Getting Started Guide

Rami Alshafi
VTM GROUP

David Kinder
INTEL CORPORATION

OPEN CONNECTIVITY FOUNDATION™

IoTivity Development on Raspberry Pi* 3

GETTING STARTED GUIDE

Version 2, January 8, 2018

Table of Contents

Summary

This Getting Started Guide shows you how to set up an IoTivity development environment on a Raspberry Pi* 3 board. You will also build and run sample server and client applications that verify the build environment is set up properly, and can interact with an Internet of Things (IoT) device, in our case an LED. These sample applications are a baseline and reference for new developers to explore the IoTivity API framework and learn how to write secure server and client applications that can pass the OCF Certification Test Tool (CTT) and implement OCF Introspection (how devices discover and communicate their capabilities to each other, enabling interoperability)

OCF and IoTivity

IoTivity is an open source software project enabling seamless device-to-device connectivity where billions of wired and wireless Internet of Things (IoT) devices can securely connect to each other and to the internet. The Open Connectivity Foundation (OCF) develops specification standards, interoperability guidelines, and a certification program for these devices. IoTivity is an open source reference implementation of the OCF specification. You can learn more about OCF at http://openconnectivity.org and IoTivity at http://iotivity.org.

The IoTivity APIs expose the OCF framework to developers, and are available in several languages and for multiple operating systems. The framework supports dedicated and optimized protocols for IoT devices, with specific considerations for constrained devices, and addressing many types of devices, form-factors, companies, and markets.

The IoTivity framework operates as middleware across supported operating systems and connectivity platforms and has these four essential building blocks:

- **Discovery**: supporting multiple mechanisms for discovering devices and resources in proximity and remotely.
- **Data transmission**: supporting information exchange and control based on a messaging and streaming model.
- **Data management**: supporting the collection, storage, and analysis of data from various resources.
- **Device management**: supporting configuration, provisioning, and diagnostics of devices.

The IoTivity Development on Raspberry Pi 3 Kit

This kit provides you with a Raspberry Pi 3 board and additional hardware to get you started with IoT development using IoTivity APIs and interacting with sensor devices. You'll be building and running the sample client and server applications (written in C/C++) on the Raspberry Pi 3 board itself, by following the instructions in this getting started guide. After following these instructions, you will have verified the on-board development environment and tools are set up and working. You can read through the sample applications source code to see how it works and make changes on your own to learn more!

Parts List

The kit contains the following hardware. (You can also assemble a hardware kit from this parts list):

- Raspberry Pi 3, Model B V1.2, with a quad-core 64-bit Arm*v8 processor, HDMI, USB, Ethernet, on-board Wi-Fi* and Bluetooth* Low Energy support, and 1GB RAM.

- Pimoroni Enviro pHAT, with two LEDs, four different sensors, (letting you measure temperature, pressure, light level, color, 3-axis motion, compass heading), and analog inputs. In the supplied kits, we've soldered a female GPIO header for connecting directly to the Raspberry Pi's header pins.

- Micro-USB 5V, 2.5A power supply

- 8 GB microSD card, with Raspbian OS (a variant of Linux* built for the Raspberry Pi) loaded and configured to boot with SSH enabled. (An appendix has instructions for downloading needed software and creating a configured microSD card from scratch.)

- Ethernet cable for connecting the Raspberry Pi board to your network. You'll need an active connection to the Internet as well.

- Optionally, an HDMI monitor, HDMI cable, and a USB keyboard are recommended but not required (and not included in the kit).

Network Security Warning

You'll be connecting the Raspberry Pi to your local network and the Internet. If you're in a corporate environment or using a corporate laptop, it may be against corporate guidelines to do this because of security concerns. Please consult with your network admin. Corporate environments may also require proxy settings to allow tools such as Git to get through the firewall to GitHub and Gerrit repositories. Configuring your proxy settings is beyond the scope of this guide.

The Raspberry Pi setup uses a default login and password shared on every Raspberry Pi running Raspbian. If you expose the board to the internet with SSH enabled (as this Getting Started Guide does, to get access to source code and to download needed tools), you need to make some basic security changes, as documented by the Raspberry Pi folks at https://www.raspberrypi.org/documentation/configuration/security.md. This list is not exhaustive though. Secure setup of your development board is out of scope for this guide.

Working directly or indirectly with the Raspberry Pi

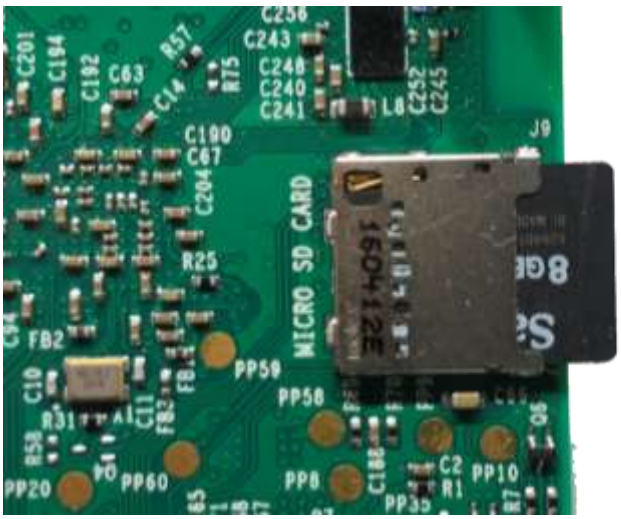
We've found it convenient to setup and communicate directly to the Raspberry Pi with an attached USB keyboard and HDMI monitor (not included in the kit). We do provide instructions in an appendix, for working with a "headless" Raspberry Pi via ssh to indirectly connect to the board from a host computer. In either case, we use the Raspbian command line interface so we won't need a mouse.

Setting up the Hardware

Let's get started! (Note: you may have received a pre-assembled kit with the boards connected and the microSD card inserted. In this case, simply verify it's been assembled as described here.

1. Align the female connector on the Enviro PHAT board with the male header pins on the Raspberry Pi board and press firmly to connect the two boards together:

2. Flip the board over and fully insert the microSD card into the microSD card slot on the edge of the Raspberry Pi board (note the microSD card orientation, label up):



The microSD card in the kit has been formatted and loaded with a bootable Raspbian Stretch Lite OS, configured with SSH enabled. This version of Raspbian does not include

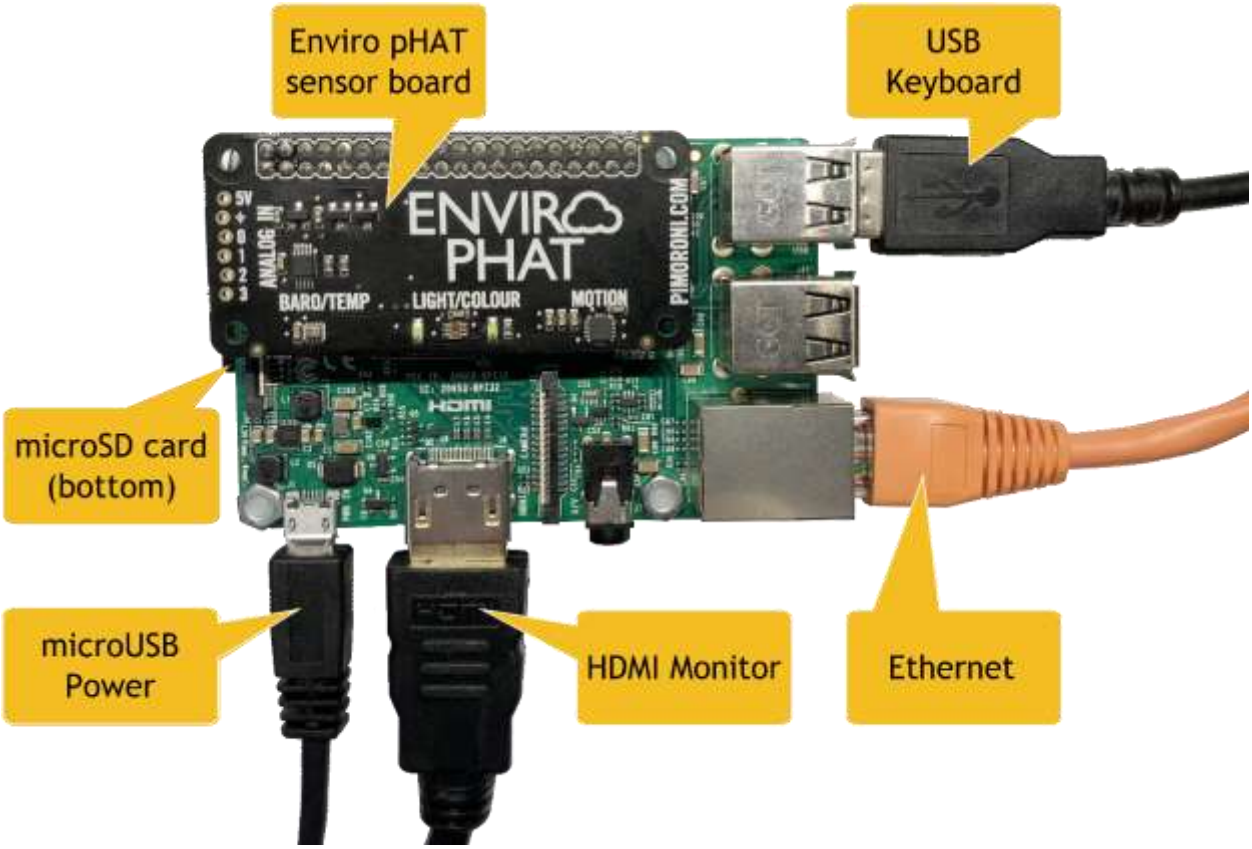
the graphical desktop environment packages since we'll be working with the command line.

