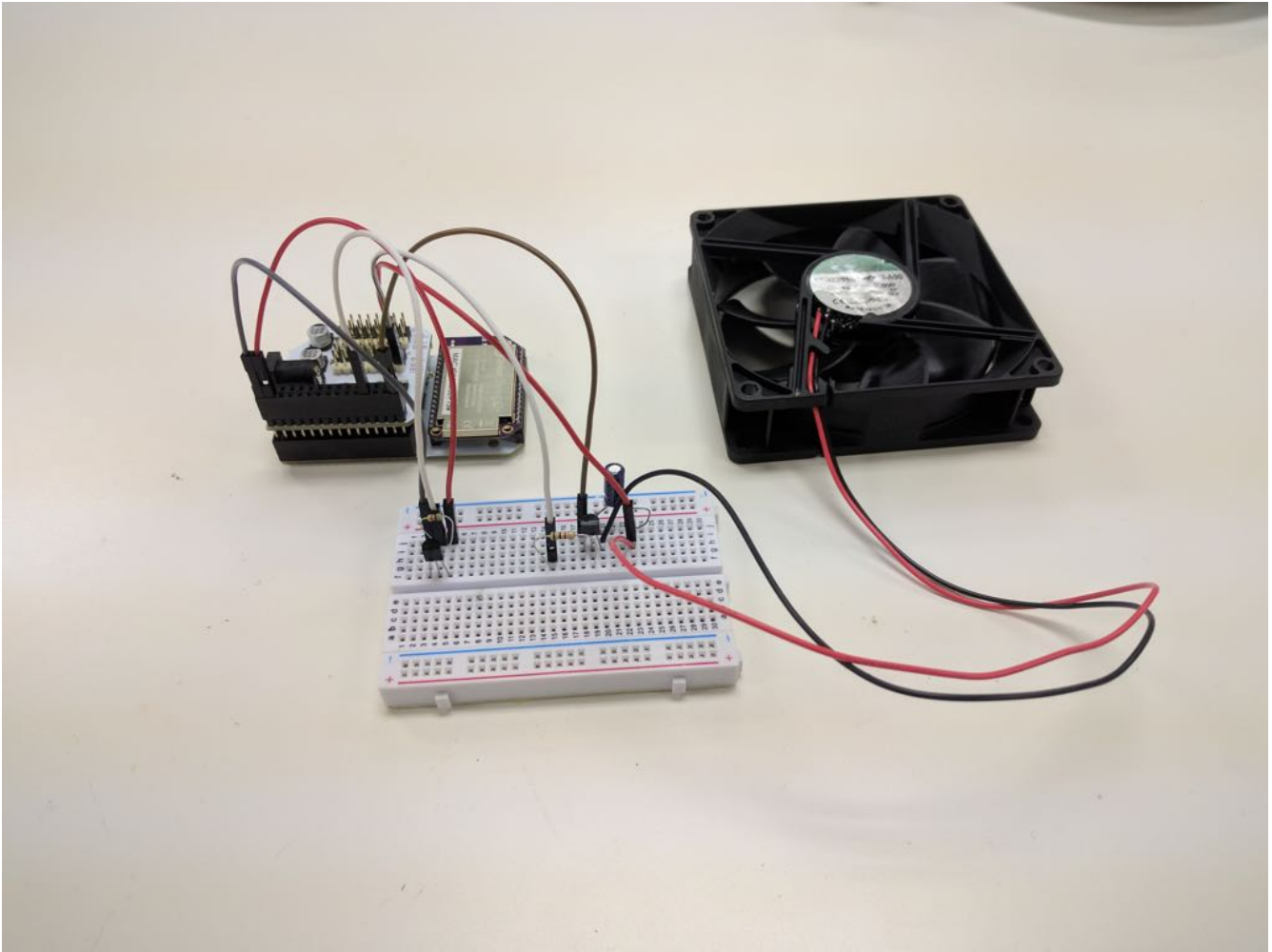
**NOTE:** the second graphic is a **bottom** view, where the pins are pointing towards you (we may have fried a sensor by misreading this one).

Now we can connect the sensor to the Expansion Headers.

1. First, connect the temperature sensor to the breadboard across another three empty rows.
  - Leave some space from the transistor so you can easily interact with it!
2. Connect the **GND** pin of the sensor to a **GND** pin on the Expansion Header using a M-M jumper wire.
3. Next, connect the middle pin (**DQ**) to **GPIO1** on the Expansion Header using a M-M jumper.
4. Connect the **VDD** pin to a **3.3V** pin on the Expansion Header using a M-M jumper.
5. Finally, connect the **5.1k $\Omega$**  resistor across the sensor's **VDD** and **DQ** pins (right and middle respectively).



Your setup is now complete!



## 5. Get the Project Code

The code for this project is all done and can be found in Onion's [iot-smart-fan repo](#) on GitHub. Use [git to download the code to your Omega](#): navigate to the /root directory, and clone the GitHub repo:

```
cd /root
git clone https://github.com/OnionIoT/iot-smart-fan.git
```

### 5.5. Using a Different Sensor

There's a good bit of setup for the temperature sensor - initialization, communicating, and parsing.

If you have a different sensor than the the one we're using, you'll have to modify the project code. The code that sets up the D18B20 1-wire sensor can be found in the lines between `#~~~ SENSOR SETUP BEGIN` and `#~~~ SENSOR SETUP END`.

Additionally, you'd probably need to change the function used to get the sensor data:

```
temp = sensor.readValue()
```

One important thing to note is that the values assigned to the `temp` variable must be integer or float.

## 6. Calibrate and Customize

You can edit the `config.json` to change the possible speed range of the fan and restrict the temperature range to which the fan reacts:

```
{
  "tempMax" : "40",
  "tempMin" : "18",
  "dutyMin" : "60",
  "dutyMax" : "100",
  "frequency" : "1000",
  "fanType" : "case"
}
```

The `dutyMin` and `dutyMax` parameters control the minimum and maximum duty cycle of the signal being sent to the fan, thereby controlling the fan speed. The `tempMin` and `tempMax` parameters specify the temperature range in which to enable the fan. The fan speed has a linear relationship with the temperature when it is between the min and max temperature.

If you find that the fan does not spin when current is applied, you may have to increase the `dutyMin` to overcome the static friction in the fan's shaft bearing. Once it gets up to speed, you can then lower the duty and the fan will still be able to spin.

### Using A Different Fan

If you would rather use the H-Bridge and DC Motor setup, you'll have to make some changes to the code. Namely, you'll have to swap out the `OmegaPwm` class with the `hBridgeMotor` class from `omegaMotors.py`. Check the pin-outs that we've put in by default in `iotSmartFan.py` to make sure you're correctly connecting the H-Bridge to the Servo Expansion.

For a detailed guide on how to set this up, check out the wiring instructions in the [Maker Kit DC Motor experiment](#).

To change up the code, open up `iotSmartFan.py` and change this line:

```
fan = OmegaPwm(FAN_PWM_CHANNEL)
```

To this:

```
fan = hBridgeMotor(FAN_PWM_CHANNEL, H_BRIDGE_1A_CHANNEL, H_BRIDGE_2A_CHANNEL)
```

And this line:

```
fan.setDutyCycle(duty)
```

To this:

```
fan.spinForward(duty)
```

### Code Highlight

Two of the key components in this project are the temperature sensor and the motor drivers, found in `temperatureSensor.py` and `omegaMotors.py`.

The output from the 1-Wire temperature sensor contains a lot of unnecessary information such as the device address, connection acknowledgements, and other fields. The `__readOneWire()` internal method of the `TemperatureSensor` class extracts the temperature value and converts it to degrees Celsius:

```
def __readOneWire(self):
    # device typically prints 2 lines, the 2nd line has the temperature sensor at the end
    # eg. a6 01 4b 46 7f ff 0c 10 5c t=26375
    rawValue = self.driver.readDevice()

    # grab the 2nd line, then read the last entry in the line, then get everything after the
    value = rawValue[1].split()[-1].split("=")[1]

    # convert value from string to number
    value = int(value)

    # DS18B20 outputs in 1/1000ths of a degree C, so convert to standard units
    value /= 1000.0
    return value
```

The method to set the duty cycle for a servo fan, `setDutyCycle()` uses the Onion `pwmExp` class to easily control it:

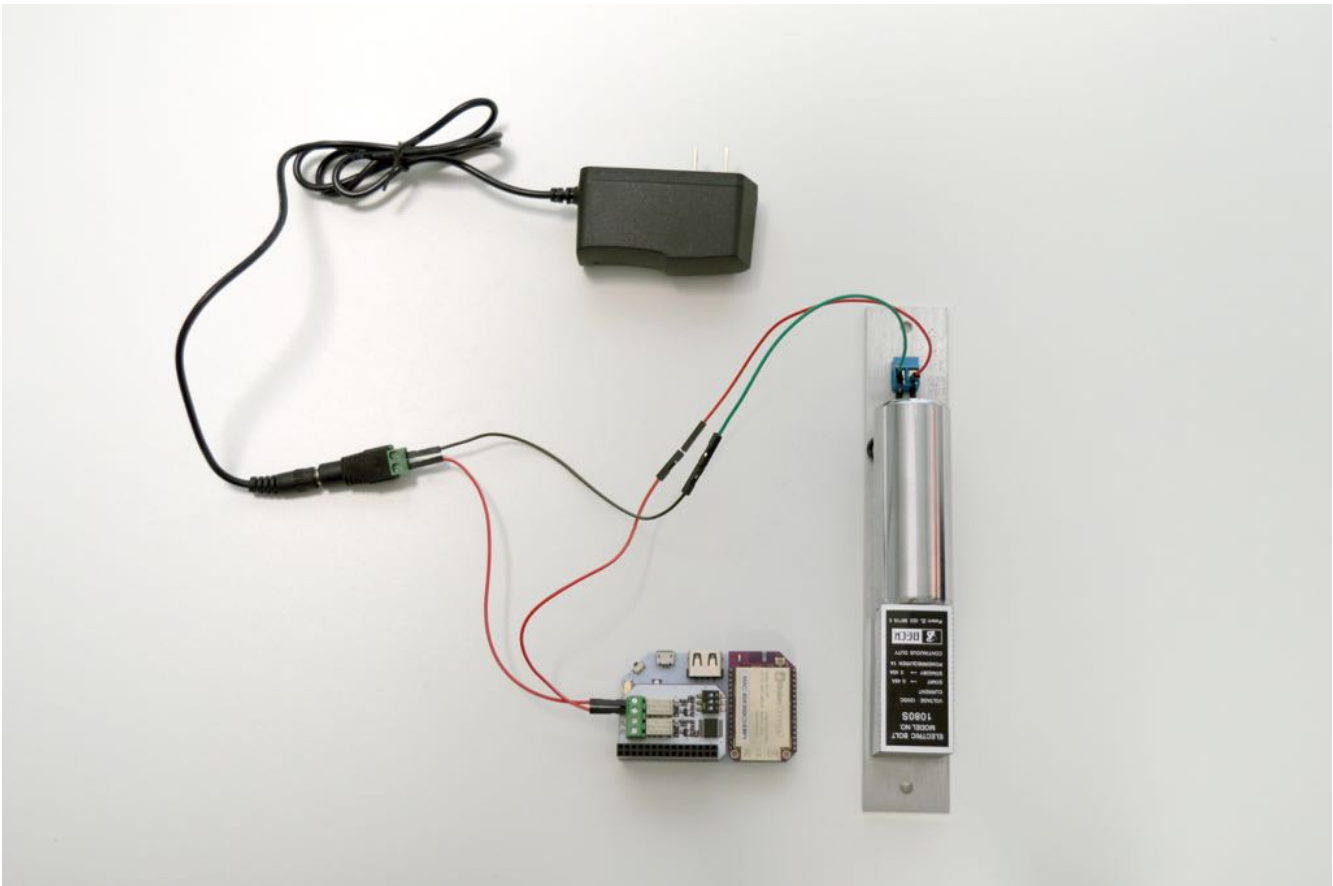
```
def setDutyCycle(self, duty):
    """Set duty cycle for pwm channel"""
    ret = pwmExp.setupDriver(self.channel, duty, 0)
    if (ret != 0):
        print 'ERROR: pwm-exp setupDriver not successful!'

    return ret
```

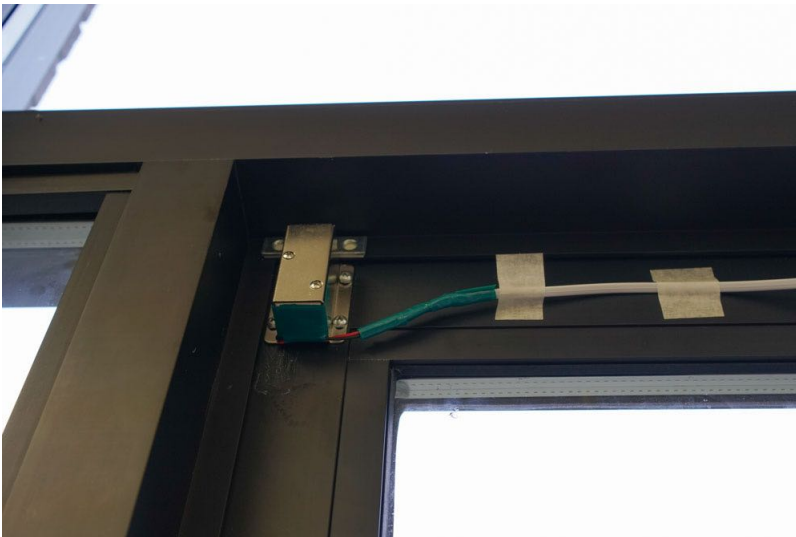
## IoT Lock

Keys are so last year. With the Omega and the internet, we can unlock things with our keyboard or touchscreen!

In this project, we'll be building an electric lock system with the Omega:



In fact, we use this very setup to control a secondary lock at Onion HQ:



Note: in fact, keys are still very useful. We still recommend you to use a normally-open lock and a key-lock in conjunction, as power failure will result in a fail-safe backup instead of locking you out.

**Disclaimer:** This security-related project is just that, a *project*. This is not intended to be a fully-featured or robust home security solution. Use your own judgment when applying this project to securing your belongings, property, etc. By doing this project, you accept all risk and Onion cannot be held responsible for any damages or misuse.



## Overview

**Skill Level:** Intermediate

**Time Required:** 1.5 hours

To accomplish this, we'll use the HTTP server, `uhttpd` on the Omega to listen for the unlock signal through a request and `cgi-bin` scripts to control the lock. When it's set up, we'll be able to unlock by accessing a web page through a phone, a laptop, a tablet, a tv, anything!

The web page will allow us to:

- unlock
- lock
- toggle - unlock and then lock again after some time
  - If the lock is currently locked. If not, this action does nothing

## Ingredients

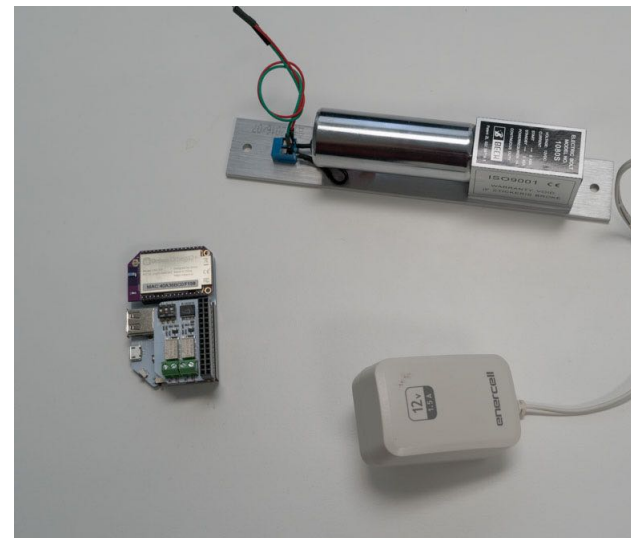
- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
- Onion [Relay Expansion](#)
- An [electric solenoid lock](#)
  - We recommend a simple power locking, normally unlocked lock so you don't get locked out when there's no power.
- Appropriate DC power supply for your lock
  - We found a [12V 1A DC power supply](#) to be compatible with most locks

Lock Mounting Tools:

- Screws
- Bolts
- Extra wires
- Appropriate tools

This really depends on where and how you're mounting your lock

Here's what our list looked like - minus the mounting tools and parts.



## Step-by-Step

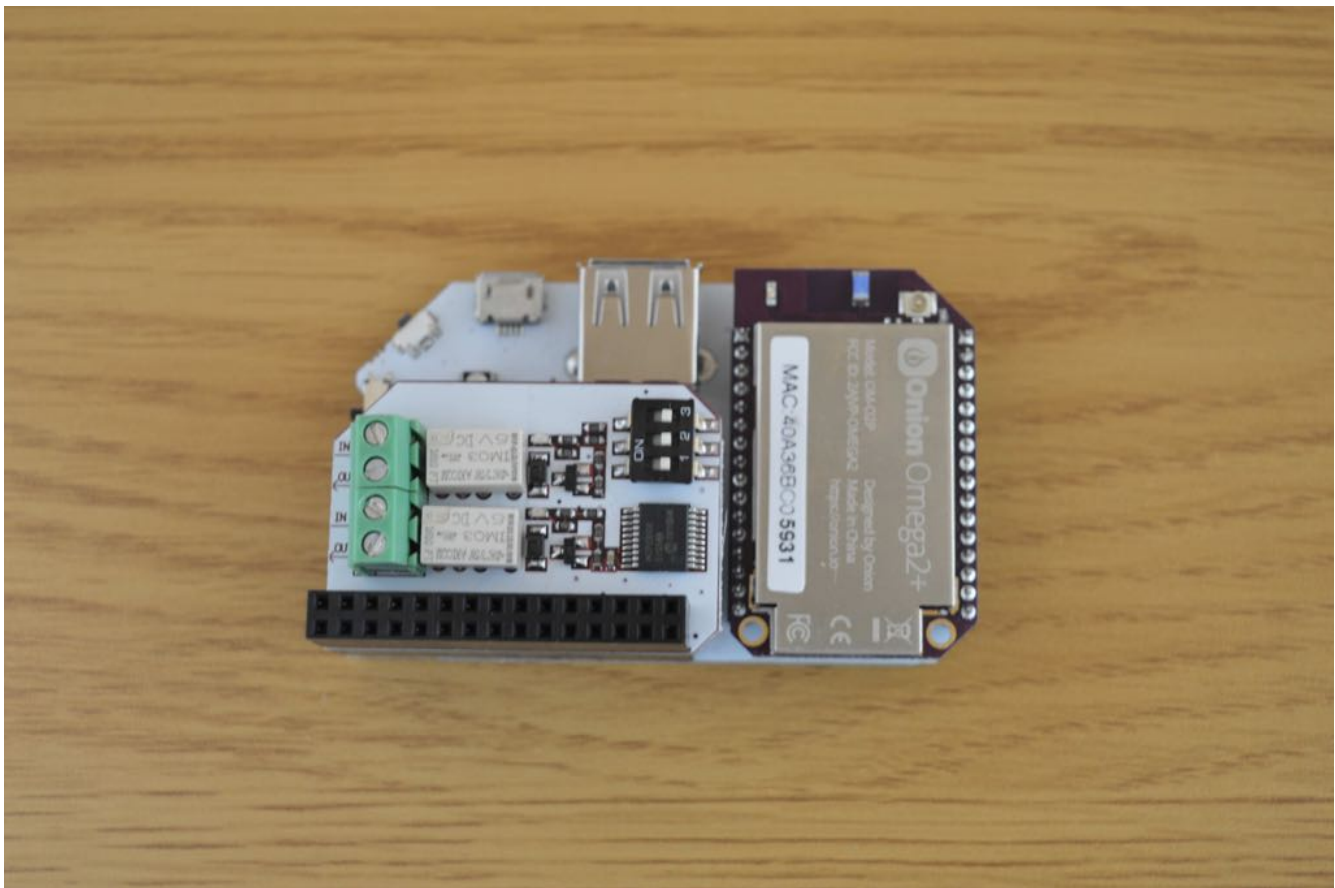
Our instructions will be based on the recommended lock type. If you have an advanced electric lock with multiple settings, you can adjust the instructions as you see fit.

### 1. Prepare

To get started, we need to set up the Omega and our lock.

First we need an Omega2 ready to go. If you haven't already, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

Plug in the Relay Expansion, and that's it for the Omega.



### 2. Test the lock

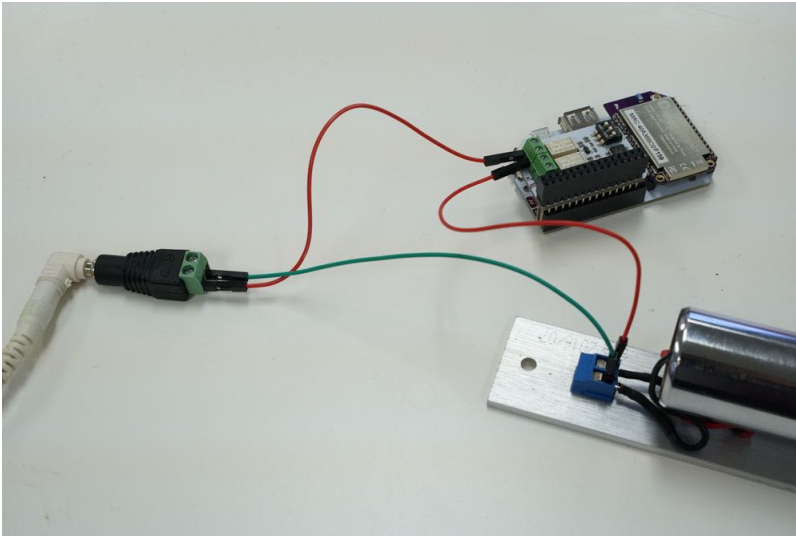
Next, read up on the operation of the lock of choice. Our code is based on a simple on/off switch system so it helps to know if it will work with your chosen lock.

