



## Smart Builds, Azure IoT Hub, Raspberry Pi for Mac OS Developers (v.7)

This guide is intended for developers with an Apple Mac running Mac OS and building “Internet of Things” solutions on Azure [IoT Hub](#), [Stream Analytics](#), [Power BI](#), [Machine Learning](#), [Functions](#), Data services and more.

The JSON data streamed from this Smart Building sample is compatible with, and intended to be used in conjunction with Section 4 “Microsoft Azure Cloud Development” of the [Maker Den User Guide](#).

This guide assumes you have a working understanding of Mac OS, some development experience and some knowledge of [Python](#).

The Smart Building Environment sample is written in Python3 and has been tested on Mac OS, Raspberry Pi, Windows and Ubuntu Linux. It should run on any platform supporting Python3 and the required pip3 packages such as a Beagleboard.

When the Smart Building Environment sample is run on your Apple Mac the environmental data is requested from the free [Open Weather Map](#) service.

Running the Smart Building Environment sample on your Apple Mac will enable you to make a fast start streaming sensor data to Azure IoT Hub.

With sensor data in the Azure cloud you can use advanced Azure services such as Stream Analytics, Power BI, Machine Learning, Functions, Data services and more to build amazing solutions.

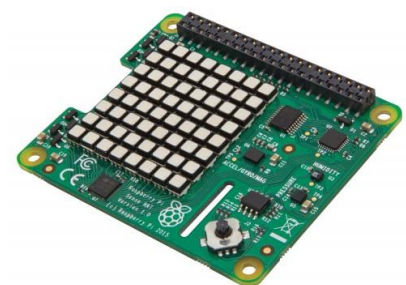
The goal is to extend the Smart Building sample app and run on the Raspberry Pi using local environmental data from a Raspberry Pi Sense HAT.

The Raspberry Pi Sense HAT has a great selection of sensors, a display and a joystick.

There are temperature, barometric and humidity sensors. You could use the accelerometer to simulate an earthquake, a lift vibrating in need of maintenance.

Joystick presses could be used to simulate the number of people in an area or using a facility such as the bathroom.

The magnetometer (compass) to simulate the sun or wind direction. You could use the LED display to graph data, simulate blinds opening and closing or lights being turned off and on. These are just some ideas, let your imagination run wild.



---

## This Guide Covers

1. Setting up Raspbian Linux for the Raspberry Pi from an Apple Mac
2. Setting up Internet Sharing on Mac OS
3. Setting up your Raspberry Pi
4. Installing Recommended Mac OS Packages
5. Provisioning an Azure IoT Hub Device
6. Azure IoT Hub Device Identity Management
7. Smart Building Environment Sample Configuration
8. Developer Workflow

---

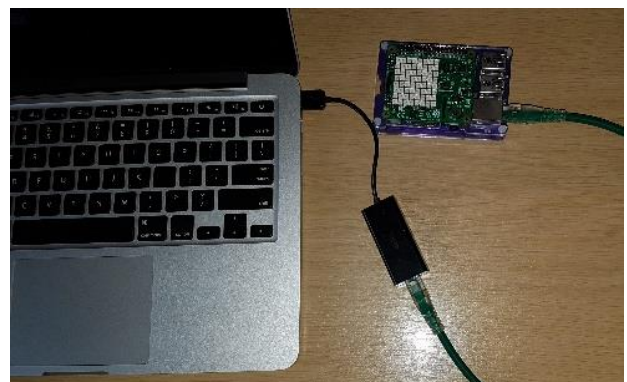
## Setting up Raspbian Linux for the Raspberry Pi from an Apple Mac

- 1) Download from <https://www.raspberrypi.org/downloads/raspbian/>
- 2) Setting up the Raspberry Pi operating system on an SD Card from Mac OS  
<https://www.raspberrypi.org/documentation/installation/installing-images/mac.md>
- 3) Eject the SD card from Mac OS

---

## Powering up your Raspberry Pi

- 1) Ensure the SD Card is properly seated in the Raspberry Pi
- 2) Connected the Raspberry Pi via the Ethernet cable to the USB Ethernet Dongle that is plugged into your Apple Mac
- 3) Power on the Raspberry Pi
- 4) Wait for a couple of minutes for the Raspberry Pi to perform its initial boot up sequence.
- 5) The green disk activity will stop flashing when it is ready