3. Connect the Ethernet cable from your network to the Ethernet connector on the Raspberry Pi board, and the micro-USB power cable to the board (but don't plug in the power yet).

   If you're using a monitor and keyboard (recommended), connect the HDMI cable to your monitor and the HDMI connector on the Raspberry Pi board, and connect the USB keyboard to any of the USB connectors.

   If you're not using a connected monitor and keyboard, we explain in an appendix how to connect to the board from a host computer using SSH.

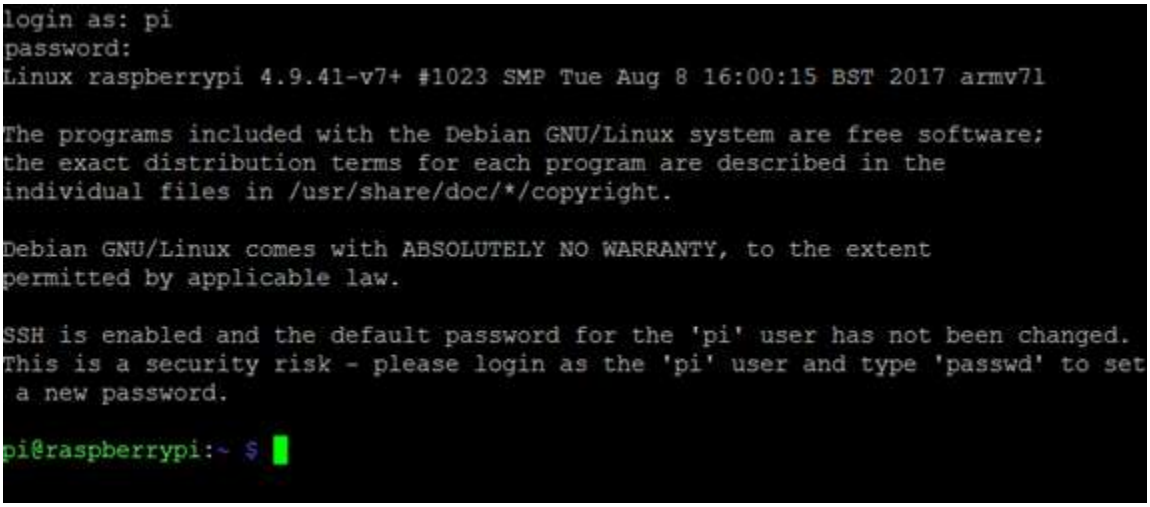
   Here's the Raspberry Pi 3 board fully populated with kit components and cable connections (if you're accessing the board remotely via ssh, then you won't need the monitor or keyboard connection):



4. Now you're ready to plug in the power and boot the board, as explained in the next section.

Booting the Raspberry Pi with Raspbian

Plug in the power supply connected to the micro-USB power connector and (with a connected monitor) you'll see the Raspbian OS boot and print log messages. After about 20 seconds, you'll be prompted to login. **The default username is "pi" and password is "raspberry".** Type those in, and you'll be at a (hopefully familiar looking) Linux command line prompt:



If you're interacting indirectly with the board via ssh, simply wait about 20 seconds before connecting from your host computer, and logging in to the Raspberry Pi. (See the appendix for connecting via ssh.)

Password, Keyboard, and Date/Time Setup

Here are some recommended changes you should do after you login for the first time.

1. By default, the Raspbian lite OS is configured with British (UK) settings for the keyboard and locale. This can be confusing for folks because of keyboard differences (for example, if you press the " | " key on a US keyboard it shows up as "~" with a UK keyboard layout).

   Edit the default keyboard configuration file `/etc/default/keyboard` and change the XKBLAYOUT setting. Use "sudo" with your editor since this is a system-owned file. (Raspbian provides both "vi/vim" and "nano" text editors.)

   ```
   sudo vi /etc/default/keyboard
   ```

   Look for the line:

   ```
   XKBLAYOUT="gb"
   ```

and change the "gb" to your desired two letter country code, e.g., "us" for United States. (Vi tip: use the arrow keys to move around, "R" command to replace characters, <escape> to stop replacing characters, and ":x" command to save and exit changes.)

Reboot the Raspberry Pi now so you'll be using the intended keyboard layout for the next step:

```
sudo reboot
```

And login again.

2. For security reasons, it's a good idea to change the default login password for the "pi" user. Use the "passwd" command, enter the current password ("raspberry") and change it to a new password you'll remember:

```
pi@raspberrypi:~$ passwd
Changing password for pi.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
pi@raspberrypi:~$
```

3. This is also a good time (pun intended) to verify the date and time is set correctly on the Raspberry Pi, by using the "date" command. Set the date and time if it's not correct, with your local time, something like this for Pacific Daylight Time (PDT):

```
sudo date -s "Jan 6 14:33:00 PDT 2018"
```

An incorrect date and time, can cause system problems maintaining the file system, using security protocols, or with the build system.

You can also install network-time-protocol (ntp) packages that will keep the Raspberry Pi time synchronized automatically:

```
sudo apt-get update

sudo apt-get install ntp
```

4. Reboot the Raspberry Pi to have these changes go into effect:

```
sudo reboot
```

Setting up the IoTivity Development Environment

1. Before you start setting up the IoTivity-specific software development environment, you'll need to login again and then make sure the base Linux OS is up to date by first "updating" information about available packages from the internet, then then upgrading installed packages if updates are available:

```
sudo apt-get update

sudo apt-get upgrade
```

Reply "y" and press enter when asked to continue. You'll see something like this:

```
pi@raspberrypi:~ $ sudo apt-get update
Get:1 http://archive.raspberrypi.org/debian stretch InRelease [25.3 kB]
Get:2 http://mirrordirector.raspbian.org/raspbian stretch InRelease [15.0 kB]
Get:3 http://archive.raspberrypi.org/debian stretch/main armhf Packages [117 kB]
Get:4 http://mirrordirector.raspbian.org/raspbian stretch/main armhf Packages [1
1.7 MB]
Get:5 http://archive.raspberrypi.org/debian stretch/ui armhf Packages [26.9 kB]
Fetched 11.9 MB in 19s (623 kB/s)
Reading package lists... Done
pi@raspberrypi:~ $ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  bluez dhcpcd5 libperl5.24 libwbclient0 perl perl-base perl-modules-5.24
  raspi-config raspi-copies-and-fills samba-common
10 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 8,143 kB of archives.
After this operation, 9,216 B of additional disk space will be used.
Do you want to continue? [Y/n]
```

2. Next, use the same apt-get tool to install the tools and external libraries needed for the IoTivity build environment. As above, reply "y" and press enter when asked to continue. (Note this is a long command line you can cut and paste from below. If you're typing it in, the "\" at the end of a line means the next line is a continuation, or you can type it all in (without the "\" characters as one long line.)

```
sudo apt-get install build-essential git scons libtool \
autoconf valgrind doxygen wget unzip cmake libboost-dev \
libboost-program-options-dev libboost-thread-dev uuid-dev \
libexpat1-dev libglib2.0-dev libsqlite3-dev libcurl4-gnutls-dev
```

This command will install, if not already there, the gcc c++ compiler, Git version control system, build tools (such as scons, libtool, and auto conf), analysis tools (such as valgrind), API documentation tool (doxygen), utilities (such as wget and unzip), and other tools and external libraries needed by the project.