It's a good idea to start with a simple circuit to test the hardware. Using a two-wire lock, we'll connect it to our power supply through the Relay Expansion.

To set up the terminals on the Relay Expansion, turn the screw on the terminal counterclockwise until the metal clamp inside is sitting a bit less than halfway in the bottom of the housing, not too much or the screw might pop out. The screw terminal on the barrel jack adapter is a

bit different, it will rise and sink depending on the clamp position. When the screw is roughly flush with the top, it is open. To close it, turn clockwise until the screw sinks to about halfway, or until it becomes difficult to continue turning.

- First, connect the **negative (ground) terminal** (usually the black wire) of the lock to the **negative (ground) terminal** of the power supply.
- Next, connect the **positive terminal** of your supply to the **IN** screw terminal of Channel 0 on the Relay Expansion
- Finally, connect the **positive (power) terminal** (usually red) to the **OUT** screw terminal of Channel 0 on the Relay Expansion.



Once the lock is wired, connect to the Omega's **command line** and then switch on the relay:

```
relay-exp -i 0 on
```

If the lock's state changes, you're all set to continue! Before proceeding, You can disable the lock with:

```
relay-exp -i 0 off
```

#### 4. Plan out the Lock Placement

Before getting to software, you should make sure the lock chosen can be mounted to the door with good fit. Take some measurements and plan out the wiring and placement of the Omega/supply so we can quickly follow through once the software is ready to go.

*Measure twice, cut once.*

#### 5. Mount the lock

Now that the pieces work together, it's time to mount your lock! Keep all the components powered off, and take the testing rig apart



At Onion HQ, we've extended the wiring of the lock and routed it to an Omega and power supply right next to the door, but depending on the situation, you may have to do something completely different.

## 6. Download the Project Code

The code for this project can be found in Onion's [iot-door-lock repository](#) on GitHub. We'll use [git to download the code to your Omega](#): navigate to the `/root` directory on the Omega, and clone the GitHub repo:

```
opkg update
opkg install git git-http ca-bundle
cd /root
git clone https://github.com/OnionIoT/iot-door-lock.git
```

If your lock has more modes/controls, feel free to take a look at the code (specially `www/cgi-bin/door.sh`) and make changes that control your lock more effectively.

## 7. Adjust the Code for your Lock

The code assumes two things:

- That the lock in use is a normally closed lock
  - ie when there is no current (the Relay Expansion channel is off) the lock will be in the **locked** state
- When toggling the lock, it assumes a delay of **24 seconds** between unlocking and locking again

To adjust either of the above, you'll need to edit the `www/cgi-bin/door.sh` script:

- To change the Relay Expansion channel values for the lock, adjust the `LOCKED` and `UNLOCKED` variables
- To change the delay between unlocking and locking during a toggle, adjust the `TOGGLE_TIME` variable

## 8. Serve the Lock Webpage

In order to serve up the webpage that we'll use to send the lock commands, copy the contents of the `www` directory of the project directory to the `/www` directory on your Omega, and you should be good to go!

```
cp -r iot-door-lock/www/ /
```

By virtue of `uhttpd`, the HTTP server running on the Omega, all of the files in the `/www` directory will be served up as a website.

## Using the IoT Lock

Now the truly IoT part, using the IoT Lock!

1. Connect your Omega to your WiFi network, or connect your computer to the Omega's WiFi network.
2. In a web browser, navigate to `omega-ABCD.local/lock.html`, where `ABCD` is the last 4 digits on the sticker on the Omega.

- On some Android and PC devices, the `omega-ABCD.local` address doesn't always work. Follow our [guide on finding your Omega's IP Address](#) and use the IP address instead of `omega-ABCD.local` when connecting the web interface. It will be something along the lines of `192.168.1.109/lock.html`
3. Hit any of the buttons to carry out the action indicated
    - As a refresher, we can Toggle the lock (unlock momentarily and then lock again), unlock it, or lock it



## Bonus: Automatically Lock & Unlock

To make this truly useful & automated, we can schedule when the IoT lock will unlock and lock using the `cron` Linux utility!

Check out the cron example that sets up the lock to turn on and off at 11AM and 6PM respectively but only on weekdays:

```
0 11 * * 1,2,3,4,5 sh /www/cgi-bin/door.sh unlock
0 18 * * 1,2,3,4,5 sh /www/cgi-bin/door.sh lock
#
```

This is included in the project code repo as `crontab.txt`

Here's a quick overview of cron job definitions work:

```
# * * * * * command to execute
#
#
#
#          day of week (0 - 7) (0 to 6 are Sunday to Saturday, or use names; 7 is Sunday, the sa
#          month (1 - 12)
#          day of month (1 - 31)
#          hour (0 - 23)
#          min (0 - 59)
```

# The hash (#) denotes a comment that will be ignored

So the first line specifies that the lock will be unlocked at 11am on weekdays, and the second line specifies that it will be locked at 6pm on weekdays.

To apply this scheduling to your IoT lock, type `crontab -e` to add a task to the `cron` daemon, it will open a file in `vi`, enter in the command listed up above. Then restart the `cron` daemon for the changes to take effect:

```
/etc/init.d/cron restart
```

Great! Your IoT lock now runs on a schedule!

Check out the Omega documentation for more info on [using cron](#)

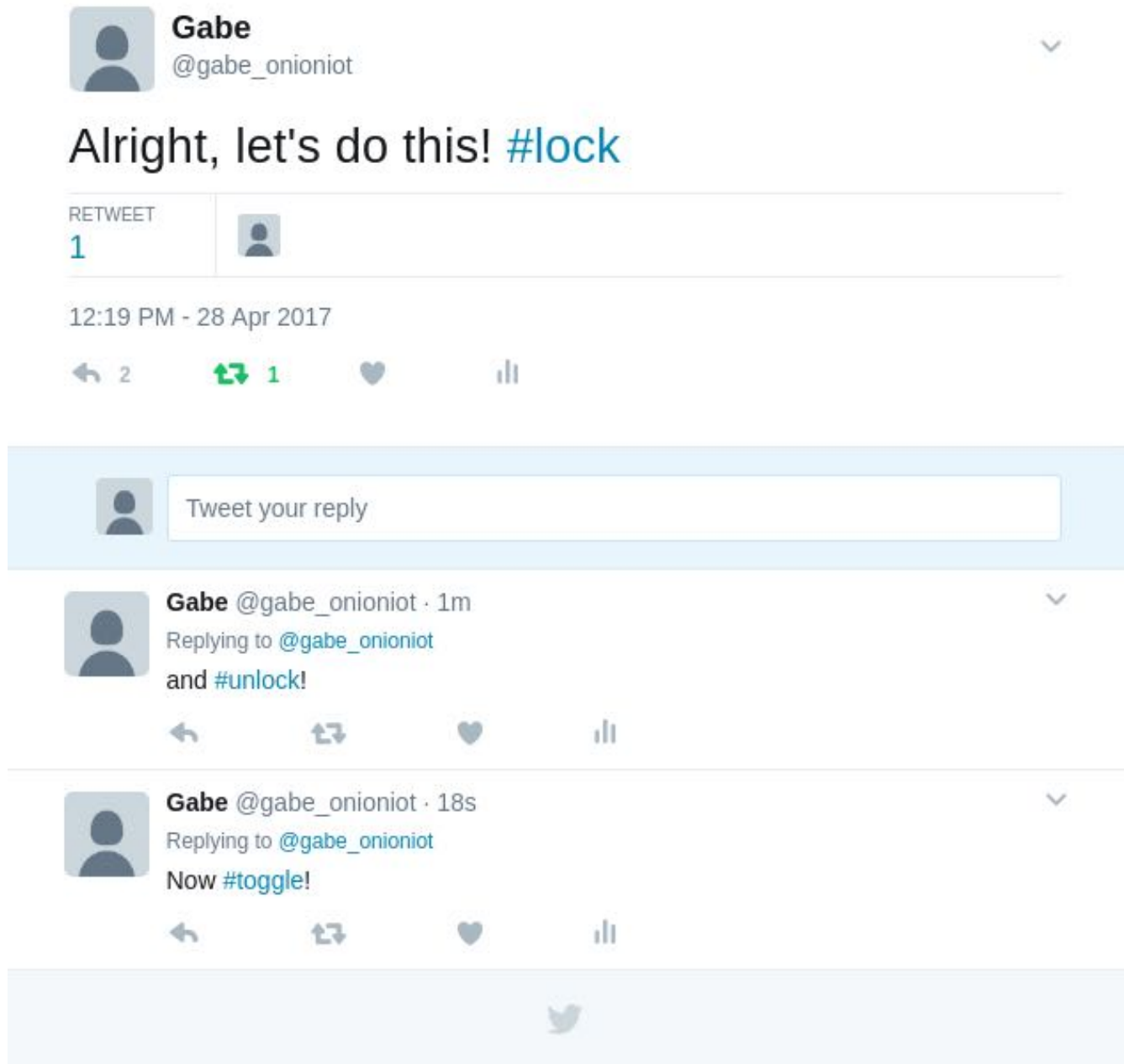
## Going Further

While this is really useful, [next](#) we'll make the lock react to Tweets from authorized users, so you can lock and unlock your IoT lock from anywhere, and even give access to your friends!

## IoT Lock - Control with a Tweet

Let's take what we did in Internet Lock - Part 1, and add a program to control it in real time from Twitter!





The screenshot shows a Twitter thread. At the top is a tweet from Gabe (@gabe\_onioniot) with the text "Alright, let's do this! #lock". Below it, there is a "RETWEET" section showing 1 retweet. The tweet is dated "12:19 PM - 28 Apr 2017" and has 2 replies, 1 retweet, and 0 likes. Below the tweet is a light blue box with a "Tweet your reply" input field. Underneath are two replies from Gabe: the first says "Replying to @gabe\_onioniot and #unlock!" and the second says "Replying to @gabe\_onioniot Now #toggle!". The bottom of the screenshot shows the Twitter logo on a light blue background.

**Disclaimer:** This security-related project is just that: a *project*. This is not intended to be a fully-featured nor robust home security solution. Use your own judgment when applying this project to securing your belongings, property, etc. By doing this project, you accept all risk and Onion cannot be held responsible for any damages or misuse.

## Overview

**Skill Level:** Intermediate

**Time Required:** 30 minutes

The code will be written in Python and we'll be making use of [Twitter's Streaming APIs](#) to grab Tweet data. Specifically, the code uses the [Tweepy library](#) to manage authentication and keeping a persistent

HTTP connection to Twitter.

Same as before, the code used to handle this setup can be found in the [iot-door-lock repository](#) on GitHub.

## Lock Access Rules

First, this app will be listening for new tweets from the users you specify in the configuration file. Please note that all public tweets will be received over the Internet by your Omega when the main script is running.

By default, the Omega is configured to change the state of the lock when it detects a tweet from an authorized user with a corresponding hashtag. The list of allowed users and hashtags for each command are configured in a separate JSON file, `config.json`.

The default hashtags that correspond to lock actions are:

Hashtag	Lock Action
#lock	Program the Relay to set the lock to <b>locked</b>
#unlock	Program the Relay to set the lock to <b>unlocked</b>
#toggle	Unlock, wait 5 seconds, lock again (Does nothing if lock is already unlocked)

Some examples of authorization are shown below (all of the command hashtags follow the same rules):

Event	Lock Response
@authorizedUser1 posts a status containing #unlock	Unlock
@authorizedUser2 replies to a status and includes #toggle	Toggle (briefly unlock, and then lock again)
@authorizedUser2 retweets a status from @authorizedUser1 that contains #lock	Lock
@unauthorizedUser1 posts or retweets a status containing #toggle	None
@unauthorizedUser2 posts retweets a status from @authorizedUser1 containing #unlock	None

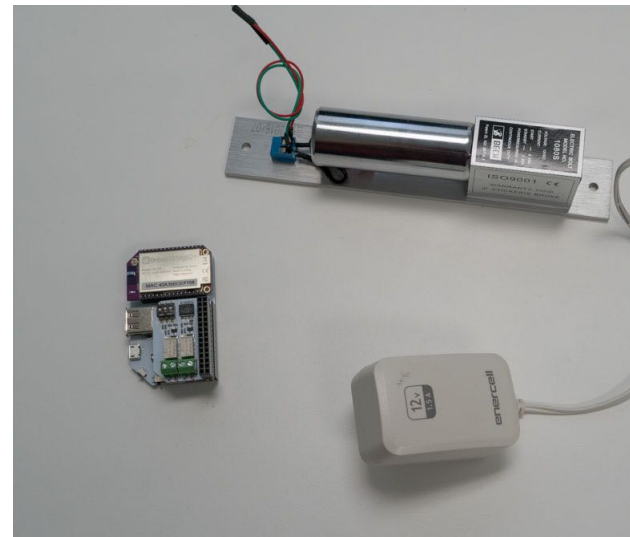
**Note:** The lock will respond to the **first** command hashtag it finds in each status, so posting “#lock #unlock” will activate the lock.

## Ingredients

Unlike Part 1, the dependencies for the Python Twitter software requires more space than is available on the Omega2 standard model by default. You will either have to **boot from external storage** or use an Omega2+ instead.

We will be using the same components and setup as in the first part:

- Onion **Omega2** or **Omega2+**
- Any Onion Dock that supports Expansions: **Expansion Dock**, **Power Dock**, **Arduino Dock 2**
- Onion **Relay Expansion**
- An **electric solenoid lock**
  - We recommend a simple power locking, normally unlocked lock so you don't get locked out when there's no power.
- Appropriate DC power supply for your lock
  - We found a **12V 1A DC power supply** to be compatible with most locks



Here's what our list looked like - minus the mounting tools and parts.

## Step-by-Step

Follow these instructions to control the smart lock from Twitter on your very own Omega!

### 1. Prepare

You'll have to have your Omega2 ready to go, complete the **First Time Setup Guide** to connect your Omega to WiFi and update to the latest firmware.

### 2. Complete Part 1 of the Project

This project builds on the first part of the IoT Lock project. If you haven't already completed the **first part**, go back and do it now!

### 3. Install Dependencies

Connect to the Omega's command line and run the following commands:

```
opkg update
opkg install python-pip
pip install --upgrade setuptools
pip install tweepy
```

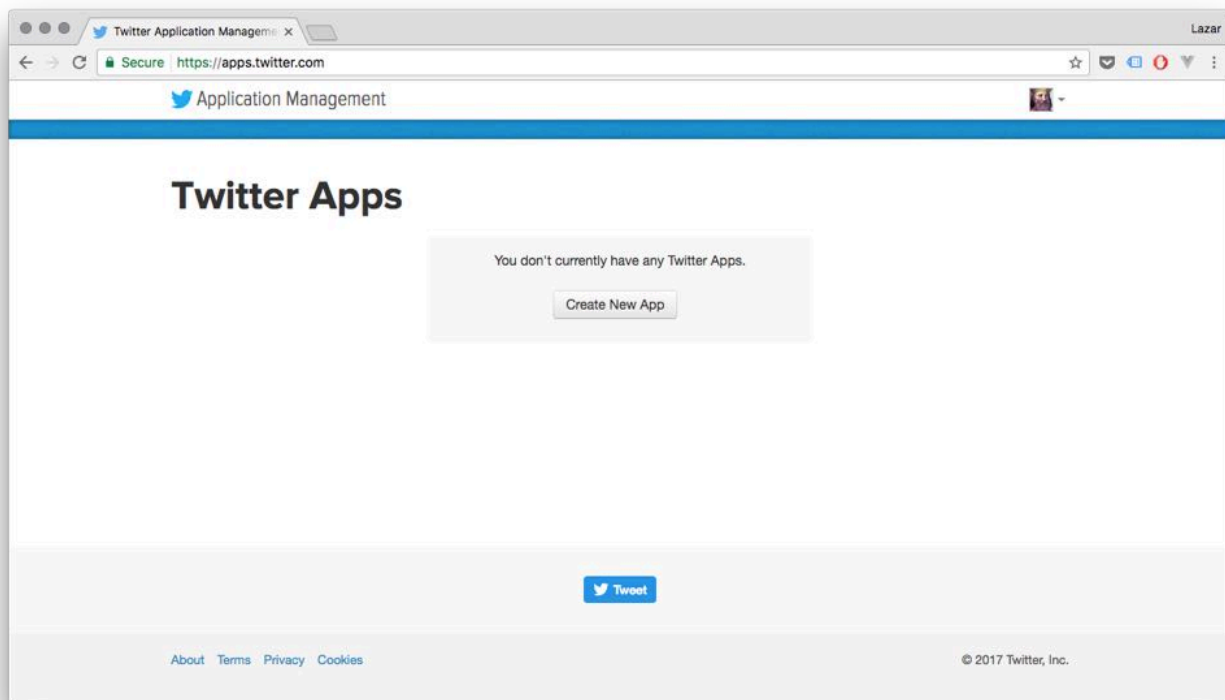
### 4. Create a Twitter Application

We'll need to create a Twitter Application in order to be able to use Twitter's APIs to grab Tweets. Specifically, our code needs the following information:

- an API Key
- an API Secret
- an Access Token
- an Access Token Secret

in order to authenticate with Twitter before we can use the APIs.

1. If you don't have a Twitter account, [create one now](#).
2. Head over to <https://apps.twitter.com> and sign in with your Twitter handle



1. Fill in the form details for your application. Twitter app names must be unique globally, so try calling it `omega-ABCD-door-lock`, where ABCD are the 4 digits in your Omega's hostname.

Create an application | Twitter x Lazar

Secure <https://apps.twitter.com/app/new>

Application Management

## Create an application

**Application Details**

**Name \***

Omega Twitter Reader

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

**Description \***

Reading the latest Tweet from a specified user

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

**Website \***

https://onion.io

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

**Callback URL**

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth\_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

1. Read and agree to the Twitter Developer Agreement and hit Create your Twitter application.

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

**Website \***

https://onion.io

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

**Callback URL**

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth\_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

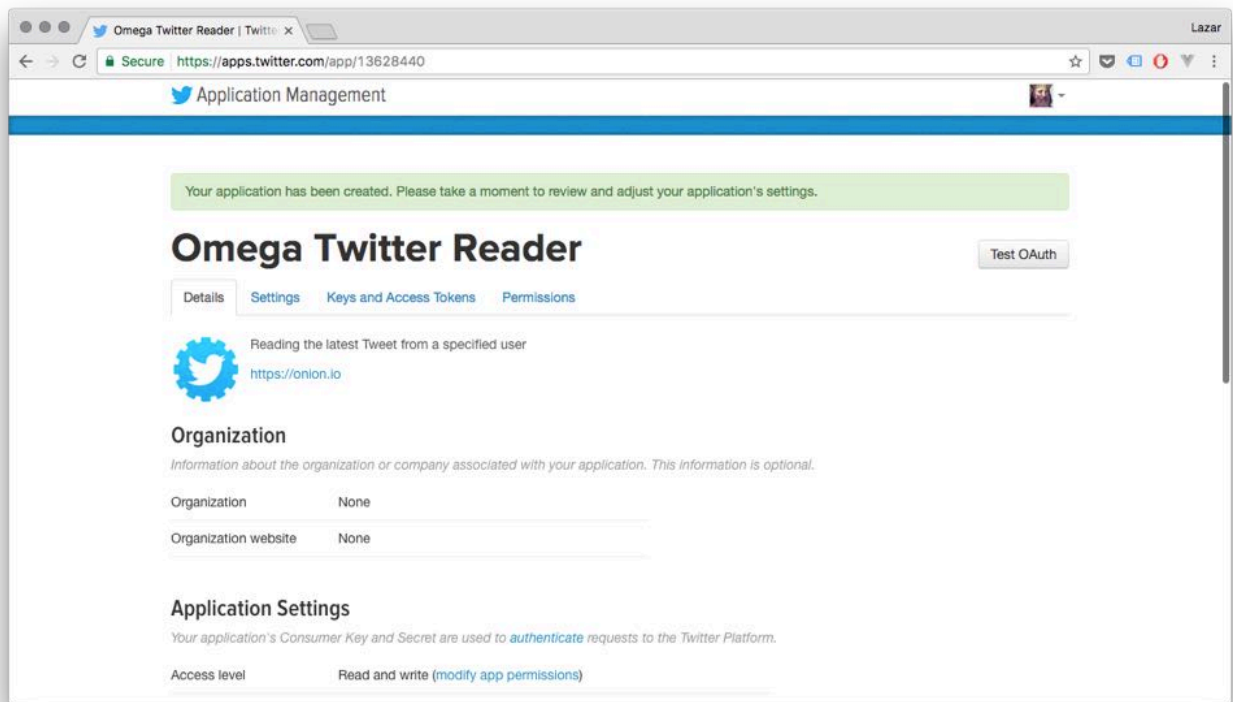
**Developer Agreement**

Yes, I have read and agree to the [Twitter Developer Agreement](#).