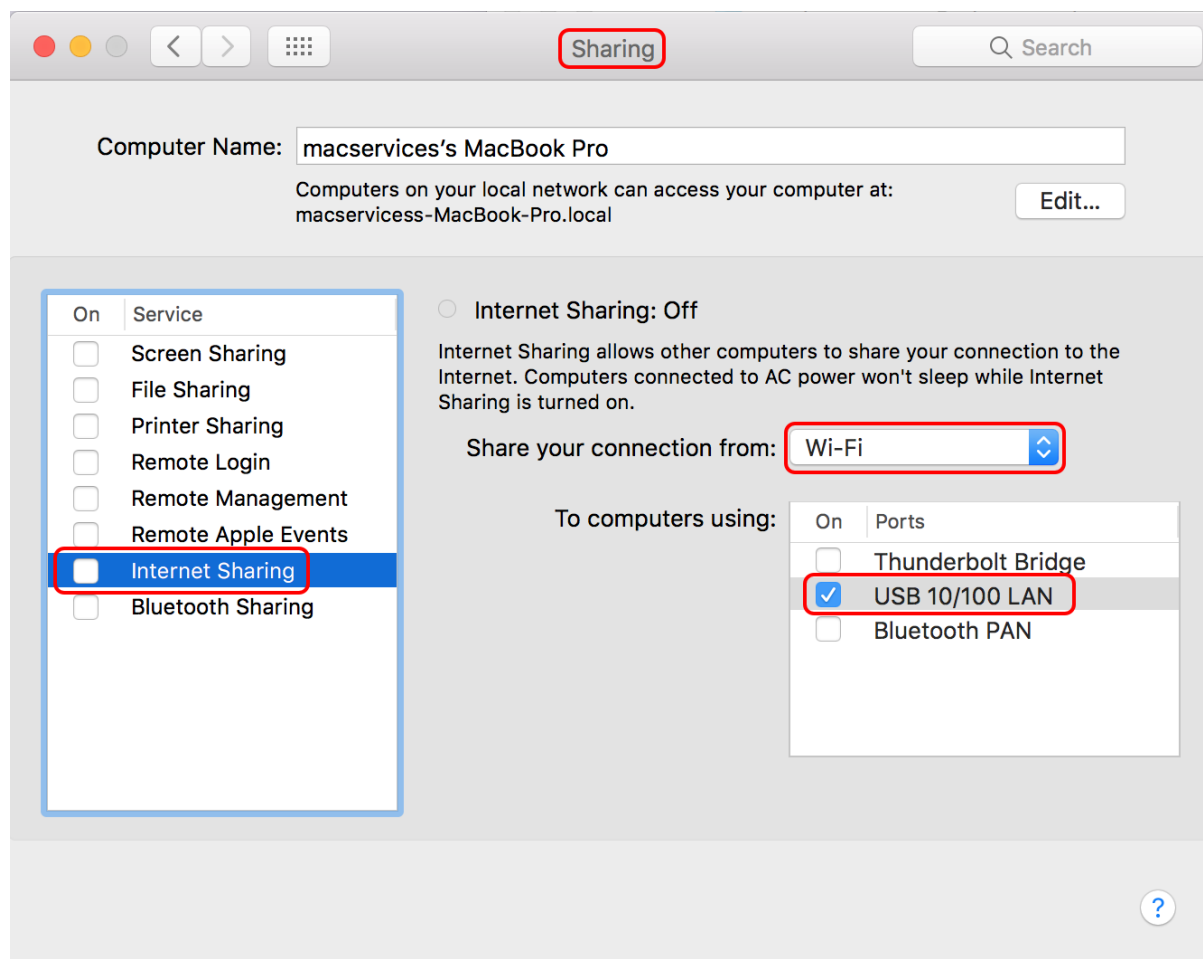
---

## Setting up Internet Sharing on Mac OS

Enabling Internet Sharing on your Apple Mac will allow internet requests from the Raspberry Pi to pass through your Apple Mac and its Wifi connection to the Internet. This connection is needed to update the Raspberry Pi with the latest patches and allow IoT traffic to flow between the Raspberry Pi and Azure IoT Hub.

See [https://support.apple.com/kb/PH18704?locale=en\\_AU](https://support.apple.com/kb/PH18704?locale=en_AU) for more information.

- 1) Ensure you have an active Wi-Fi connection on your Apple Mac
- 2) From the Apple menu > System Preferences, then click Sharing
- 3) Share your Wi-Fi connection
- 4) To computers using the USB 10/100 LAN
- 5) Enable Internet Sharing



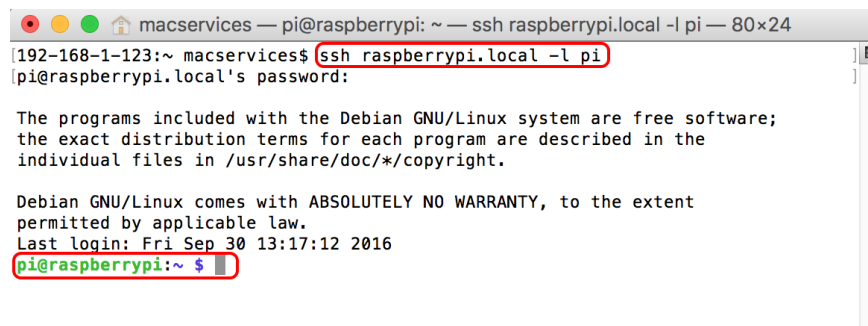
---

## Setting up your Raspberry Pi

From a Mac OS Terminal session, SSH (Secure Shell) to the Raspberry Pi as user “pi”.

**ssh raspberry.local -l pi**

You will be prompted to enter the password for user “pi”. The default password is “raspberry”.



```
macservices — pi@raspberrypi: ~ — ssh raspberrypi.local -l pi — 80x24
192-168-1-123:~ macservices$ ssh raspberrypi.local -l pi
pi@raspberrypi.local's password:

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: Fri Sep 30 13:17:12 2016
pi@raspberrypi:~ $
```

From the SSH session run the following commands on the Raspberry Pi.

- 1) Update Raspbian Linux with the latest updates. Depending on the number of updates this process can take between 5 and 20 minutes.

**sudo apt update**  
**sudo apt dist-upgrade**  
**sudo reboot**

- 2) Rebooting the Raspberry Pi will drop the SSH Session. You will need to re-establish the SSH session.