Getting the IoTivity source code

As with many open source projects, IoTivity is distributed as source code that you'll build in your development environment. These next steps connect to the internet to fetch the IoTivity source code and some additional tools.

1. In your home directory, make an "iot" working directory:

```
mkdir ~/iot

cd ~/iot
```

2. Clone the iotivity repository code using Git commands, change to the directory created, and check out the IoTivity 1.3 release branch:

```
git clone https://gerrit.iotivity.org/gerrit/iotivity

cd iotivity

git checkout 1.3-rel
```

3. Download additional tool sources from these repos:

```
git clone https://github.com/01org/tinycbor.git \
extlibs/tinycbor/tinycbor -b v0.4.1

git clone https://github.com/ARMmbed/mbedtls.git \
extlibs/mbedtls/mbedtls -b mbedtls-2.4.2
```

4. Download the application sample source code (Note: this step won't be needed when the sample code is included in the IoTivity repo, but it's required for now):

```
git fetch origin refs/changes/13/22513/15

git checkout FETCH_HEAD
```

Setting up MRAA library

MRAA is a C Linux library with bindings to C++, Java, Python, and Node.js (JavaScript) that lets you write portable code accessing low speed IO interfaces for sensors and actuators across a variety of hardware platforms. The mraa library is provided as sources that you'll build to create the needed runtime library. You'll also build and run a short sample that verifies mraa has been set up properly for the Raspberry Pi hardware.

1. Clone the mraa source into the "iot" working directory we created earlier:

```
cd ~/iot

git clone https://github.com/intel-iot-devkit/mraa.git
```

2. Make a working directory in the mraa directory we've cloned, change to that directory, build the mraa library, and install it:

```
mkdir mraa/build && cd mraa/build

cmake .. && make && sudo make install
```

Note: You'll see some warnings reported during the cmake run that are expected and OK.

3. Verify mraa is installed properly by running a blink-io example (provided with mraa). Because the application directly accesses hardware, you need to run it as root with the "sudo" command:

```
cd ~/iot/mraa/build/examples && sudo ./blink-io 7
```

If all is good, the two LEDs on the Enviro pHAT board will blink on and off every second, along with the words on and off displaying on the command line console. (Press CTL-C to stop.)

```
pi@raspberrypi:~/iot/mraa/build $ cd ~/iot/mraa/build/examples && sudo ./blink-io 7
MRAA Version: v1.8.0-21-g64fe50f
Starting Blinking on IO7
Initialised pin7
off
on
off
on
off
on
off
on
off
on
off
on
off
```

Building the IoTivity Server/Client Sample Applications

Now that all the required sources and tools are on the Raspberry Pi system, and mraa is working, it's time to build the IoTivity sample application code.

1. Change back to the iotivity directory and build the sample code using the "scons" tool:

```
cd ~/iot/iotivity

scons examples/OCFSecure -j 2 TARGET_TRANSPORT=IP
```

Here's an explanation of this command:

- **scons** is an open source software construction tool, an improved and more functional substitute for the classic "make" utility.
- The "`examples/OCFSecure`" parameter restricts the building process to the sample code directory and its dependencies.
- The "`-j 2`" flag will utilize two (of the four) processor cores available on the Raspberry Pi 3, improving performance.
- The "`TARGET_TRANSPORT=IP`" parameter restricts building to only implement the IP transport protocol and not others such as such as Bluetooth Low Energy (BLE) or Near Field Communication (NFC).

2. This build should take less than 10 minutes. Once you see the message: "`scons: done building targets.`", and there are no obvious errors, the build was successful and we can try running the OCFSecure sample server and client applications.

Running the IoTivity Server/Client Sample Applications

A pair of client and server application samples are included in the software for this kit to give you a working example of building a server for an OCF hardware "switch" device (in our case an LED), together with a client that interacts with this server.

Now, let's run the sample applications:

1. Change to the output directory where the sample application executable files were created. (Note the directory name armv7l ends with the lower-case letter "l" not a digit one.) Because the application directly accesses hardware (as did the blink-io mraa test), it needs to run as root (administrator) with the "sudo" command. Note too, you're going to run the server as a background app so we can run the client app in the foreground in the next step:

```
cd ~/iot/iotivity/out/linux/armv7l/release/examples/OCFSecure

sudo ./server &
```

If you get an error saying "`./server: error while loading shared libraries: libmraa.so.1: cannot open shared object file: No such file or directory`", use this command to fix the error and try running the server application again:

```
sudo env LD_LIBRARY_PATH=~/iot/mraa/build/src && sudo ldconfig
```

If you forgot to add the trailing "&" to tell the shell to run the server in the background, press CTRL-C to end the server, and start the server app again with the trailing "&".

The server application will report that it's initializing and reading some files, creating the SWITCH resource (for the LED), and finally reporting, "`Server is running, press ctrl+c to stop…`"

```
pi@raspberrypi:~/iot/iotivity/out/linux/armv7l/release/examples/OCFSecure $ sudo ./server
57:53.778 DEBUG: SERVER_APP: [main] Initializing and registering persistentstorage
57:53.778 DEBUG: SERVER_APP: [main] Initializing IoTivity stack for server
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: wb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.788 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: ocf_svr_db_server.dat with mode: wb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: device_properties.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: device_properties.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: device_properties.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: device_properties.dat with mode: rb
57:53.798 DEBUG: SERVER_APP: [ServerFOpen] reading file: device_properties.dat with mode: wb
57:53.808 INFO: SERVER_APP: [CreateResource] Created SWITCH resource (0): OC_STACK_OK
57:53.808 INFO: SERVER_APP: [main] Server is running, press ctrl+c to stop...
```

2. With the server app running in the background, you can run the client application in the foreground. Since the client doesn't (directly) access the hardware, it doesn't need to run with sudo (admin) access. If you don't see the command prompt, press return, and then type:

```
./client
```

Both the server and client applications will write INFO and DEBUG information to the console, identified as SERVER_APP or CLIENT_APP.

The client application has a very simple interface that lets you interact with resources via IoTivity API calls to the server. Using the client, you identify by <resource number>, the resource you want to interact with, together with a request to either GET (read) or POST (write) information to that resource.

If you press a return, the client displays the list of discovered resources. Press return again if you don't see the discovered /switch resource when the client first starts up. As shown below, the /switch resource was discovered as resource number 21:

```
============================================================
Discovered Resources:
        #0: /oic/res    @224.0.1.187:5683       resource type: nil
        #1: /oic/sec/doxm       @fe80::996e:7dc6:37f3:5311%eth0:57495   resource type: oic.r.doxm
        #2: /oic/sec/pstat      @fe80::996e:7dc6:37f3:5311%eth0:0       resource type: oic.r.pstat
        #3: /oic/sec/acl2       @192.168.1.24:57791     resource type: oic.r.acl2
        #4: /oic/sec/cred       @192.168.1.24:57791     resource type: oic.r.cred
        #5: /oic/sec/crl        @192.168.1.24:57791     resource type: oic.r.crl
        #6: /oic/sec/csr        @192.168.1.24:57791     resource type: oic.r.csr
        #7: /oic/sec/roles      @192.168.1.24:57791     resource type: oic.r.roles
        #8: /oic/d      @192.168.1.24:57791     resource type: oic.wk.d
        #9: /oic/p      @192.168.1.24:0 resource type: oic.wk.p
        #10: /introspection     @192.168.1.24:0 resource type: oic.wk.introspection
        #11: /oic/sec/doxm      @fe80::996e:7dc6:37f3:5311%eth0:54464   resource type: oic.r.doxm
        #12: /oic/sec/pstat     @fe80::996e:7dc6:37f3:5311%eth0:0       resource type: oic.r.pstat
        #13: /oic/sec/acl2      @192.168.1.24:43330     resource type: oic.r.acl2
        #14: /oic/sec/cred      @192.168.1.24:43330     resource type: oic.r.cred
        #15: /oic/sec/crl       @192.168.1.24:43330     resource type: oic.r.crl
        #16: /oic/sec/csr       @192.168.1.24:43330     resource type: oic.r.csr
        #17: /oic/sec/roles     @192.168.1.24:43330     resource type: oic.r.roles
        #18: /oic/d     @192.168.1.24:43330     resource type: oic.wk.d
        #19: /oic/p     @192.168.1.24:0 resource type: oic.wk.p
        #20: /introspection     @192.168.1.24:0 resource type: oic.wk.introspection
        #21: /switch    @192.168.1.24:43330     resource type: oic.r.switch.binary

Request Methods:        1: GET  4: POST
Usage:<resource number> <request method> :
```

