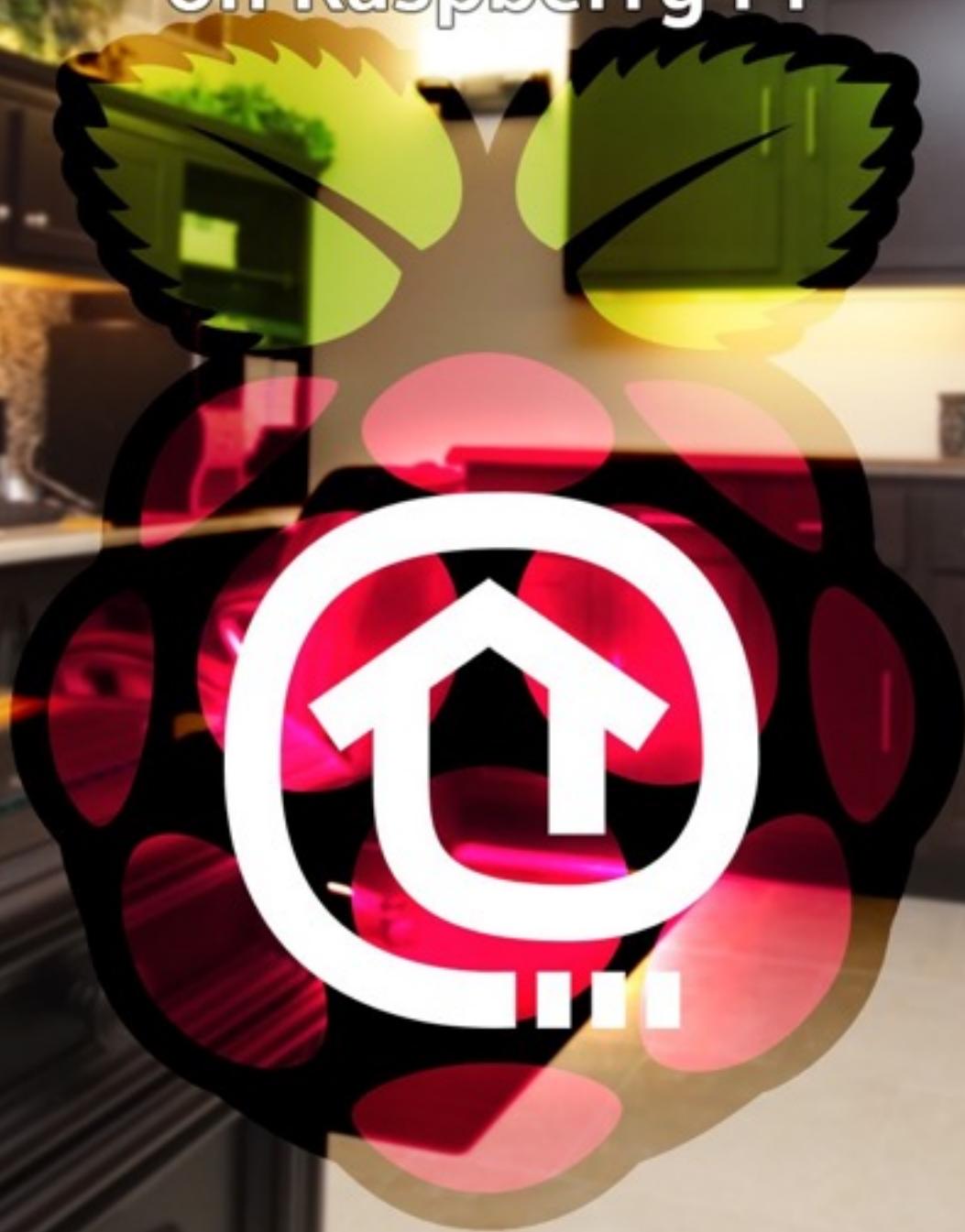


makeuseof

Getting started with  
**OPENHAB**  
**HOME AUTOMATION**  
on Raspberry Pi



by James Bruce

# Getting Started with OpenHAB Home Automation on Raspberry Pi

Written by James Bruce

Published September 2015.

*Read the original article here: <http://www.makeuseof.com/tag/getting-started-openhab-home-automation-raspberry-pi/>*

This ebook is the intellectual property of MakeUseOf. It must only be published in its original form. Using parts or republishing altered parts of this ebook is prohibited without permission from [MakeUseOf.com](http://MakeUseOf.com).

---

Read more stories like this at [MakeUseOf.com](http://MakeUseOf.com)

---

# Table of contents

What You'll Need	4
Installing OpenHAB	5
Install the Demo House	8
So How Does OpenHAB Work?	9
Enable Debug Mode	11
Adding Philips Hue	11
Remote Access, and IFTTT with My.OpenHAB	15
Bluetooth Presence Sensor using REST	18
OpenHAB Mobile App	20
Moving Forward and Getting Help	21

OpenHAB is a mature, open source home automation platform that runs on a variety of hardware and is protocol agnostic, meaning it can connect to nearly any home automation hardware on the market today. If you've been frustrated with the number of manufacturer specific apps you need to run just to control your lights, then I've got great news for you: OpenHAB is the solution you've been looking for – it's the most flexible smart home hub you'll ever find.

Unfortunately, it's about as far as you can get from consumer friendly – but as ever, that's where MakeUseOf comes in: we'll show you how to get up and running with the ultimate smart home system money needn't buy (because OpenHAB is 100% free – just supply the hardware).