**ssh raspberry.local -l pi**

- 3) Install Apple Talk support on the Raspberry Pi. This will allow you to browse the file system on the Raspberry Pi from the Mac OS Finder.

**sudo apt install netatalk**

- 4) Install Remote Desktop Services on the Raspberry Pi.

**sudo apt install xrdp**

- 5) Install the following pip3 ([https://en.wikipedia.org/wiki/Pip\\_\(package\\_manager\)](https://en.wikipedia.org/wiki/Pip_(package_manager))) packages required for the Python3 Weather Data Sample to run on the Raspberry Pi

**sudo pip3 paho-mqtt**  
**sudo pip3 pyowm**

- 6) Clone the Python3 Smart Building Environment sample to the Raspberry Pi

**git clone https://github.com/gloveboxes/Smart-Building-Environmental-Data--Azure-IoT-Hub--Python3--MQTT.git iothub**

---

## Installing Recommended Mac OS Packages

### Visual Studio Code IDE

Visual Studio Code is a cross platform light weight pluggable IDE that supports a wide range of programming languages. You will also need to add the Python language extension.

- 1) Download and install from <https://code.visualstudio.com>
- 2) Add the Python Language Extension from <https://marketplace.visualstudio.com/items?itemName=donjayamanne.python>
- 3) Review the other extensions available for Visual Studio Code from <https://marketplace.visualstudio.com/VSCode>

### Python3 Support

Mac OS ships with Python 2.x support only. To run the Python3 Weather Data Simulator on your Apple Mac you will need to install Python3.

To install Python3 support see <https://www.python.org/downloads/mac-osx> and install the latest version of Python 3. At the time of writing this is 3.5.2.

### Clone the Python3 Smart Building Environment sample to your Apple Mac

The Smart Building Environment sample runs both on the Raspberry Pi and on your Apple Mac. When run on your Apple Mac the weather data is obtained from the Open Weather Map Service.

Running the Smart Building Environment sample will enable you to get started quickly streaming data to Azure IoT Hub. The ultimate goal is to stream real weather from the Raspberry Pi using the Pi Sense HAT.

**cd Documents**

**git clone https://github.com/gloveboxes/Smart-Building-Environmental-Data--Azure-IoT-Hub--Python3--MQTT.git iothub**

### Mac OS Python3 PIP3 Libraries

The following pip3 packages are required on your Apple Mac to run the Python3 Smart Building Environment sample on your Apple Mac.

**pip3 paho-mqtt**  
**pip3 pyowm**

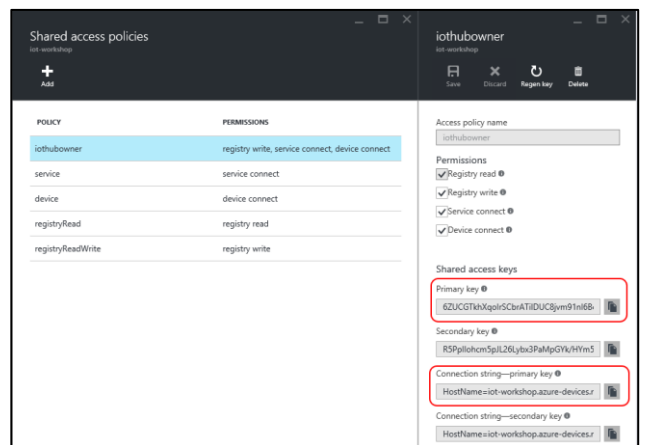
## Provisioning an Azure IoT Hub Device

All devices need an identity before they can connect and stream data to or from Azure IoT Hub.

Before you can create a device identity in Azure IoT Hub you need the IoT Hub Connection String. This will either be provided to you by your workshop mentor or from your own instance of Azure IoT Hub.

### To obtain your own IoT Hub connection string.

- 1) Sign in to azure at <http://portal.azure.com> with your credentials and either create or navigate to your IoT Hub.
- 2) Create or navigate to your instance of IoT Hub.
- 3) Go to **Settings -> Shared Access Policies -> iothubowner**
- 4) Copy the connection string



## Device Identity Management with Node.js [Azure IoT Hub Explorer](#)

[Azure IoT Hub Explorer](#) is an easy Node.js **command line** tool to:-

- 1) Create, delete, and list device identities
- 2) Send cloud to device messages
- 3) Listen to device to cloud messages