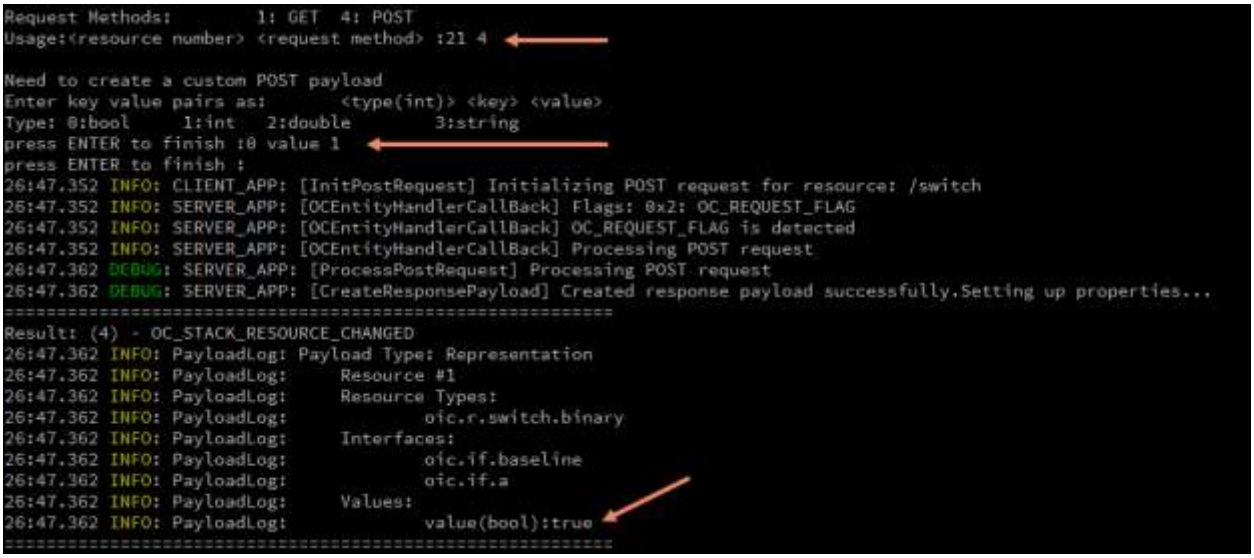
3. Send a GET request to the /switch resource by typing "21 1" and press return: 21 for the /switch resource and 1 for a GET request. After some INFO and DEBUG messages go by, you'll see the result as:

```
============================================================
Result: (0) - OC_STACK_OK
05:47.637 INFO: PayloadLog: Payload Type: Representation
05:47.637 INFO: PayloadLog:     Resource #1
05:47.637 INFO: PayloadLog:     Resource Types:
05:47.637 INFO: PayloadLog:             oic.r.switch.binary
05:47.637 INFO: PayloadLog:     Interfaces:
05:47.637 INFO: PayloadLog:             oic.if.baseline
05:47.637 INFO: PayloadLog:             oic.if.a
05:47.637 INFO: PayloadLog:     Values:
05:47.647 INFO: PayloadLog:             value(bool):false
============================================================
```

This response indicates the result was OC_STACK_OK and displays the payload returned. You can see the value of a Boolean property named "value" is false, indicating the LED is off.

(If you don't see this response on your screen, but you do see the "Discovered Resources" list, the response may have just scrolled off your screen. If you're connecting via ssh, try making your terminal screen taller, or simply scroll the windows up to see the output.)

4. Next, try changing the property "value" to turn the LED on, by sending a POST request to the /switch resource. Start by typing "21 4" and enter. You'll be prompted to create a custom payload to POST (send) to the server. Select a property type from the list (in our case Boolean, so 0), and enter the property key (in our case "value") and the property value (1 for true), and press enter. If the custom payload required multiple key/value pairs to POST, we could enter more, but for our example that's all, so press enter to finish:



Notice that the response message indicates the result is OC_STACK_RESOURCE_CHANGED and the values that were affected, namely the Boolean property "value" is now true (and the LED is on). You can verify this by repeating the GET request you previously did.

5. Exit the client application by pressing CTRL-C. Then, exit the server app running in the background, by using the "fg" command to bring the background job to the foreground and press CTRL-C to exit the server.

Shutting down the Raspberry Pi

Like all computers, just pulling the power plug on the Raspberry Pi is a bad way to turn it off, and can cause data loss on the microSD card. You must shut down the Raspberry Pi properly and give the operating system a chance to cleanly close down system services and the file system.

From the command line use this command:

```
sudo shutdown -h now
```

Then wait until the green LED near the micro-USB power connector on the Raspberry Pi blinks a few times and then stays off. (This green LED flashes when the board is writing to the microSD

card.) At this point it's best to unplug the power supply from the wall rather than unplug the micro-USB connector – connectors like these have a limited service lifetime.

Wrap-up and Next Steps

With that, you're done setting up the IoTivity development environment on the Raspberry Pi 3 and verified the sample applications can be built and run. Over time, we plan to add additional samples along with updated documentation to go with them, to give you more examples of using the IoTivity APIs.

Now it's up to you to explore the source code for the sample code (in the ~/iot/iotivity/examples/OCFSecure directory). Read through the source comments and the included README file for more information.

If you're looking for some ideas, try modifying the server app to add access to another device on the Enviro pHAT card, for example returning the temperature or reading the light and color value detected by the on-board sensors.

OCF Membership Resources

The OCF basic membership level is a no-cost way to get read only rights for members-only materials and access the OCF Certification Test Tools (CTT) for pre-testing purposes. Visit the OCF membership page to learn more. If your company is already an OCF member, you can also get access to additional resources and participate in the many OCF work and task groups that are helping to create this interoperable and secure "network of everything" environment.

Appendix A: Headless access to the Raspberry Pi 3 via ssh

If it's not convenient to connect a monitor and keyboard to the Raspberry Pi 3, you can also get command line access via ssh to the board from another computer connected to the same network. Alas, to use ssh, you'll need to know the IP address of the Raspberry Pi 3 and this can be tricky.

Finding the IP address of the Raspberry Pi

Because of the nearly endless varieties of network configurations and equipment, there are many different ways to find the IP address of your board on the network. (You may need to contact a system administrator or customer support for your particular network setup.) Here are some recommended approaches.

Try a local name

If your Raspberry Pi is the only Raspberry Pi board on your local network, you may be able to connect using the board's hostname (raspberrypi by default) instead of using its IP address. If you're already familiar with using ssh or an ssh client such as PuTTY, try connecting using the name **raspberrypi.local**

For example, use the command line:

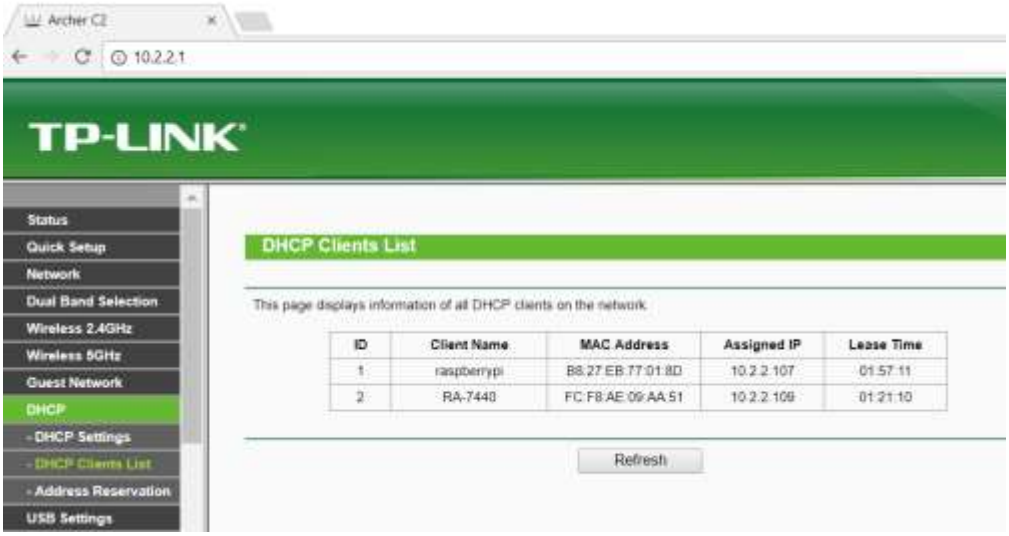
```
ssh pi@raspberrypi.local
```

or use an ssh client and specify **raspberrypi.local** as the hostname to connect to. If you changed the boards default hostname, use that new name instead.