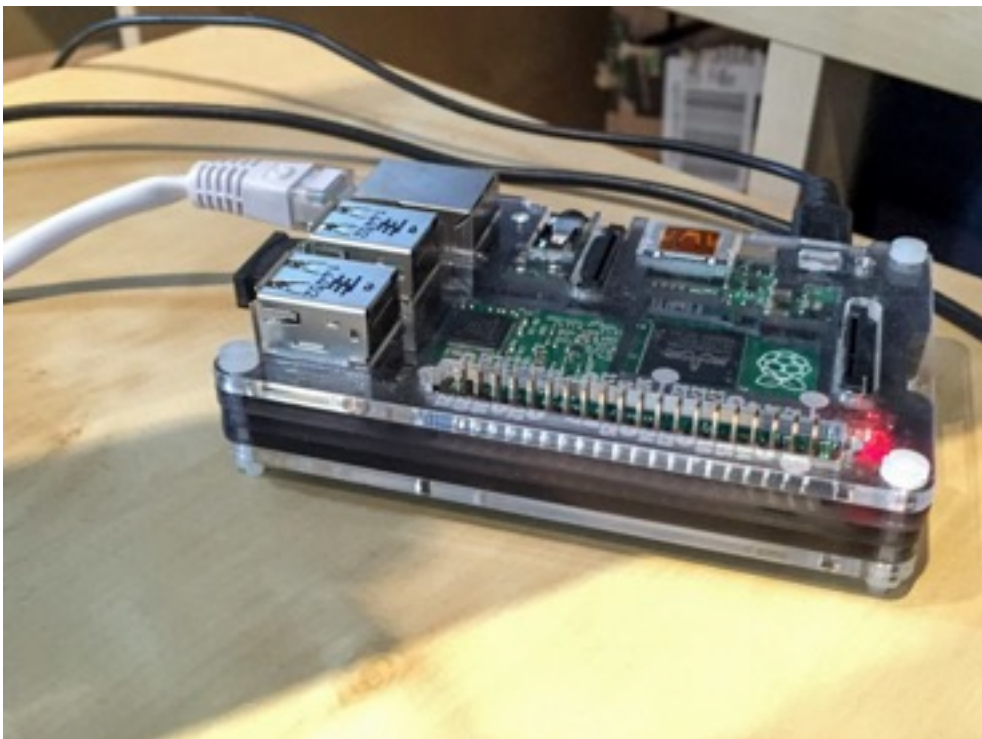
The first part of this guide focuses specifically on how to get OpenHAB setup with a Raspberry Pi 2, but further on, the tutorials and advice can be applied to anywhere OpenHAB is installed.

This guide covers three introductory topics, and one slightly more advanced.

- Getting OpenHAB up and running on the Pi, and installing the demo house configuration to check core systems are working.
- How to add bindings, and profiles for devices. I'll be working with Philips Hue.
- Enabling remote access, and connecting to IFTTT.
- Adding a DIY presence sensor using Bluetooth, and an introduction to the REST interface.
- Configuring the OpenHAB mobile app.

## What You'll Need

At the very least, you'll need a Raspberry Pi (v2, preferably), and an Ethernet or wireless adapter (Ethernet preferred, this guide won't include instructions on making your Wi-Fi adaptor work). Everything else is optional. Note that OpenHAB will run on the original Raspberry Pi too, but there's a known issue with slower processing and Z-Wave devices. If you have no need of Z-Wave, you can safely ignore this warning and go ahead with a Raspberry Pi model B or B+, because everything else seems to work fine. You can always upgrade to the latest Pi if and when you do add Z-Wave.



At the time of writing, the latest stable version of OpenHAB is version 1.71; version 1.8 is expected soon, and everything in this guide should still be relevant, though certain bindings may have more features. Version 2 is also currently available as a very early alpha preview, but adopts a dramatically different architecture to the OpenHAB 1 series: this guide is not compatible with version 2.

**I strongly suggest you follow this guide through slowly and methodically** – do not attempt to jump in at the deep end and add everything at once. Yes, it's a long guide – OpenHAB is a difficult system that often requires tweaking for your needs, and the best way to ensure success is to work slowly and complete one piece at a time.

The good news is that once it's working, it's a rock solid experience and incredibly rewarding.

## Installing OpenHAB

There's no pre-configured image for OpenHAB, so installation is done the old fashioned way via a command line. I suggest you work headless on the RPi – the overhead of managing a GUI which you'll rarely use isn't worth it.

Start with the [latest Raspbian SD image](#). Get your network cable plugged in, then boot up, and navigate through SSH. Run:

```
sudo raspi-config
```

Expand the filesystem; and from the advanced menu, change the memory split to 16. When you're done, restart, and as good practice, run a full update

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

The easier way to install the OpenHAB runtime is via *apt-get*, but first we need to add a secure key and the new repository:

```
wget -qO - 'https://bintray.com/user/downloadSubjectPublicKey?username=openhab' | sudo apt-key add -
```

```
echo "deb http://dl.bintray.com/openhab/apt-repo stable main" | sudo tee /etc/apt/sources.list.d/openhab.list
```

```
sudo apt-get update
```

```
sudo apt-get install openhab-runtime
```

```
sudo update-rc.d openhab defaults
```

Curiously, everything was installed as owned by “root”. We need to fix that with the following commands.

```
sudo chown -hR openhab:openhab /etc/openhab

sudo chown -hR openhab:openhab /usr/share/openhab
```

Next, we’ll install Samba and share the configuration and user folders – this will make it easier to install add-ons and change the sitemap remotely.

```
sudo apt-get install samba samba-common-bin

sudo nano /etc/samba/smb.conf
```

Change the workgroup name if needed, but otherwise enable WINS support:

```
wins support = yes
```

(you’ll need to uncomment the line, and change no to yes)

then add the following to the share definitions section (scroll all the way down to the bottom of the long file):

```
[OpenHAB Home]

comment= OpenHAB Home

path=/usr/share/openhab

browseable=Yes

writeable=Yes

only guest=no

create mask=0777

directory mask=0777

public=no

[OpenHAB Config]

comment= OpenHAB Site Config
```

```
path=/etc/openhab
```

```
browseable=Yes
```

```
writeable=Yes
```

```
only guest=no
```

```
create mask=0777
```

```
directory mask=0777
```

```
public=no
```

I also commented out the Printers section. I've made two shares, since the configuration files are actually stored separately to the add-ons.

