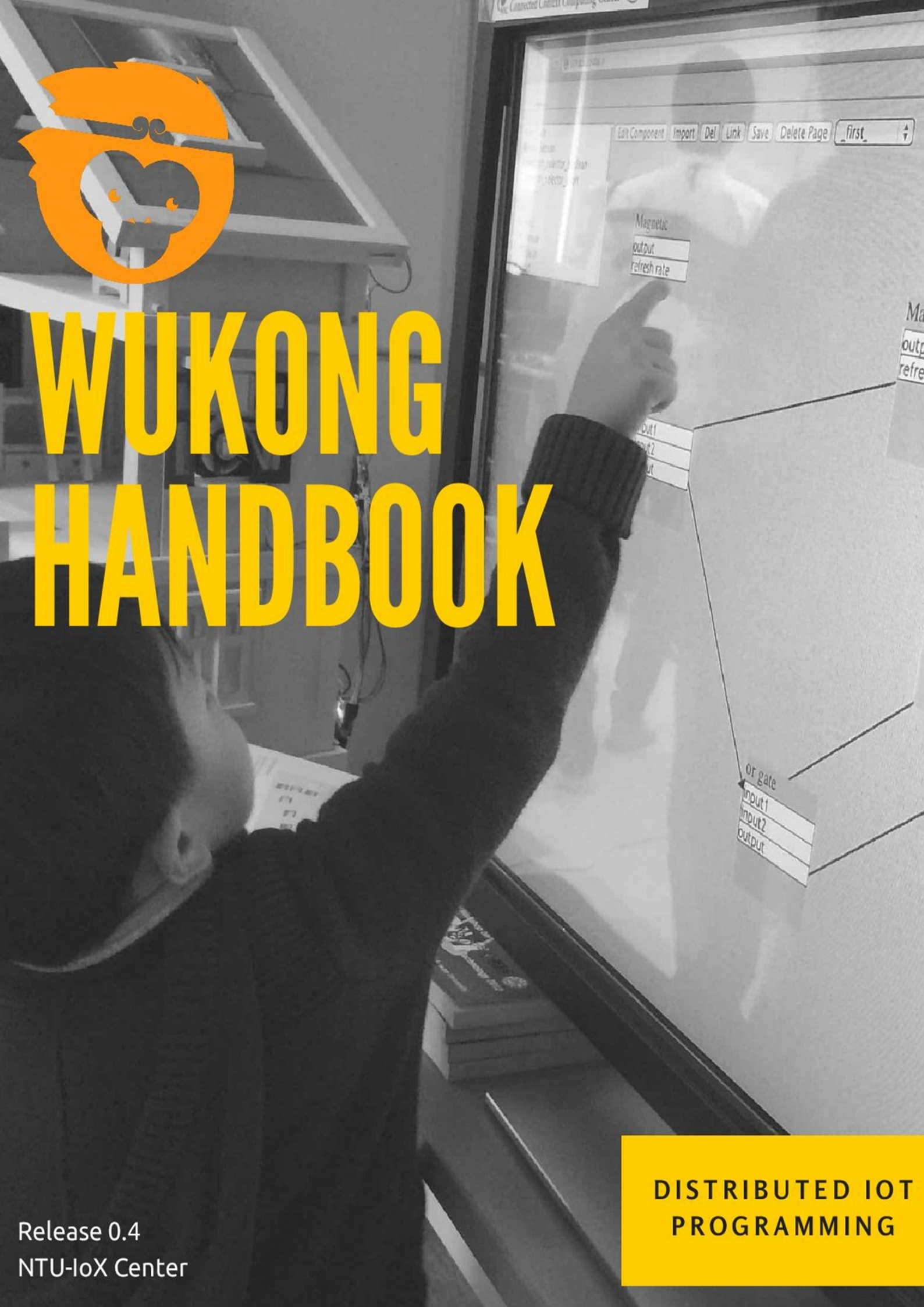


WUKONG HANDBOOK

DISTRIBUTED IOT PROGRAMMING

Release 0.4

NTU-IoX Center

Table of Contents

Version History

Date	Version	Content
May 6, 2016	1	Initial version

Chapter 1: Introduction

The WuKong project at the NTU-IoX Center has built an intelligent middleware for IoT systems. The goal of the WuKong project is to help users design and develop hardware-independent IoT applications, so that they can be easily configured and dynamically deployed on vendor-independent IoT platforms.

The WuKong IoT middleware has the following major features:

- **Virtualized IoT Devices**: Virtualizing IoT devices allows hardware-independent application design and simplifies IoT services migration among devices without redefining applications. Consequently, one can deploy an IoT application on different hardware platforms without using hardware- and/or network-dependent application codes.
- **Flow-Based Programming environment**: WuKong provides a graphical flow-based programming (FBP) tool. In each FBP, a user can define the data and control flow to build an IoT application. All the user has to do is to select the type of services from a predefined WuKong component library, drag and drop them on the programming canvas, and then connect them with directed links. The application flow diagram will then be used to map logical services onto appropriate physical devices during application deployment.
- **Heterogeneous and Virtual services**: The WuKong middleware provides virtual machines on heterogeneous platforms to simplify IoT application deployment and migration. Virtual IoT services can also be implemented (in Python) to support Web-based data services or UI running on servers, computers and smartphones. In addition, Darjeeling-based Java Virtual Machine is included in the WuKong middleware so that a system can dynamically add bytecodes on each device.
- **Deployment-time Service Mapping**: To support heterogeneous and constant-evolving hardware platforms, WuKong delays the binding between logical IoT services and physical IoT devices until deployment. Consequently, during application development, platform-dependent properties and configurations such as port assignment and pin assignment are minimized. The platform-dependent properties are collected by the WuKong Master when a device registers itself in a WuKong system. The Master will use these properties to produce a proper configuration and generate required executable code for each IoT application.

In this document, we present the instructions and examples on how to set up a WuKong environment and develop WuKong IoT applications. The document includes the description on the required system toolchains, the device hardware and firmware setup, Master installation and operations, device sensing connections, and advanced IoT device support. All hardware and software used are openly available for access and installation. By making this project open, we hope to make IoT much easier to build and to deploy in the future.

1.1 How to use this book

This book is designed to walk you through (a) configuring the WuKong middleware and development environment, (b) setting IoT devices for WuKong adoption, and (c) developing new IoT applications on WuKong. The book is organized in seven chapters. Chapter 1 is an introduction of the WuKong middleware and the system building environment. Chapters 2 and 3 show the steps for building configuring the software and hardware environment. Chapter 4 gives two simple WuKong execution examples, one without using any actual IoT device and the other using a single IoT device. Chapter 5 go through various WuKong development and management capabilities. In Chapter 6 and 7, we show how to implement and add specific device capabilities (called WuClasses) in your projects.

Suggested Chapters

This book is designed for three groups of target readers: *device builders*, *application developers*, and *system installers*. Different chapters may be useful for people with different roles when building WuKong-based IoT systems.

Below are the recommended materials for each role.

- Device Builder (hardware specific, wuclass programming): Chapters 2, 3 and 6
- Service Installers (Master and network setup): Chapters 4.*.1, 4.*.2 and 5.1
- Application Developers: Chapters 4.*.3, 5.2, and 5.3

Target Hardware

WuKong is a distributed middleware consisting of three system components: Master, gateway, and IoT device. Each component can be run independently on specific or shared hardware platforms.

- Master must be run on Linux-based computers with network connections and Web server capability. It is used to produce target code for specific IoT devices and send executable codes to these devices. A machine with an adequate computing capability is desired for running Master.
- Gateways are used to connect to IoT devices and forward network packets among devices on different subnets and/or using different networking protocols. They can be run on Linux-based servers or simple gateway machines.
- IoT devices are designed to provide sensing inputs and actuation controls. They may be run on tiny sensing and control boards, such as those without operating system support. Some IoT boards with sensing and control capabilities may have vendor-supported

RTOS or embedded Linux. In addition, WuKong can define virtual devices running on PC or servers to collect user input and to display user interface or media.

Figure 1(a-c) show these possible configurations for WuKong systems. A WuKong-based IoT application can be run using only a Linux-based computer or server, with Master, gateway, and virtual devices all on the same machine (Figure 1a). Some WuKong-based IoT applications may be run on several physical sensing devices and a Linux server running Master and gateway (Figure 1b). A full scale WuKong system may have a dedicated Master running on a (local or cloud) server, several gateway units each managing devices on different subnets or protocols (e.g. WiFi, ZWave, Zigbee), and many IoT devices running on physical boards and/or virtual software modules on servers (Figure 1c).