Ask your router for connected devices

If you're in a home environment, you likely have physical access to your home router and can connect the Raspberry Pi's network cable directly to an available router port. Once you do that, use your computer's web browser to access the router's web-based admin page to login to the router and show you the IP address of connected devices. Most home routers are set up at the same IP address as your client, but with the last number as "1", if your computer has an IP address of 10.2.2.109, your router would likely be at http://10.2.2.1.

For example, this home's router DHCP client list shows two connected devices:



In this home environment, the device named "raspberrypi" has the IP address 10.2.2.107.

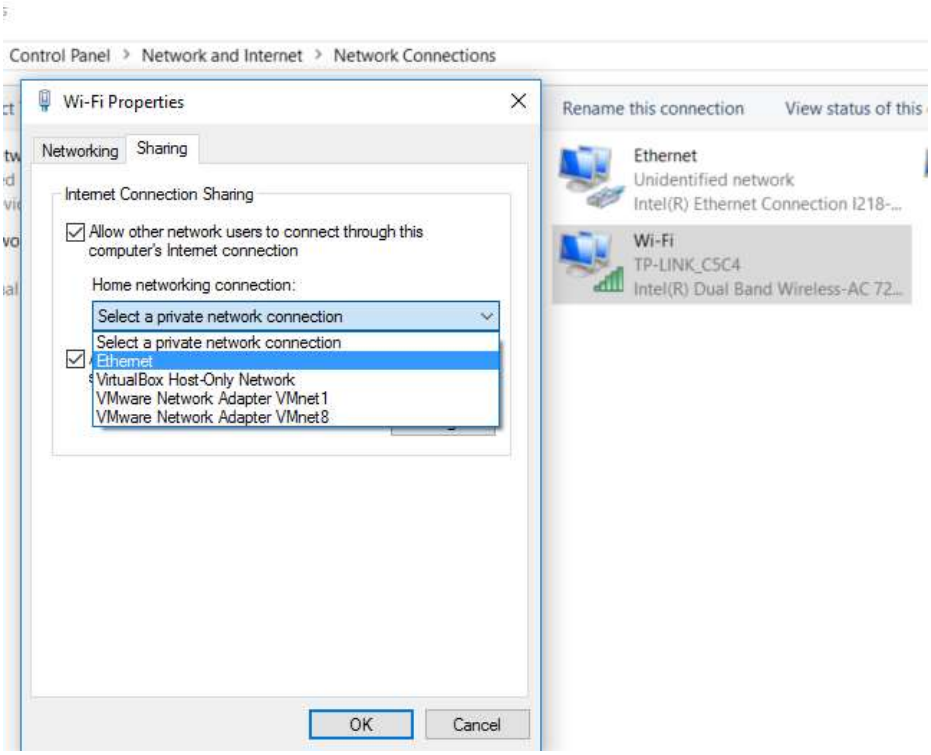
Direct network connection to your PC

If a direct connection to a router is not an option, and your host computer has an available Ethernet port, you can share your host computer's internet connection with the Raspberry Pi. You will need to use a wireless network connection for your host computer to free up its network port.) If you're in a corporate environment, sharing your host computer's network access might not work or not be allowed, so check with your corporate network administrators.

Connect an Ethernet cable from the Raspberry Pi to your host computer's Ethernet port. Then you're going to map the host computer's internet connection from Wi-Fi to the Ethernet port to reach the Pi. How you do this depends on your host computer's operating system:

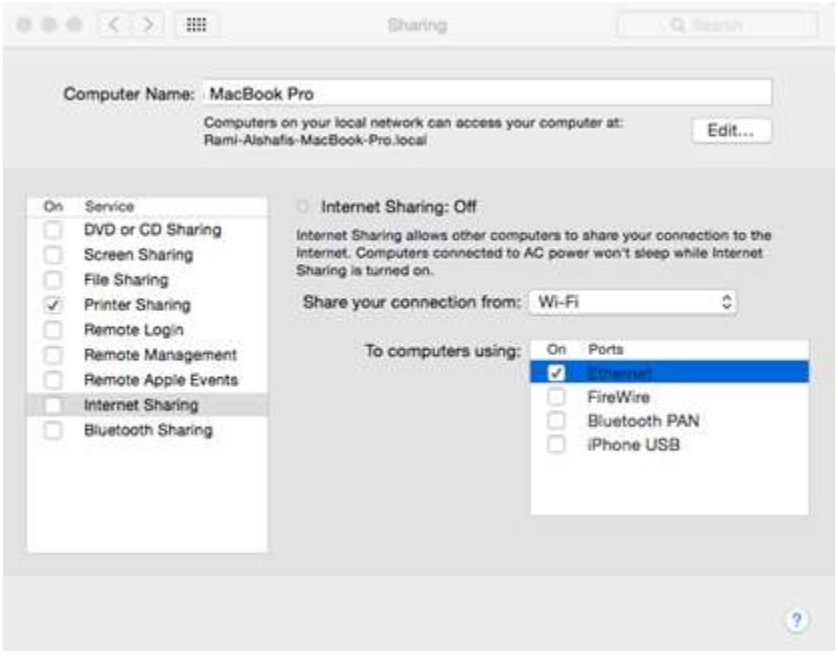
- On a **Windows** host, open up the network connections (control panel -> Network and Internet -> Network Connections) and you should see both Ethernet and Wi-Fi.

  Right-click on the Wi-Fi, select properties, click on the sharing tab and check both allow options, and from the drop down menu select the Ethernet connection that is attached to the Raspberry Pi. Sometimes it is called Ethernet or Ethernet4. Other times, it is called LAN or Local Area Network. (You can connect and disconnect the Raspberry Pi network cable and watch which Ethernet port is responding to that change in the list of the network adaptors in the Network Connections window.)

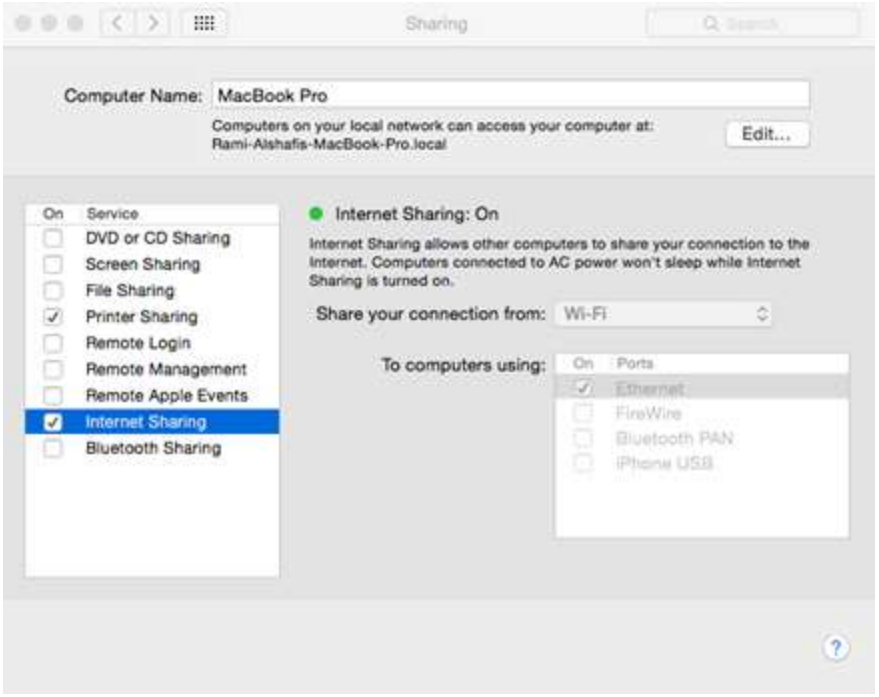


Remember this name for a later step! Click OK to make these changes and then restart your windows PC.