Save and exit. We finally need to set a Samba password for the openhab user:

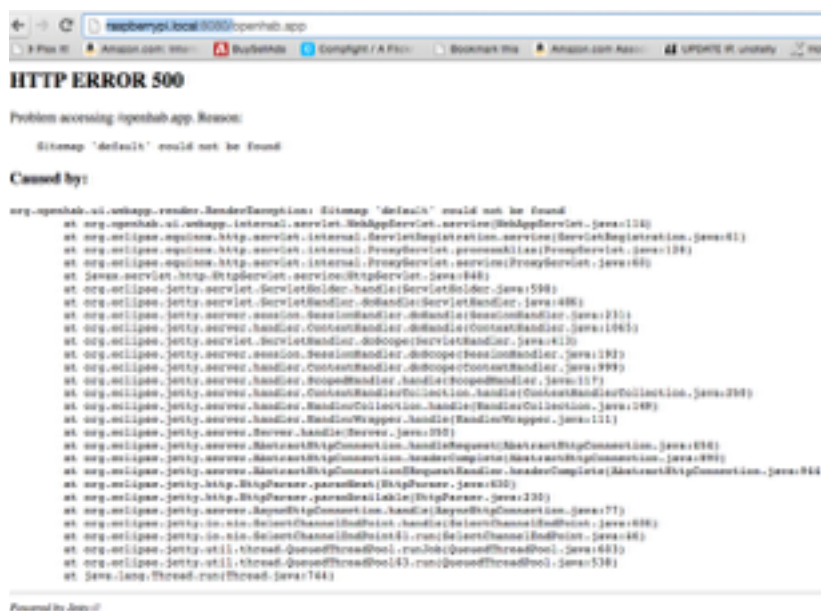
```
sudo smbpasswd -a openhab
```

I'd suggest "openhab" as the password just for ease of use, but it doesn't really matter.

After restarting Samba (sudo service samba restart), test you can access the shared drive. It might not be auto-discovered on a Mac; but you can use the **Finder** -> **Go** -> **Connect to Server** and the address

```
smb://openhab@raspberrypi.local
```

Authenticate with username openhab and your chosen password, then open up both your shares to have a look around. You should even be able to open <http://raspberrypi.local:8080/> in your web browser, but you'll be met with an error because we haven't create a sitemap yet. That's normal.



Now would be a good time to learn the command to tail the OpenHAB log so you can keep an eye on errors.

```
tail -f /var/log/openhab/openhab.log
```

Keep that running and open in a separate SSH window at all times while you continue with the guide.

## Install the Demo House

Before we delve into the intricacies of configuration files, adding devices and bindings etc; let's check everything is working by adding the demo content. You'll find "Demo Setup" under the [downloads](#) section of [OpenHAB.org](#).

Once you've unzipped it, there's two folders: **addons** and **configurations**.

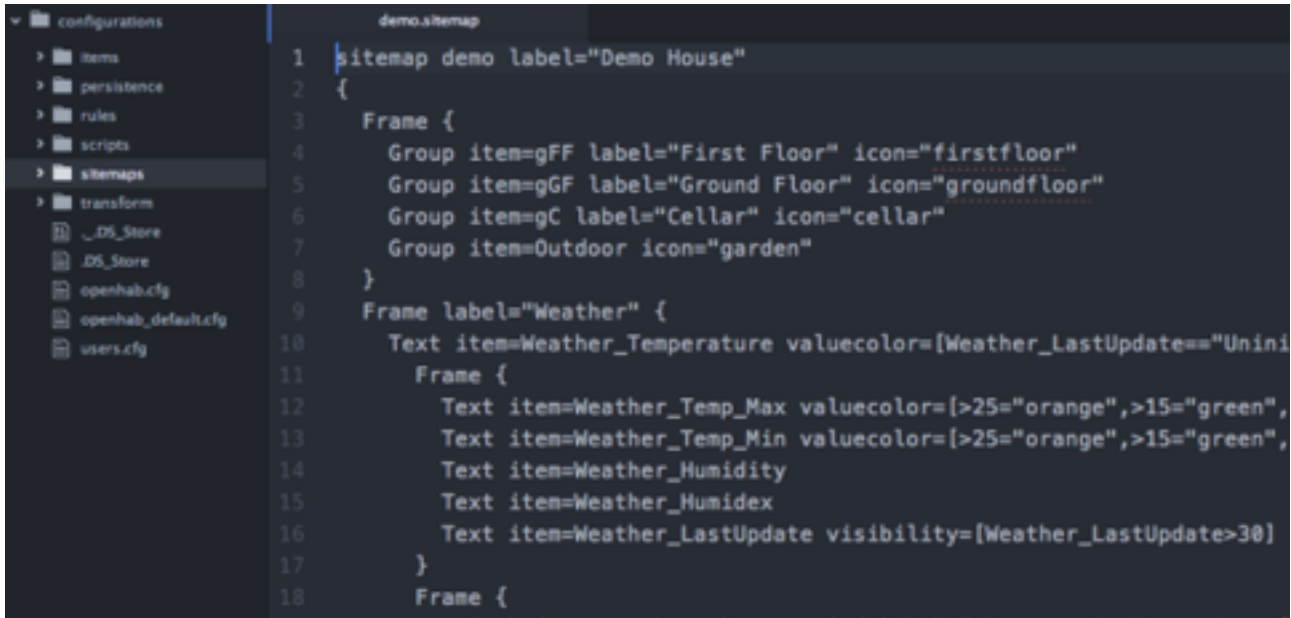
▼	Folder	addons	31 July 2015 23:43	--	Folder
	Java archive	org.openhab.binding.http-1.7.1.jar	31 July 2015 23:38	20 KB	Java archive
	Java archive	org.openhab.binding.ntp-1.7.1.jar	31 July 2015 23:38	10 KB	Java archive
	Java archive	org.openhab.persistence.exec-1.7.1.jar	31 July 2015 23:40	6 KB	Java archive
	Java archive	org.openhab.persistence.logging-1.7.1.jar	31 July 2015 23:40	7 KB	Java archive
	Java archive	org.openhab.persistence.rrd4j-1.7.1.jar	31 July 2015 23:40	625 KB	Java archive
▼	Folder	configurations	31 July 2015 23:43	--	Folder
▶	Folder	items	31 July 2015 23:43	--	Folder
▶	Folder	persistence	31 July 2015 23:43	--	Folder
▶	Folder	rules	31 July 2015 23:43	--	Folder
▶	Folder	scripts	31 July 2015 23:43	--	Folder
▶	Folder	sitemaps	31 July 2015 23:43	--	Folder
	text	README.TXT	31 July 2015 23:02	657 bytes	text

Using the network shares, copy **configurations** to the **OpenHAB Config** share and overwrite the existing folder. Copy **addons** to the other **OpenHAB Home** share, again, overwriting the existing folders. If you aren't prompted to overwrite something, you're doing it wrong. If you've got your eye on the debug log file, you should see a flutter of activity as it notices the new bindings and whirs into action. Open [raspberrypi.local:8080/openhab.app?sitemap=demo](http://raspberrypi.local:8080/openhab.app?sitemap=demo) to see the demo.