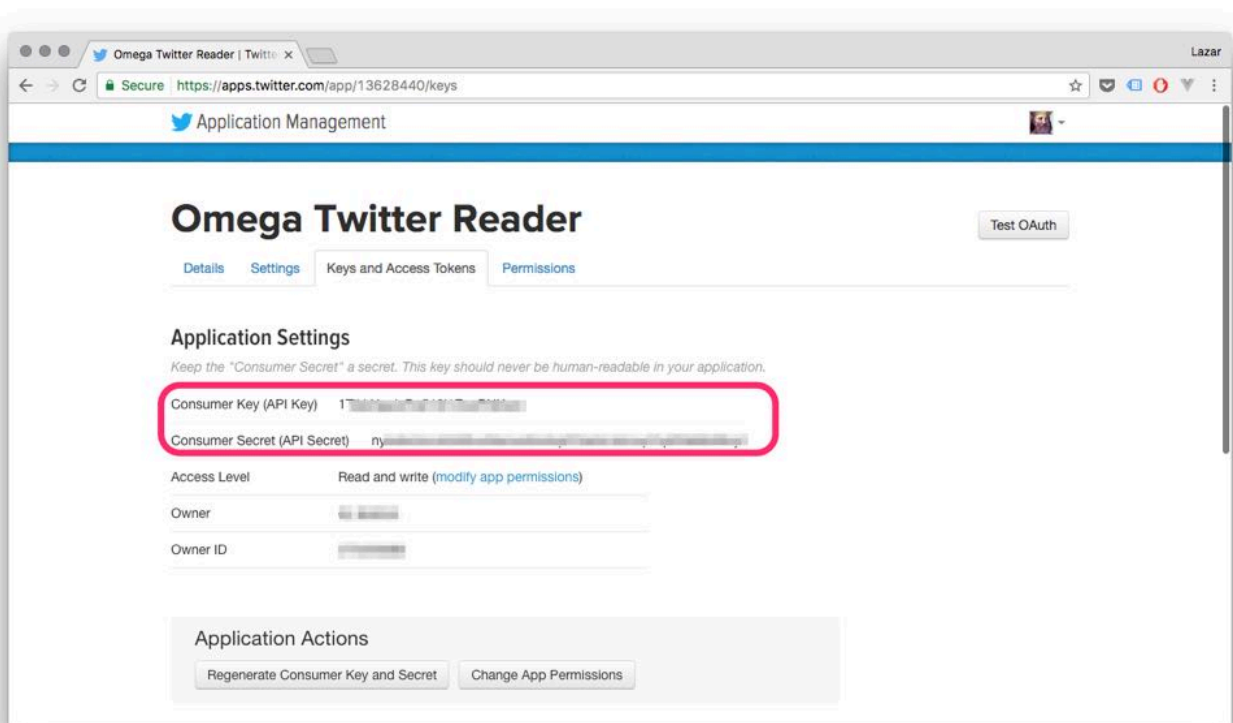
Create your Twitter application

Note that your Twitter account must have an associated mobile phone number before Twitter will allow you to create an application!

1. Your Application is now created!



1. Head over to the **Keys and Access Tokens** tab to grab the info we need



1. Scroll down to the section called "Your Access Token", and click "Create my access token".

## Your Access Token

*You haven't authorized this application for your own account yet.*

*By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.*

### Token Actions

Create my access token

We will be using the 2 pairs of keys and secrets to authorize our app to connect to Twitter, so copy and paste or write them down somewhere for later.

## 5. Edit the Configuration File

Open the `config.json` file and edit or paste in the following information:

config.json	Value
consumerKey	Consumer Key in the Twitter application menu
consumerSecret	Twitter Consumer Secret
accessToken	Twitter Access Token
accessTokenSecret	Twitter Access Token Secret
allowedUsers	A comma-separated list/array of screen names who will have access to your lock. eg.: ["@john_smith", "@jane_doe"]
hashtags	Customize the hashtag you want for each action. The defaults are #lock, #unlock, and #toggle.

If you trust them, you can add friends or family to the list of allowed users.

## 6. Running the Project

Navigate to the repo directory and run:

```
python tweetLock.py
```

The script will then run forever, listening for new tweets until you exit by pressing **Ctrl-C**.

Now try tweeting from some of the allowed accounts and include one of the hashtags you configured. You should see your lock reacting very quickly!

The screenshot shows a Twitter interface. At the top is a tweet by **Gabe** (@gabe\_onioniot) with the text "Alright, let's do this! #lock". Below the tweet, it shows "RETWEET 1" and "12:19 PM - 28 Apr 2017". There are icons for replies (2), retweets (1), likes, and a share icon. Below the tweet is a "Tweet your reply" input field. Underneath are two replies from Gabe: "Replying to @gabe\_onioniot and #unlock!" (1m ago) and "Replying to @gabe\_onioniot Now #toggle!" (18s ago). The bottom of the screenshot shows the Twitter logo on a light blue background.

Tell your friends to try it out too!

## 7. Rate Limiting

The Twitter Streaming API that pushes new tweets to the Omega limits the amount of **new** sessions you can initiate within a certain period of time. If you restart the program too often in a short window



of time, you will receive a 420 error. You will see a warning on the command line, and the program will automatically disconnect and retry according to Twitter's recommended backoff policy; see the Rate Limiting section on [Twitter's documentation](#).

The rate limiting criteria are not made public, so we recommend playing it safe and relaxing for about 5-10 minutes each time you need to restart the script.

**Note:** Too many connection attempts may result in your IP being banned from connecting to Twitter!

## 8. Running the Program on Boot

We can automate this project to run when the Omega is turned on, and we can also make it run in the background so you can use the Omega for other things while it's running! To do this, we'll place a script in `/etc/init.d`.

In the repo folder, make the `etc/init.d/tweet-lock` file executable, copy it to `/etc/init.d`, then enable it to run on boot:

```
cd etc/init.d
chmod +x tweet-lock
cp tweet-lock /etc/init.d
/etc/init.d/tweet-lock enable
```

Wait for 5-10 minutes, reboot the Omega, and you will automatically be able to tweet at your lock again!

The `/etc/init.d/tweet-lock` script registers the IoT Lock Python script as a service with `procd`, the process management daemon of the system. `procd` then ensures that the process gets started at boot and continues to run until the service is disabled.

## Code Highlight

This project uses the Twitter Streaming API, made easily accessible by the Tweepy library. Streaming, as opposed to using the REST API, lets Twitter push data such as new statuses or direct messages immediately after they happen in real-time. This eliminates the overhead of repeatedly polling Twitter for new information; it'll get sent to us as it happens.

When opening a stream, you must specify some way to filter the incoming tweets. This is because thousands of tweets are sent every second on average; that's a **lot**! The two most common ways of doing this are by:

- following particular users
- tracking keywords

Here we use the first method by only receiving tweets from our list of authorized users.

The streaming functions have been further abstracted by the `StreamListener` and `TwitterApp` classes in `twitterHelper.py`. Any class that is used to receive and process tweets must extend the `tweepy.StreamListener` class by redefining its callback methods, such as `on_status()` and `on_error()`. These functions are called when an event such as a new status occurs. They do nothing by default, so you need to tell them what to do!

For full details on the Tweepy library, visit the [documentation page](#).



## 5 | Audio Projects

Like a regular computer, the Omega2 IoT computer is equipped to handle sound output and input. The fact that the Omega runs a Linux Operating System, makes the endeavor easier since there is a wide variety of already available software for this very purpose.



### Concepts

A highlight of some of the concepts that will be covered in these projects:

- Interfacing with a USB audio adapter
- Configuring Linux programs and utilities
- Bluetooth pairing with a device

### Projects

Audio projects for the Omega:

1. **AirPlay Speaker**
  - Stream music to your Omega over WiFi using Apple's AirPlay protocol
2. **Bluetooth Speaker**
  - Use your Omega as a Bluetooth speaker and stream music from your devices

## AirPlay Speaker

Since the Omega can be set up to receive AirPlay streams, we can turn our Omega into a WiFi speaker that can be controlled with a laptop or phone using a USB audio device.



### Overview

**Skill Level:** Intermediate

**Time Required:** 15 Minutes

There are three main pieces of the puzzle here:

To get AirPlay working, we will set up `shairport-sync` on the Omega. The audio stack on the Omega will work out of the box with USB devices, so we simply need to plug in any USB audio device. Finally, to actually stream music, a device with AirPlay controller capabilities must be set up to stream to the Omega.

Reference configuration files can be found on Onion's [audio-airplay-receiver](#) repo on GitHub.

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock with a USB host connector: [Expansion Dock](#), [Power Dock](#), [Mini Dock](#), [Arduino Dock 2](#)
  - We found the [Power Dock](#) especially useful since you can take it on the go!
- [USB Audio Adapter](#)
  - We used an adapter but USB speakers will likely work too
- Headphones or a small speaker



## Step-by-Step

Follow these steps and we'll have audio streaming to the Omega in no time!

### 1. Prepare the Ingredients

For this project, we'll need an Omega2 ready to go. If needed, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 2. Replace Avahi

The version of Avahi that comes installed on the Omega does not have `dbus-daemon` support, this needs to be fixed!

[Connect to the Omega's command line](#) to uninstall the pre-existing avahi package. Then we can get the `avahi-dbus-daemon` package to replace it.

First uninstall avahi:

```
opkg remove avahi-nodbus-daemon --force-depends
```

Once that's finished,

```
opkg update
opkg install avahi-dbus-daemon --force-overwrite
```

### 3. Install Shairport Sync

The `shairport-sync` package runs an Airplay Receiver server to listen and process AirPlay streams. Fortunately, it is available in the Onion Repositories, so we can install with `opkg`:

```
opkg install shairport-sync
```

### 4. Configure Shairport Sync

Shairport Sync requires some setup to work properly. To configure it, we'll be editing `/etc/config/shairport-sync`. Open it up and you should see something like this:

```
# Use your own config file
config shairport-sync 'shairport_sync_file'
    option disabled '1'
    option respawn '1'
    option conf_custom '1'
    option conf_file '/etc/shairport-sync.conf'

# Use OpenWrt UCI config
config shairport-sync 'shairport_sync'
    option disabled '1'
    option respawn '1'
    ...
    ...
```

The first block isn't useful to us. We want to edit options under the `# Use OpenWrt UCI config` line.

Specifically, these following lines:

```
    option disabled '1'
    ...
    option name 'Shairport-Sync-%v-%h'
    option password ''
    ...
    option mdns_backend '' # avahi/external-avahi/dns-sd/external-dns-sd/tinysvcmdns
```

The Steps:

- First let's set `disabled` to `'0'` to enable the UCI configuration.
- Optionally, pick out a new name to display in your AirPlay devices menus.
- Optionally, add a password to ensure only trusted users can access your Airplay receiver - Not Recommended
- Finally, make sure the `mdns_backend` is set to `'tinysvcmdns'`
- The rest of the options can be kept as their default values.

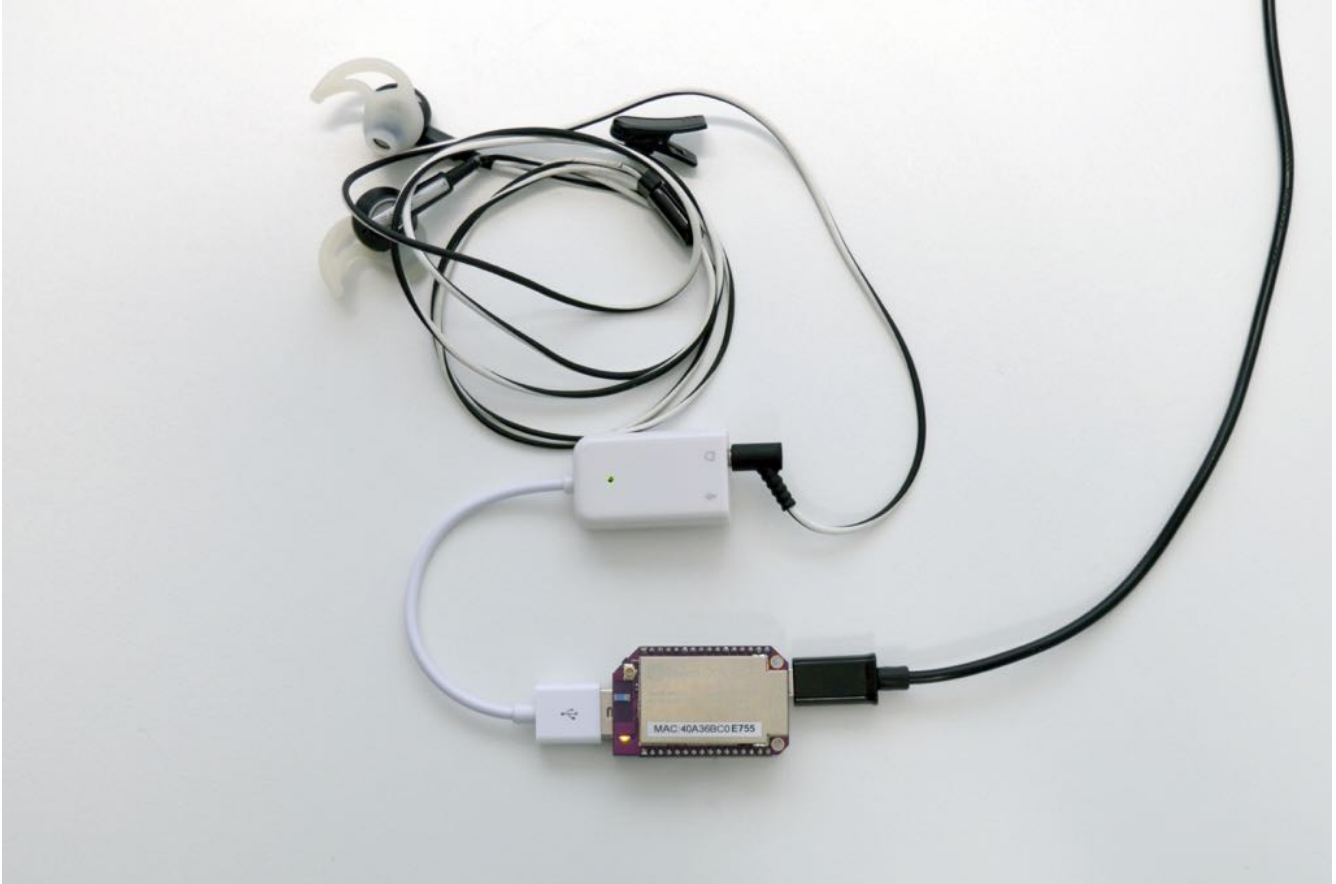
If you need a reference for the configuration files, we've put them into the [audio-airplay-receiver](#) repo on GitHub.

Restart the Omega for the changes to take effect, and we'll plug in some speakers!

## 5. Set up your Speakers

In our setup, we used a USB-based Audio Adapter. It has a built in Digital-Analog Converter (DAC) that receives digital audio data from the Omega through the USB port and converts it into an analog audio signal for speakers or headphones.

But any USB speaker setup should work out of the box thanks to Linux's audio stack. Plug it into the dock, and we'll be good to go!



## 6. Prepare your controller

AirPlay works out of the box for iOS devices, so if you own one there's no set up needed.

If you wish to use an Android device, we found AllStream and DoubleTwist to have stable AirPlay integration.

## 7. Fire up Shairport Sync

Now that everything's ready to go, enter `shairport-sync -d` to start up the shairport-sync server in the background.

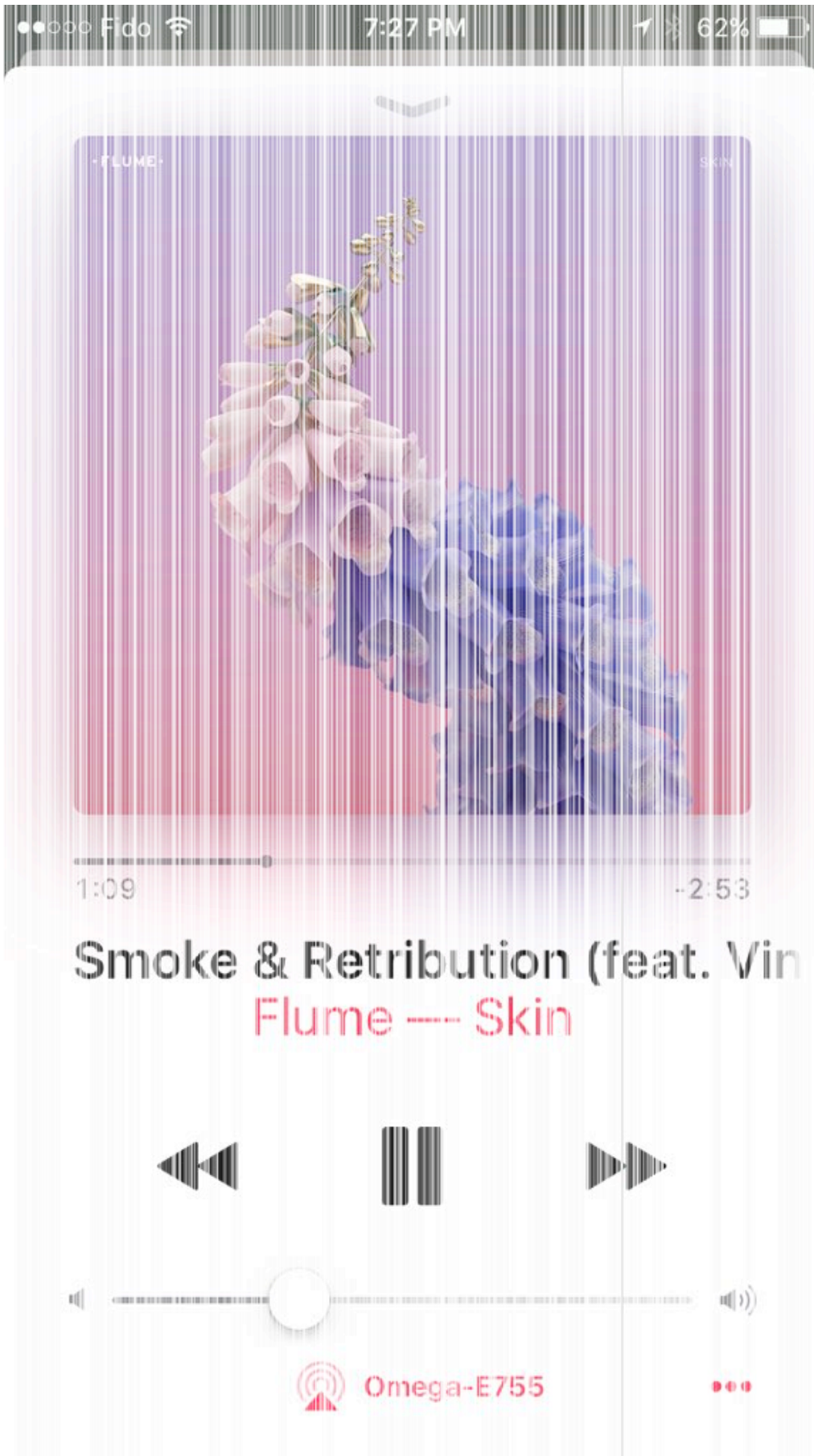
## Listening to Tunes

Now take a look at your AirPlay device, and you should see the Omega pop up as a receiver!

On an iPhone:







On an Android phone using the AllConnect App:



Play some tunes and enjoy your AirPlay-powered WiFi audio streaming brought to you by your Omega.

## Acknowledgements

A big thank you to Mike Brady, who keeps the [shairport-sync project](#) alive, making this project possible!

## Bluetooth Speaker

The Omega can communicate with other devices using the Bluetooth Low Energy wireless protocol. In this project, we're going to turn it into a Bluetooth speaker that you can play music and control it from your phone or tablet!



## Overview

**Skill Level:** Intermediate

**Time Required:** 15 minutes

We'll first install the necessary Bluetooth and audio drivers. Then we'll learn how to pair Bluetooth devices with the Omega. Then we'll connect a speaker and play our favourite music!

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock with a USB host connector: [Expansion Dock](#), [Power Dock](#), [Mini Dock](#), [Arduino Dock 2](#)
  - We found the [Power Dock](#) especially useful since you can take it on the go!
- Onion Bluetooth Expansion
- [USB Audio Adapter](#)
  - We used an a adapter but USB speakers will likely work too
- [USB hub](#) with at least 2 ports
- Standard headphones or speakers with a 3.5mm audio jack

## Step-by-Step

Here's how to turn your Omega into a Bluetooth speaker!

### 1. Prepare the Ingredients

For this project, we'll need an Omega2 ready to go. If needed, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 2. Setup the Hardware

1. Connect the Omega to the Dock.
2. Plug in the USB hub to the large USB host port.
3. Plug in the Bluetooth Expansion and USB Audio Expansion into the USB hub.
4. Do not connect your speaker just yet, as there will be loud popping and crackling when the USB Audio Expansion initializes!



After assembling all the components, turn on the Omega.

### 3. Install Software

Connect to the Omega's command line and install the necessary packages by running the commands below:

```
opkg update
opkg install bluez-libs bluez-utils pulseaudio-daemon pulseaudio-tools alsa-lib alsa-utils
```

- The bluez packages are for controlling the Bluetooth radio.
- The pulseaudio and alsa packages are audio drivers for Linux.

### 4. Setting Up the pulseaudio daemon

Run the following commands to initialize the daemon:

```
mkdir /run
udev --daemon
chmod 0777 /dev/snd/*
mkdir -p /var/lib/pulse
pulseaudio --system --disallow-exit --no-cpu-limit &
```

## After Rebooting

If you reboot the Omega, the daemon may still be running. Check using:

```
ps | grep pulse
```

If you see something like:

```
1124 pulse    10956 S <  /usr/bin/pulseaudio --system --disallow-exit --no-cp
```

Then it's running. You only need to run:

```
udevd --daemon
chmod 0777 /dev/snd/*
```

before moving to the next step again.