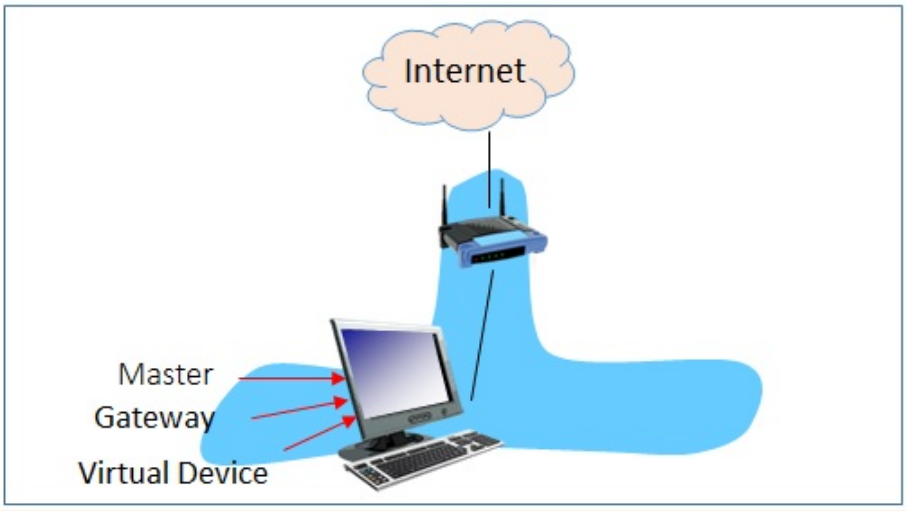


Fig 1a. A WuKong-based IoT application can be run using only a Linux-based computer, with Master, gateway, and virtual devices all on the same machine.

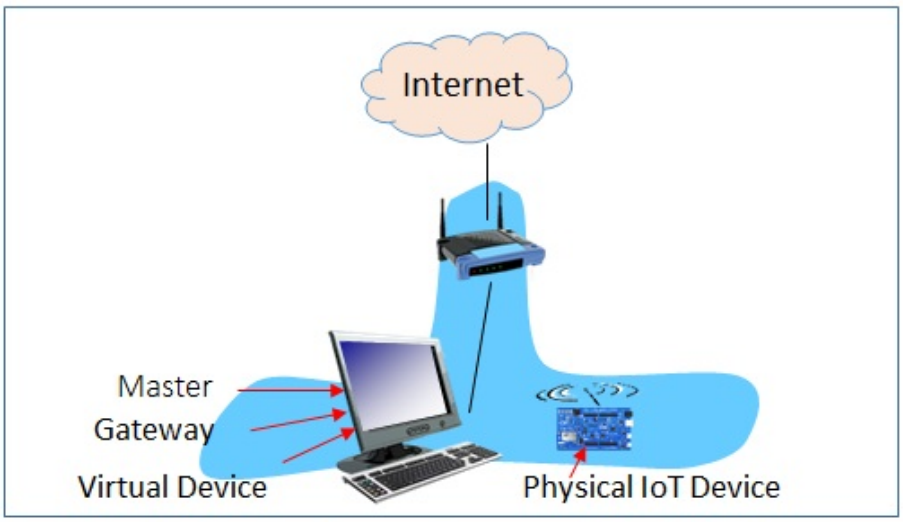


Fig 1b. WuKong-based IoT applications can be run on physical IoT devices connected to a Linux server running Master and gateway.

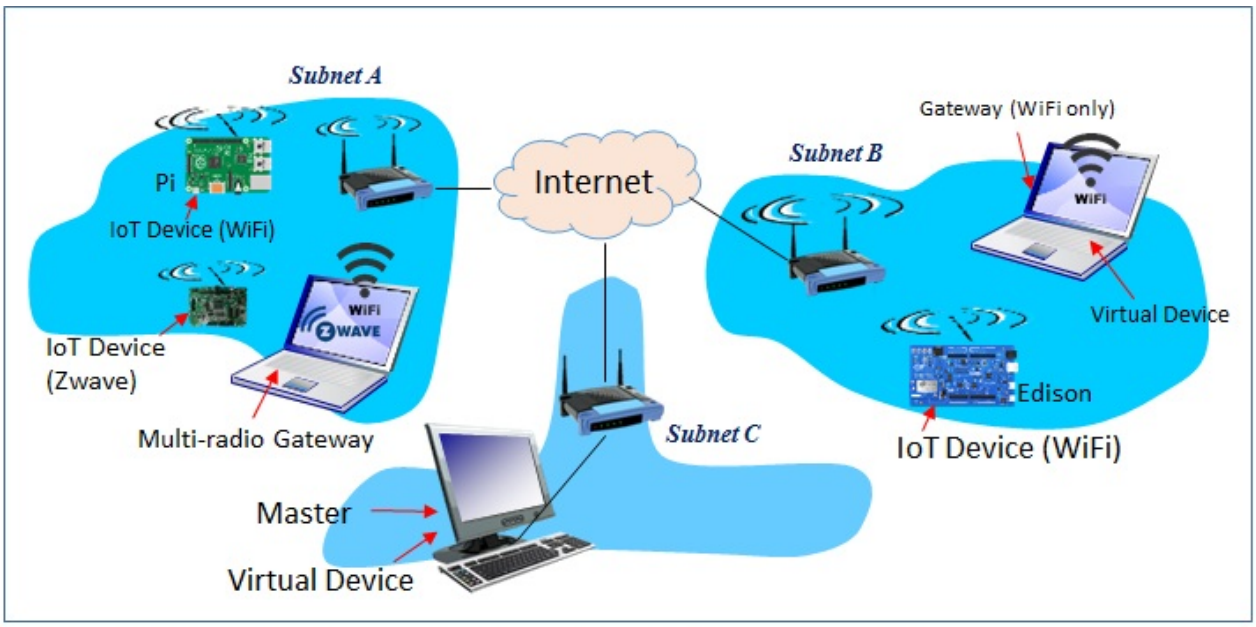


Fig 1c. A full scale WuKong system has a dedicated Master, several gateways each managing devices on different subnets or protocols (e.g. WiFi, ZWave, Zigbee), and IoT devices running on physical and/or virtual machines.

For information on how to run a WuKong-based IoT application using only Linux-based computers, please read Chapters 2, 4.1, 5, and 6.

For information on how to run a WuKong-based IoT application both on Linux-based computers and IoT boards, please read Chapters 2, 3, 4.2, 5, and 6.

Chapter 2: Preparing the Development Environment on Server

This chapter presents the instructions for preparing the WuKong development environment on a Linux-based computer. The instructions are tested on computers running **ubuntu 14.04 LTS**. We plan to provide the installation guide for OSX and Windows in the future. Before then, a possible option for PC users is to install Virtualbox to run ubuntu on their PC's.

Four tools are required to run WuKong. Git is used to get the source code from the WuKong project. Java is necessary to run the Master software. Gradle is used for building the project. And Python is used for implementing the WuKong Profile Framework. You can skip any step if your server has already installed that specific software.

- **Install Git**

```
sudo apt-get install git-core
```

- **Install Java**

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update && sudo apt-get install oracle-java7-installer
```

- **Install Gradle**

  Download gradle 2.4 HERE and extract it in a directory, for example, ~/gradle-2.4
  Append this gradle directory path to environment variable PATH as below.