It's a little basic looking at the moment, but the open nature of OpenHAB means we can install a lovely new theme later or an alternative interface entirely. For now, we just needed to know it's all working. Note that what we're looking at is called a **sitemap** (nothing to do with a website site map). A sitemap describes the user interface – not the actual devices on your network or sensors – just the interface to view them. Every part of it is completely customizable. To have a look at how this one has been created, open up the **sitemaps/demo.sitemap** file on the OpenHAB Config share.



It's pretty daunting, but for the most part you'll be copy pasting code fragments from examples elsewhere to create your own custom interface. Here's the [technical overview](#) of all possible sitemap elements, but for now it will suffice just to start thinking about what kind of interface you want to build and what information you want to display.

While you're in there, open up **items/demo.items** too. Again, looks scary, but this is where you create items to control and define sensors to track.

## So How Does OpenHAB Work?

Now that you've had a quick peruse of the sitemap and items folder, let's break down exactly what these files are and the other main components of OpenHAB that combine to create your complete smart home. You'll find subdirectories for each of these in the OpenHAB Config shared folder.

**Items** is an inventory of every control device, sensor, or information element you want in your system. It needn't be a physical device either – you might define a web source such as weather or stock prices. Each item can be named, assigned multiple groups (or none), and connected to specific binding. (*Beginner tip: Capitalization is important when it comes to bindings. I spent a long time trying to work out why my "Hue" bulbs weren't working; it was because they should have been "hue" instead.*)

**Sitemaps** is concerned only with the interface you'll see when you open the OpenHAB mobile or web app. You can control precisely how you want the buttons laid out and information presented. You could define top level groups for each room of your house; clicking on each would show you a list of every device in that room. Or you might prefer to show groups for each type of device: a button for lights, another for electrical outlets. There might be some devices you use so often that you just want a switch for them right on the home screen.

**Rules** is where the home automation aspect comes into play, where you can define schedules or conditions for an action to happen. Simple events like turning on the bedroom lights at 10pm to a

warm red color; or more complex logic like turning on a space heater if the temperature is less than 0 and someone is present in that room. You'll also find a **scripts** folder, which offers similar functionality to rules but at a more complex level of programmable logic.

**Persistence** is an advanced topic that we won't be covering in this guide, but persistence defines data you want to keep a record of. By default, OpenHAB is only going to show the current state of something; if you want to track that value over time, you need to setup a persistence definition for that data source. In this you'll specify things like how often a data point should be measured, or when to discard old data points – you'll also need to tell it what kind of persistence engine to use, such as MySQL or simple logging to a file.

**Transform** contains mappings for data values to labels. For instance, the **humidex.scale** file defines a range of humidity index values and how they should be shown in English: 29-38 is "some discomfort".

The **sitemap** and **items** files are essential for OpenHAB to run; the rest are optional. You can have multiple sitemaps and items, so you can keep the demo content and refer back to it at any time, or try out a new layout for your home control interface. Don't worry if this all seems a bit overwhelming right now, we'll break it down into manageable pieces and I promise by the end of this guide you'll have confidence to create your own OpenHAB setup.

Next up, we'll walk you through adding some common smart home kit, starting from scratch in a new sitemap. Each one will introduce some core concepts such as how to install bindings and item definitions, so I'd strongly encourage you to read through these instructions *even if you don't own those particular devices*.

Start by creating a new (blank) **home.items** file, and a new **home.sitemap** file in the relevant directories. Open up **home.sitemap** and paste in the following code. This just acts as a basic skeleton to which we'll be adding bits later.

```
sitemap home label="My Home"

{

}

}
```

You should see a notice to report that OpenHAB has identified a new sitemap and items file.

```
2015-08-17 08:10:50.007 [INFO ] [c.internal.ModelRepositoryImpl] - Loading model 'home.items'
2015-08-17 08:11:00.039 [INFO ] [c.internal.ModelRepositoryImpl] - Loading model 'home.sitemap'
2015-08-17 08:11:10.055 [INFO ] [c.internal.ModelRepositoryImpl] - Refreshing model 'home.sitemap'
2015-08-17 08:11:10.070 [INFO ] [c.internal.ModelRepositoryImpl] - Refreshing model 'demo.items'
█
```

## Enable Debug Mode

While you're still trying to get OpenHAB working correctly, it can be useful to enable a more verbose debug log that lists everything, and not just the important stuff. To enable this mode, open up the OpenHAB Config shared folder, and edit the **logback.xml**. On line 40, change the following line to read DEBUG instead of INFO. You'll need to restart after changing this.

```
<logger name="org.openhab" level="INFO"/>
```

This is a global change, so you'll be getting a lot more info once you tail the log file.

## Adding Philips Hue

I'm going to start with [Philips Hue](#). Like most things you'll want to interact with in OpenHAB, Hue bulbs require the you to install a **binding** – think of bindings as like a device driver. At the time of writing, there's around 160 bindings available for OpenHAB 1, which is why OpenHAB is such a powerful system – it can interface with anything, combining all of those disparate control systems into a single unified interface. Here's a [demo and quick overview](#) of the steps involved.

Bindings must first be downloaded, and the easiest way to do this on the Pi is using **apt-get**, then force ownership to the openhab user.

```
sudo apt-get install openhab-addon-binding-hue
```

```
sudo chown -hR openhab:openhab /usr/share/openhab
```