## 5. Pairing the Omega to Your Bluetooth Device

Check that your Bluetooth Expansion is properly detected by the Omega by running:

```
hciconfig -a
```

You should see some lines with information about your device.

```
root@Omega-F12D:/# hciconfig -a
hci0:  Type: BR/EDR  Bus: USB
      BD Address: 00:1A:7D:DA:71:13  ACL MTU: 310:10  SCO MTU: 64:8
      DOWN
      RX bytes:574 acl:0 sco:0 events:30 errors:0
      TX bytes:368 acl:0 sco:0 commands:30 errors:0
      Features: 0xff 0xff 0x8f 0xfe 0xdb 0xff 0x5b 0x87
      Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
      Link policy: RSWITCH HOLD SNIFF PARK
      Link mode: SLAVE ACCEPT
```

Run the following commands to turn on your Bluetooth Expansion:

```
hciconfig hci0 up
hciconfig hci0 sspmode enable
hciconfig hci0 piscan
```

Enable Bluetooth on the device you wish to connect to the Omega. Then on the Omega, enter the `bluetoothctl` command to be taken into a new command prompt. Then run:

```
agent on
discoverable on
pairable on
scan on
```

You should see success messages, and a list of Bluetooth devices.



```
root@Omega-F12D:/# hciconfig hci0 up
root@Omega-F12D:/# hciconfig hci0 sspmode enable
root@Omega-F12D:/# hciconfig hci0 piscan
root@Omega-F12D:/# bluetoothctl
[NEW] Controller 00:1A:7D:DA:71:13 OnionBLE [default]
[bluetooth]# agent on
Agent registered
[bluetooth]# discoverable on
Changing discoverable on succeeded
[bluetooth]# pairable on
Changing pairable on succeeded
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:1A:7D:DA:71:13 Discovering: yes
[NEW] Device 54:60:09:38:05:A9 54-60-09-38-05-A9
[NEW] Device 24:DF:6A:4C:27:42 Nexus 6P
[bluetooth]#
```

If you do not see these messages try removing the Bluetooth Expansion, rebooting your Omega, and trying again.

Take note of the device address you wish to pair, or copy it down in a text editor on your computer somewhere. In this example, the device's address is the string of numbers, letters, and colons in the red bubble above.

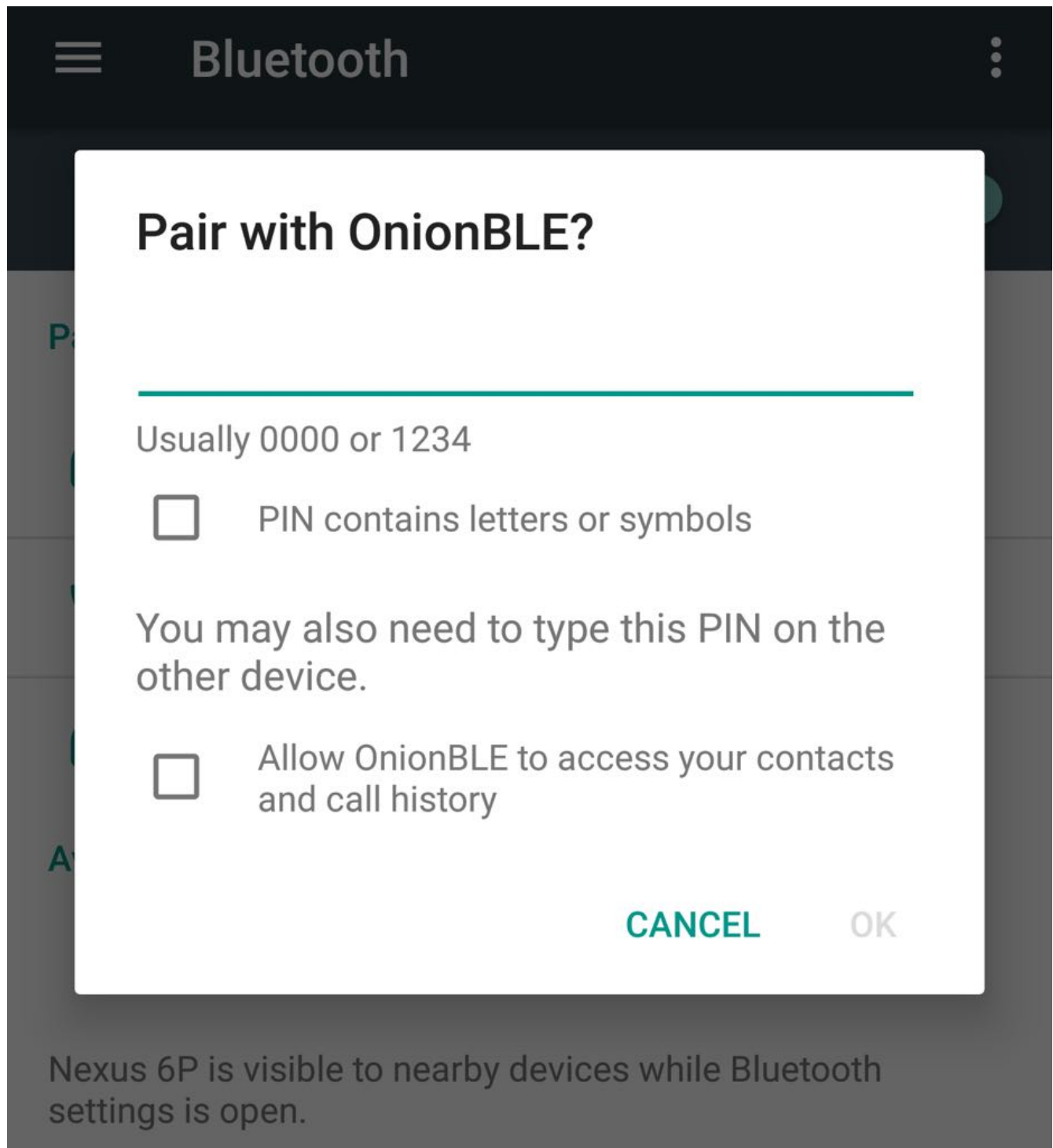
Then run:

```
pair YOURDEVICEADDRESS
```

It will then prompt you for a PIN to secure this connection. You can enter 0, 0000, 1234, or anything you like; make sure to remember it!

```
[bluetooth]# pair 24:DF:6A:4C:27:42
Attempting to pair with 24:DF:6A:4C:27:42
[CHG] Device 24:DF:6A:4C:27:42 Connected: yes
Request PIN code
[agent] Enter PIN code: 0
```

You should then get a prompt on your device asking you to connect to **OmegABLE** and enter a PIN. Enter the PIN you just provided on the command line to finish connecting.

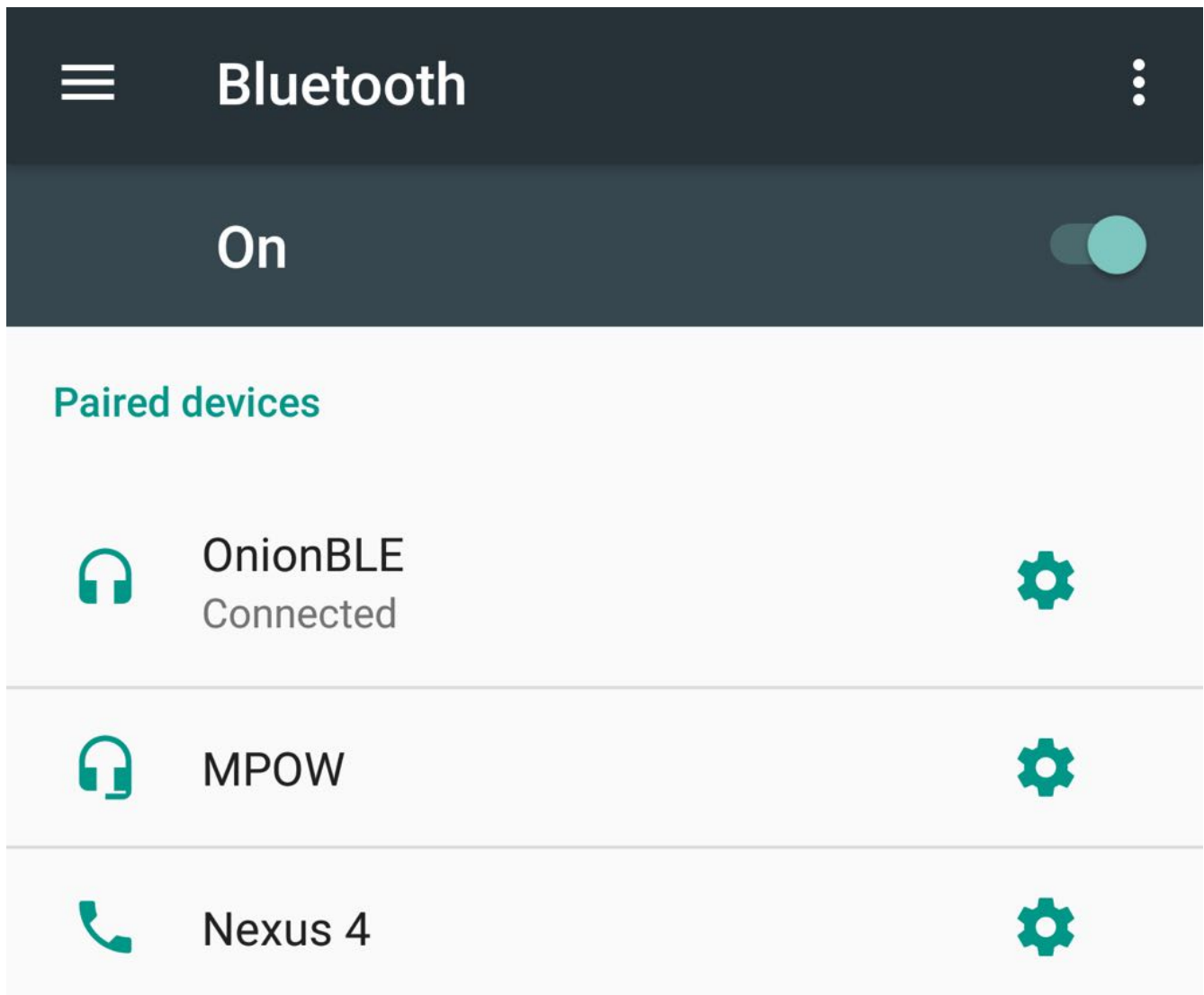


Complete the prompt on your device, then on the Omega run:

```
trust YOURDEVICEADDRESS
```

Your device has been **paired** with the Omega, meaning it can connect at any time. However, they are still not connected yet.

Now on your device, tap on the connection again and it should connect.



You can now quit the `bluetoothctl` program with the command:

```
quit
```

Check to make sure your device is still connected by running this command:

```
hcitool con
```

Your device should be listed. If you see nothing, try initiating the connection again from the remote device. If that doesn't work, go back to the previous step.

```
root@Omega-F12D:/# hcitool con
Connections:
  > ACL 24:DF:6A:4C:27:42 handle 71 state 1 lm MASTER AUTH ENCRYPT
root@Omega-F12D:/# █
```

## 6. Set Up Audio Streaming From the Device

We will use a command called `pactl` to set up audio streaming from the Bluetooth connection to the USB Audio Expansion. First run:

```
pactl list sources
```

And look for the Source with `bluez_source` in the Name field. Copy that entire label down for later.

```
Source #2
  State: SUSPENDED
  Name: bluez_source.24_DF_6A_4C_27_42
  Description: Nexus 6P
  Driver: module-bluez5-device.c
  Sample Specification: s16le 2ch 44100Hz
  Channel Map: front-left,front-right
  Owner Module: 13
  Mute: no
  Volume: front-left: 65536 / 100% / 0.00 dB, front-right: 65536 / 100% / 0.00 dB,
         balance 0.00
  Base Volume: 65536 / 100% / 0.00 dB
  Monitor of Sink: n/a
  Latency: 0 usec, configured 0 usec
  Flags: HARDWARE DECIBEL_VOLUME LATENCY
  Properties:
    bluetooth.protocol = "a2dp_source"
    device.description = "Nexus 6P"
    device.string = "24:DF:6A:4C:27:42"
    device.api = "bluez"
    device.class = "sound"
    device.bus = "bluetooth"
    device.form_factor = "phone"
```

Substitute the source name into the following command:

```
pactl load-module module-loopback source=SOURCENAME sink=alsa_output.default rate=44100 adjust_ti
```

The Omega is now ready to stream Bluetooth audio!

## 7. Using the Bluetooth Audio Streamer

Before plugging in your speaker, make sure the volume is set as low as possible. Then start playing music or audio on your device. Gradually turn up the volume on the speaker until you can hear it. And you're done!

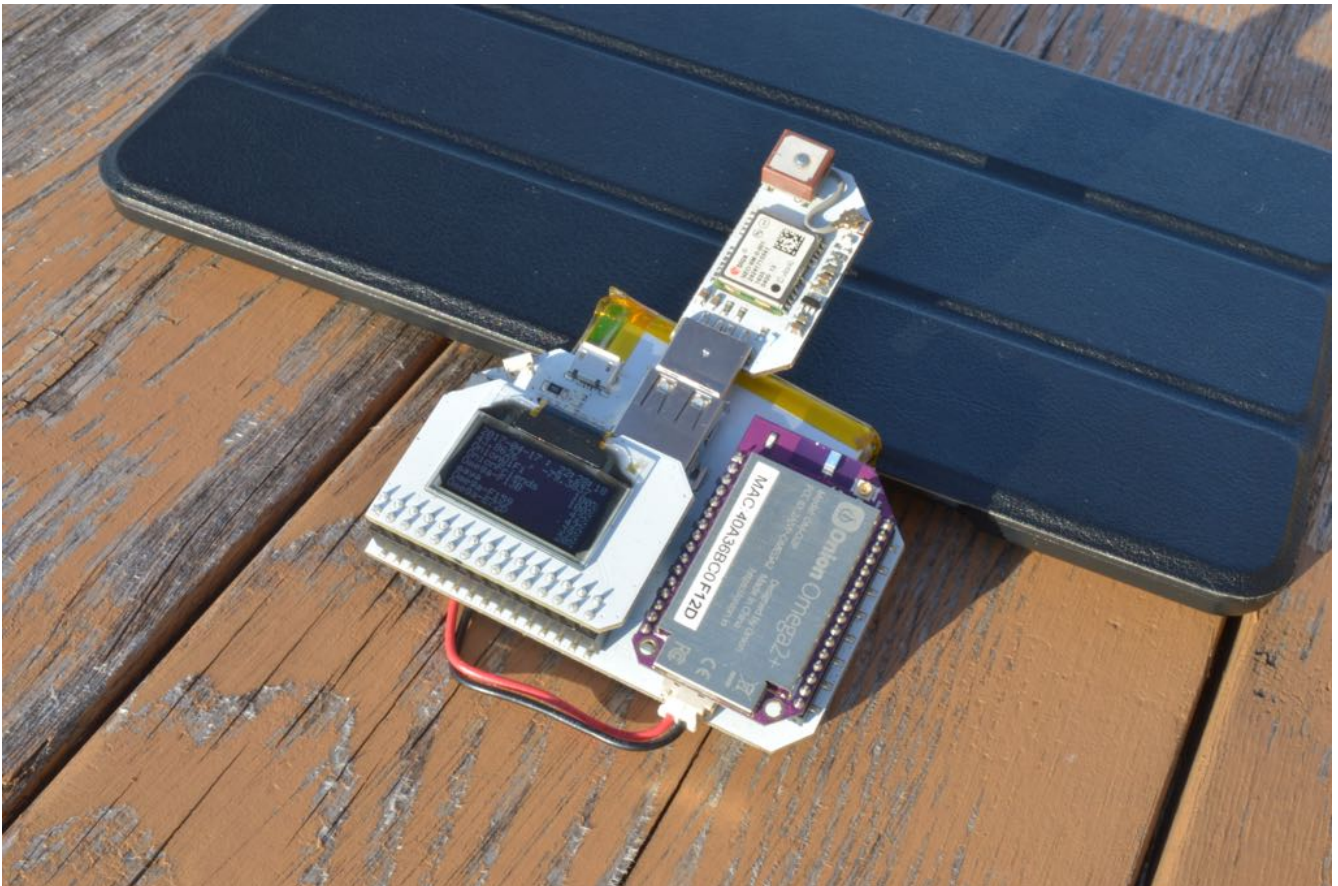


Enjoy your Omega-powered Bluetooth speaker!



## 6 | Wireless Projects

The Omega has pretty extensive networking capabilities, both wireless and wired when used with an [Ethernet Expansion](#). On the wireless side, the Omega can join existing WiFi networks as well as create and host its own WiFi network access point. In fact, it can even do both simultaneously. This opens the door to some novel scenarios especially since network connectivity can be forwarded between the Omega's network and the existing network to which it is connected. When used with the Ethernet Expansion, the Omega can join wired networks or share connectivity to the wireless network to which it is connected. And finally, since the Omega runs Linux, there are many, many useful tools available which can be used in creative ways, as you'll see in the upcoming projects.



### Concepts

A highlight of some of the concepts that will be covered in these projects:

- Using the GPS Expansion to get precise location data
- Using the `ubus` utility
- Modifying existing software and compiling it on the device
- Booting the Omega's OS from an SD Card (external storage)
- Sharing files on the local wireless network
- Network configuration for operation as a router
- Network configuration for operation as a WiFi range extender
- Network configuration for operation as a WiFi ethernet bridge

## Projects

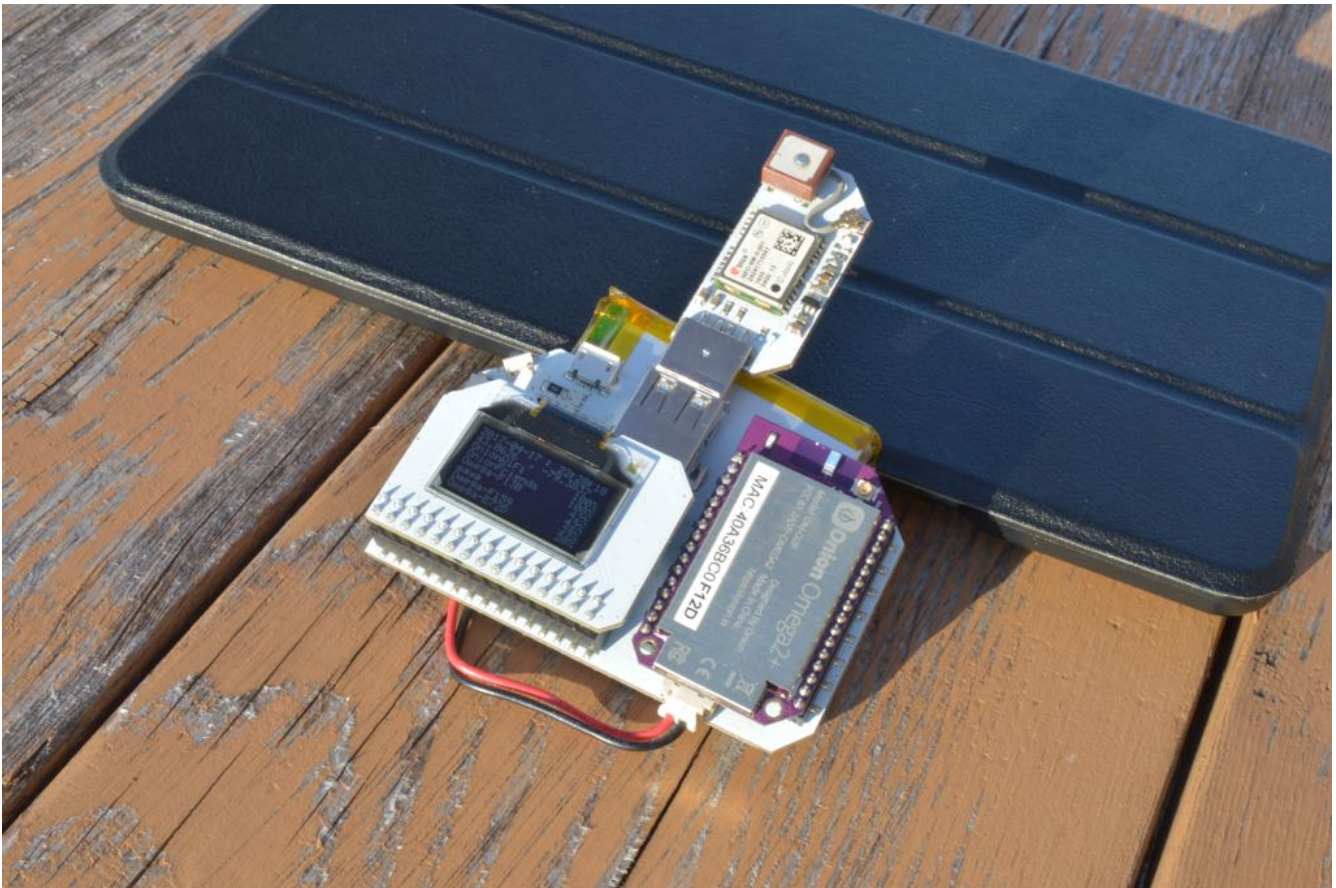
Projects that turn your Omega into a wireless networking tool:

1. **Mobile WiFi Network Scanner**
  - Collect and display the GPS location, signal strength, and more of WiFi networks in your surrounding area. Take your scanner on the go!
2. **OctoPrint 3D Printing Server**
  - Run the OctoPrint 3D Printing server on your Omega and add wireless connectivity to your 3D printer
3. **Mobile Network File Server**
  - Share files from a USB device on a WiFi network. Can take this on the road and provide access to data on the go!
4. **Omega as a WiFi Router**
  - Setup your Omega to act just like a WiFi router
5. **Omega as a WiFi Range Extender**
  - Is your WiFi network spotty if you get too far from your router? Setup your Omega to act as a range extender for your WiFi network
6. **Omega as an WiFi Ethernet Bridge**
  - Use your Omega to connect a computer

## Mobile WiFi Network Scanner

The Omega can scan nearby WiFi networks and report information such as their SSID, encryption type, and signal strength. In this project, we'll be using the Omega to scan local WiFi networks, record the GPS coordinates where they're found, display the networks with the strongest signal on the OLED Expansion, and save the rest of the data to a spreadsheet file.