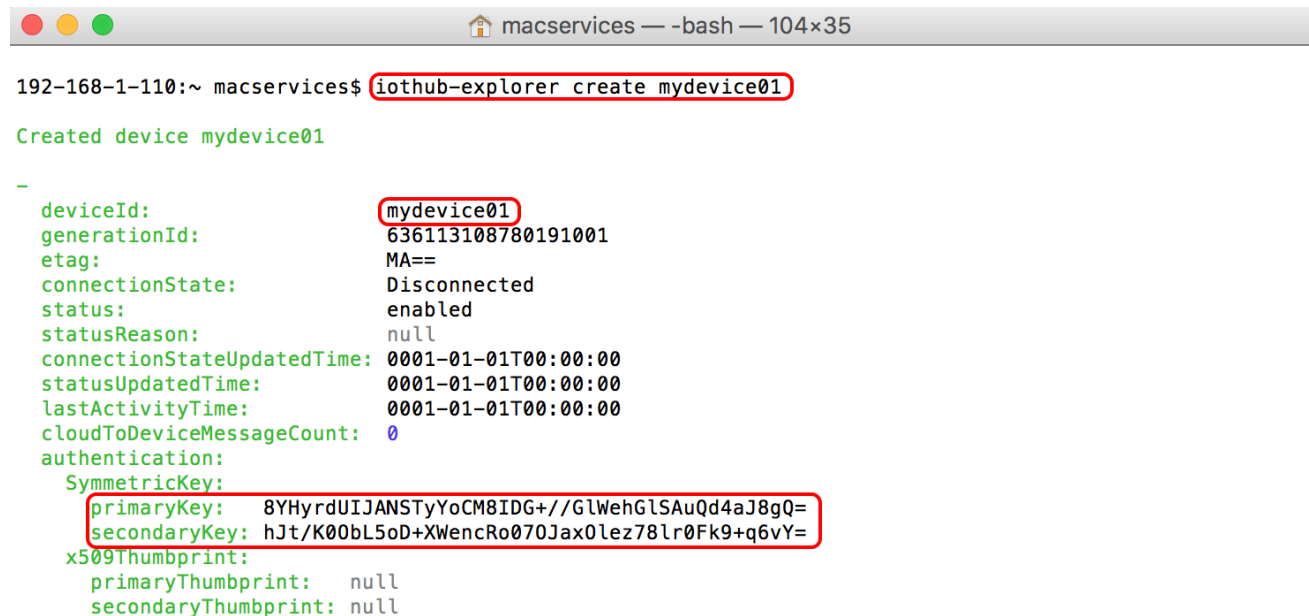
Node.js Azure IoT Hub Explorer	
Install Node.js	<a href="http://www.nodejs.org">www.nodejs.org</a>
Install Azure IoT Hub Explorer	<code>npm install -g iothub-explorer@latest</code>
Help	<code>iothub-explorer help</code>
Cache the Connection String	<code>iothub-explorer login &lt;connection-string&gt;</code>
Create a new device identity	<code>iothub-explorer [&lt;connection-string&gt;] create &lt;device-id&gt;</code>
Delete a device identity	<code>iothub-explorer [&lt;connection-string&gt;] delete &lt;device-id&gt;</code>
Get device information	<code>iothub-explorer [&lt;connection-string&gt;] get &lt;device-id&gt;</code>
Send a cloud to device message	<code>iothub-explorer [&lt;connection-string&gt;] send &lt;device-id&gt; &lt;msg&gt;</code>
Monitor device to cloud messages	<code>iothub-explorer &lt;connection-string&gt; monitor-events &lt;device-id&gt;</code>

Example usage

```
iothub-explorer login "HostName=YourIoTHub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=iVfr5G12UcCBQuju8OD98TAP0gmORQsLSXeOTPmaNEw="
```

## Create an Azure IoT Hub Device Identity

**iothub-explorer create mydevice01**



```
192-168-1-110:~ macservices$ iothub-explorer create mydevice01

Created device mydevice01

-
deviceId: mydevice01
generationId: 636113108780191001
etag: MA==
connectionState: Disconnected
status: enabled
statusReason: null
connectionStateUpdateTime: 0001-01-01T00:00:00
statusUpdateTime: 0001-01-01T00:00:00
lastActivityTime: 0001-01-01T00:00:00
cloudToDeviceMessageCount: 0
authentication:
  SymmetricKey:
    primaryKey: 8YHyrdUIJANSTyYoCM8IDG+//GlWehGlSAuQd4aJ8gQ=
    secondaryKey: hJt/K00bL5oD+XWencRo070Jax0lez78lr0Fk9+q6vY=
  x509Thumbprint:
    primaryKeyThumbprint: null
    secondaryThumbprint: null
```

**Be sure to copy one of the keys as you'll need when configuring the Smart Building Environment app. For the purposes of this lab either the primary or secondary key can be used.**

---

## Smart Building Environment Sample Configuration

The main Smart Building Environment sample is located in the **IoTHub/smartbuilding** directory.

A configuration file is passed to the environment.py app at start-up time. It contains the Azure IoT Hub address, the device identity including the shared access key, the sensor module to load, along with the Open Weather Map API key and location.

Sample code includes support for the following HATS

- 1) [Open Weather Map](#) Virtual Environmental Sensor HAT: sensor\_openweather. This virtual Environment HAT can be used on your Apple Mac or the Raspberry Pi as it does not require any specific hardware and gets environmental data from the free Open Weather Map service.
- 2) [Raspberry Pi Sense HAT](#): sensor\_envirophat
- 3) [Enviro pHAT](#): sensor\_sensehat

The following is an example for the Raspberry Pi Sense HAT (**sensor\_openweather.json**)

```
{
  "IoTHubAddress": "YourIoTHub.azure-devices.net",
  "DeviceId": "rpi3mlb",
  "SharedAccessKey": "uJ21qp9LUvjkohipkXycvb7RoYwmUDE+4gXylYS00feZg=",
  "SensorModule": "sensor_openweather",
  "OpenWeatherMapApiKey": "c2044448a2f55555925f27b9e21296dd",
  "OpenWeatherMapLocationId": "Melbourne, AU"
}
```

The configuration file is passed in as a start-up argument. The following are examples of running the Smart Buildings sample app using the three HAT configuration files included in the sample.