Next you need to tell OpenHAB to load in that binding, and configure any variables needed. Browse to the configurations folder and make a copy of **openhab-default.cfg**, naming it **openhab.cfg**. Open that up, search for *HUE* and replace the whole section with the following code. The only thing you need to change is the IP value of your bridge – if you don't already know it, try the [online discovery tool](#). The secret value doesn't really matter, it's just a kind of username that OpenHAB will use to identify itself to the bridge.

```
##### HUE Binding
#####

# IP of the Hue bridge

hue:ip=192.168.1.216

hue:secret=makeuseofdotcom

hue:refresh=10000
```

```

openhab.cfg - /Volumes/OpenHAB Config/configurations - Atom
configurations
├── items
├── persistence
├── rules
├── scripts
├── sitemap
├── transform
├── openhab.cfg
├── openhab_default.cfg
└── users.cfg
openhab.cfg
808
809 ##### PLC Bus Binding #####
810 #
811 # PLCBus adapter serial port
812 #plcbus:port=
813
814 ##### DMX Binding #####
815 #
816 # The combined connection String, e.g. 'localhost:9010' (optional, defaults to
817 # 'localhost:9010' or 'localhost:9020' depending on the chosen connection type)
818 #dmx:connection=
819
820 ##### HUE Binding #####
821 #
822 # IP of the Hue bridge
823 hue:ip=192.168.1.216
824 hue:secret=makeuseofdotcom
825 hue:refresh=10000
826
827 ##### RFXCOM Binding #####
828 #
829 # Serial port of RFXCOM interface
830 # Valid values are e.g. COM1 for Windows and /dev/ttyS0 or /dev/ttyUSB0 for Linux
831 #rfxcom:serialPort=
832
833 # Set mode command for controller (optional)
834 # E.g. rfxcom:setMode=0000000035300000C2F0000000
835 #rfxcom:setMode=
836
837 ##### Pulseaudio Binding #####
838 #
839 # PulseaudioServer IP address
840 #pulseaudio:Main.host=

```

Save and exit. Like any third party Hue application, you'll need to approve OpenHAB on the Hue Bridge by pressing the button on the front – you only need to do this once. You'll see a message about *waiting to be paired* if you're tailing the log file, but if you've forgotten or missed the count down, just reset the Pi – you'll get a 100 second timer from when the Hue binding is initiated. Make sure you've successfully paired before you continue.

Next, open up the **home.items** file, to which we'll add some Hue bulbs. Here's an example item definition:

```
Color Bedroom_Hue "Bedroom Hue" <hue> (Bedroom) {hue="1"}
```

- The **Color** word specifies what kind of control we have over this item. RGB Hue bulbs are “Color”, since we have full color control of them. Other lights may just be a Switch.
- Next is the codename of the item: I chose **Bedroom\_Hue**, but literally anything is fine – just something descriptive that feels natural to you, because you'll need to remember it later when making the sitemap. The codename should have no spaces.
- Between the quote marks is the label. Ours is simple in this case, but for some items like temperature or something that reports a value, you'll add some special code that tells it how to display that value or using what *transform*. The label is for the interface, and it can have spaces.
- Between the angle brackets is the name of the icon. You'll find all the available icons in the OpenHAB share, under the **webapps/images** directory. There's actually a whole range of hue icons that represent different brightnesses or on/off. Just specify the base icon name – OpenHAB will know to automatically look for the different on/off icons if this is a switched item. This is optional.

- In the round brackets, we tell it which groups to be a part of – in this case, just the **Bedroom** group.
- Finally and crucially, we connect the item to the appropriate binding with any variables needed. In this case, the **hue** binding, and the number of the bulb is 1. You can find the number by opening up the official Hue application and looking at the lights tab. Each bulb has a unique number.

I've added a total of four bulbs, as well as a simple declaration of the groups that we'll expand on later. Here's my complete **home.items** at this point:

```

Group Bedroom

Group Office

Group Kai

Group Living_Room

Group Cinema

Group Secret

Group Lights

/* Lights */

Color Bedroom_Hue "Bedroom Hue" <hue> (Bedroom,Lights) {hue="1"}

Color Office_Hue "Office Hue" <hue> (Office, Lights) {hue="2"}

Color Secret_Hue "Secret Hue" <hue> (Secret, Lights) {hue="3"}

Color Kai_Hue "Kai's Hue" <hue> (Kai, Lights) {hue="4"}

```

The **/\* Lights \*/** text is just a comment, it has no function other than to help us scan the file later when it gets bigger. Now we have the devices added, but opening up <http://raspberrypi.local:8080/?sitemap=home> results in a blank interface – of course, because we haven't actually created interface elements in the sitemap yet. Let's start really simple for now. Open up **home.sitemap**.

The code used to describe the interface is different to items, but for now we'll create a new "frame" and add a couple of group controls along with some icons.

```

sitemap home label="My Home"

{

```

```

Frame {

    Group item=Lights label="All lighting" icon="hue"

    Group item=Bedroom label="Bedroom" icon="bedroom"

    Group item=Office label="Office" icon="desk"

}

}

```

Groups are a useful tool for quick testing, but in reality you'll want more control over how the items are display. For now, this will suffice. Save and reload your home sitemap in the browser. You should see this (or, whatever groups you've added).



Click on **All lighting** to see every Hue light, since we've defined them all as belonging to that overarching lights group.



Notice that the Office Hue item is displayed with a different icon – that’s because my office light is already on, and OpenHAB knows this when it spoke to the Hue bridge and was smart enough to adjust the icon the “on” version of the file. Unfortunately, it doesn’t reflect the color, but if you have mobile app installed, that will reflect the current color.

If you’re seeing more items than you thought you’d defined or receiving errors about multiple definitions, know that although you can only load one sitemap at a time onto the page *all sitemaps will pull items in from all .item files*, so if you’ve left the demo items file there, you may have some additional items show up in your groups as well. I’d suggest at this point backing up the demo items content and moving it out of the folder to avoid duplication errors.

## Remote Access, and IFTTT with My.OpenHAB

Right now, you need be on the same local network to access your OpenHAB system, but what if you want to control your devices and check on sensors when out of range of your Wi-Fi? For that we’ll need to set up remote access – and we’ll do it the easy way, with the [My.OpenHAB web service](#), which bypasses the need to mess around with port forwarding and [router](#) configurations. As a bonus, the My.OpenHAB service also has an IFTTT channel, giving you infinite possibilities for remote control and automation.