```
vim ~/.bashrc
```

  Add

```
export PATH=$PATH:~/gradle-2.4/bin
```

  after the last line of bash startup file ~/.bashrc, and then use the source command to execute the script

```
source ~/.bashrc
```

- **Install Python Tools**

```
sudo apt-get install libevent-dev
sudo apt-get install python-dev
sudo apt-get install python-setuptools
sudo apt-get install libxml2-dev libxslt1-dev zlib1g-dev
sudo apt-get install python-pip
sudo pip install configobj simplejson gevent greenlet tornado jinja2 pyserial \
lxml
sudo pip install netifaces
sudo pip install python-cjson
sudo apt-get install python-pyaudio
```

In the next chapter, we will show how to set up IoT devices (such as Intel Edison, Galileo and Raspberry Pi 2) to develop applications that can sense and control the physical environment. But using the toolchains in this chapter, you can already develop a WuKong-based IoT application using just your computer as shown in the Section 4.1.

Chapter 3: Preparing IoT Boards for WuKong

Many IoT boards and devices are now available on the market. Each IoT device or sensing board must be pre-configured with the WuKong middleware before it can be managed by the WuKong Master as a WuKong device. In this chapter, we present the set up procedure for three commonly available boards: **Intel Edison**, **Intel Galileo Gen2**, and **Raspberry Pi 2**.

3.1 Board Setup for Intel Edison

There are **four** steps to set up an Intel Edison as a WuKong device.

Step 1. Board Hardware Assembly: To assemble the Intel Edison module on its Arduino extension board.

Step 2. Board Configuration: To setup a password on the Intel Edison so that we can use the ssh command to access the board.

Step 3. Firmware Update: To obtain the compatible Yocto image, which is 159.devkit at the time of this writing, to support a larger root partition size and better Python tools.

Step 4. Python Tool Installation: To install the required Python libraries for running WuKong.

Step 5. Download WuKong Source Code: To clone the WuKong source code from GitHub to the Edison board.

- **Step 1. Board Hardware Assembly**

  There are two types of extension boards designed for the Intel Edison module: the Arduino extension board, and the mini breakout board. In this chapter, we use the Arduino extension board since it is more convenient to connect sensor kit without soldering.

  Please see the following website on how to assemble an Intel Edison board: https://software.intel.com/en-us/assembling-intel-edison-board-with-arduino-expansion-board

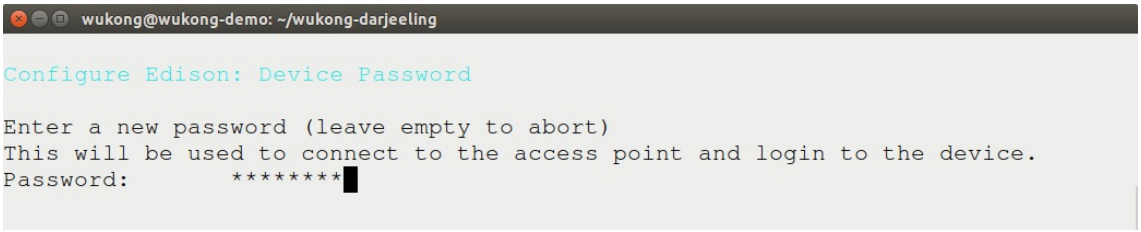
  After that, you can see the following website on how to set up a serial terminal to communicate with the board. https://software.intel.com/en-us/setting-up-serial-terminal-intel-edison-board

---

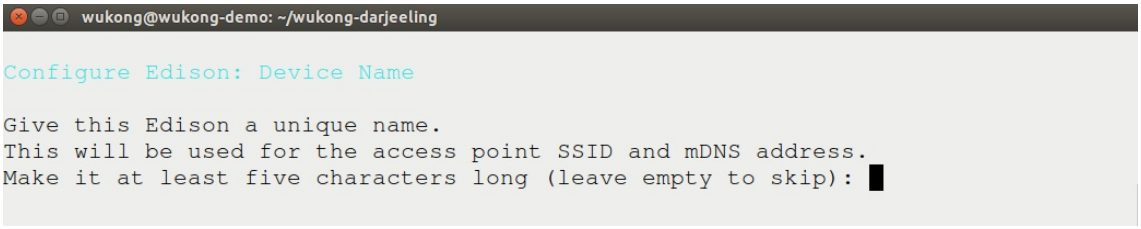
- **Step 2. Board Configuration**
  - To configure the board, enter the following command in the serial terminal opened in Step 1:

    ```
    configure_edison --setup
    ```

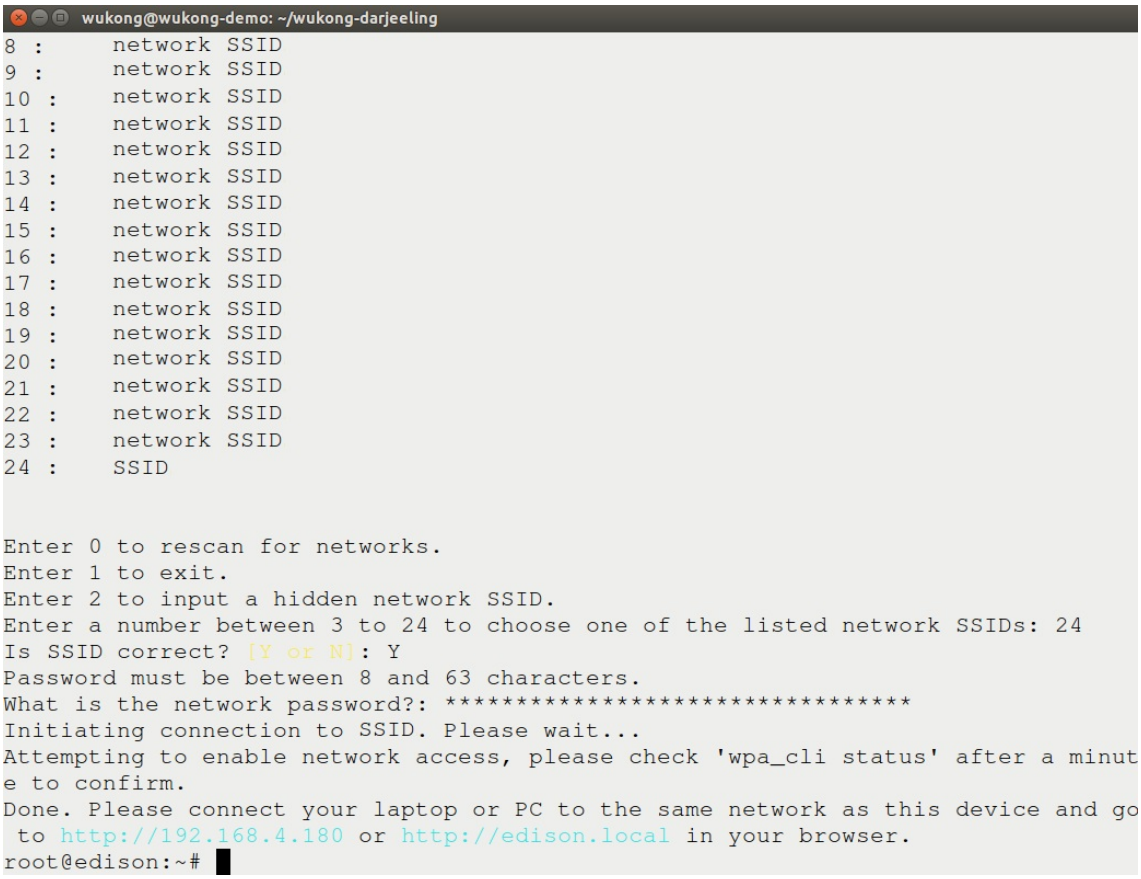
- When prompted, you must enter a chosen password so that the board can be accessed through the **ssh** command later on.



- When prompted, you can enter a name for the board, or the name **"root"** will be given if it is left empty.



- The next step is the WiFi configuration. Type "Y" to setup WiFi and the board will start to scan network services. Once the board finishes scanning, choose a network SSID and then enter the network password if required. The following screen shows an example setting.



- If you need to connect to a different network service, you can later use the following command to configure it again:

```
configure_edison --wifi
```

- **Step 3. Firmware Update**

  Please download this image, iot-devkit-prof-dev-image-edison-20160315, for Edison. At the time of writing, this is a relative stable version.

  Then, choose and download the installer from the following website:
  `https://software.intel.com/en-us/iot/hardware/edison/downloads`

  Follow instructions of the installer and flash the firmware with previous downloaded image.

- **Step 4. Python Tool Installation**

```
pip install twisted python-cjson gevent netifaces
```

- **Step 5. Download WuKong Source Code**
  - Clone the source code and download to your preferred working directory on the IoT board.