## Overview

**Skill Level:** Intermediate

**Time Required:** 10 minutes

The WiFi scanner will:

- Scan for any WiFi networks in range using a `ubus` call
- Retrieve location data from the GPS Expansion, again using the `ubus`
- Sort the scanned networks by signal strength and display the six networks with the strongest signal on the OLED Expansion

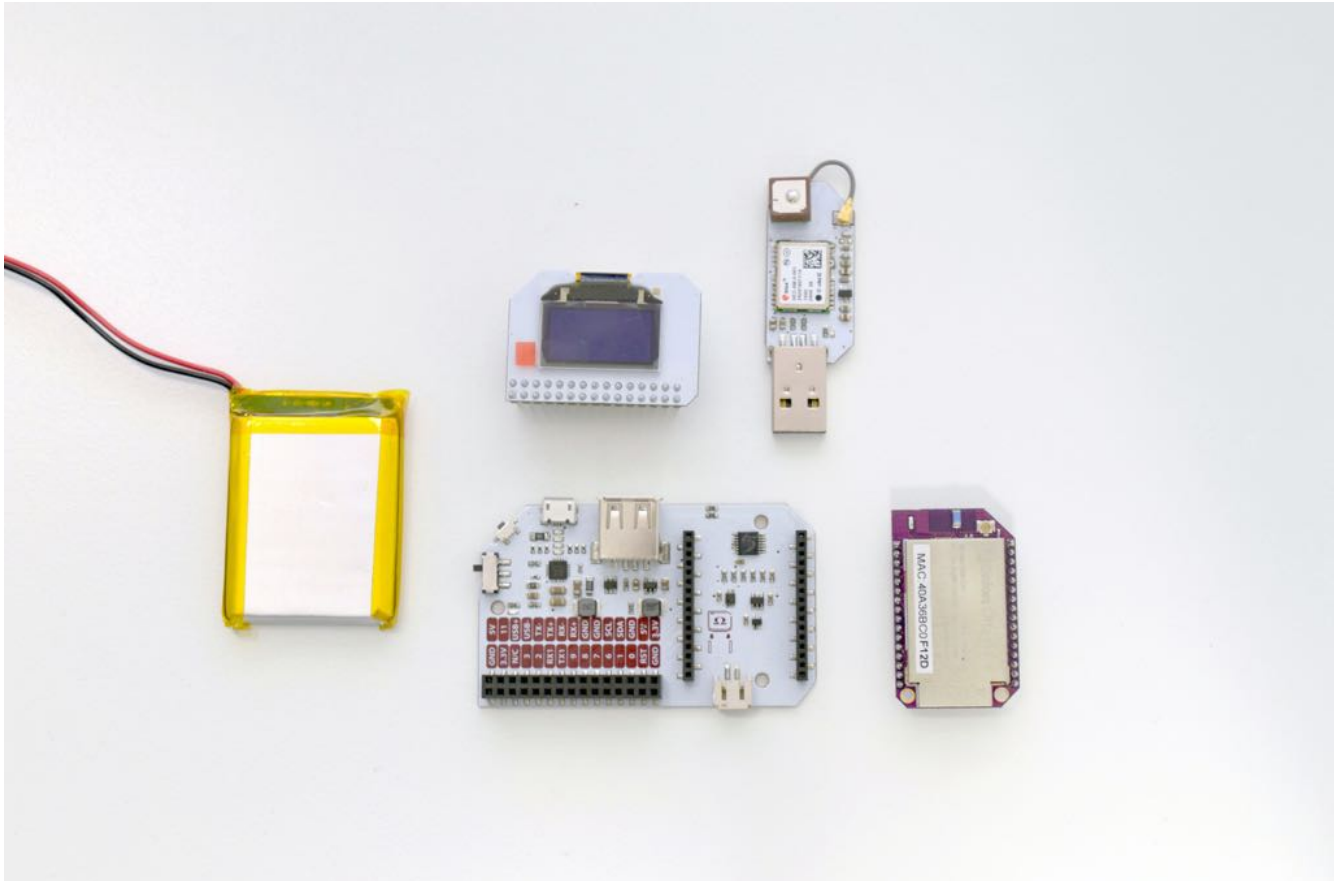
It will then save the following data for each network into a comma separated value (CSV) file that can be imported into a spreadsheet program:

- Date & time scanned
- Latitude and longitude
- SSID
- BSSID
- Encryption type
- Signal strength

Using the Power Dock, you will be able to use your scanner out in the world without needing a USB power supply.

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Onion [Power Dock](#)
  - The [Expansion Dock](#) and [Arduino Dock 2](#) will work as well, they just won't be mobile
- Onion [OLED Expansion](#)
- Onion [GPS Expansion](#)
- [External GPS Antenna](#) with a [u.Fl connector](#) (optional, but gets better reception indoors)
- A [3.7V LiPo battery](#)
  - We found 1200 mAh to be good for several hours of use



## Step-by-Step

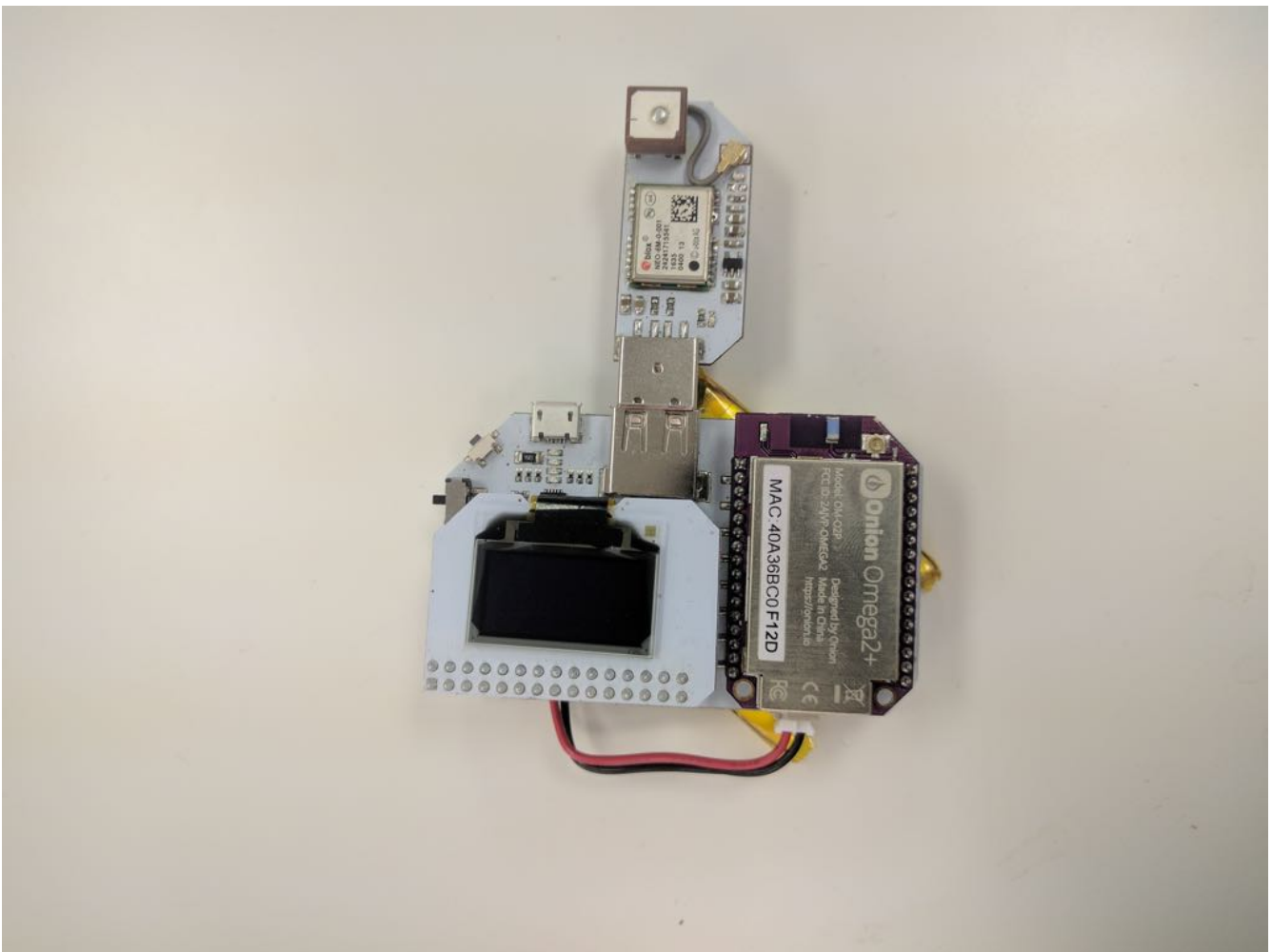
Here's how to turn your Omega into a WiFi scanner!

### 1. Prepare

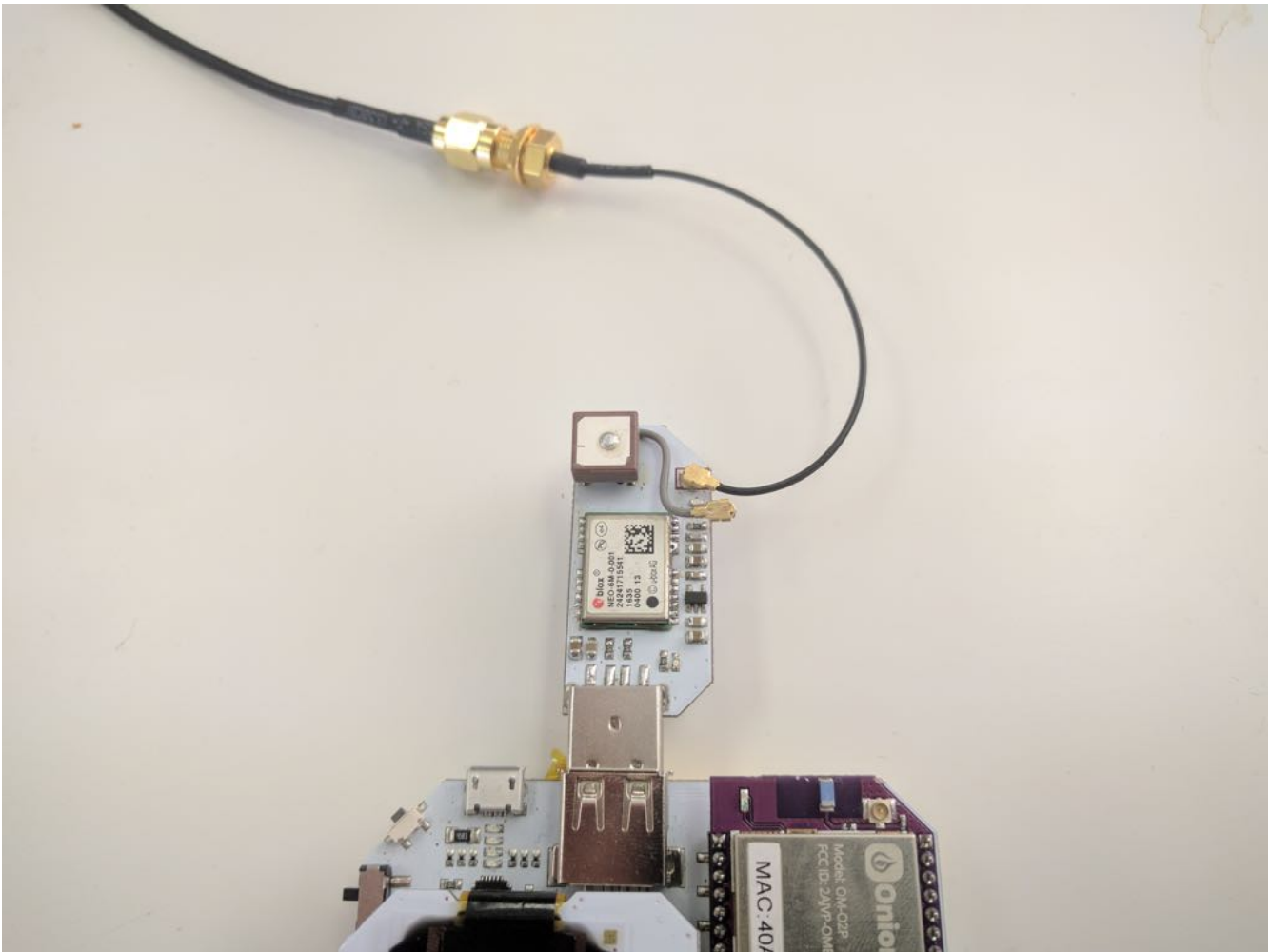
You'll need to have an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

## 2. Setup the Hardware

Connect your Omega to the Power Dock, then plug in the OLED Expansion into the Expansion Header. Then plug in the GPS Expansion into the USB host port as shown below.



The GPS Expansion's antenna is connected via a Hirose U.FL connector. If you have your own antenna with the appropriate connector that you would like to use, you can gently unplug the included antenna (the large square piece with a wire) and replace it with your own.



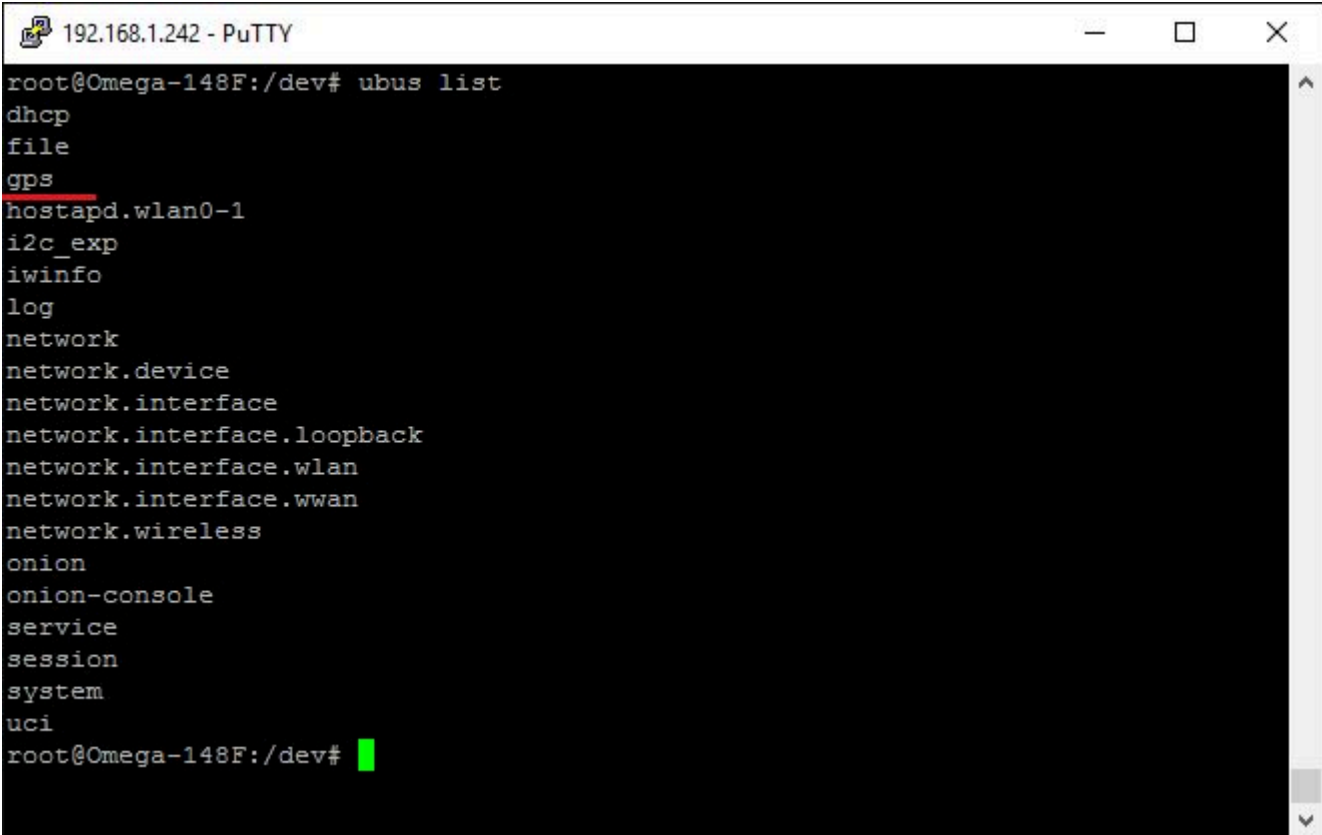
### 3. Install Packages

Connect to the Omega's command line and install Python as well as some of the packages we need:

```
opkg update
opkg install python-light py0ledExp ogps git git-http ca-bundle
```

The `py0ledExp` package gives us control of the OLED Expansion, while the `ogps` package will provide a `ubus` service that lets us easily get data from the GPS Expansion. The `git`, `git-http`, and `ca-bundle` packages will allow us to download the project code from GitHub.

After installing `ogps`, check that the `ubus` `gps` service is listed by running `ubus list`:



```
192.168.1.242 - PuTTY
root@Omega-148F:/dev# ubus list
dhcp
file
gps
hostapd.wlan0-1
i2c_exp
iuserinfo
log
network
network.device
network.interface
network.interface.loopback
network.interface.wlan
network.interface.wwan
network.wireless
onion
onion-console
service
session
system
uci
root@Omega-148F:/dev#
```

If you don't see `gps` listed, you'll need to restart your `rpcd` service in order to refresh the list:

```
/etc/init.d/rpcd restart
```

If this doesn't work, try reinstalling the `ogps` package by running the following commands:

```
opkg remove ogps
opkg update
opkg install ogps
```

#### 4. Download and Install the Project Software

The code for this project is all done and can be found in Onion's [wifi-hotspot-scanner repo](#) on GitHub. Use [git to download the code to your Omega](#): navigate to the `/root` directory, and clone the GitHub repo:

```
cd /root
git clone https://github.com/OnionIoT/wifi-hotspot-scanner.git
```

#### 5. Running the Project on Boot

Next we'll setup the Omega to automatically run the scanner when it turns on. Edit the `/etc/rc.local` file and add the following line above `exit 0`:

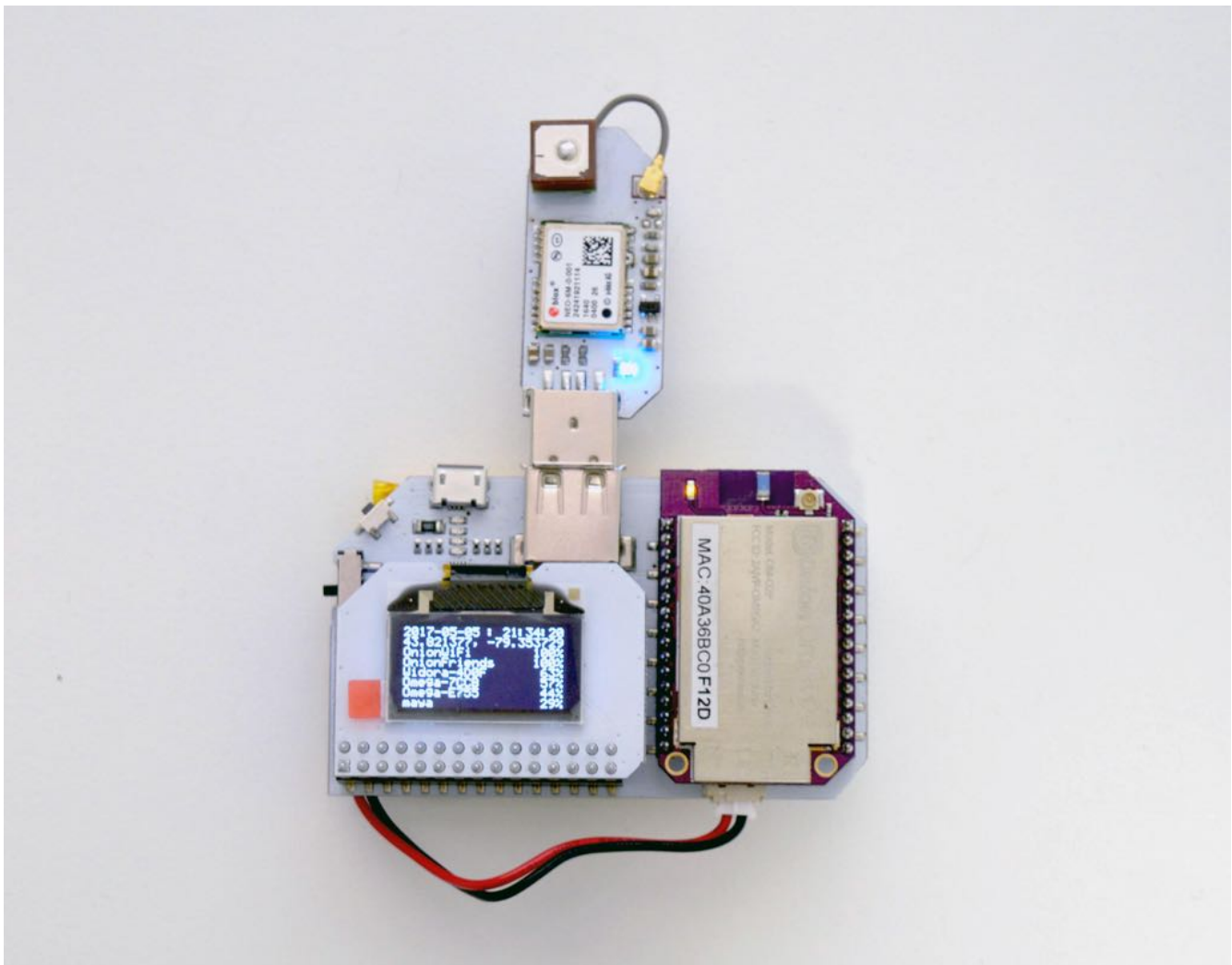
```
python /root/wifi-hotspot-scanner/main.py &
```

This way, when you flip the power switch, the Omega will run the code in the background after it completes the initialization process.