First: install the binding. Quick tip: if you don’t know the exact name of a particular install package, try searching for it with apt-cache.

```
pi@raspberrypi ~ $ apt-cache search myopenhab
openhab-addon-io-myopenhab - my.openHAB Connection Service
pi@raspberrypi ~ $ █
```

```
sudo apt-get install openhab-addon-io-myopenhab
```

```
sudo chown -hR openhab:openhab /usr/share/openhab
```

Before you can register on the My.OpenHAB site, you’ll need to create a secret key, and find your UUID, which uniquely identifies your installation. Check under the **OpenHAB Home share** -> **webapps** -> **static** and you should find a UUID file containing your unique identifier. It’s at this point that I discovered my Pi was using an older version of Java which doesn’t correctly create the secret key. Type

```
java -version
```

to check. If it doesn’t say 1.7 or higher, you have the wrong version. Oddly, the latest version of Raspbian does come with Oracle Java 8 installed, but not set as default.

```
sudo update-alternatives --config java
```

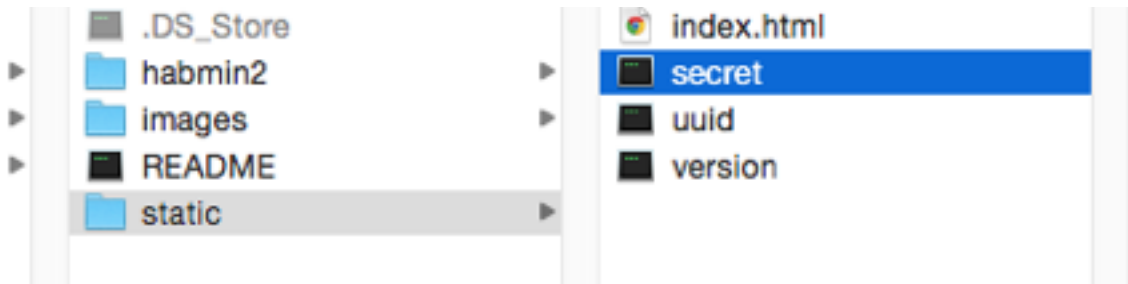
```
pi@raspberrypi /usr/bin $ sudo update-alternatives --config java
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                          Priority  Status
  -----
* 0            /usr/lib/jvm/java-6-openjdk-arahf/jre/bin/java 1057     auto mode
  1            /usr/lib/jvm/java-6-openjdk-arahf/jre/bin/java 1057     manual mode
  2            /usr/lib/jvm/jdk-8-oracle-ara-vfp-hflt/jre/bin/java 318     manual mode

Press enter to keep the current choice[*], or type selection number:
```

Choose the option that indicates **jdk-8-oracle**, then restart OpenHAB. Bonus: Oracle Java 8 is faster than the default OpenJDK!

Now you should also find a secret file in the **webapps/static** folder. Open both the **secret** and **uuid**, and be ready to copy paste.



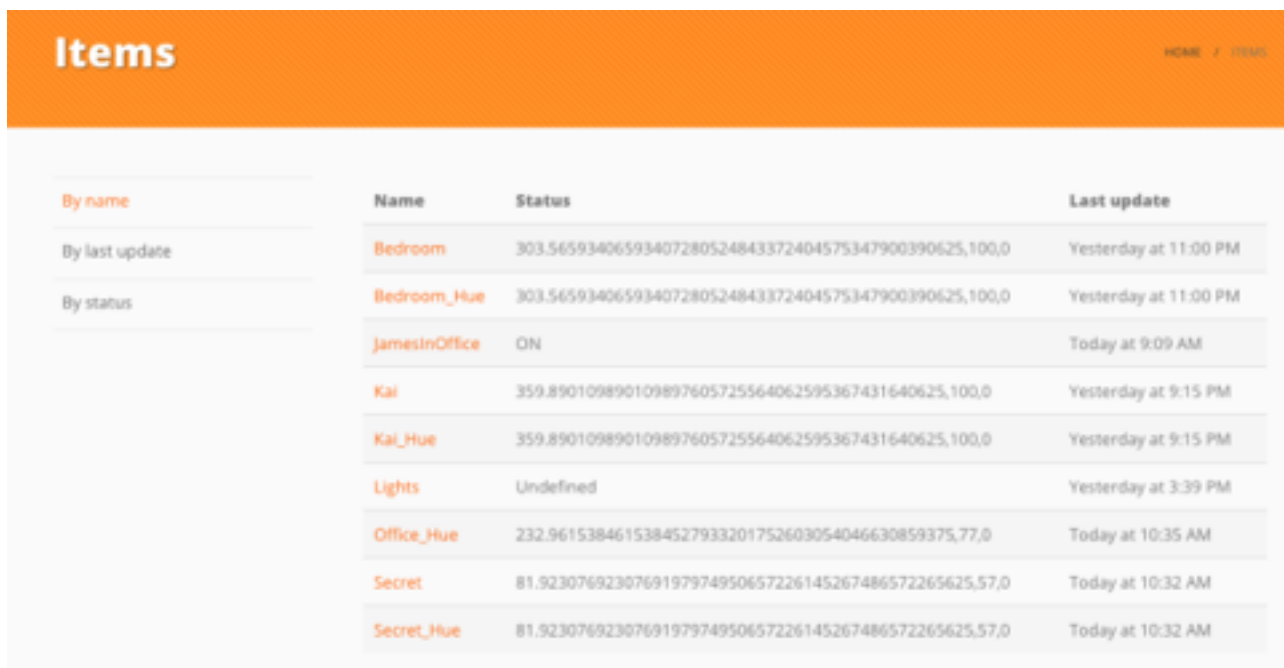
Now go create a My.OpenHAB account, using those details, then come back – you’ll also need to confirm your email before anything works. There’s a couple more steps to this one. First, we need to set the default persistence engine to myopenhab (persistence is something for a later guide, but regardless, we need to set up something basic in order to “export” our data to the online service and make it visible to IFTTT). To do this, open up openhab.cfg, and find the variable that says **persistence:default=** and change it to **persistence:default=myopenhab**. Save.