- **python3 environment.py sensor\_openweather.json**
- **python3 environment.py sensor\_sensehat.json**
- **python3 environment.py sensor\_envirophat.json**

### Open Weather Sample in Actions

When the app starts successfully it will display the sent message count and the device id.

The data is serialised in the following JSON format.

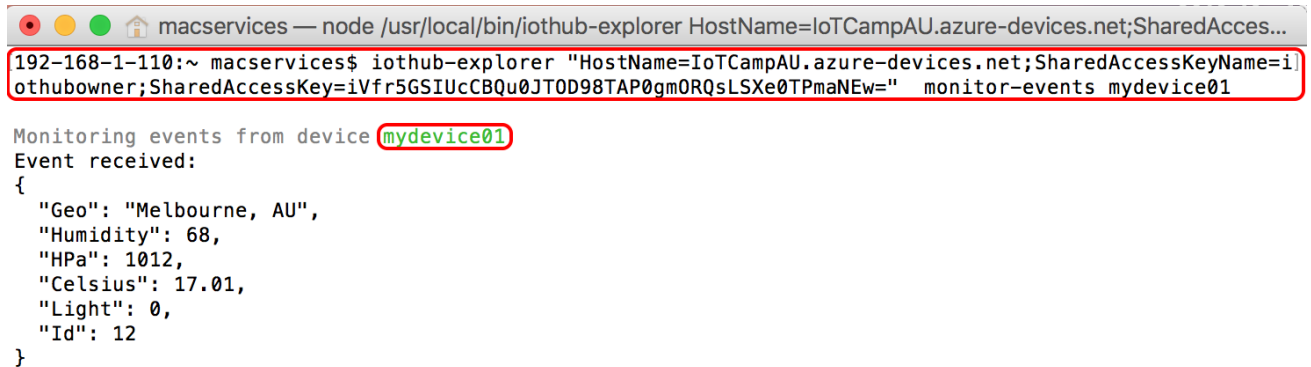
```
{"Geo": "Melbourne, AU", "Humidity": 50, "HPa": 1011, "Celsius": 18.40, "Light": 0, "Id": 199783}
```



---

## Monitoring Azure IoT Hub Device to Cloud Messages

**iothub-explorer** "HostName=YourIoTHub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=iVfr5G12UcCBQuju8OD98TAP0gmORQsLSXe0TPmaNEw=" **monitor-events** mydevice01



```
macservices — node /usr/local/bin/iothub-explorer HostName=IoTCampAU.azure-devices.net;SharedAccessKey=iVfr5G12UcCBQuju8OD98TAP0gmORQsLSXe0TPmaNEw=" monitor-events mydevice01

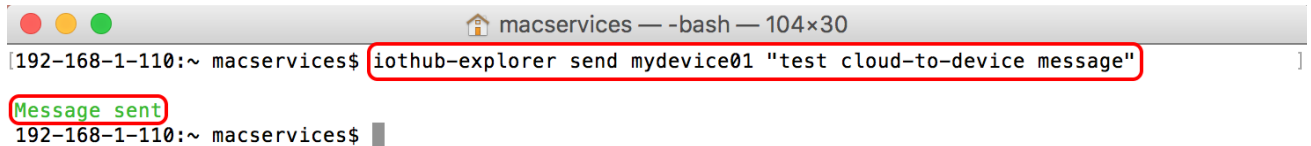
192-168-1-110:~ macservices$ iothub-explorer "HostName=IoTCampAU.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=iVfr5G12UcCBQuju8OD98TAP0gmORQsLSXe0TPmaNEw=" monitor-events mydevice01

Monitoring events from device mydevice01
Event received:
{
  "Geo": "Melbourne, AU",
  "Humidity": 68,
  "HPa": 1012,
  "Celsius": 17.01,
  "Light": 0,
  "Id": 12
}
```

---

## Send a Cloud to Device Message

**iothub-explorer send mdevice01 "test cloud-to-device message"**




```
macservices — -bash — 104x30

192-168-1-110:~ macservices$ iothub-explorer send mydevice01 "test cloud-to-device message"

Message sent
192-168-1-110:~ macservices$
```

Message received by the Smart Building Environment Python3 app.



```
smartbuilding — Python environment.py config_openweather.json — 80x24

Message 37 sent from mydevice01
Message 38 sent from mydevice01
Message 39 sent from mydevice01
Message 40 sent from mydevice01
Message 41 sent from mydevice01
Message 42 sent from mydevice01
Message 43 sent from mydevice01
devices/mydevice01/messages/devicebound/%24.to=%2Fdevices%2Fmydevice01%2Fmessage%2Fdevicebound - b'test cloud-to-device message'
Message 44 sent from mydevice01
Message 45 sent from mydevice01
```

---

## Developer Workflow

Visual Studio Code with the Python Extension added is an excellent way to edit the Smart Building sample.

If you are running the environment app on your Apple Mac with the Open Weather Sensor Virtual HAT then you can edit and debug the app all from your Apple Mac.