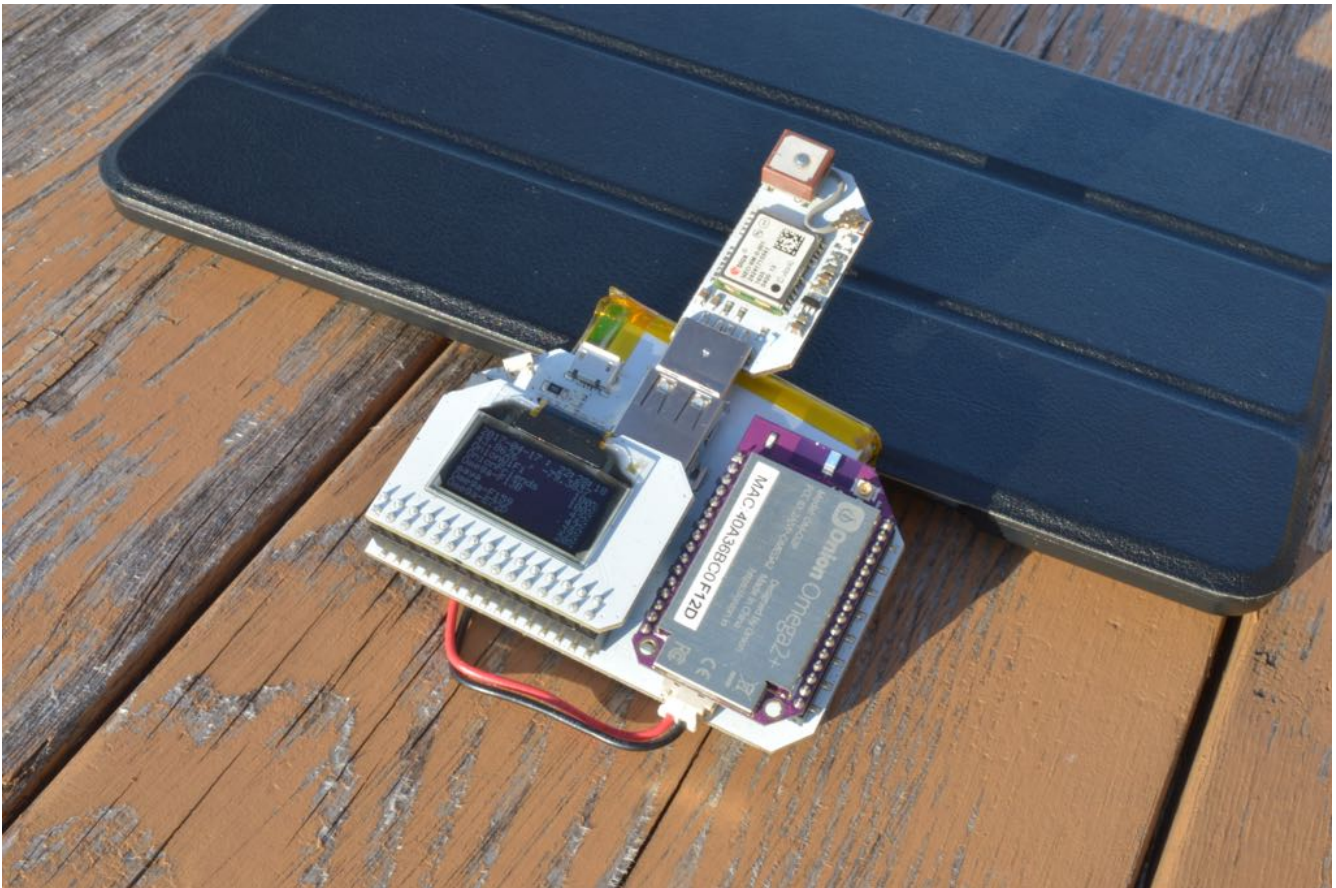
## 6. Using the WiFi Scanner

Here's the fun part! Press the reset button and the Omega will run the program.

If the GPS Expansion is able to lock onto a satellite signal, you'll see the time, the GPS coordinates, and the 6 WiFi networks with the strongest signal available nearby.



The Omega will then save data about all of the discovered networks to a file called `wifiData.csv` in the project directory. You can then import this into a spreadsheet or navigation program for mapping later!



### Unable to Lock Signal

If the GPS Expansion cannot lock onto a satellite, you'll see an error message on the OLED. The program will try again in a few seconds.

### Saved Data

Assuming the project code was downloaded to the `/root` directory, the collected wifi data will be saved to: `/root/wifi-hotspot-scanner/wifiData.csv`. It is a Comma Separated Value (CSV) file and can be opened with any spreadsheet program. It stores data about the surrounding networks for every single scan:

```

root@Omega-F12D:~/wifi-hotspot-scanner# cat wifiData.csv
date,latitude,longitude,ssid,bssid,encryption,signalStrength
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,d0:03:4b:60:79:c6,WPA1PSKWPA2PSK,0
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,68:b6:fc:a7:53:88,WPA2PSK,0
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,c8:91:f9:c1:09:e6,WPA2PSK,0
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,b8:62:1f:53:da:47,WPA2PSK,13
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,a4:6c:2a:a8:56:c1,WPA2PSK,0
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,Omega-7CCB,40:a3:6b:c0:7c:cb,WPA2PSK,50
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,OnionWiFi,00:25:9c:13:9b:6b,WPA2PSK,100
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,Omega-E755,40:a3:6b:c0:e7:55,WPA2PSK,47
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,OnionFriends,02:25:9c:13:9b:6b,WPA2PSK,100
2017-05-05 21:34:12.107230,43.821182,-79.353691,██████,i,90:72:40:21:29:4c,WPA1PSKWPA2PSK,29
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,d0:03:4b:60:79:c6,WPA1PSKWPA2PSK,0
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,68:b6:fc:a7:53:88,WPA2PSK,0
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,c8:91:f9:c1:09:e6,WPA2PSK,0
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,b8:62:1f:53:da:47,WPA2PSK,13
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,Omega-7CCB,40:a3:6b:c0:7c:cb,WPA2PSK,57
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,OnionWiFi,00:25:9c:13:9b:6b,WPA2PSK,100
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,a4:6c:2a:a8:56:c0,WPA2PSK,0
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,Omega-E755,40:a3:6b:c0:e7:55,WPA2PSK,44
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,OnionFriends,02:25:9c:13:9b:6b,WPA2PSK,100
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,i,90:72:40:21:29:4c,WPA1PSKWPA2PSK,29
2017-05-05 21:34:21.661293,43.821377,-79.353729,██████,a4:6c:2a:a8:56:c1,WPA2PSK,0

```

This data can be used in a variety of creative ways: creating a map of your neighbourhood that shows the strength of the local WiFi networks, creating a database of open networks around the city, the sky is the limit.

## Code Highlight

The `ubus` system utility is a key part of the firmware on which the Omega is based. It allows you to call services and functions on the Omega as if you were sending data to a web API. The basic syntax goes like this:

```
ubus call (service) (function) '{{(JSON parameters)}}'
```

The WiFi and GPS scanning functions are available as `ubus` functions so that they can be called by any program.

You can see how they work in the `ubusHelper.py` module:

```

# basics of running a command
# returns a dict as ubus functions return json objects
def runCommand(command):
    output, err = shellHelper.runCommand(command)
    responseDict = json.loads(output)
    return responseDict

# often used commands
# add more if you need
def call(args):
    command = ["ubus", "call"]
    command.extend(args)
    return runCommand(command)

```



and the `helpers.py` module:

```
# scan wifi networks in range
# returns a list of wifi dictionaries
def scanWifi():
    device = json.dumps({"device": "ra0"})
    args = ["onion", "wifi-scan", device]
    return ubus.call(args)["results"]

# read the GPS expansion
# returns a dictionary with gps info
def readGps():
    args = ["gps", "info"]
    response = ubus.call(args)

    # check if the GPS is locked
    if "signal" in response and response["signal"] == False:
        return False
    # else return the data
    return response
```

In essence, the `scanWifi()` function above runs the following command:

```
ubus call onion wifi-scan '{"device":"ra0}"'
```

And the `readGps()` function runs this command:

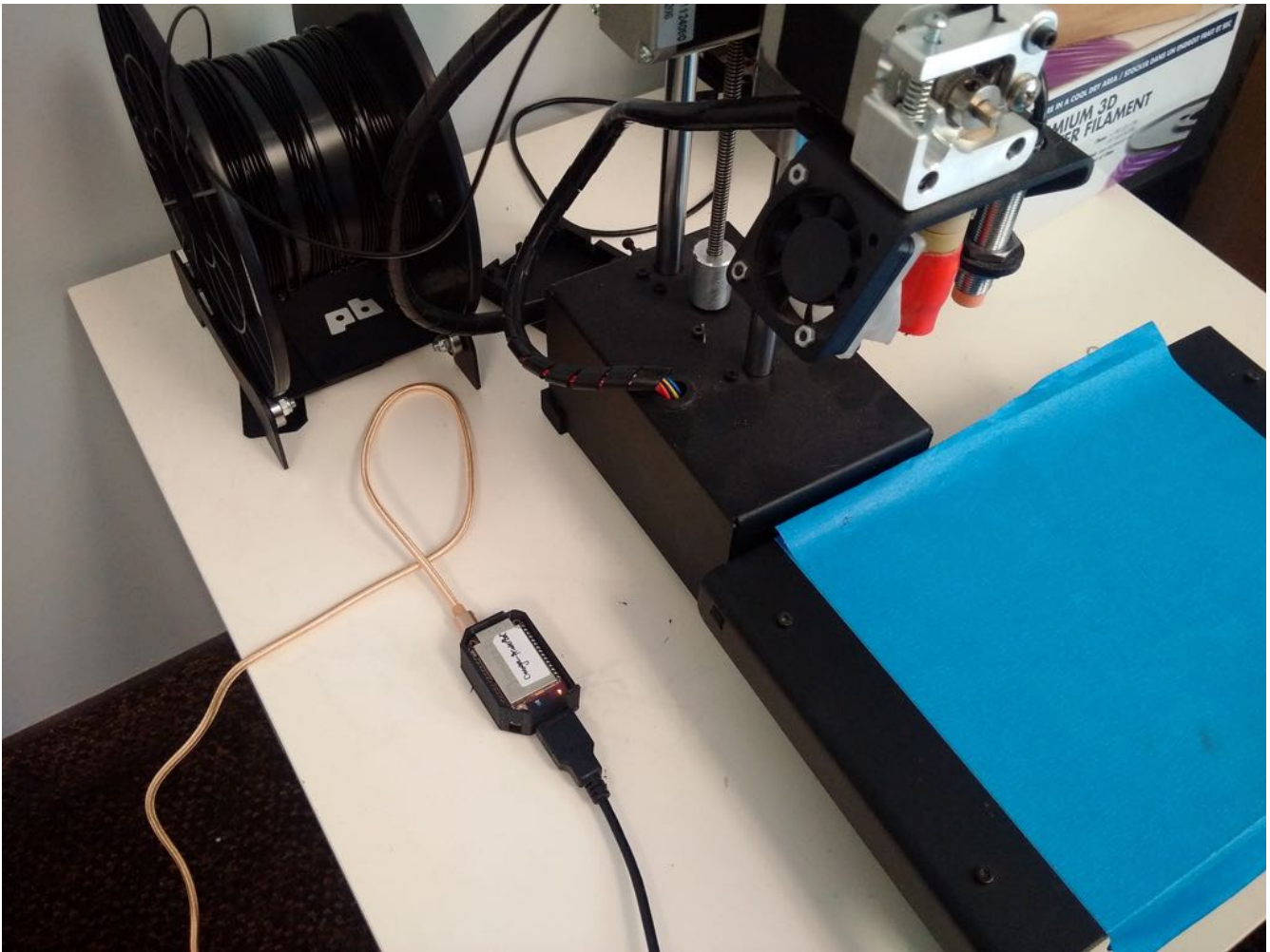
```
ubus call gps info
```

Try running these two commands on your Omega's command line by hand and take note of the output.

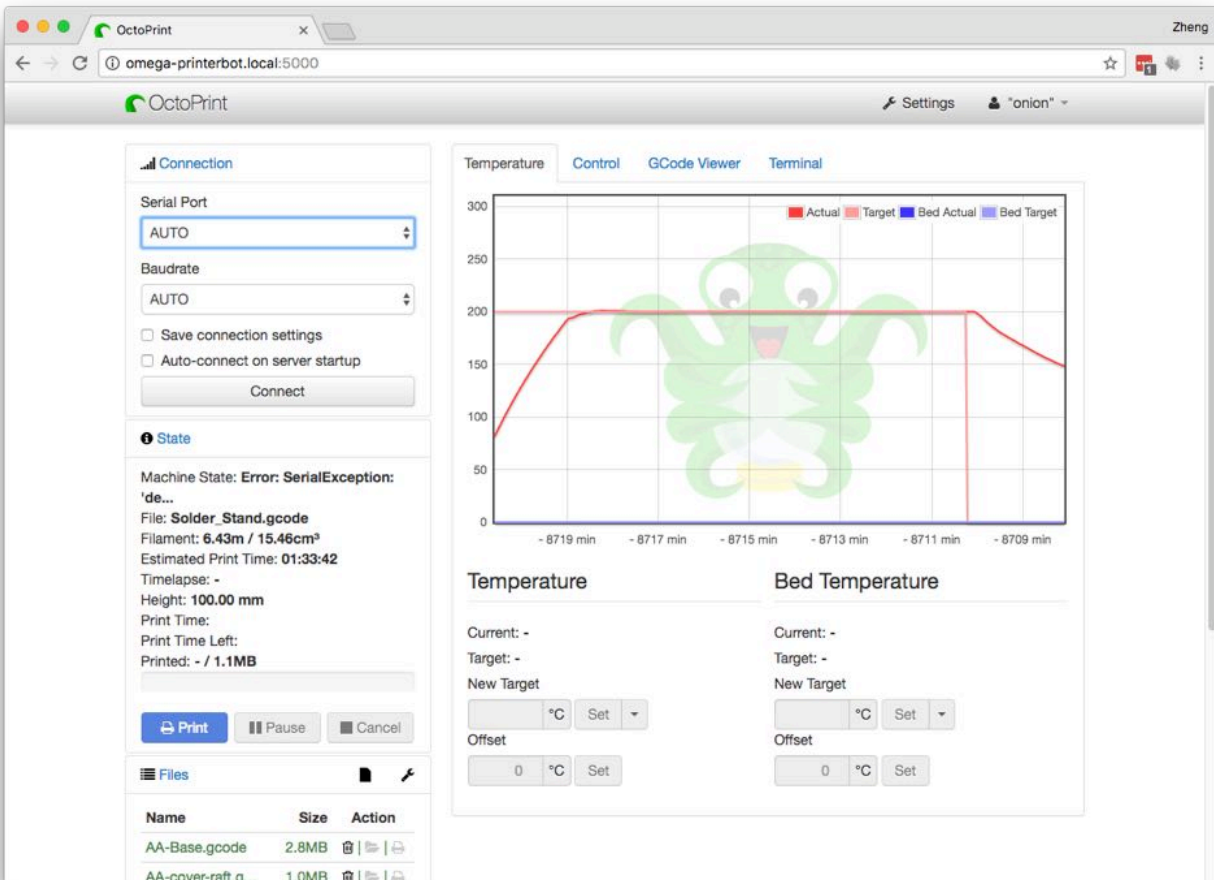
## OctoPrint 3D Printing Server

This project allows you to use the Omega to wirelessly control your 3D printer.

Instead of having to connect a computer directly to our 3D Printer, we can have the Omega run Octoprint, a 3D printing server that will control the printer.



Octoprint serves up a web interface that we can access from any device in our Local Area Network that allows you to control and monitor every aspect of your 3D printer and your printing jobs right from within your browser.



## Overview

**Skill Level:** Intermediate

**Time Required:** 30 minutes

To get our printer server up and running, we'll have to use some packages from the LEDE repository. Additionally, we're going to build a custom version of OctoPrint to fix some compatibility issues.

## Ingredients

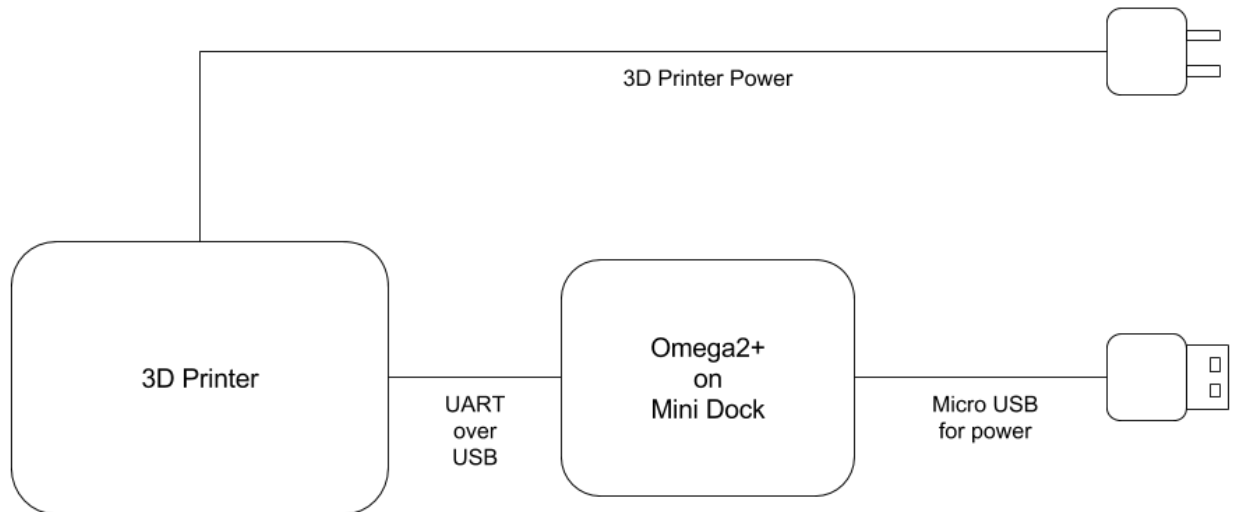
- Onion Omega2+
- Any Onion Dock with a USB host connector: [Expansion Dock](#), [Power Dock](#), [Mini Dock](#), [Arduino Dock 2](#)
  - We liked the [Mini Dock](#) for this project since it's so compact
- 3D Printer that is [supported by Octoprint](#)
- [Micro SD card](#)
- USB cable to connect the Omega and 3D printer

## Step-by-Step

Follow these instructions to turn your 3D printer wireless!

### 1. Set up the hardware

Here's a handy connection diagram of the way the pieces are put together:



First, connect the printer to the Omega with a USB cable. Next, connect the 3D printer to power. Finally, power on the Omega and we'll be good to go.

### 2. Prepare the Omega

If you need, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 3. Expand Omega storage with SD card

Octoprint is a rather large program and will not fit on the on-board storage of the Omega2 or Omega2+. To remedy this, we will setup the Omega so that it boots the operating system from external storage.

The below procedure is a condensed implementation of our [Booting from External Storage guide](#). We recommend taking a look at the full guide before proceeding.

First, plug a micro SD card into the Omega2+, it will show up as `/dev/mmcblk0` and `/dev/mmcblk0p1`

To make it useable, we'll have to format the SD card to EXT4. **Warning: This will delete all data on the card!**

Install disk utilities:

```
opkg update
opkg install fdisk e2fsprogs block-mount
```

Now unmount and format the SD card:

```
umount /dev/mmcblk0p1
mkfs.ext4 /dev/mmcblk0p1
```

Mount the freshly formatted SD card in a more legible location:

```
umount /dev/mmcblk0p1
mkdir /mnt/SD
mount /dev/mmcblk0p1 /mnt/SD
```

Copy current `/overlay` directory:

```
tar -C /overlay -cvf - . | tar -C /mnt/SD/ -xf -
umount /mnt/SD
```

Now we'll set up the `/overlay` directory to automount on startup. First, we'll copy our current setup to the filesystem record:

```
block detect > /etc/config/fstab
```

Next, we edit `/etc/config/fstab` to tell the Omega to mount the SD card as the `/overlay` partition, ie run the Omega's operating system from the SD card's storageL

- Change option `target '/mnt/mmcblk0p1'` to option `target '/overlay'`
- Change option `enabled '0'` to option `enabled '1'`

Reboot for our changes to take effect.

After the Omega starts again, you can verify that you are indeed running the operating system from the SD card by running `df -h`.

You should see something like this:

Filesystem	Size	Used	Available	Use%	Mounted on
.					
.					
.					
/dev/mmcblk0p1	7201.9M	32.5M	6784.3M	0%	/overlay

#### 4. Build Octoprint

Octoprint requires some packages that are not in the Onion package repository, so we'll pull them from the LEDE repo instead.

To do so, we need to edit `/etc/opkg/distfeeds.conf` and uncomment this line:

```
src/gz reboot_packages http://downloads.lede-project.org/snapshots/packages/mipsel_24kc/packages
```

Once done, we can install the packages we need:

```
opkg update
opkg install python python-pip unzip
pip install --upgrade setuptools
```

Now we need to expand `/tmp` folder on the Omega:

```
mkdir /overlay/tmp
rm -rf /overlay/tmp/*
cp -a /tmp/* /overlay/tmp/
umount /tmp
[ $? -ne 0 ] && {
umount -l /tmp
}
mount /overlay/tmp/ /tmp
```

Once we have the packages and enough space, we can download and build Octoprint:

```
cd /root
wget https://github.com/foosel/OctoPrint/archive/1.0.0.zip
unzip 1.0.0.zip
cd OctoPrint-1.0.0
pip install -r requirements.txt
```

Note: Currently we've only able to successfully get Octoprint 1.0.0 to compile and work

Now we need to edit a few files to resolve some compatibility issues.