Lastly, create a new file in the **configurations/persistence** folder called **myopenhab.persist**, and paste in the following rule.

```
Strategies {  
  
    default = everyChange  
  
}  
  
Items {  
  
    * : strategy = everyChange  
  
}
```



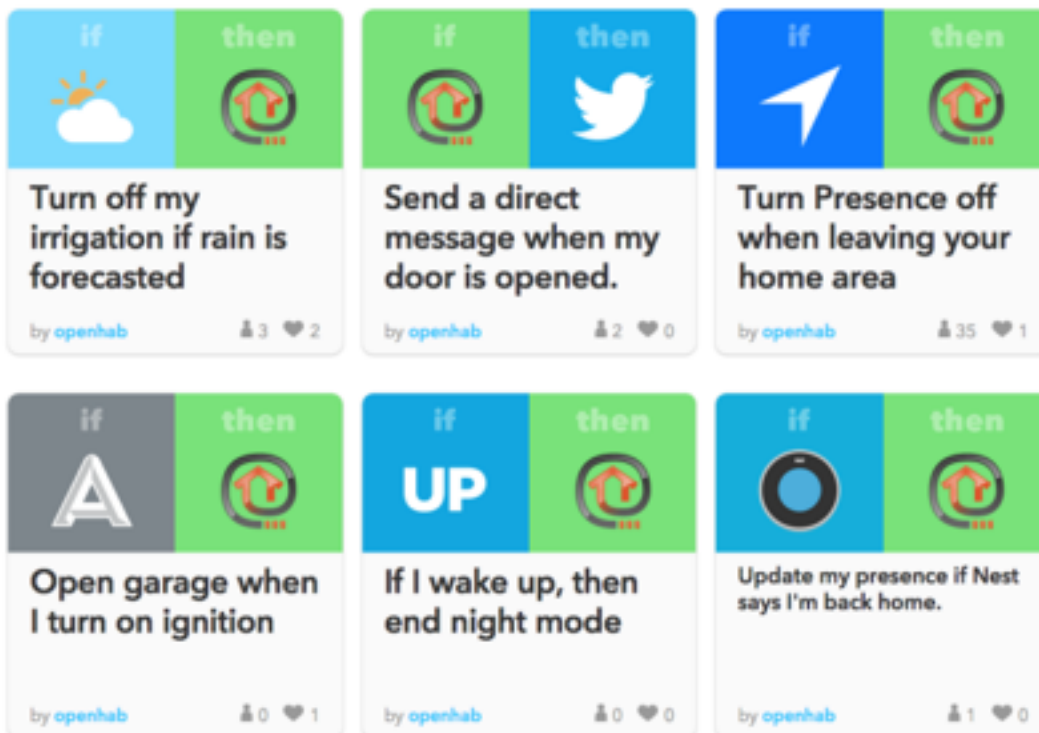
You don't need to understand this for now, but know that it says "save every item state when it changes".



	Name	Status	Last update
By name	Bedroom	303.56593406593407280524843372404575347900390625,100,0	Yesterday at 11:00 PM
By last update	Bedroom_Hue	303.56593406593407280524843372404575347900390625,100,0	Yesterday at 11:00 PM
By status	JamesInOffice	ON	Today at 9:09 AM
	Kai	359.890109890109897605725564062595367431640625,100,0	Yesterday at 9:15 PM
	Kai_Hue	359.890109890109897605725564062595367431640625,100,0	Yesterday at 9:15 PM
	Lights	Undefined	Yesterday at 3:39 PM
	Office_Hue	232.9615384615384527933201752603054046630859375,77,0	Today at 10:35 AM
	Secret	81.9230769230769197974950657226145267486572265625,57,0	Today at 10:32 AM
	Secret_Hue	81.9230769230769197974950657226145267486572265625,57,0	Today at 10:32 AM

To connect with IFTTT, head over to the [OpenHAB channel](#) – you'll need to authenticate and give it access to your MyOpenHAB account. Also note that until your items have changed at least once, they won't be visible in the items list on IFTTT, so if it's not visible, toggle something on and off, then reload. Congratulations, you now have complete IFTTT access to everything in your OpenHAB system!

### Popular openHAB Recipes



- Turn off my irrigation if rain is forecasted**  
by openhab 3 users, 2 hearts
- Send a direct message when my door is opened.**  
by openhab 2 users, 0 hearts
- Turn Presence off when leaving your home area**  
by openhab 35 users, 1 heart
- Open garage when I turn on ignition**  
by openhab 0 users, 1 heart
- If I wake up, then end night mode**  
by openhab 0 users, 0 hearts
- Update my presence if Nest says I'm back home.**  
by openhab 1 user, 0 hearts

# Bluetooth Presence Sensor using REST

A short while ago I showed you how to make an [automatic office door lock using Bluetooth](#) scanning to detect presence of the user – I wanted to bring something like that into OpenHAB.

On any platform other than Raspberry Pi, this would be simple thanks to a ready-made Bluetooth binding; unfortunately, it doesn't work on Pi due to a crucial Java file that would need to be recompiled for the ARM architecture, added to the binding, and then rebuild the binding. Suffice to say, I tried that, and it was hideously complicated and didn't work. There is however a much easier solution that also serves as a good introduction to the sheer extensibility of OpenHAB: we'll simply adapt our previous Python script to have it report directly to the OpenHAB RESTful interface.

*Aside: a RESTful interface means you can interact with a system using it's built in web server, simply by calling URLs and passing in or fetching data. You can visit this URL to see a simple example of this on your own server: <http://raspberrypi.local:8080/rest/items> – which outputs a coded list of all your defined items. This is incredibly powerful, as it exposes the full potential of OpenHAB and allows you to write custom interfaces; or in used reverse, to report the status of sensors without having a specific binding. We'll use this capability to report the presence of a specific Bluetooth device without resorting to the Bluetooth binding.*

Start by adding a new *Switch* item to your *home.items* file. I've called mine "JamesInOffice", and I've made it a switch rather than a simple on/off contact so that I can manually control my presence in case my phone dies.

```
Switch JamesInOffice "James in Office" (Office)
```

Notice that I've not defined an icon, or associated a specific binding. It's just a generic switch.