- On a **macOS** host, open up system preferences, click on the Sharing settings, click on the "Internet Sharing" and then select the Ethernet port (where the Raspberry Pi is connected):

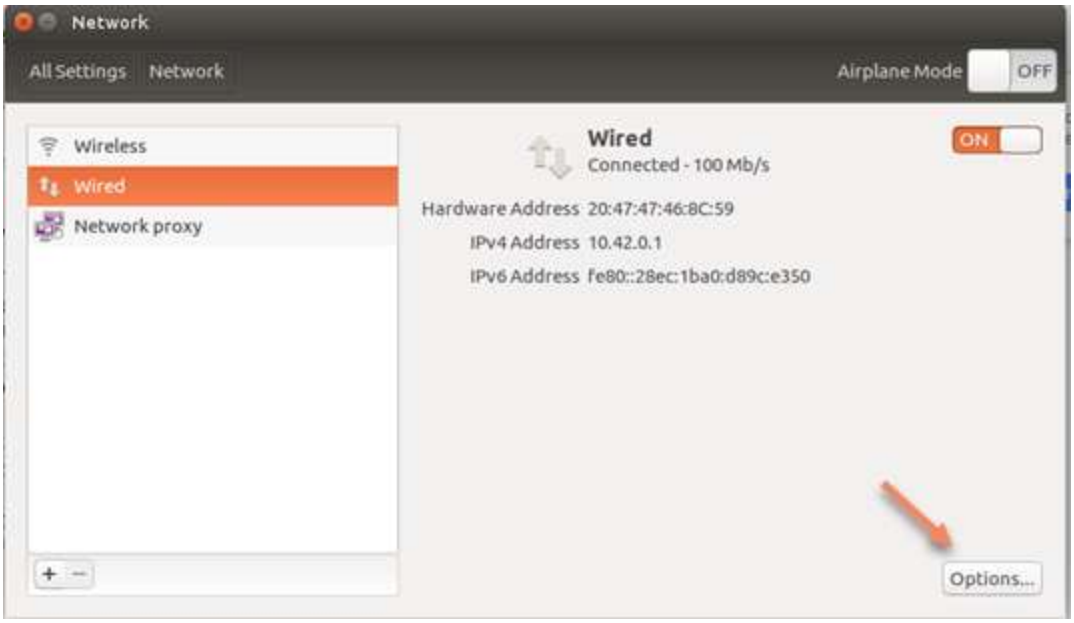


Next, click again on the "Internet Sharing" option and confirm the changed settings.

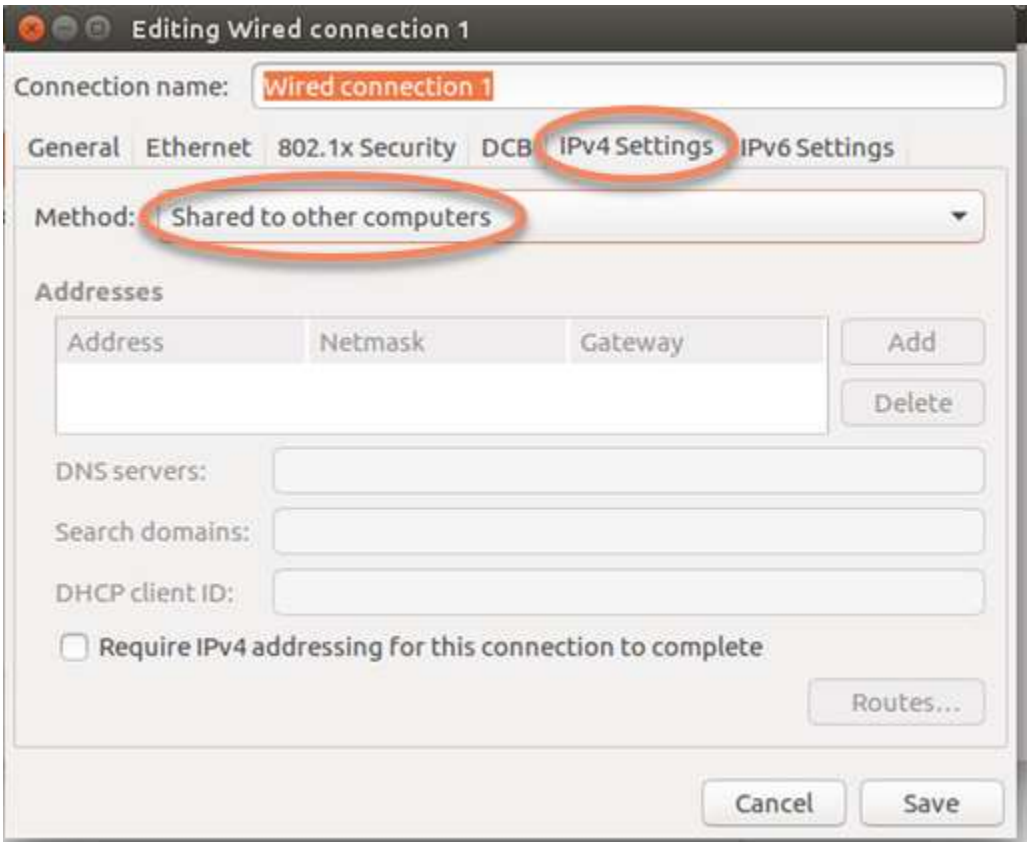


If you see the "Internet Sharing:" flag is set to On and lit green then you are good to go. Restart your host computer now.

- For a **Linux (Ubuntu)** host, open up System Settings, select Network, then select Wired, click on "Options…".



Select the "IPv4 Settings" tab. From the Method drop menu, select "Shared to other computers" and click save.



Restart your host computer now.

After restarting your host computer, your Raspberry Pi will be sharing your host computer's network connection and get its own IP address. Now we have to discover what that IP address is.

Scan your network using Nmap

The nmap tool is a free and open source command line utility for network discovery and can be run on Windows, Linux, and macOS systems. (Again, if you're in a corporate environment or using a corporate laptop, you may need to verify with your system admin before installing this tool.)

- For **Windows** users, go to https://nmap.org/download.html#windows and download the latest stable release self-installer (currently nmap-7.60-setup.exe). Run the downloaded installer and use the defaults.
- For **macOS** users, go to https://nmap.org/download.htm#macosx and download the latest stable release installer (currently https://nmap.org/dist/nmap-7.60.dmg). Run the downloaded installer and use the defaults.
- For **Linux** users, use your system's package manager to fetch and install nmap. On Ubuntu, for example, you can use:

```
sudo apt-get install nmap
```

or on RedHat, use:

```
sudo dnf install nmap
```

For nmap to do its work, the Raspberry Pi and your host computer must be on the same network and the Raspberry Pi must be powered on. You'll be using nmap to scan the entire local network looking for devices. But first, you need to find the IP address of your host computer:

- On **Linux**, use the hostname command to display your system's IP address:

```
hostname –I
```