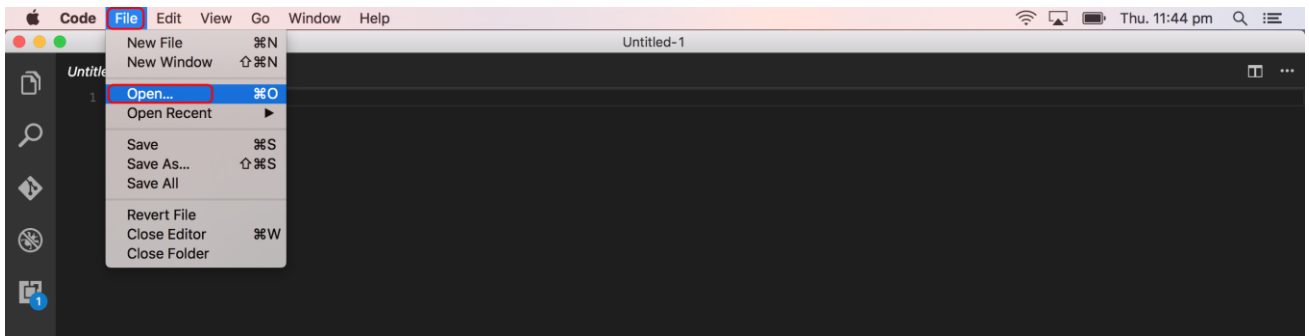
If you are editing the Smart Building sample on the Raspberry Pi then there are a couple of options.

### Visual Studio Code

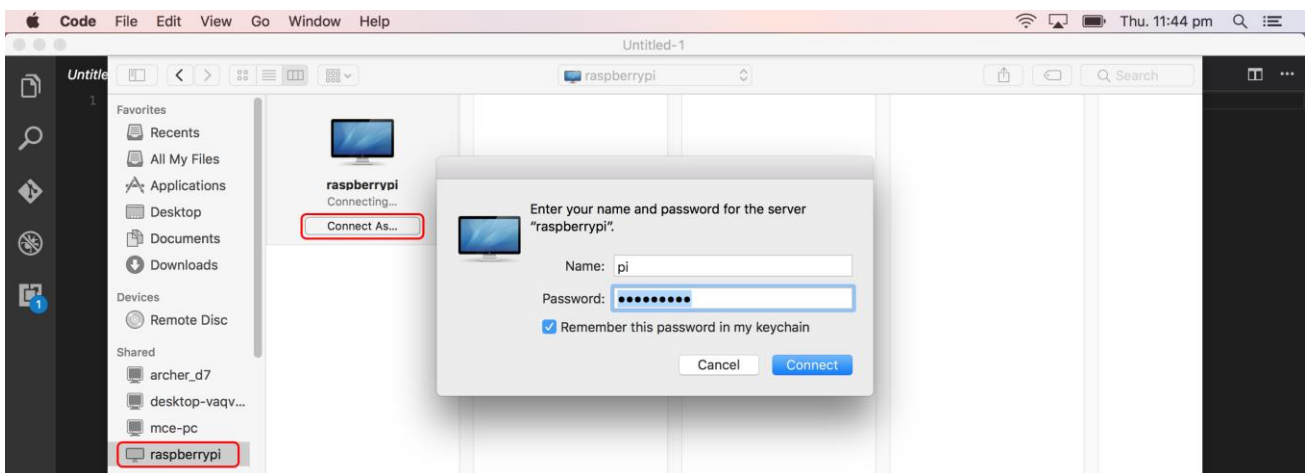
Given you installed the Apple Talk package on the Raspberry Pi the Raspberry Pi file system will appear in the Mac OS Finder. You will need to provide network credentials for the Raspberry Pi, which are user 'pi' and password 'raspberrypi'. Then open the IoTHub\smartbuilding at the directory level, this will load all the files in the solution and you can start editing.

This approach will provide you with a rich editing intellisense environment but you will need to SSH in to the Raspberry Pi separately to run the app.

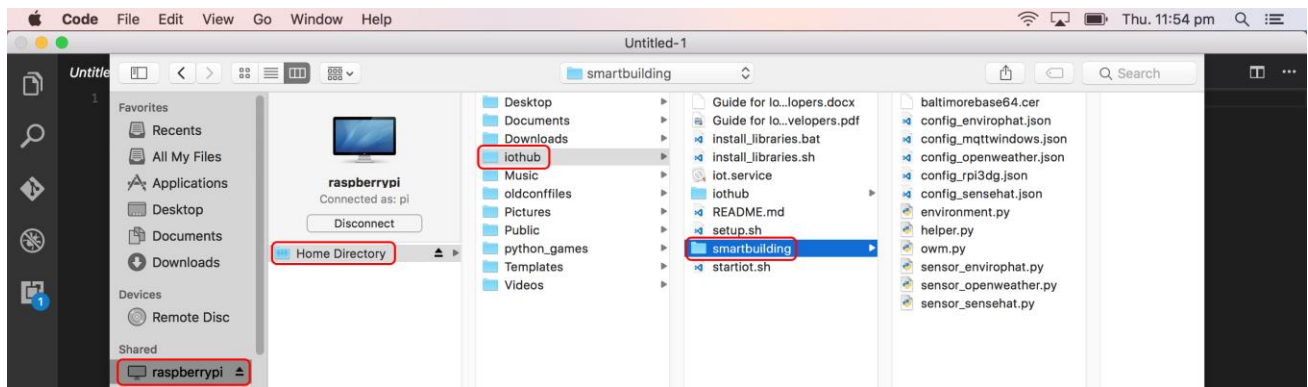
Start Visual Studio Code. Then File -> Open