```
git clone -b release0.4 http://github.com/wukong-m2m/wukong-darjeeling
```

3.2 Board Setup for Intel Galileo Gen 2

This section shows **four** steps to set up an Intel Galileo Gen2 for running WuKong.

Step 1. Linux Image Preparation: To prepare a bootable microSD card with the Linux environment.

Step 2. Board Assembly: To assemble the Intel Galileo board.

Step 3. Board Configuration: To setup a password for Intel Edison so that we can use ssh command to access Intel Edison.

Step 4. Python Tool Installation: To install required Python libraries for WuKong.

Step 5. Download WuKong Source Code: To clone the WuKong source code from GitHub to the Galileo board.

- **Step 1. Linux Image Preparation**

  Unlike Edison, Galileo Gen2 doesn't have a built-in flash memory so that we need to prepare a bootable microSD card.

  Please see the following website to make a bootable microSD card:
  https://software.intel.com/en-us/get-started-galileo-linux-step1

- **Step 2. Board Assembly**

  According to the Intel official IoT website, the required hardware includes:

  - Intel Galileo Gen 2,
  - A SD card (>2GB),
  - 6 pin Serial to Type A USB cable (e.g. FTDI cable # TTL-232R-3V3)
  - a 7-15V DC power supply rated at least 1500mA.
  - (Optional) WiFi adaptor (e.g. Intel® Centrino® Wireless-N 135 or Intel® Centrino® Advanced –N 6205).

  Once the hardware is prepared, please follow the lower part of the following website to assemble an Intel Galileo Gen 2 board:
  https://software.intel.com/en-us/get-started-galileo-linux-step2

After assembling the Intel Galileo Gen2 board, please see the following website to set up the serial communication with the board:
https://software.intel.com/en-us/get-started-galileo-linux-step3

- **Step 3. Board Configuration**
  If you have a recommended WiFi module as mentioned in Step 2, see the following website on how to install the WiFi adapter and to configure the WiFi setting.
  https://software.intel.com/en-us/get-started-galileo-linux-step4

  If not, see the following website to set up an ethernet connection. Instead of using a serial terminal, it uses Arduino IDE to communicate with board.
  https://software.intel.com/en-us/articles/intel-galileo-getting-started-ethernet

---

- **Step 4. Python Tool Installation**
  - Copy the following instructions to `/etc/opkg/base-feeds.conf`

```
src/gz all http://repo.opkg.net/edison/repo/all
src/gz edison http://repo.opkg.net/edison/repo/edison (?edison?)
src/gz core2-32 http://repo.opkg.net/edison/repo/core2-32
```

  - Install **pip** and other tools

```
opkg update
opkg install python-pip
pip install twisted
pip install python-cjson
pip install gevent
opkg install sqlite3
pip install netifaces
opkg install git
```

- **Step 5. Download WuKong Source Code**
  - Clone the source code and download to your preferred working directory on the IoT board.

```
git clone -b release0.4 http://github.com/wukong-m2m/wukong-darjeeling
```

3.3 Board Setup for Raspberry Pi Board

This section shows the **four** steps to set up an Raspberry Pi for running WuKong.

Step 1. Raspbian Linux Image Preparation: To prepare a bootable microSD card with the Linux environment.

Step 2. Board Hardware Assembly: To assemble the Raspberry Pi hardware and its accessories.

Step 3. Board Configuration: To setup a password for Raspberry Pi.

Step 4. Python Tool Installation: To install required Python libraries for running WuKong.

Step 5. Download WuKong Source Code: To clone the WuKong source code from GitHub to the Raspberry Pi board.

- **Step 1. Raspbian Linux Image Preparation**

  You can install with NOOBS as introduced in the following website:
  https://www.raspberrypi.org/help/noobs-setup/

  We also recommend to download the image and write it to the SD Card by instructions in the official website.
  https://www.raspberrypi.org/documentation/installation/installing-images/README.md

  The image by default fills only 4 GB of the SD card. If you need more space, please refer to the solution in the official website.
  https://www.raspberrypi.org/documentation/configuration/raspi-config.md

---

- **Step 2. Board Hardware Assembly**

  The required hardware materials include:

  - Raspberry Pi Board,
  - A SD card (>4GB),
  - HDMI cable
  - a 7-15V DC power supply rated at least 1500mA.
  - Optional USB WiFi adaptor.
  - Optional Grove Pi extension board

You can watch video such as https://www.youtube.com/watch?v=JiTNdwD1fS0 for the instructions to set up the Raspberry Pi hardware.

- **Step 3. Board Configuration**
  Once powering on the RPi, You can either use GUI or login via SSH with account **pi** and password **raspberry**.
  - Change the keyboard layout by typing commands in a terminal.

    ```
    sudo nano /etc/default/keyboard
    ```

    Then find the following line:

    ```
    XKBLAYOUT=”gb”
    ```

    and change the "gb" to the two letter code for your country, say "us". Enter Ctrl+X to save the file and leave.

    Here (https://en.wikipedia.org/wiki/ISO_3166-1#Current_codes) is the list of current country codes from Wikipedia if you don't know your country code (use the codes in the column labeled alpha-2).
  - Change the update mirror site of Raspbian repository

    ```
    sudo nano /etc/apt/sources.list
    ```

    Replace the link "http://mirrordirector.raspbian.org/raspbian" with the closest server from the official list. https://www.raspbian.org/RaspbianMirrors

    ```
    deb http://mirrordirector.raspbian.org/raspbian jessie ...
    ```

- **Step 4. Toolchain Installation**
  - Python Tool Installation Install **pip** and other tools

    ```
    sudo apt-get update
    sudo apt-get install python-twisted python-cjson python-gevent
    sudo apt-get install sqlite3 python-netifaces
    ```

- Optionally, install grovepi library and enable the I2C interface for the Grove Pi extension board in the terminal.

```
sudo apt-get install python-smbus i2c-tools
sudo raspi-config
```

Once the menu shows up, select the **Advanced Options** and then **I2C** by Arrow and Enter keys. Select **Yes** to enable I2C interface and **Yes** to load the module by default. Finally, reboot the RPi.

```
sudo reboot
```

Check if the I2C is enabled by entering command in the terminal after reboot. If you have a Pi Model A, B+ or 2 B (512MB memory).

```
sudo i2cdetect -y 1
```

Otherwise, for Pi 1 B (256MB memory), run

```
sudo i2cdetect -y 0
```

It should show some address(es) like 20, 40, or 70 within the table.

```
git clone https://github.com/DexterInd/GrovePi.git
cd <source code>/GrovePi/Software/Python
sudo python setup.py install
```

- **Step 5. Download WuKong Source Code**
  - Clone the source code and download to your preferred working directory on the IoT board.

```
git clone -b release0.4 http://github.com/wukong-m2m/wukong-darjeeling
```

Chapter 4: Simple Examples of Deploying WuKong

This chapter gives two simple examples on how to deploy an IoT application in a WuKong system. The first application runs on a computer only, without connecting to any IoT board. The second application shows how to build an application on Intel Edison or Raspberry Pi boards. Both examples run on a local area network with no Internet connection required.

4.1 Using a Computer Only to Play Music

As WuKong master, gateway, node programs can all run on a computer, we start the first example by using only software modules on the computer.
The example is built in four steps:

Step 1. Start Master and gateway programs

Step 2. Include New Device

Step 3. Build the FBP.

Step 4. Running the Application

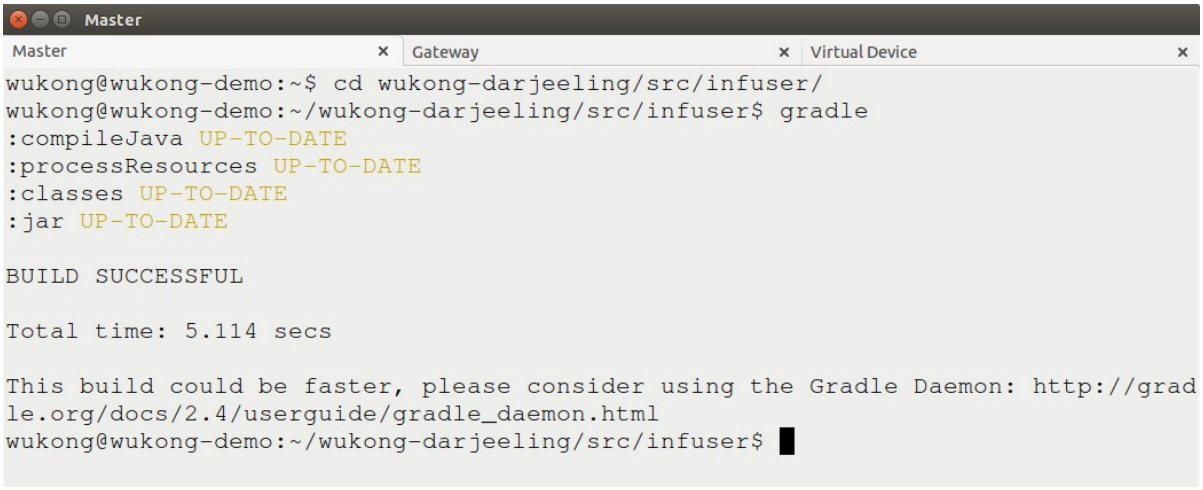
4.1.1 Start the Master and Gateway

1. On your computer, download the source code from github as below:

```
git clone -b release0.4 http://github.com/wukong-m2m/wukong-darjeeling
```

2. Build infuser

```
cd <path_of_source_code>/wukong-darjeeling/src/infuser/
gradle
```

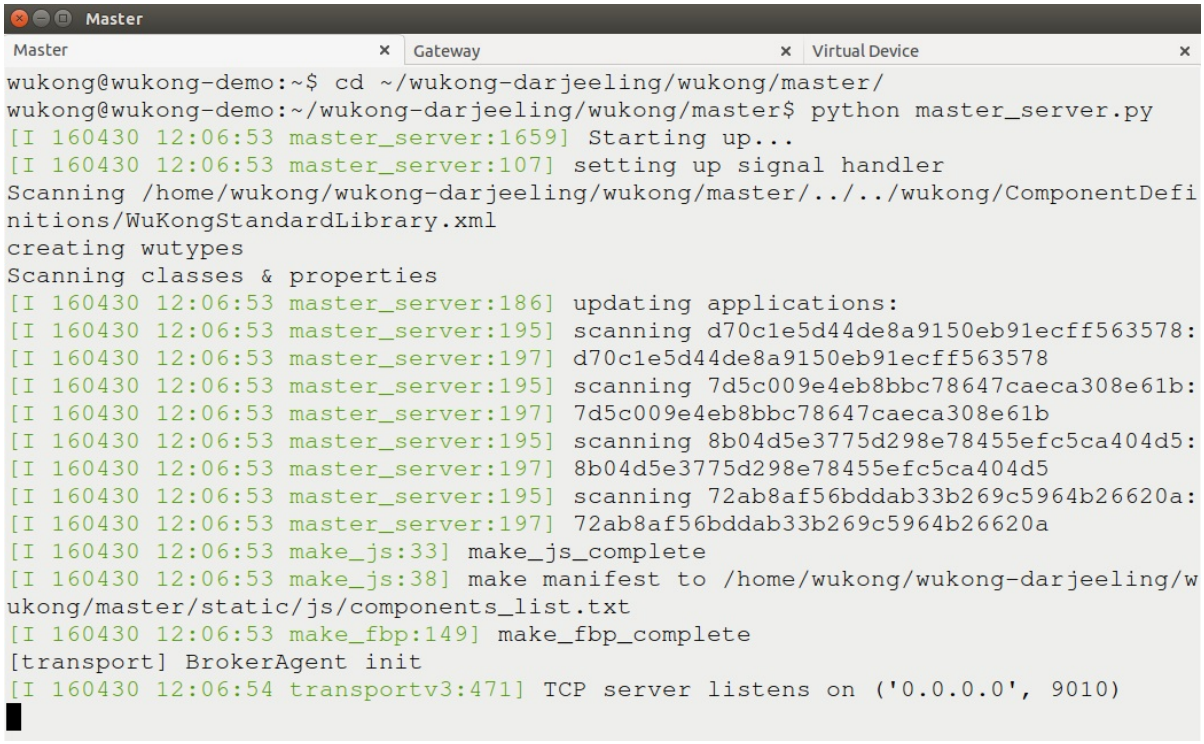


3. Copy the configuration file for Master

```
cd <path_of_source_code>/wukong-darjeeling/wukong/config/
cp master.cfg.dist master.cfg
```

4. Run the Master

```
cd <path_of_source_code>/wukong-darjeeling/wukong/master/
python master_server.py
```

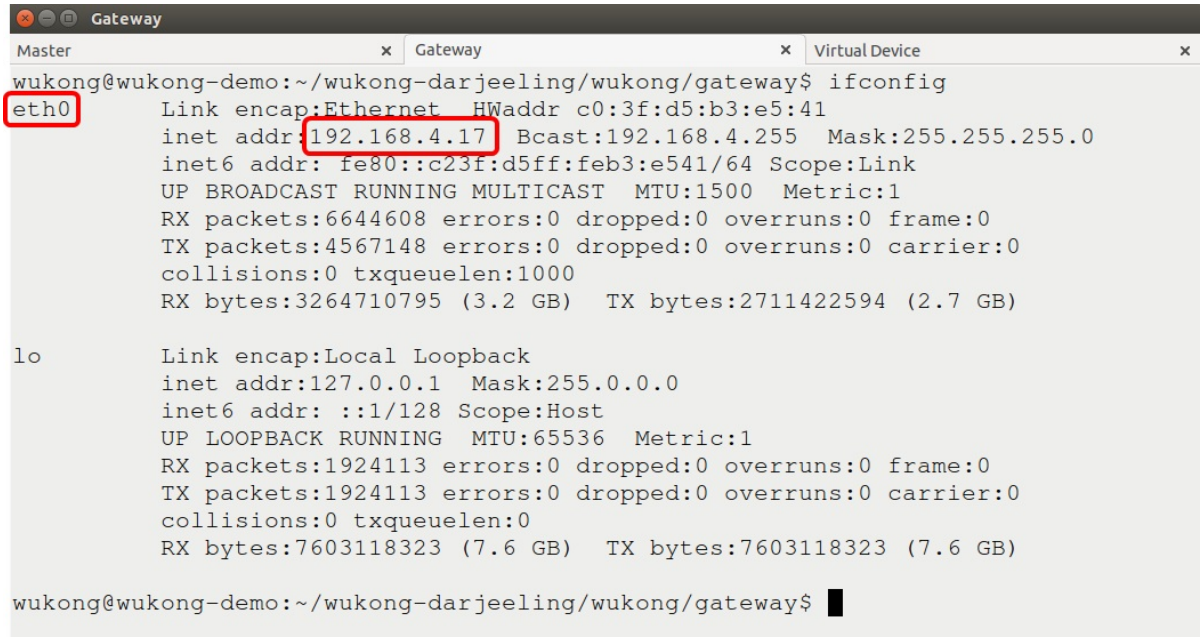


5. On the computer, open a new terminal to copy the configuration file for gateway program

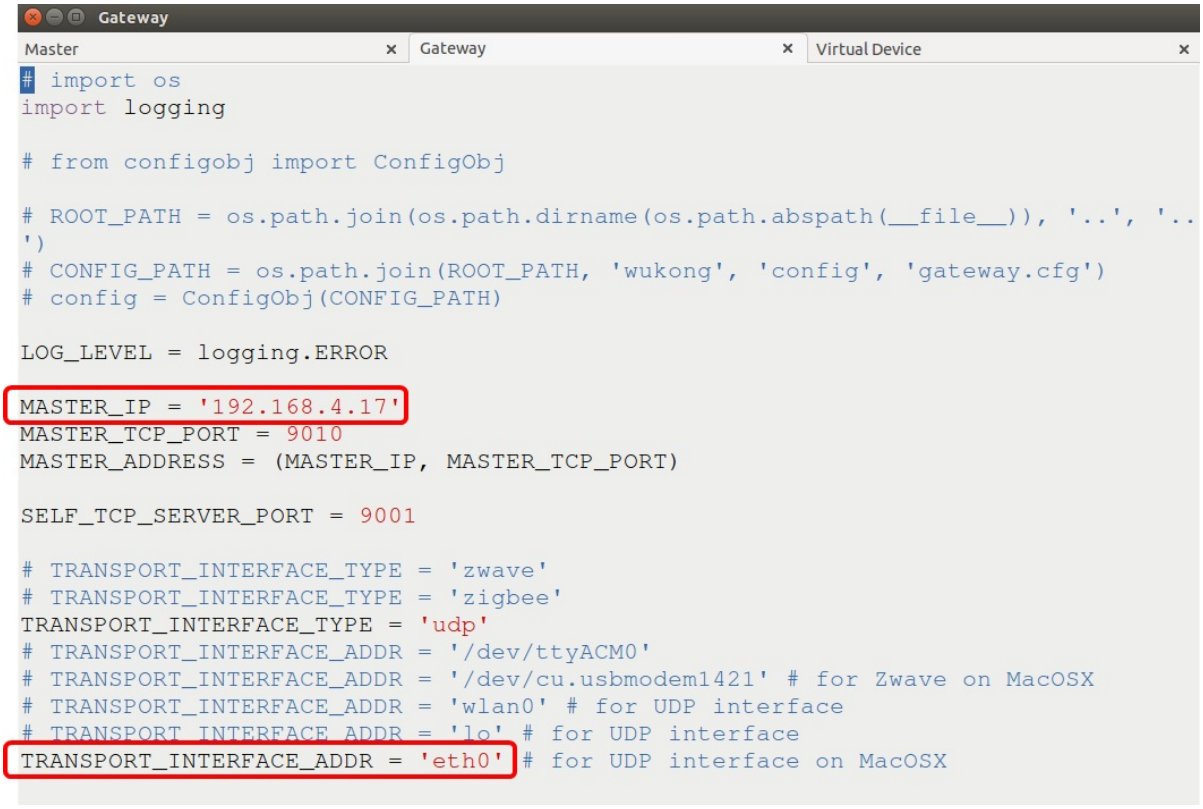
```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
cp gtwconfig.py.dist gtwconfig.py
```

6. Configure gtwconfig.py