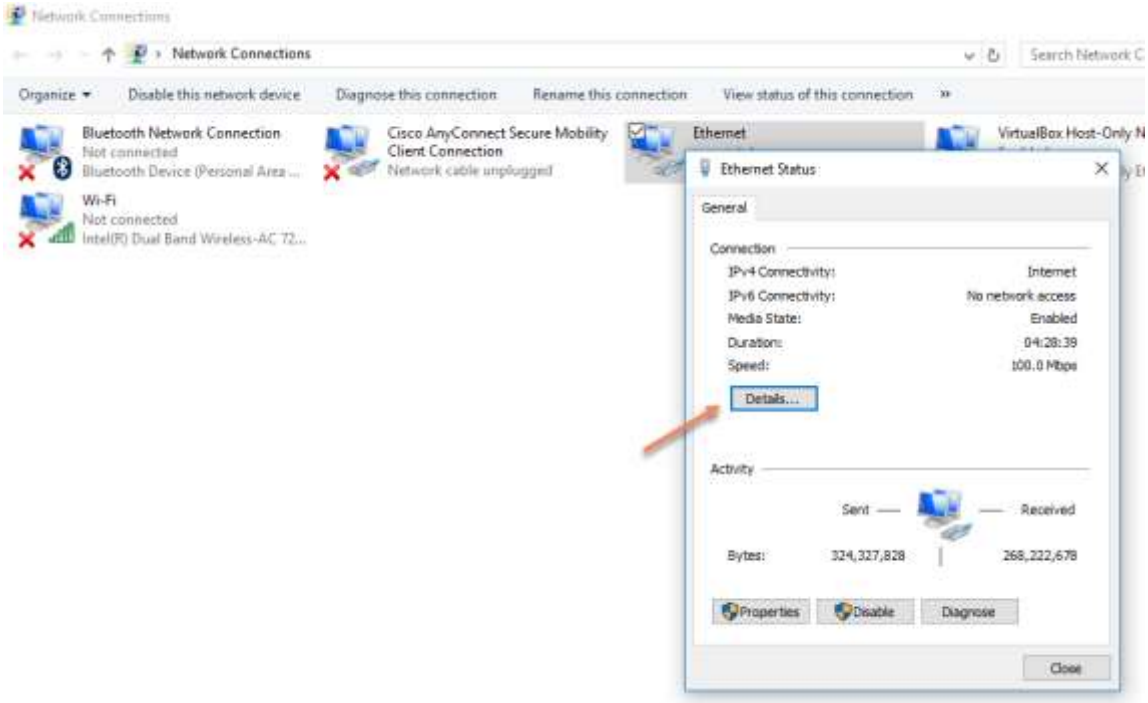
- On **Windows**, from a command line, use the "ipconfig" command:

```
C:\Users\gsg>ipconfig

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : mycompany.com
   IPv4 Address. . . . . . . . . . . : 10.54.74.154
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.54.74.0
```

or go through Control Panel to examine your active Network Connection status details:



- On **macOS**, go to **System Preferences**, then **Network**, and select your active network connection to view the IP address.

With your host computers IP address in hand, run the "nmap" command line tool to find systems on the same network. Specify the IP address of your host computer with /24 appended and use grep to look for output mentioning raspberry, for example:

```
nmap -n -sV 10.54.74.154/24 | grep raspberry
```

and you'll see output something like this:

```
Nmap scan report for raspberrypi.mycompany.com (10.54.74.130)
```

Note the IP address of your Raspberry Pi system so we can connect to it. It's worth mentioning here that your Raspberry Pi will keep this IP address only as long as it's powered up. If you turn it off for a while and then back on, it will likely get a different IP address.

If nmap reports an "All hosts (65536) are up" message, this means something went wrong in sharing the internet connection. This error is common on a corporate network or on corporate laptops. If this is not the case, then check that the Ethernet driver is not disabled or corrupted and it is updated. Also check that the firewall is disabled (for debugging you need to configure the firewall to allow the right traffic). For more information about nmap, read the nmap documentation at http://nmap.org.

If you don't see any output, then no device on this subnet reported as your Raspberry Pi. Make sure your Raspberry Pi is connected to the same network as your host computer, and is powered

on. Try running the nmap command again, without the "grep", and look through what systems were discovered.

Connecting to the Raspberry Pi 3 via ssh

Once you have the IP address of the Raspberry Pi, you can connect to it via ssh using an ssh client (such as PuTTY) on Windows, or the ssh command in a terminal windows on Linux or macOS.

If your ssh or PuTTY connection fails in the steps below, this could mean the Raspberry Pi's IP address has changed from when you last discovered it. This can happen if it's been powered off for a while and you turn it on and try to connect with an old IP address. In this case, simply use the steps noted above to find its IP address again.

Using ssh on Linux or macOS command line

The ssh command is available on Linux and macOS. Open a terminal windows and run the ssh command with the **pi** username and IP address of the Raspberry Pi 3.

The first time you connect to a new computer with SSH you'll see a security alert, to which you should reply "yes". Enter the password (**raspberry**, unless you changed it) and you're connected:

```
~$ ssh pi@10.54.74.130

The authenticity of host '10.54.74.130 (10.54.74.130)' can't be established.
ECDSA key fingerprint is SHA256:btHbcrzQVfYw6QwMaVM2pW7yvPBTkybpkO78hFdYPpA.
Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '10.54.74.130' (ECDSA) to the list of known hosts.

pi@10.54.74.130's password:
Linux raspberrypi 4.9.41-v7+ #1023 SMP Tue Aug 8 16:00:15 BST 2017 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 28 23:24:30 2017 from 10.7.200.147

SSH is enabled and the default password for the 'pi' user has not been
changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to
set a new password.
```

```
pi@raspberrypi:~ $
```

You'll see the security reminder that the default username and password have not been changed (if you didn't do that yet). See the Raspberry Pi security change recommendations listed in https://www.raspberrypi.org/documentation/configuration/security.md.

You can configure the ssh client to prevent the SSH session from closing because of inactivity by adding the following line to your ~/.ssh/config file:

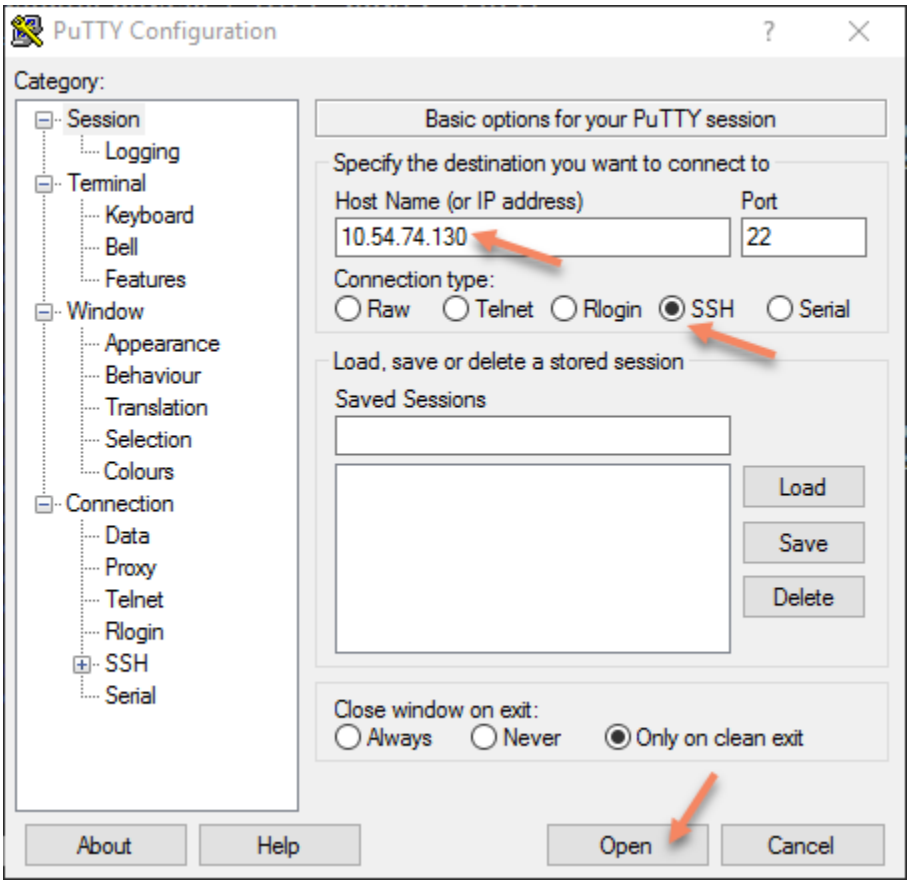
```
ServerAliveInterval 300
```

This configuration will send a no-op null packet to the server, to keep the connection alive, every 300 seconds (5 minutes).