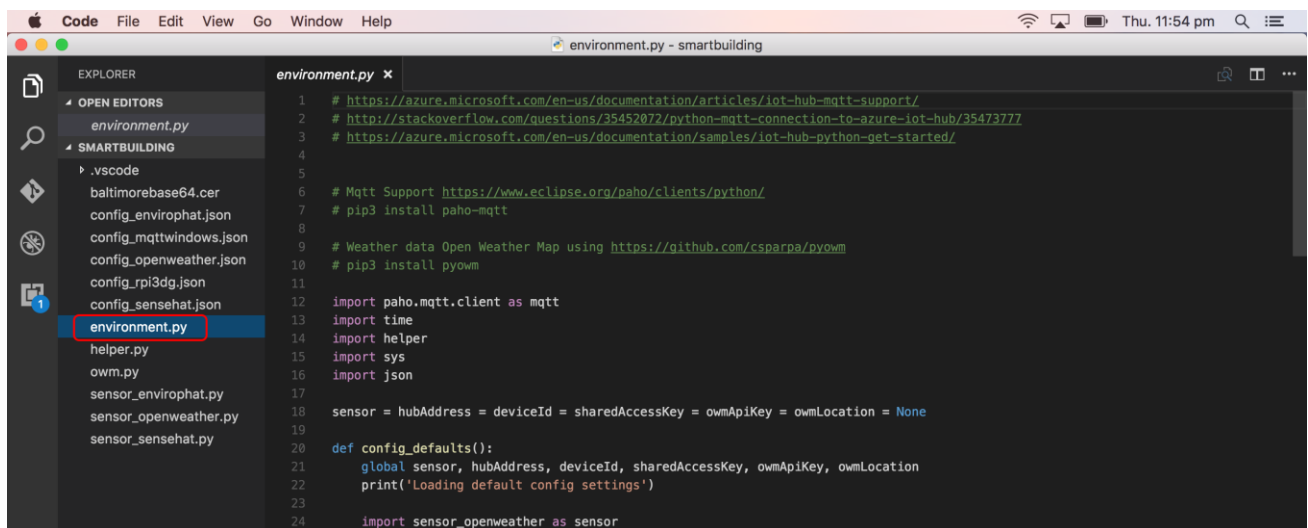
From Finder, select the Raspberry Pi, then Connect As, then credentials, user 'pi', password 'raspberrypi' then connect.



Then select Home Directory, iotHub, then the smartbuilding directory.



Visual Studio Code will open the smartbuilding directory and will display all the project files. Select environment.py and start editing.

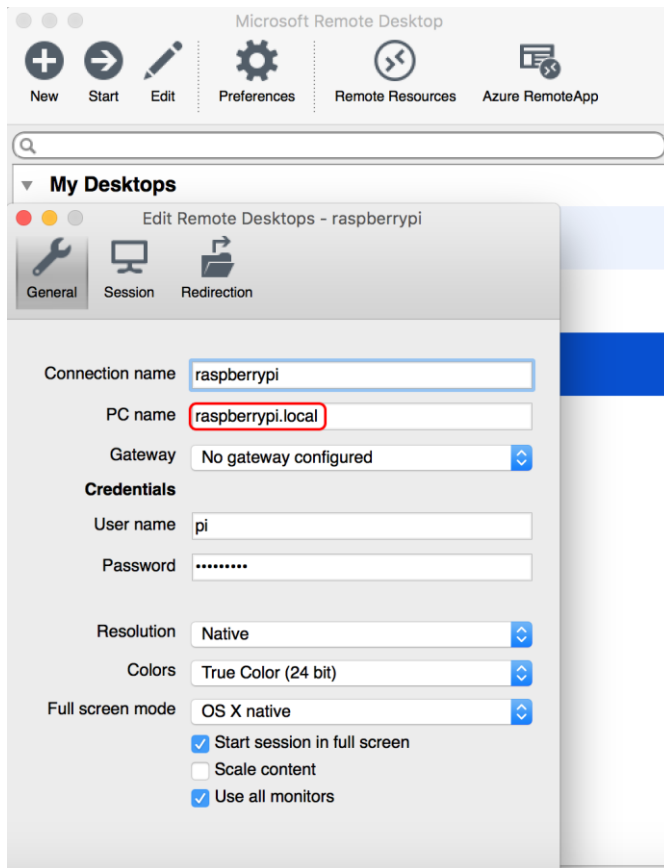


To run the project on the Raspberry Pi be sure to save your changes in Visual Studio Code, start an SSH session to the Raspberry Pi. Change directory to iotHub/smartbuilding and start the environment.py by running `python3 environment.py config_sense.json`.