```
# Use ifconfig command to check the network information.
ifconfig
```

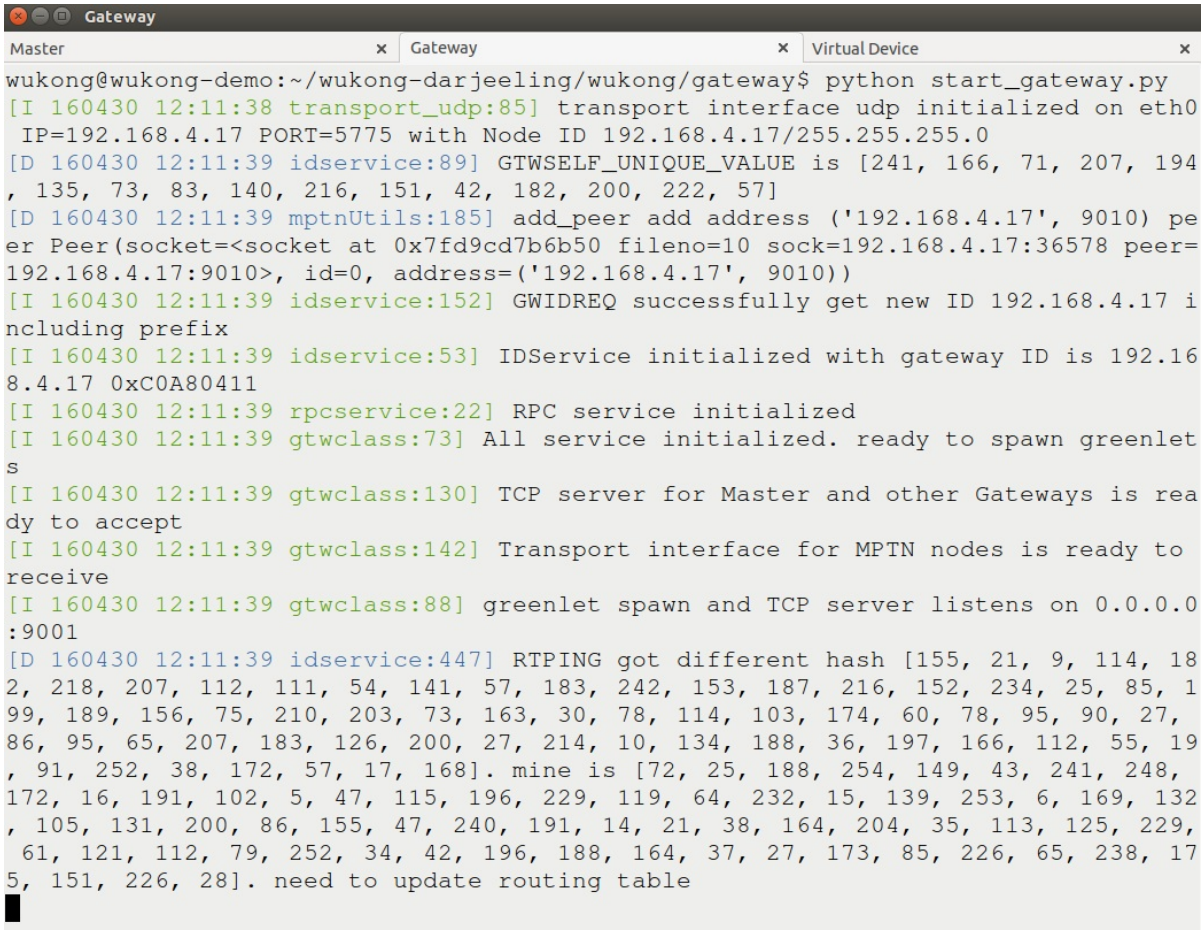