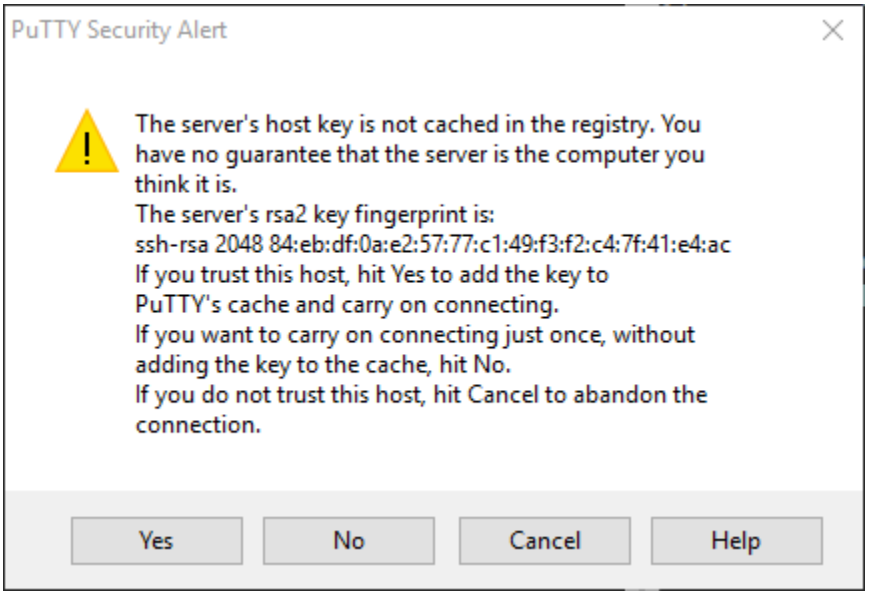
Using PuTTY on Windows

On Windows, you'll need to download an SSH client to your Windows computer. One of the most commonly used clients is PuTTY and can be downloaded from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. Look for the MSI Windows Installer links for your OS version (most likely 64-bit windows). You can also read their online documentation for full information about PuTTY.

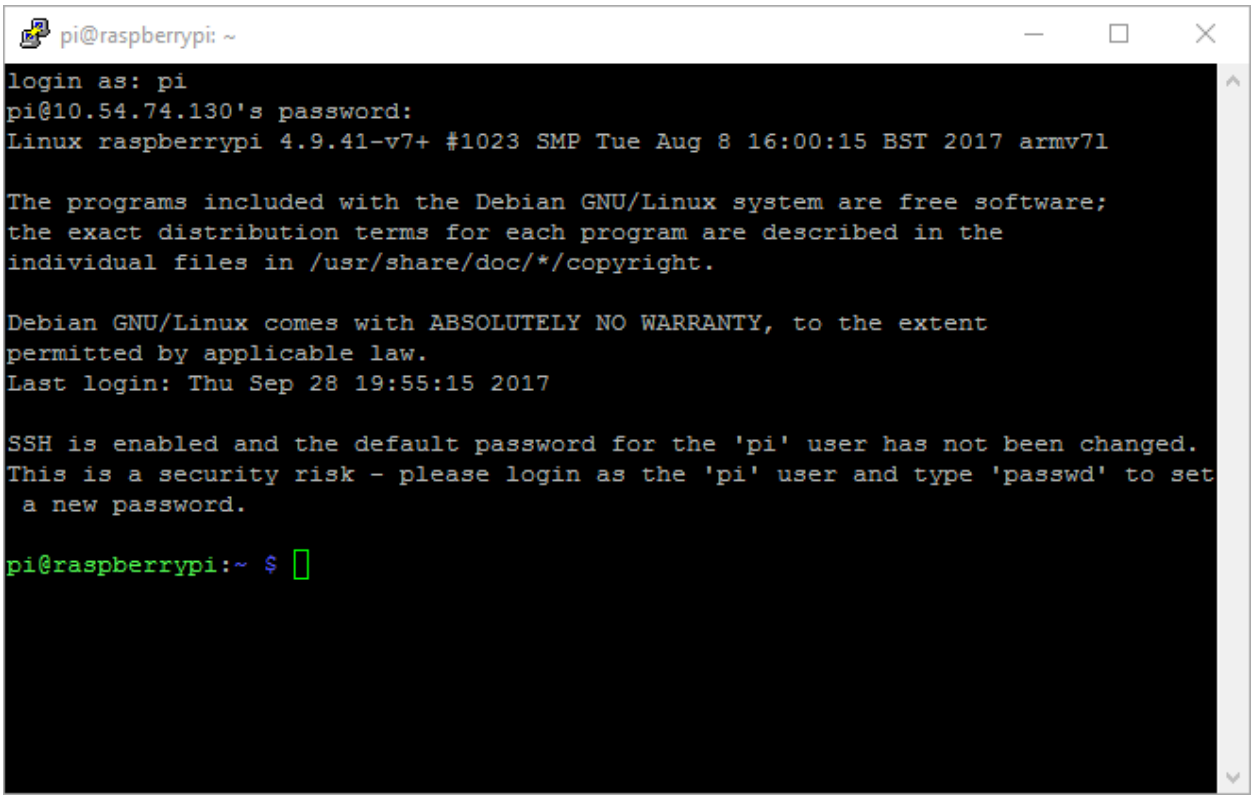
After installing PuTTY, launch the PuTTY application and input the Raspberry Pi IP address in the hostname field, verify the SSH connection type is selected, and click open:



The first time you connect to a new computer with SSH you'll see a security alert:



Click on Yes to accept the connection. Next you'll be prompted for the username (**pi**) and password (**raspberry**, unless you changed it), and you'll be connected:



You'll see the security reminder that the default username and password have not been changed (if you haven't done that yet). See https://www.raspberrypi.org/documentation/configuration/security.md for security change recommendations.

Refer to the PuTTY documentation to configure options such as "keep alive" that will prevent the SSH session from closing because of inactivity. In your session properties, go to **Connection** and under **Sending of null packets to keep session active**, set **Seconds between keepalives (0 to turn off)** to 300 (5 minutes).

From here, you can interact with the Raspberry Pi as if you were connected with a keyboard and monitor.

Type "exit" to close the PuTTY window when done. (Be sure to properly shut down the Raspbian OS before powering off the Raspberry Pi, as explained in the "Shutting Down" section earlier in this guide.)

Appendix B: Creating the microSD card bootable image

The kit is distributed with an 8GB microSD card, preloaded with a bootable Raspbian Lite image (without a graphical desktop) and configured to boot with SSH enabled. The following instructions will show how you can create your own microSD card with the same bootable image.

Download the Raspbian OS Image

Raspbian is a Linux variant designed and configured to work well with the Raspberry Pi. Official Raspbian OS images are available from the https://www.raspberrypi.org/downloads/raspbian/ web page. On your Windows, macOS, or Linux host, use your web browser to download the ZIP file for Raspbian Stretch Lite:

Write the image to the microSD card

You'll need special image burning software to write this file to the microSD card in a way that creates a bootable image. As described in the Raspberry Pi installing images guide, we recommend using a free open-source graphical SD card writing tool called Etcher. This tool runs on Windows, macOS, and Linux systems and is available for download from http://etcher.io.

Here's a summary of image burning instructions:

1. Download Etcher and install it on your host computer.
2. Connect an 8GB (or larger) microSD card to your host computer with an SD or USB carrier or an external adapter. If you're using an SD-card carrier, verify the write-protect switch is not enabled.
3. Open Etcher and select the .ZIP file you downloaded with the Raspbian image as the source.
4. Select the SD card you're writing to as the destination. Note you'll be completely overwriting the card's contents. If Etcher can't find an SD card to write to, you may

need to reformat the microSD card. (Instructions for doing a low-level format for an SD card can be found on the SD Association website.)

5. Review your selections and click "Flash!" to begin writing to the microSD card.

6. When done, exit Etcher, unmount or eject the microSD card, and remove it from your computer.

Enabling SSH

For security reasons, Raspbian has secure shell (SSH) access disabled by default. In a "headless" configuration of the Raspberry Pi (without a monitor and keyboard connected), we'll need SSH access enabled when the board is turned on. (If you're going to use a keyboard and monitor and not connect via ssh, you can skip this step.)

Here's how to automatically enable ssh access on the Raspberry Pi microSD card every time it powers on:

1. Connect the microSD card to your host computer. (Note that Etcher will unmount the microSD card when finished, so you'll need to remove and reconnect the microSD card.)
2. Create an empty file named "ssh" (no file extension) in the root directory of the microSD drive. This file must be named "ssh" and not, for example, "ssh.txt". On a Windows* system for example, you can double click the microSD card drive, then right-click within the open folder and create a new text document (named "ssh.txt" and delete the ".txt" extension before saving).
3. Unmount or eject the microSD card, and remove it from your computer

And with that, the microSD card is ready to use to boot the Raspberry Pi and access it remotely via ssh.