Unicode characters cause problems with Python on the Omega, so we have to replace one of the author's names to be only ASCII characters (Sorry Gina Häußge)

```
s/Häußge/H\./g
sed -i 's/Häußge/H\./g' /root/OctoPrint-1.0.0/src/octoprint/util/comm.py
sed -i 's/Häußge/H\./g' /root/OctoPrint-1.0.0/src/octoprint/util/virtual.py
```

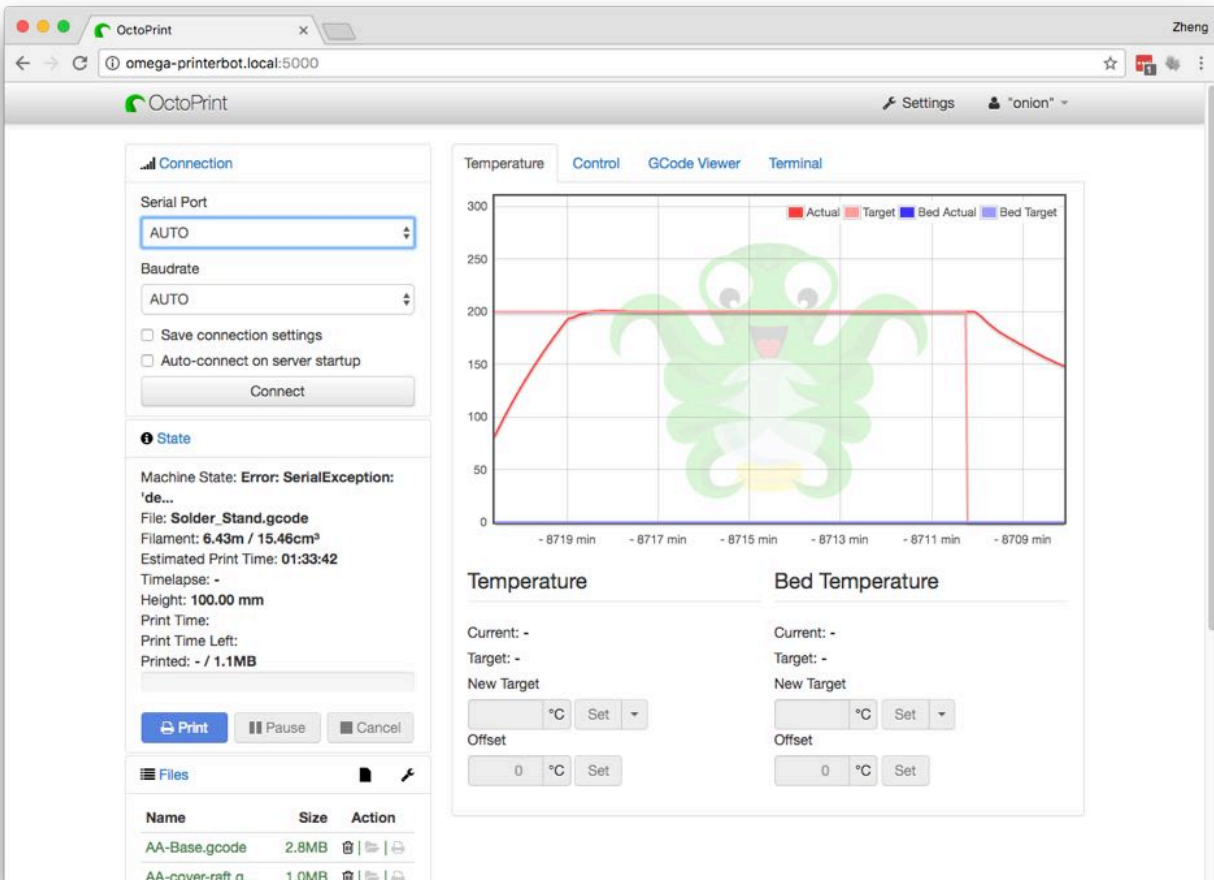
The Omega only has a single user, and that is the root user. Octoprint does not let you run as root, so we have to suppress that

```
sed -i 's/exit("You should not run OctoPrint as root!)/pass/g' /root/OctoPrint-1.0.0/src/octopri
```

That's it! Now we can test drive our Octoprint Installation:

```
./run
```

Open a browser, connect to port 5000 on your Omega. We renamed our Omega to `omega-printerbot` so the address we use is `http://omega-printerbot.local:5000:`



## 5. Auto start Octoprint at startup

Move OctoPrint to a proper location:

```
mv /root/OctoPrint-1.0.0 /usr/share/OctoPrint
```

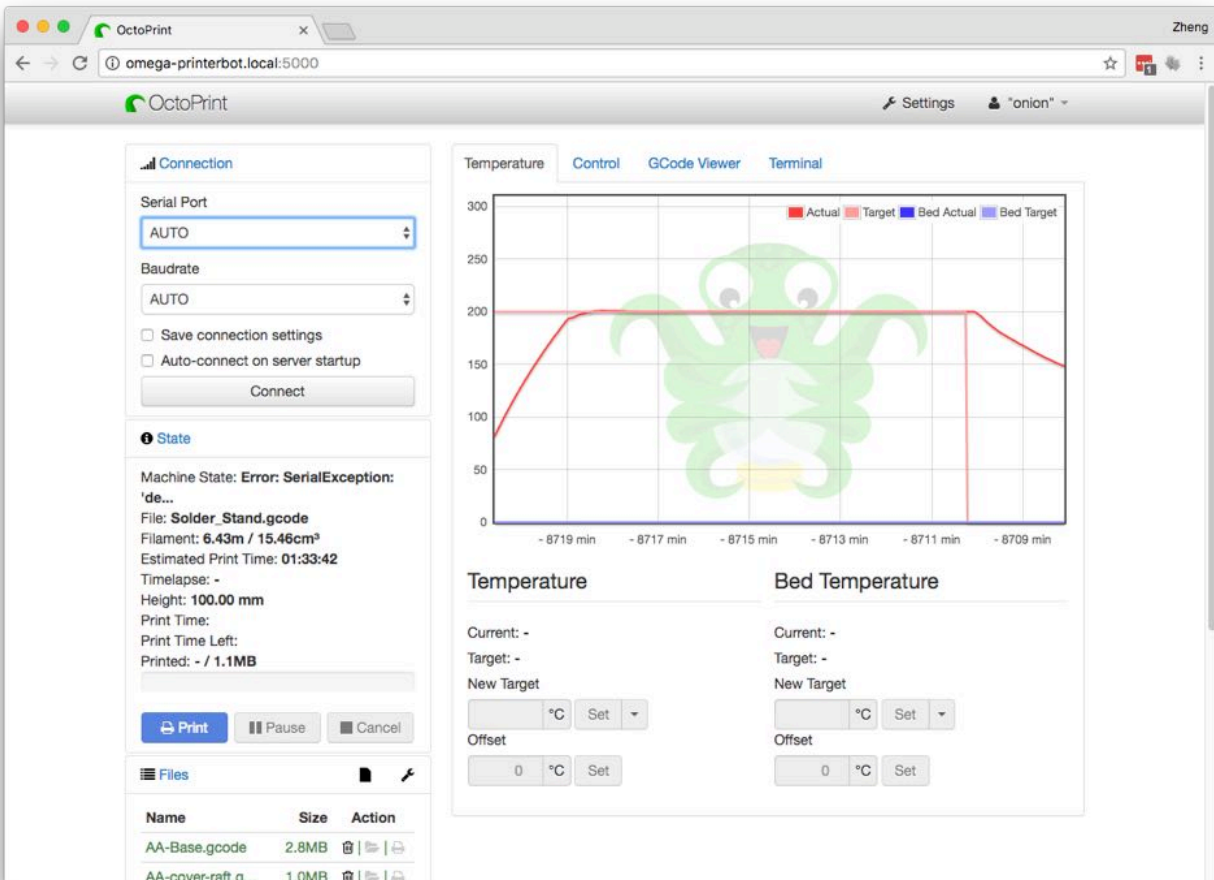
Make a symlink to the start up binary:

```
ln -s /usr/share/OctoPrint/run /usr/bin/octoprint
```

Edit `/etc/rc.local` add the following before `exit 0`:

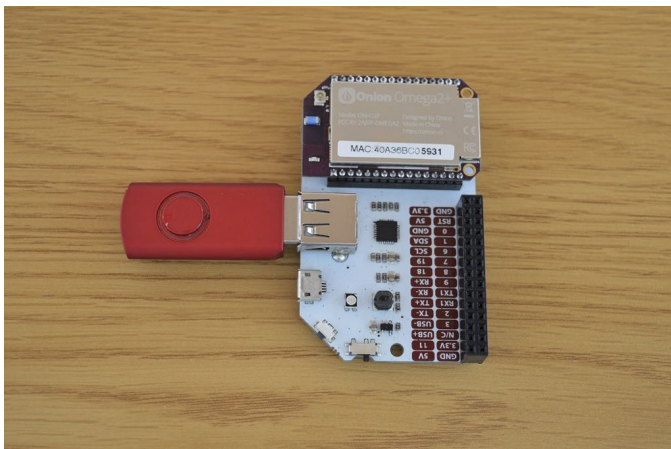
```
octoprint &
```

And bam, we're ready to control our 3D printer wirelessly from our local network!



## Mobile Network File Server

The Omega's firmware has packages available for a file sharing server program called Samba. By plugging in a USB storage device, you can turn your Omega into a mobile network file server!





## Overview

**Skill Level:** Intermediate

**Time Required:** 20 minutes

This project will walk through how to set up an external storage device, configure a Samba server on the Omega, and how to access it from other operating systems.

## Sample Configuration files

The Onion [samba-server-config GitHub repository](#) contains reference configuration files in case you need to troubleshoot your setup.

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any [Onion Dock](#) to power the Omega
  - We like the [Mini Dock](#) if you plan to keep it one place.
  - Use the [Power Dock](#) if you plan to make this a truly portable network storage device.
- A [USB storage device](#) or [Micro SD card](#) (for Omega2+)

## Step-by-Step

Let's turn your Omega into a portable network attached storage, or NAS for short!

### 1. Setup your Omega

You'll need an Omega2 ready to go, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

If you need to hook up the Omega to a new network, [connect to the command line](#) and use the `wifisetup` utility:

```
root@Omega-0104:/# wifisetup
Onion Omega Wifi Setup
```

Select from the following:

- 1) Scan for Wifi networks
- 2) Type network info
- q) Exit

Selection:

Follow the instructions to scan for WiFi networks and connect to your router's network.

## 2. Set up your Storage Device

You can share any directory on your Omega through Samba. For this project, we'll assume you have a USB storage device or microSD. Both of these devices will be automatically mounted at `/tmp/mounts`.

Usually, a USB device is mounted under `/tmp/mounts/USB-A1`, but you can make sure by calling:

```
ls /tmp/mounts
```

The directories listed should all correspond with auto-mounted devices.

Copy down the **full path** to your storage device (`/tmp/mounts/USB-A1` worked for us) - we'll need this to configure Samba.

For a detailed walk-through on how to use storage devices, take a look at the guide to [USB Storage](#) and [MicroSD Cards](#) on the Onion Docs.

## 3. Install the Required Software

We'll need Samba for this, naturally. Samba's name occasionally changes with versioning changes, to check what it is, we can do:

```
opkg update
opkg list | grep samba
opkg install samba##-server
```

At time of writing, Samba is at version 36, so we can install it like so:

```
opkg install samba36-server
```

## 4. Find the Omega's WiFi card

By default, Samba does not listen to Omega's WiFi for incoming access requests. We'll need to let Samba know which interface it should listen on to get requests for its data.

The Omega communicates with other networks through the `apcli0` interface. Note that down for later!

If you'd like to see for yourself, use `ifconfig` to list all the interfaces available. Here's an example of that the output will look like:

```
root@Omega-E755:~# ifconfig
apcli0  Link encap:Ethernet  HWaddr AA:AA:AA:AA:AA:AA
        inet addr:192.168.1.109  Bcast:192.168.1.255  Mask:255.255.255.0
        inet6 addr: fe80::40a3:6bff:fe00:e755/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:13787 errors:0 dropped:3 overruns:0 frame:0
        TX packets:5953 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:3197266 (3.0 MiB)  TX bytes:602257 (588.1 KiB)

br-wlan  Link encap:Ethernet  HWaddr AA:AA:AA:AA:AA:AA
        inet addr:192.168.3.1  Bcast:192.168.3.255  Mask:255.255.255.0
        inet6 addr: fe80::42a3:6bff:fec0:e757/64 Scope:Link
        inet6 addr: fd1d:48c4:7633::1/60 Scope:Global
```

```

UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:456 errors:0 dropped:0 overruns:0 frame:0
TX packets:746 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:58973 (57.5 KiB)  TX bytes:88296 (86.2 KiB)
...
...

```

We see `apcli0` is the interface with the most RX bytes and TX bytes, meaning it's being doing the most communication. Additionally, it has an IP Address (see the `inet addr`) that corresponds to a LAN - starting with `192.168.1`.

#### 4. Configure Samba

For Samba to start sharing our folders, it needs to know where, how, and who. It gets all these details from configuration files. Specifically, `/etc/config/samba` and `/etc/samba/smb.conf.template`.

First let's open up `/etc/config/samba` with our editor. It should look a little like this:

```

config samba
    option 'name'                'Lede'
    option 'workgroup'           'WORKGROUP'
    option 'description'         'Lede'
    option 'homes'               '1'

```

These fields are all customizable options that change how the Samba server behaves.

We recommend changing `name` option to your Omega's `Omega-ABCD` name (where `ABCD` are the bolded numbers on your Omega's cover) for easy recognition. The other options should be left as-is, and the `description` option can be changed to something helpful.

Next, we'll add in a line like this:

```

    option 'interface' 'apcli0'

```

This tells Samba it should listen on the `apcli0` interface, this sets up Samba to accept connections. We just need to let it know the location of our shared folder.

To declare a new shared folder, we'll append a block to the end like this:

```

config 'usbshare'
    option 'name'                'usb'
    option 'path'                '/tmp/mounts/USB-A1'
    option 'users'               'root'
    option 'read_only'           'no'
    option 'guest_ok'            'no'

```

The main configurations that are needed are the `name`, `path`, and `users`:

- The `name` will be the name that appears devices accessing it.
- The `path` is the directory (or file) you want to share.
- Setting `read_only` to `no` will mean that anyone accessing the shared folder can change its contents
- While having `guest_ok` set to `no` means access is only granted after authentication.

The `/etc/config/samba` file is a Universal Configuration Interface (UCI) file, LEDE uses it to simplify configuration of system services.

Next, we'll have to fiddle with `/etc/samba/smb.conf.template`. Opening it with our editor will greet us with this:

```
[global]
    ...blah
    ...blah
    enable core files = no
    guest ok = yes
    invalid users = root
    load printers = no
    ...blah
```

We'll have to change one thing here since we'll be accessing the shared folder/s as root:

```
invalid users = #
```

And now Samba is ready to go!

## More Options

Samba has a ton of configuration features that allow you to micromanage who gets access to what. For a bit more detail on how Samba works, LEDE has an excellent [guide on configuring Samba](#). For a lot more detail on how Samba works, [Using Samba](#) is available to explain the nitty-gritty details.

## 5. Users and passwords

To allow access, we'll have to set up passwords any user we specified - since we only used root, we can call `smbpasswd` like so:

```
smbpasswd -a root
```

This utility will create a `root` Samba account associated with the `root` account of the operating system. You'll be prompted to enter a new password for use with Samba.

## 6. Applying our changes

Samba needs to be restarted for our configuration to apply. We can do this with:

```
/etc/init.d/samba restart
```

Now we're ready to check out our file share!

## 7. Access your shared folder

Of course once the server is running, we'll have to actually access it somehow.

## Windows

On Windows, a Samba share can be found by opening a file explorer and going to 'Network'. The `name` we specified in `/etc/config/samba` should appear as a network location, and the folder we shared listed inside it. You'll be prompted for login details, input `root` and the password you selected in step 5, and voila!

For a more detailed tutorial on connected to a Samba-shared network drive, take a look at [this tutorial](#).

## Linux

Modern Linux distros have Samba clients well integrated in the file explorer, and the process is very similar to windows. Open up file explorer, navigate to the Network root, and find your Omega to access.

Some desktop environments have it grouped under a ‘Samba Shares’ folder or the like inside the Network root.

For a more detailed tutorial on connected to a Samba-shared network drive, take a look at [this tutorial]([https://access.redhat.com/documentation/en-US/Red\\_Hat\\_Enterprise\\_Linux/3/html/System\\_Administration\\_Guide/s1-samba-connect-share.html](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/3/html/System_Administration_Guide/s1-samba-connect-share.html))

## Mac OS X

On OS X, connecting to a Samba share can be done using Finder. Open a Finder window and hit Command+K, in the window that pops up, for Server Address type in `smb://omega-ABCD.local` where ABCD is your Omega’s unique identifier. Connect as a registered user and select the volume to which you would like to connect.

After that, it will will just be like any other directory on your computer.

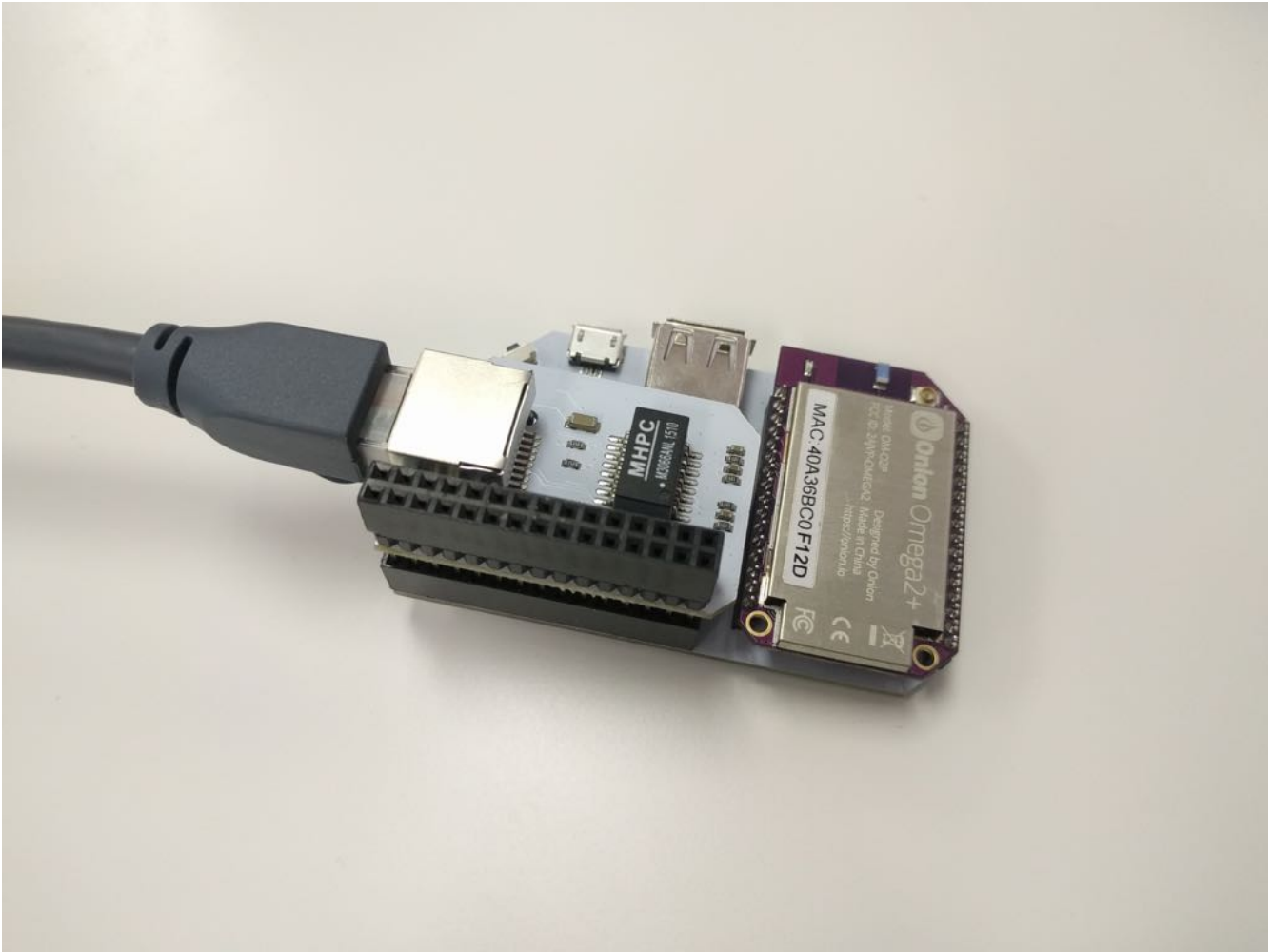
For a more detailed tutorial on connected to a Samba-shared network drive, take a look at [this tutorial](#)

## Omega WiFi Router

A router is a device that connects multiple devices on a wired or wireless network. They are very widely used with modems to allow multiple devices to connect to the Internet through the single connection provided by the modem.

We’re going to use the Omega as a wireless router that:

- Received Internet connectivity through a wired Ethernet connection
- Broadcasts a WiFi network Access Point (AP)
- Shares network access from the Ethernet network to the WiFi AP network



The Ethernet Expansion is required to give your Omega access to an Ethernet port. By using the Ethernet Expansion, we can turn our Omega into a low-cost yet effective router.



## Overview

**Skill Level:** Intermediate

**Time Required:** 10 minutes

What we are going to do is to first enable the Omega's Ethernet connection, stop the Omega from connecting to other, existing WiFi networks, then enable routing network traffic from the Omega's AP to the Internet through the Ethernet connection.

## Sample Configuration files

The Onion [omega-as-router Github repository](#) contains reference configuration files in case you need to troubleshoot your setup.

Please note that there are some placeholders such as RouterPassword and somewifissid. Make sure to copy only the relevant parts!