```
vim gtwconfig.py
# Change MASTER_IP to the IP address of the Master.
# Change TRANSPORT_INTERFACE_ADDR according to yout network interface above.
```



7. Run the gateway program

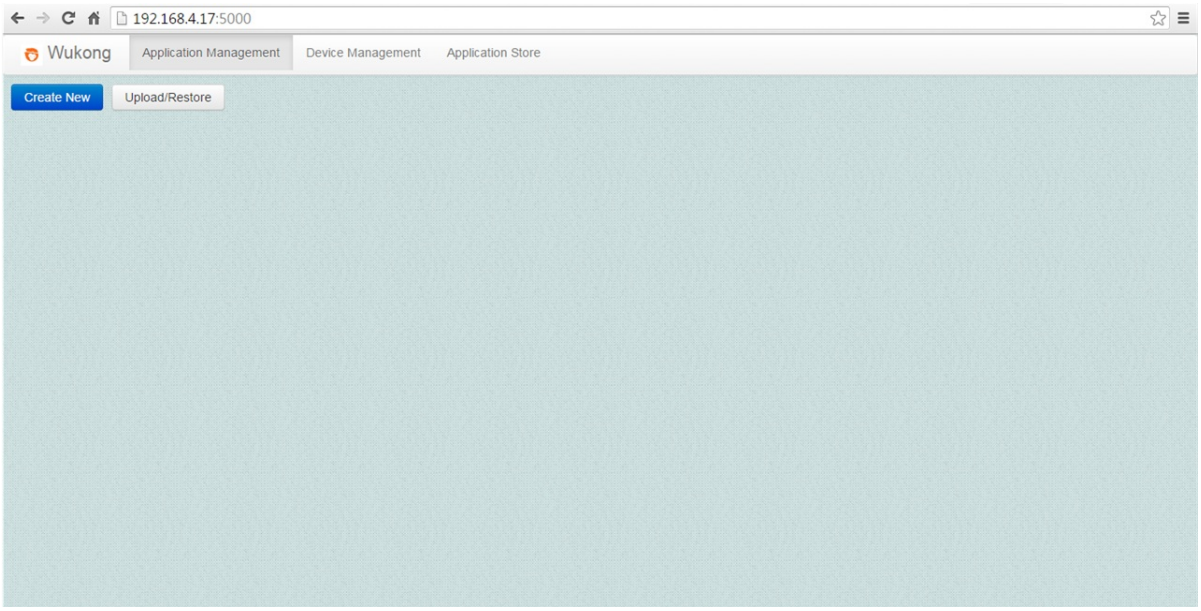
```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
python start_gateway.py
```


Gateway
Master
Gateway
Virtual Device
wukong@wukong-demo:~/wukong-darjeeling/wukong/gateway$ python start_gateway.py
[I 160430 12:11:38 transport_udp:85] transport interface udp initialized on eth0
IP=192.168.4.17 PORT=5775 with Node ID 192.168.4.17/255.255.255.0
[D 160430 12:11:39 idservice:89] GTWSELF_UNIQUE_VALUE is [241, 166, 71, 207, 194
, 135, 73, 83, 140, 216, 151, 42, 182, 200, 222, 57]
[D 160430 12:11:39 mptnUtils:185] add_peer add address ('192.168.4.17', 9010) pe
er Peer(socket=<socket at 0x7fd9cd7b6b50 fileno=10 sock=192.168.4.17:36578 peer=
192.168.4.17:9010>, id=0, address=('192.168.4.17', 9010))
[I 160430 12:11:39 idservice:152] GWIDREQ successfully get new ID 192.168.4.17 i
ncluding prefix
[I 160430 12:11:39 idservice:53] IDService initialized with gateway ID is 192.16
8.4.17 0xC0A80411
[I 160430 12:11:39 rpcservice:22] RPC service initialized
[I 160430 12:11:39 gtwclass:73] All service initialized. ready to spawn greenlet
s
[I 160430 12:11:39 gtwclass:130] TCP server for Master and other Gateways is rea
dy to accept
[I 160430 12:11:39 gtwclass:142] Transport interface for MPTN nodes is ready to
receive
[I 160430 12:11:39 gtwclass:88] greenlet spawn and TCP server listens on 0.0.0.0
:9001
[D 160430 12:11:39 idservice:447] RTPING got different hash [155, 21, 9, 114, 18
2, 218, 207, 112, 111, 54, 141, 57, 183, 242, 153, 187, 216, 152, 234, 25, 85, 1
99, 189, 156, 75, 210, 203, 73, 163, 30, 78, 114, 103, 174, 60, 78, 95, 90, 27,
86, 95, 65, 207, 183, 126, 200, 27, 214, 10, 134, 188, 36, 197, 166, 112, 55, 19
, 91, 252, 38, 172, 57, 17, 168]. mine is [72, 25, 188, 254, 149, 43, 241, 248,
172, 16, 191, 102, 5, 47, 115, 196, 229, 119, 64, 232, 15, 139, 253, 6, 169, 132
, 105, 131, 200, 86, 155, 47, 240, 191, 14, 21, 38, 164, 204, 35, 113, 125, 229,
61, 121, 112, 79, 252, 34, 42, 196, 188, 164, 37, 27, 173, 85, 226, 65, 238, 17
5, 151, 226, 28]. need to update routing table

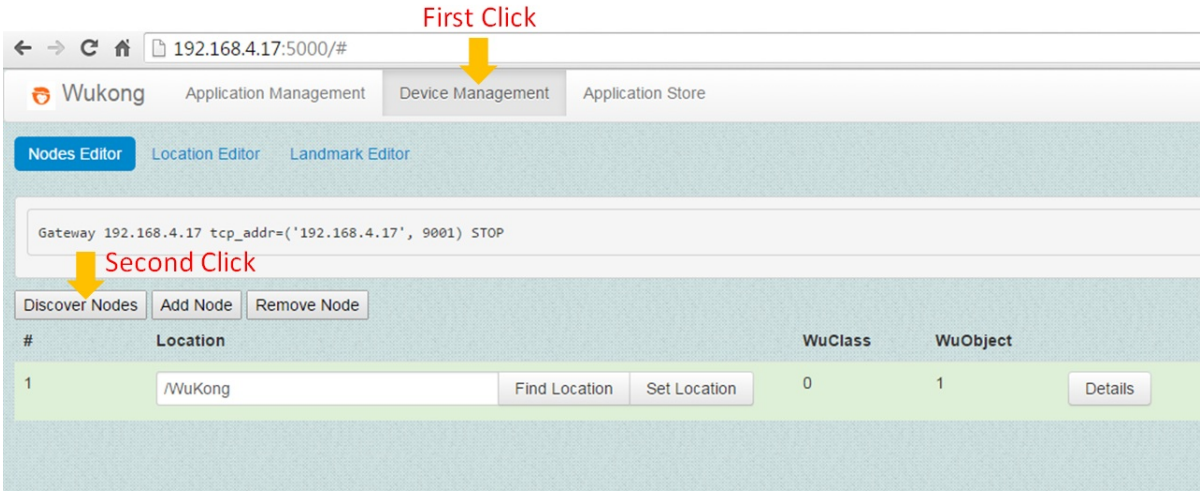
4.1.2 Include a New Device

Before we can deploy any IoT application on WuKong, Master must find the devices available to run it. In this section, we show how to use the device management capability in WuKong Master.

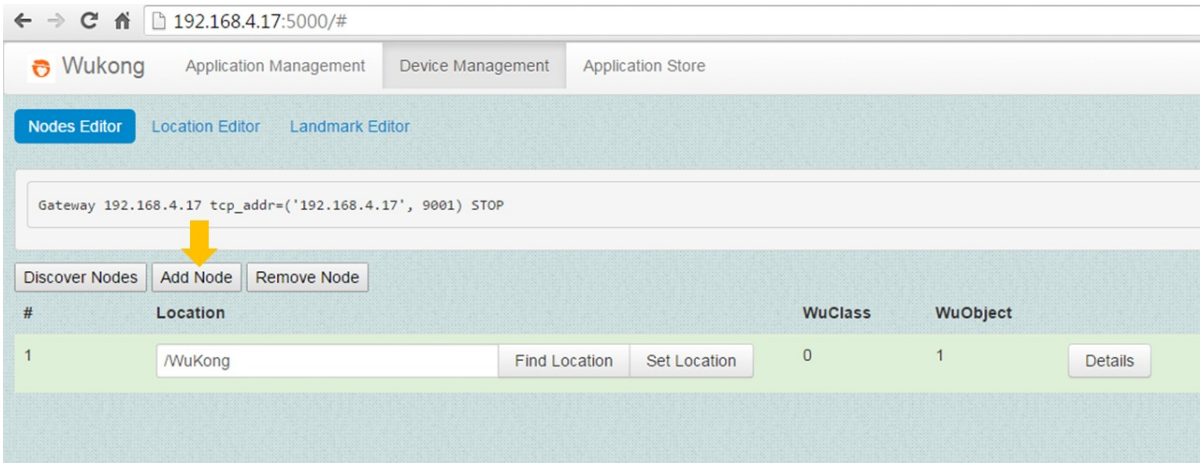
1. Use the Chrome browser to open the Master interface: http://localhost:5000



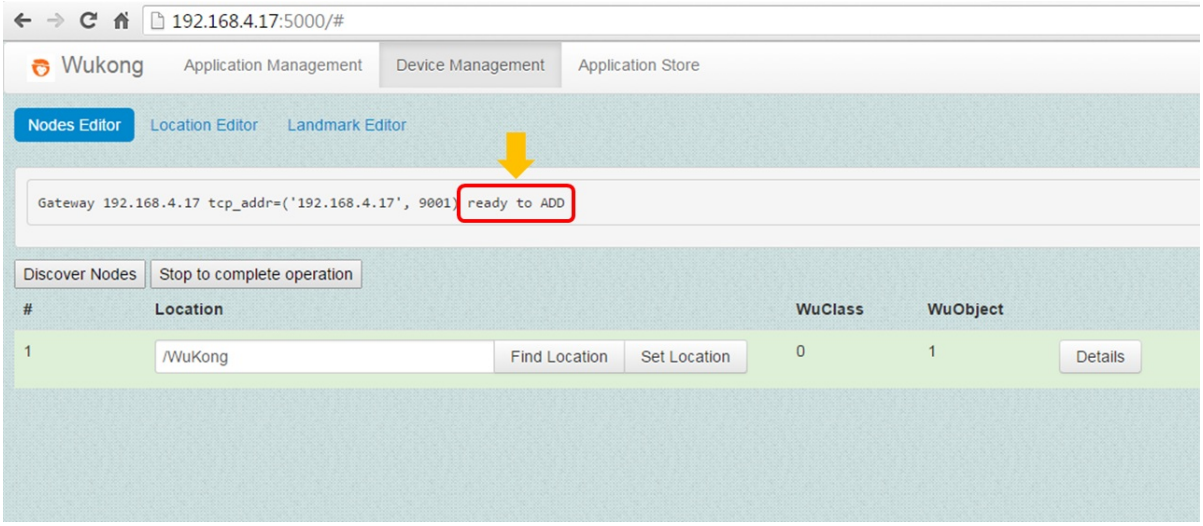
2. Click on the **Device Management** tab and the **Discover Node** button to check the initial state.



3. Click the **Add Node** button to include a new device in Master.



Note: If the message shown is not ready to ADD, press the Stop to complete operation button and try the step again.



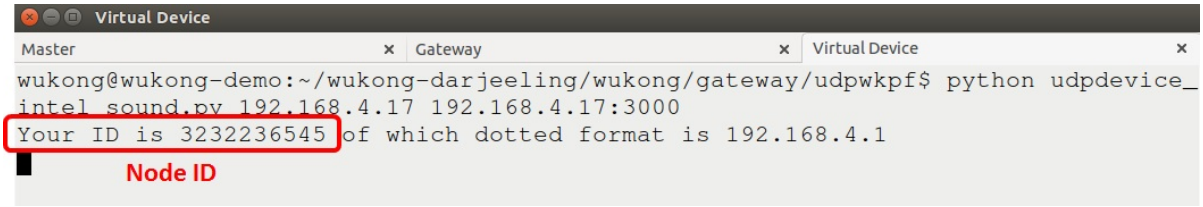
4. Open a terminal and go to the folder where the device program is stored.

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/udpwkpf/
```