Next, insert a compatible USB Bluetooth dongle, and install some basic tools for interacting with it.

```
sudo apt-get install bluez python-bluez python-pip

sudo pip install requests

hcitool dev
```

The last command should show your Bluetooth adapter. If nothing is listed, try another adapter, yours isn't compatible with Linux. The next step is to find the Bluetooth hardware address of your device.

```
wget https://pybluez.googlecode.com/svn/trunk/examples/simple/
inquiry.py

python inquiry.py
```

Ensure your phone is open on the Bluetooth settings page (which puts it into pairing/public mode), and obviously that Bluetooth is activated. You should find a hexadecimal hardware address listed.

```
pi@raspberrypi ~ $ python inquiry.py
performing inquiry...
found 1 devices
    78:9F:70:38:51:1B - James' iphone 6
pi@raspberrypi ~ $ █
```

From your Pi user home directory, create a new Python script and [paste in this code](#). There's a few things you'll need to edit, starting with your particular device address:

```
result = bluetooth.lookup_name('78:7F:70:38:51:1B', timeout=5)
```

As well as the this line, which is in two places (yes, this could probably be structured better). Change JamesInOffice to the codename of the switch you defined.

```
r = requests.put("http://localhost:8080/rest/items/JamesInOffice/  
state",data=payload)
```

The final step is to tell this script to launch at boot time.

```
sudo nano /etc/rc.local
```

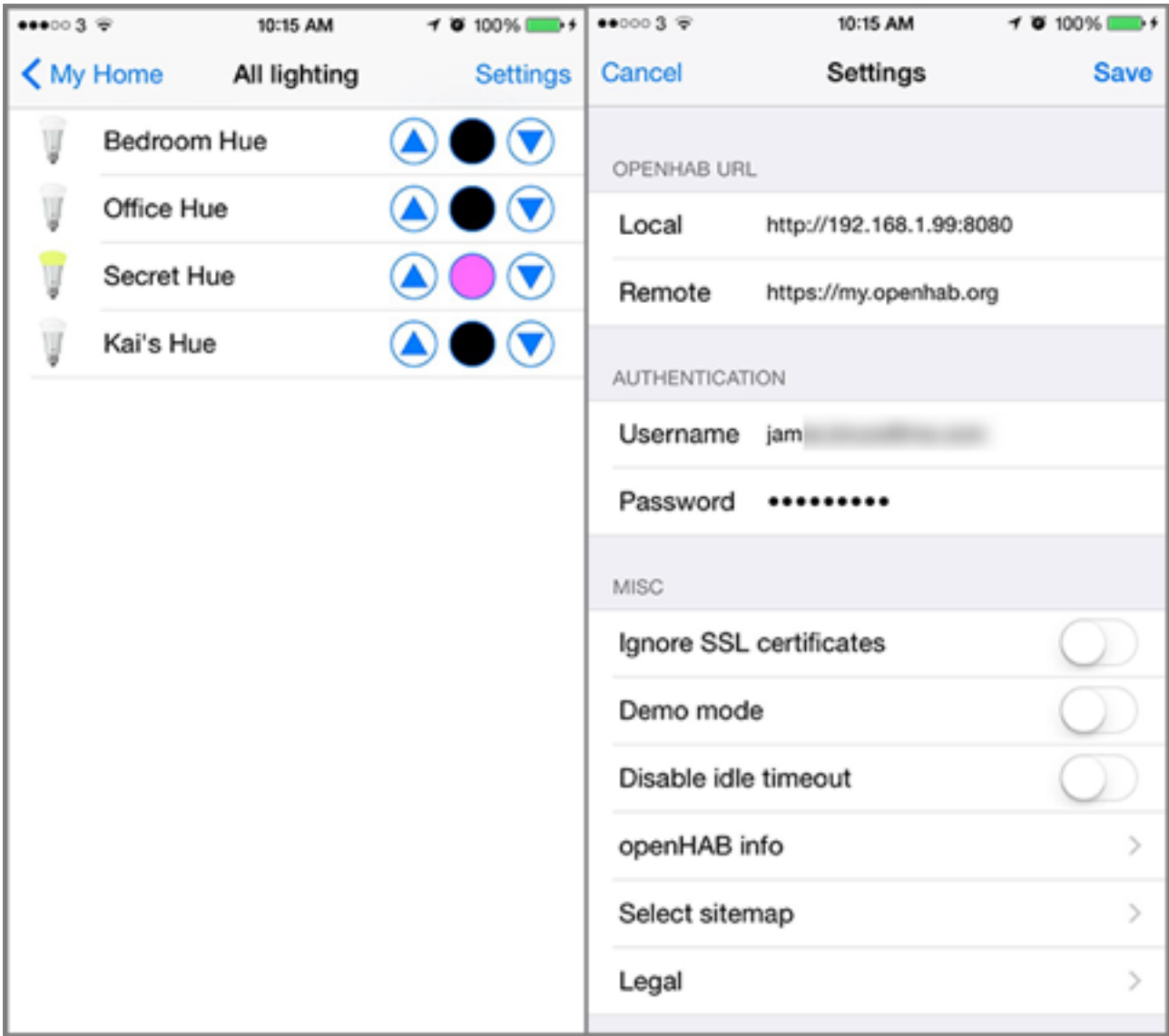
Scroll down to the bottom and just before the exit 0, add the following lines:

```
python /home/pi/detect.py &
```

The & sign means “do this in the background”. Go ahead and run the script if you haven't already, and open up your OpenHAB interface. If you've added it to a group, click through to that group. It takes about 10 seconds to update, but you'll see the default lightbulb icon come on or off depending on whether your phone is detected or not. Check the log file if nothing happens, it could be that you've used the wrong Item name.

# OpenHAB Mobile App

Though you can of course use the web interface from a mobile device, OpenHAB has native apps for both [iOS](#) and [Android](#) – and they look a *lot* nicer than the default browser interface. On the settings screen, enter the local URL as the internal IP you've been using until now, including the port number. For remote URL, enter **https://my.openhab.org**, and your username (email) and password that you entered when you signed up. If you haven't signed up for MyOpenHAB yet, just leave the authentication and remote URL blank, but you'll only be accessing your system from your local Wi-Fi.



## Moving Forward and Getting Help

The amount of customization and neat features you can add to your OpenHAB controller is really kind of epic. As well as the vast list of supported devices with bindings, you can use the RESTful interface, HTTP extensions, and IFTTT to read from or control literally any kind of IoT device, and then some (try some of our [creative lighting ideas](#)). Yes, it's an absolute pain to install, but not a single commercial system can come close to the power of a customized OpenHAB system.

That said, the ride wasn't at all easy for me, which is precisely why I wrote this guide, to ease the process for you. Stay tuned to MakeUseOf for an advanced guide that covers Z-Wave and other cool tricks you can set up.

If you need help with a specific part of this guide, please ask away in the comments. If you want help with another binding or some advanced topics we haven't covered yet, the [official OpenHAB forums](#) are a welcoming place.

---

Read more stories like this at [MakeUseOf.com](http://MakeUseOf.com)

---