## Default Configuration Files

If you ever want to revert your configuration to the original, we have a complete set of default configuration files from a factory-fresh Omega2 [in the uci-default-configs repo](#) on GitHub.

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
  - We prefer the Expansion Dock for this project since it enables [access to the command line through serial](#) even when there's no network connectivity
- Onion [Ethernet Expansion](#)

## Step-by-Step

Here's how to turn your Omega into a wireless router!

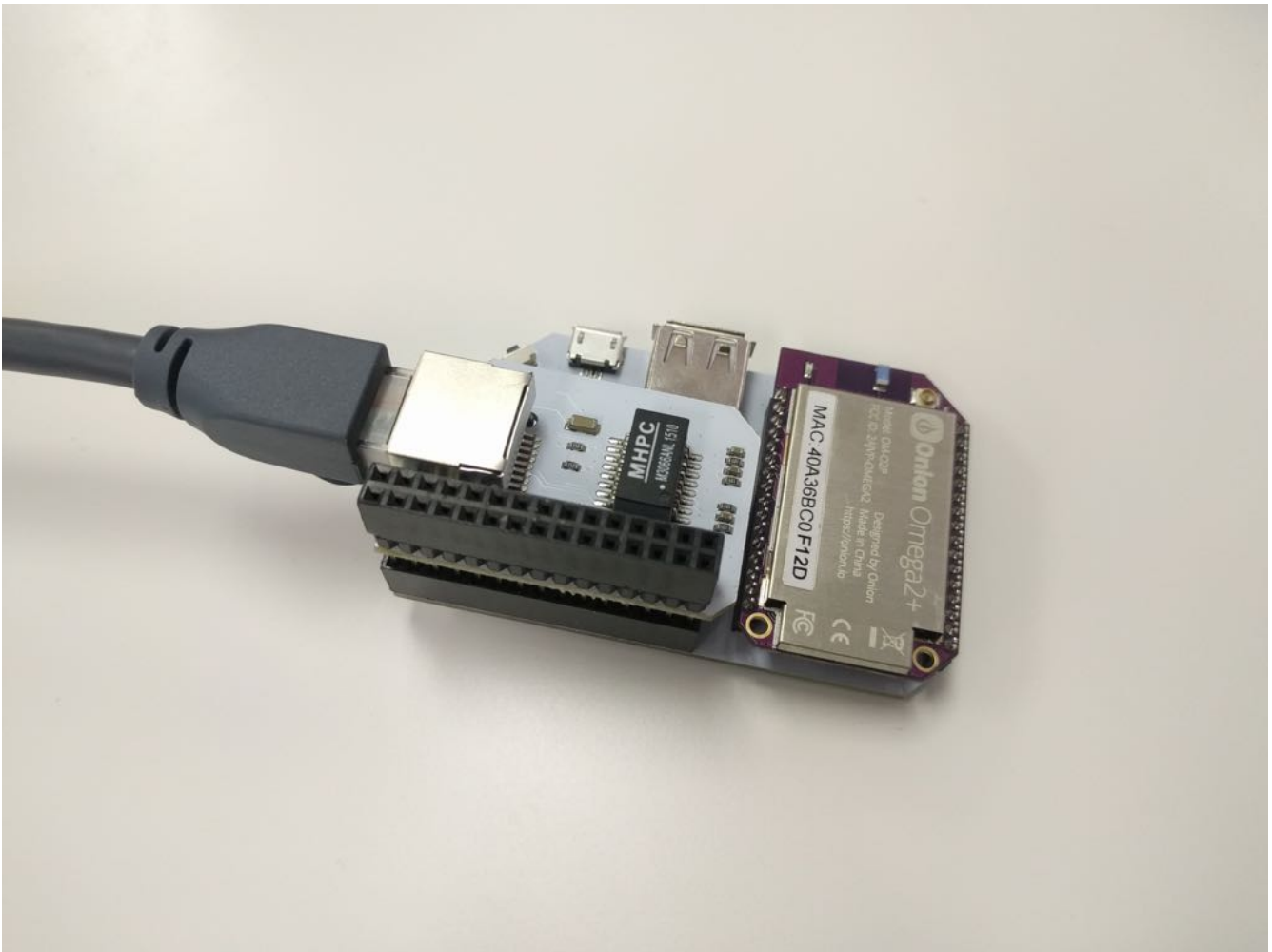
### 1. Prepare

First let's get the Omega ready to go. if you haven't already, complete the [First Time Setup Guide](#) to connect your Omega to WiFi and update to the latest firmware.

### 2. Setup the Hardware

Connect your Ethernet Expansion to the Expansion dock, and plug in the Ethernet cable, as shown below:





Connect the other end of the cable to your modem, or other device that is providing internet connectivity.

### 3. Setup the Omega

The next step is to stop the Omega from connecting to other, existing WiFi networks, as it will be using the Ethernet Expansion to access the Internet instead.

We're going to be restarting the WiFi on the Omega a few times, breaking any SSH connections in the process. To avoid this, you can try using a serial connection with your Omega. For more information, please refer to this [guide on connecting to your Omega](#).

On the Omega's command line, enter the following commands:

```
uci set wireless.@wifi-iface[0].ApCliEnable=0
uci commit wireless
```

Restart the WiFi network to apply your saved changes:

```
wifi
```

## 4. Changing the WiFi AP Configuration

Set the SSID and password of the router's WiFi network with the following commands, substituting `OmegaRouter` and `RouterPassword` with values of your choice:

```
uci set wireless.@wifi-iface[0].ssid=OmegaRouter
uci set wireless.@wifi-iface[0].key=RouterPassword
uci commit
```

### Changing the Encryption Type

If you wish to keep the default encryption type, WPA2 (psk2), which we strongly recommend, you can skip this step.

However, if you do wish to change the encryption type, find the type you want in the [UCI wireless encryption list](#), then substitute it into `YourEncryptionType` and run:

```
uci set wireless.@wifi-iface[0].encryption=YourEncryptionType
uci commit
```

Please keep in mind that 1st generation WPA is **not secure** and that WEP keys have to be a certain length in order to work properly.

### Restarting the WiFi

Run the following command to restart the WiFi network and apply your settings:

```
wifi
```

## 5. Enable Ethernet Connectivity

Enable the Ethernet interface, `eth0`, by running:

```
uci set network.wan.ifname='eth0'
uci set network.wan.hostname='OnionOmega'
uci commit
```

Then restart the network service:

```
/etc/init.d/network restart
```

This will allow the Omega to connect to the Internet via the Ethernet port.

## 6. Enabling Packet Routing in the Firewall

Now we need to enable sharing of network access between the ethernet network and the WiFi AP. Open the `/etc/config/firewall` file using `vi` and find the block that looks like the following:

```
config zone
    option name 'wan'
    option output 'ACCEPT'
    option forward 'REJECT'
    option masq '1'
```

```
option mtu_fix '1'  
option network 'wwan'  
option input 'ACCEPT'
```

and do the following:

- Change option forward 'REJECT' to option forward 'ACCEPT'
- Change option network 'wwan' to list network 'wwan'
- Add list network 'wan' after the list network 'wwan' line

The block should now look like this:

```
config zone  
  option name 'wan'  
  option output 'ACCEPT'  
  option forward 'ACCEPT'  
  option masq '1'  
  option mtu_fix '1'  
  list network 'wwan'  
  list network 'wan'  
  option input 'ACCEPT'
```

Now restart the firewall by running:

```
/etc/init.d/firewall restart
```

What we've told the firewall to do with the above configuration is to allow traffic passing between the `wwan` interface (the WiFi network) and the `wan` interface (the wired ethernet network).

## 7. Using the Omega Router

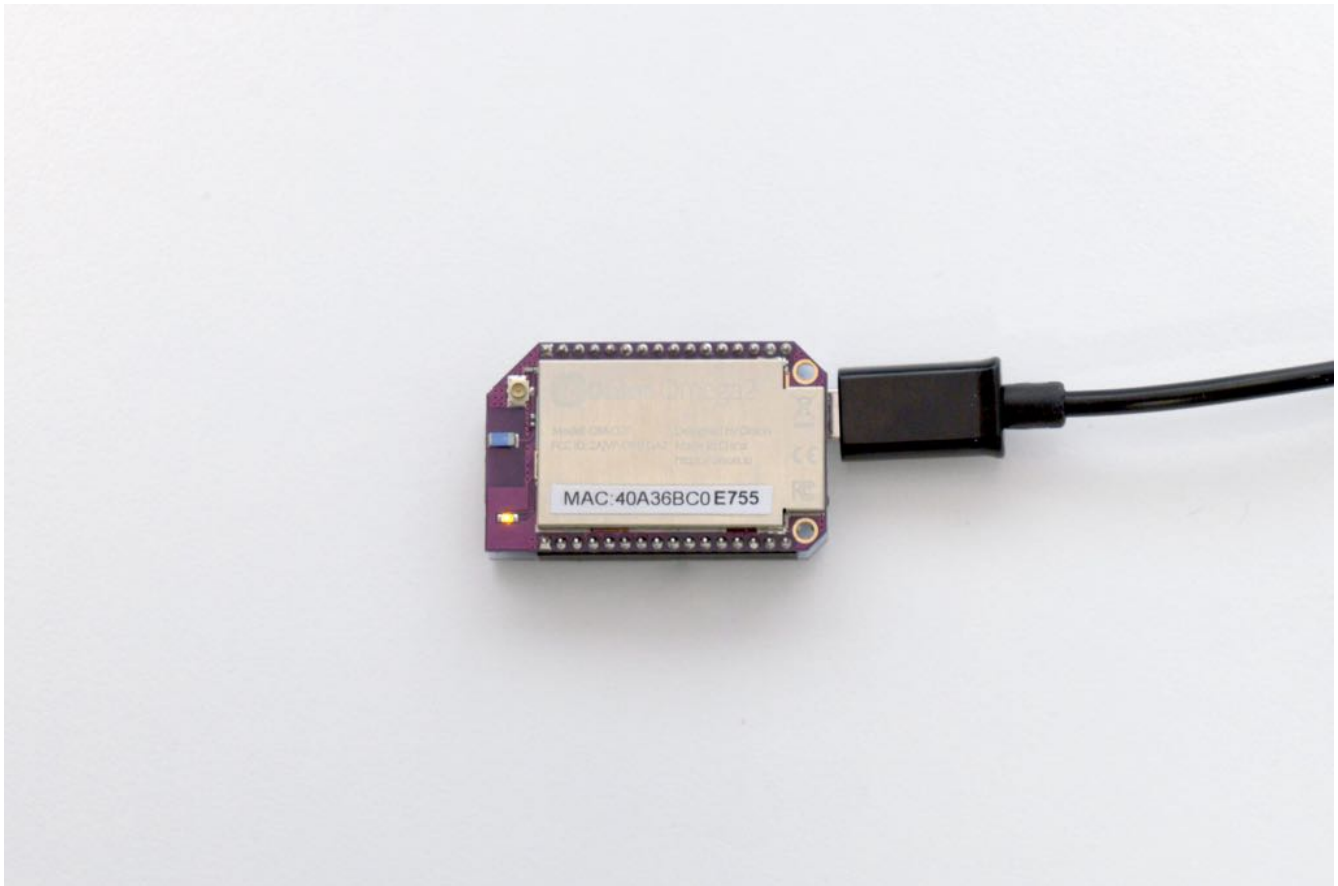
And we're ready! To use the Omega Router, you simply need to connect your computer or your smartphone/tablet to the WiFi network that you configured in Step 4, and your devices should be able to access the Internet via the Omega.



## Omega WiFi Range Extender

Do you have some places in your home where your WiFi network is slow? A WiFi range extender is a device that can increase the effective range of a router by being placed closer to the end user and acting as a relay to the router.

The Omega's powerful WiFi capabilities and incredibly small footprint allow it to be effective as a WiFi range extender.



Even though the Omega has only one physical WiFi interface, you can create two virtual interfaces and have the Omega relay the packets back and forth between the two interfaces. This allows you to set up the Omega as a WiFi range extender that relays the packets between your computer/smartphone and your router. This can be very helpful if your router has a short range and the connection has problems from beyond a certain distance.

Let's get started!

## Overview

**Skill Level:** Intermediate

**Time Required:** 10 minutes

This project will turn your Omega into a WiFi Range Extender for your WiFi network.

## Sample Configuration Files

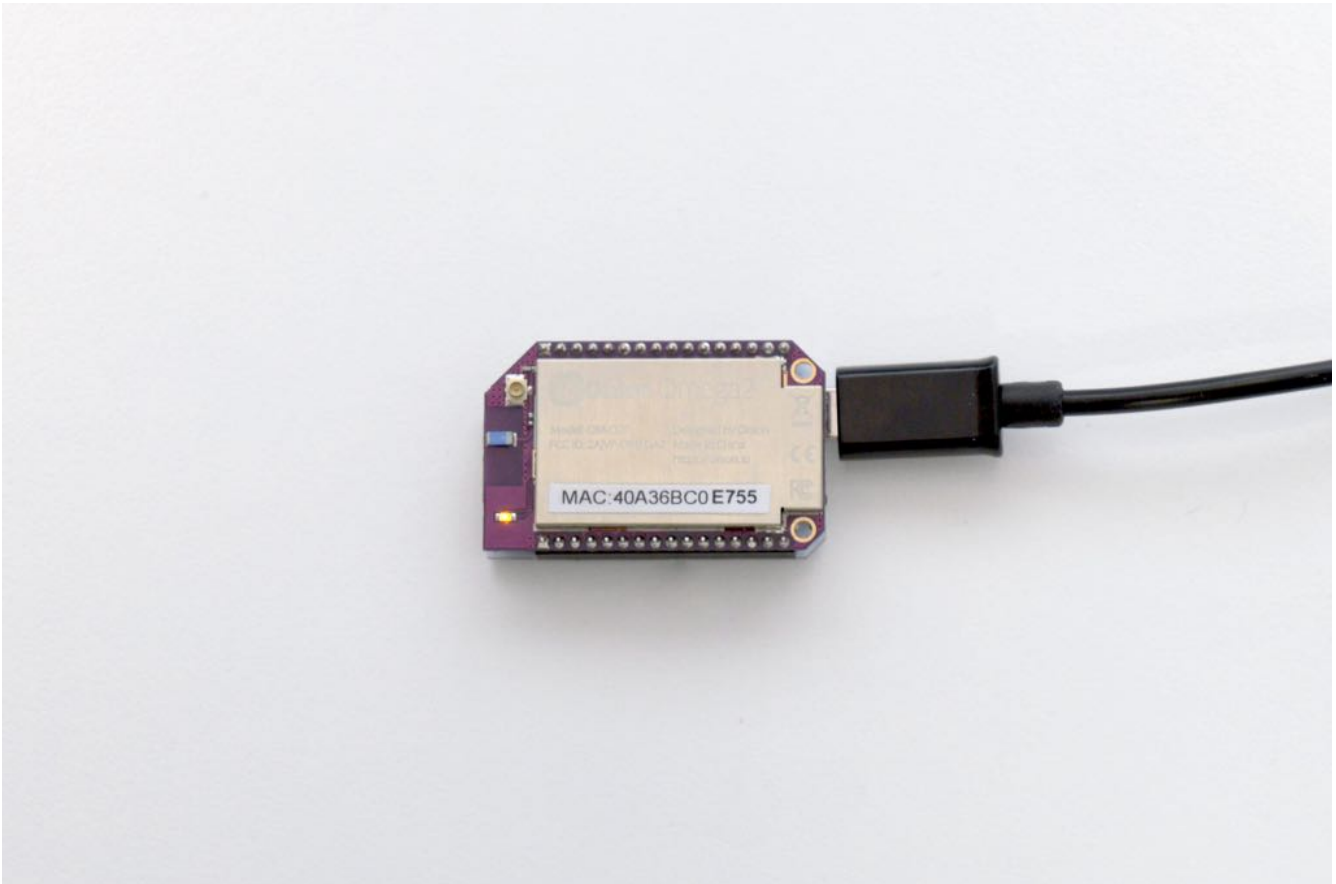
The Onion [range-extender-config Github repository](#) contains reference configuration files in case you need to troubleshoot your setup.

## Default Configuration Files

If you ever want to revert your configuration to the original, we have a complete set of default configuration files from a factory-fresh Omega2 in the [uci-default-configs repo](#) on GitHub.

## Ingredients

- Onion Omega2 or Omega2+
- Any [Onion Dock](#)
  - We really like the [Mini Dock](#) for this project because of its small footprint



## Step-by-Step

Here's how to get your Omega set up to forward packets to and from your router!

### 1. Prepare

First let's get the Omega ready to go. if you haven't already, complete the [First Time Setup Guide](#) to setup your Omega and update to the latest firmware.

## 2. Connect the Omega to the router

Now, you will need to connect the Omega to your router. To do this, connect to the command line][<https://docs.onion.io/omega2-docs/connecting-to-the-omega-terminal.html>), and use the `wifisetup` command:

```
root@Omega-0104:/# wifisetup
Union Omega Wifi Setup
```

Select from the following:

- 1) Scan for Wifi networks
- 2) Type network info
- q) Exit

Selection:

Follow the instructions to scan for WiFi and connect to your router's network.

## 3. Firewall Settings

Next, you will need to configure the Omega to route packets between it's own WiFi AP and your routers network. In other words, you're enabling the Omega to route packets from your device to the Omega to the Router, and back.

To do this, you will be editing the `/etc/config/firewall` file:

Enter the following command to edit the file:

```
vi /etc/config/firewall
```

Find the block that looks something like the following:

```
config zone
    option name 'wan'
    option output 'ACCEPT'
    option forward 'REJECT'
    option masq '1'
    option mtu_fix '1'
    option network 'wwan'
    option input 'ACCEPT'
```

and make sure that the input, output, and forwarding settings are set to ACCEPT

```
option input      ACCEPT
option output     ACCEPT
option forward    ACCEPT
```

Once you have saved and closed the file, run the following command to restart the firewall with the updated configuration:

```
/etc/init.d/firewall restart
```

What we've told the firewall to do with the above configuration is to allow traffic passing between the `wwan` network interfaces, that is, the WiFi network AP the Omega is hosting and the WiFi network to which the Omega is connected.

## Use Your Omega WiFi Range Extender

At this point, your Omega is connected to router as well as serving its own access point, and the Omega is setup to relay information back and forth between these two WiFi interfaces. This means that you can connect your computer/smartphone to the AP of your Omega, and be able to access the data coming from the router.

To use the Omega as the WiFi range extender, you would typically place the Omega somewhere between your router and your computer/smartphone. Packets will travel from your router to the Omega, and from the Omega to your computer/smartphone instead of directly from the router to your device. Effectively, extending your WiFi network's range!

Happy Surfing!

## Omega WiFi Ethernet Bridge

An Ethernet Bridge is a device that shares its WiFi network access through an Ethernet connection, kind of like an ethernet-based WiFi dongle. If the WiFi network is connected to the internet, the internet connection will be shared as well.

The Omega's flexible networking abilities and the Ethernet Expansion allow us to use the Omega as WiFi Ethernet Bridge!





As an example, this type of setup can be used to bring internet access to a desktop computer that does not have a network adapter or to a laptop with a broken wireless interface.

### Sample Configuration Files

The Onion [ethernet-bridge-config Github repository](#) contains reference configuration files in case you need to troubleshoot your setup.

### Default Configuration Files

If you ever want to revert your configuration to the original, we have a complete set of default configuration files from a factory-fresh Omega2 [in the uci-default-configs repo](#) on GitHub.

## Overview

**Skill Level:** Intermediate

**Time Required:** 10 minutes

What we're first going to do is set the Omega's `wlan` interface to use `eth0`, the wired ethernet connection. The only thing that's left is to connect the Omega and the target computer with an ethernet cable. We don't even have to adjust the firewall since the `wlan` interface is setup to route packets to all connected interfaces.

### Sample Configuration Files

The Onion [range-extender-config Github repository](#) contains reference configuration files in case you need to troubleshoot your setup.

### Default Configuration Files

If you ever want to revert your configuration to the original, we have a complete set of default configuration files from a factory-fresh Omega2 [in the uci-default-configs repo](#) on GitHub.

## Ingredients

- Onion [Omega2](#) or [Omega2+](#)
- Any Onion Dock that supports Expansions: [Expansion Dock](#), [Power Dock](#), [Arduino Dock 2](#)
  - We prefer the Expansion Dock for this project since it enables [access to the command line through serial](#) even when there's no network connectivity
- Onion [Ethernet Expansion](#)
- An Ethernet cable

## Step-by-Step

Here's how to turn your Omega into an Ethernet WiFi dongle!

## 1. Prepare the Omega

To begin, you'll need to make sure your Omega is connected to the Internet and has the latest firmware. Follow this [guide](#) if you'd like to learn more on how to set up your Omega.

Once that's done, plug in the Ethernet Expansion:



## 2. Enable the Omega's Ethernet Connection

Now connect your Omega and the target computer with an ethernet cable:



What we need to do next is change the Omega's networking configuration. Change the following code block located in `/etc/config/network`:

```
config interface 'wlan'
    option type 'bridge'
    option ifname 'eth0.1'
    option proto 'static'
    option ipaddr '192.168.3.1'
    option netmask '255.255.255.0'
    option ip6assign '60'
```

Change option `ifname 'eth0.1'` to option `ifname 'eth0'`

Restart the network service by running the following command:

```
/etc/init.d/network restart
```

Now the `wlan` network interface is using `eth0`, the physical ethernet interface.

### 3. Configure your Device to use Ethernet

Now that the Omega is configured, we should be able to get on the internet through an Ethernet cable to the Omega.

Make sure that your connection is set to Obtain IP address and DNS address Automatically. It should be set so by default.

## Windows

To do this on Windows, follow this [guide](#).

## Mac OSX

To do this on Mac OS X, follow this [guide](#).

## Enjoy

Now your computer has been given an IP address by the Omega and you can surf away!