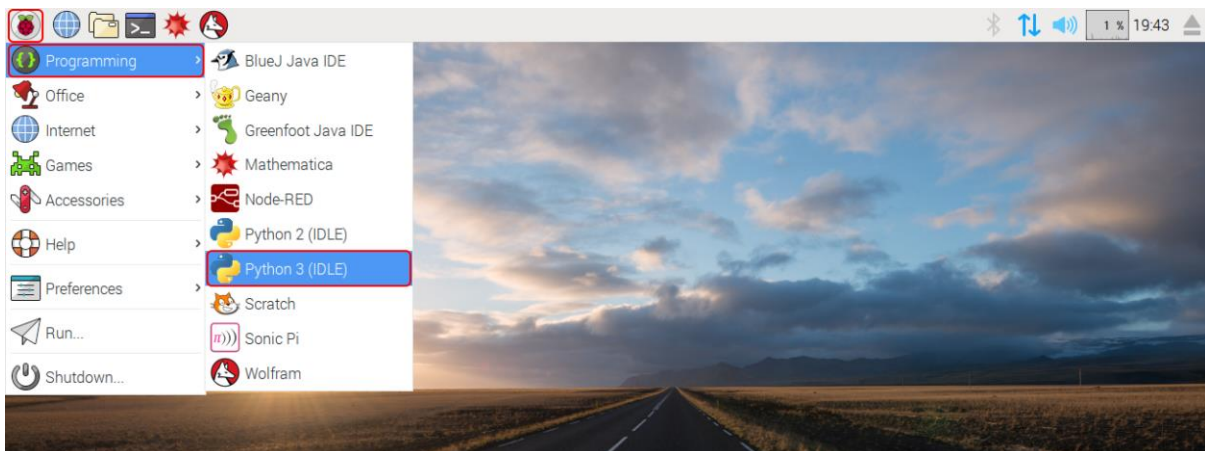
## Microsoft Remote Desktop

Microsoft Remote Desktop along with the xrdp Raspberry Pi package allows you to start a remote desktop session with your Raspberry Pi from your Apple Mac. You can edit/run/debug the Smart Building app on the Raspberry Pi itself with the editor of your choice such as nano or idle3.

Configure Microsoft Desktop as follows.



Start Python 3 (IDLE) and open the environment.py sample add from the IoTHub/smartbuilding folder.



Once the Smart Building sample app has opened you can edit/run/debug and make enhancements to the sample app.

The screenshot shows a Microsoft Remote Desktop window with two panes. The left pane displays a Python script named `weather_mqtt.py` located at `/home/pi/iot-hub/_ather_mqtt/weather_mqtt.py`. The script includes comments with links to Azure IoT Hub MQTT documentation, Stack Overflow, and GitHub. It imports `paho.mqtt.client` as `mqtt`, along with `time`, `helper`, `sys`, and `json`. It defines global variables for `sensor`, `hubAddress`, `deviceId`, `sharedAccessKey`, `ownApiKey`, and `ownLocation`. The script has two main functions: `config_defaults()` which prints 'Loading default config settings' and sets default values for the global variables, and `config_load()` which checks for command-line arguments and loads a JSON config file. The right pane shows a `*Python 3.4.2 Shell*` window. It displays the Python version (3.4.2), GCC version (4.9.1), and the output of the script: 'Loading default config settings', 'Connected with result code: 0', 'Message 2 sent from mqtt', and 'Message 3 sent from mqtt'.

```
weather_mqtt.py - /home/pi/iot-hub/_ather_mqtt/weather_mqtt.py 3.4.2
File Edit Format Run Options Windows Help
# https://azure.microsoft.com/en-us/documentation/articles/iot-hub-mqtt
# http://stackoverflow.com/questions/35452072/python-mqtt-connection-t
# https://azure.microsoft.com/en-us/documentation/samples/iot-hub-pyth

# Mqtt Support https://www.eclipse.org/paho/clients/python/
# pip3 install paho-mqtt

# Weather data Open Weather Map using https://github.com/csparpa/pyowm
# pip3 install pyowm

import paho.mqtt.client as mqtt
import time
import helper
import sys
import json

sensor = hubAddress = deviceId = sharedAccessKey = ownApiKey = ownLoca

def config_defaults():
    global sensor, hubAddress, deviceId, sharedAccessKey, ownApiKey, o
    print('Loading default config settings')

    import sensor_openweather as sensor
    hubAddress = 'IoTCampAU.azure-devices.net'
    deviceId = 'mqtt'
    sharedAccessKey = 'VZbmLwYjJdg04G6b55vbNMTq44GdDE05k5px60UIu7l8='
    ownApiKey = 'c204bb28a2f9dc23925f27b9e21296dd'
    ownLocation = 'Melbourne, AU'

def config_load():
    global sensor, hubAddress, deviceId, sharedAccessKey, ownApiKey, o
    try:
        if len(sys.argv) == 2:
            print('Loading {0} settings'.format(sys.argv[1]))

            config_data = open(sys.argv[1])
            config = json.load(config_data)

            sensor = __import__(config['SensorModule'])

Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
Loading default config settings
Connected with result code: 0
Message 2 sent from mqtt
Message 3 sent from mqtt
Ln: 6 Col: 0
```

---

## Azure IoT Resources for Apple Mac Based Developers

### Node.js IoT Hub Explorer

<https://www.npmjs.com/package/iothub-explorer>

This sample has some extended capabilities over the Python3 IoT Hub Devices sample most notably being able to send a **cloud-to-device messages**.

### Python Resources

#### Getting started with IoT Hub REST API and Python

- 1) <https://azure.microsoft.com/en-us/documentation/samples/iot-hub-python-get-started>

### Node.js

#### Getting started with IoT Hub REST API and Python

- 1) <https://azure.microsoft.com/en-us/documentation/articles/iot-hub-node-node-getstarted/>
- 2) <https://github.com/juanjperez/azure-iot-device-management>
- 3) <https://www.npmjs.com/package/iothub-explorer>