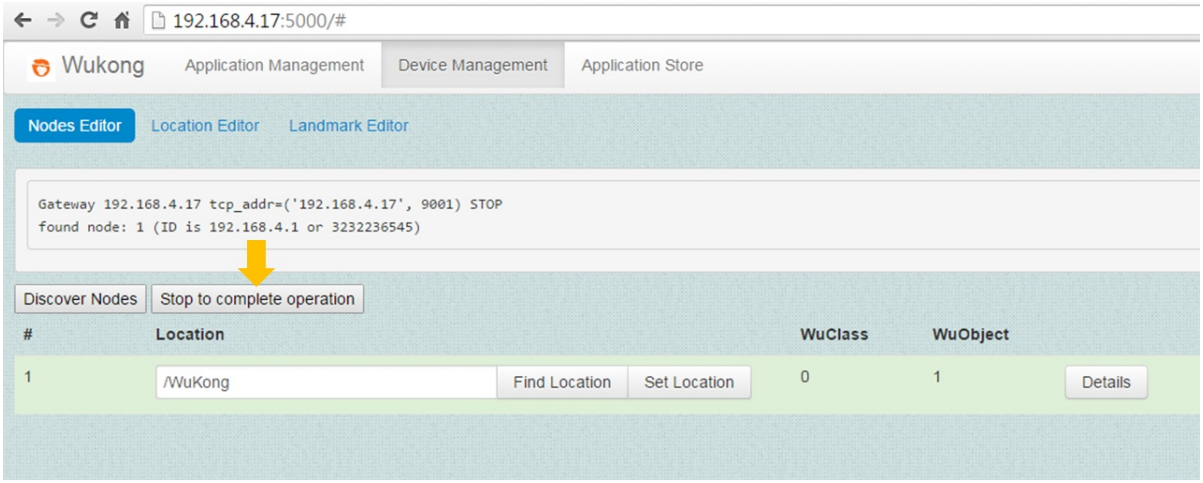
5. Run the device program udpdevice_intel_sound.py. This Python program will start the device to receive a node ID from the WuKong gateway, and then the Intel_Sound WuClass on the device.

```
python udpdevice_intel_sound.py <IP address of gateway> \
<IP address of device program>:<arbitrary port number>
```

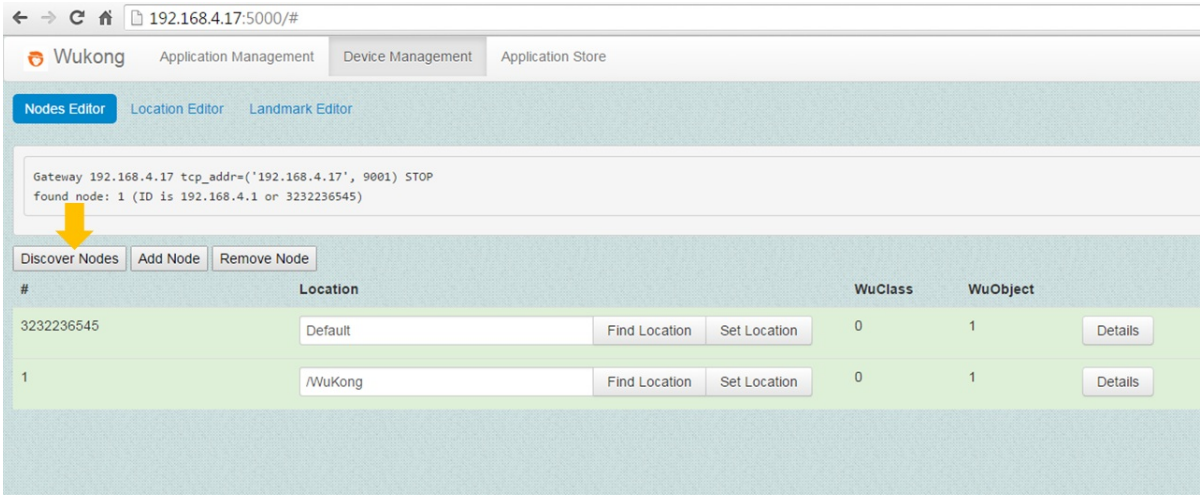
Note: When the device program is started, this device will get a Node ID assigned by the Master.



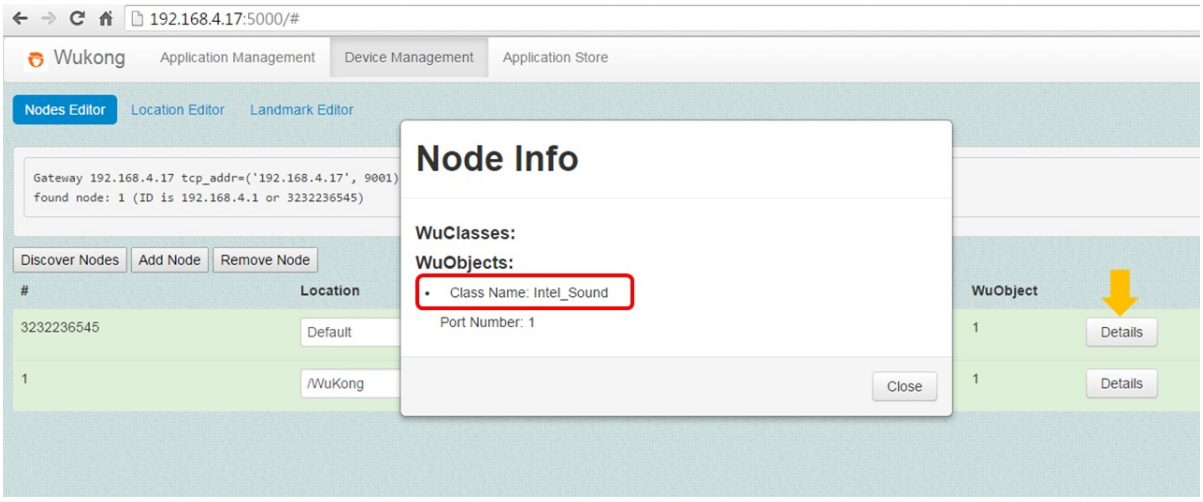
6. Click on the Stop to complete operation button.



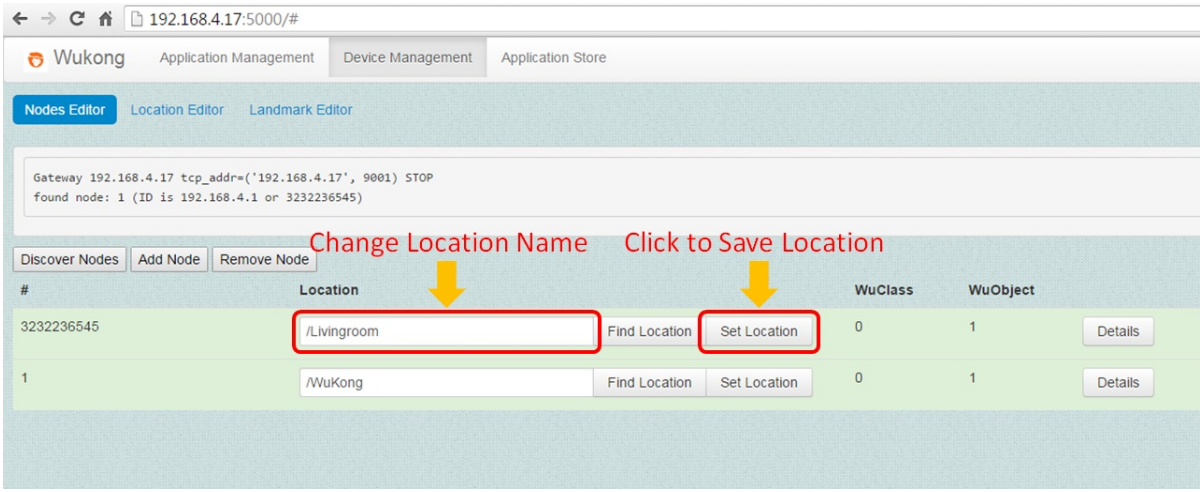
7. Click the Discover Nodes button to refresh the device list.



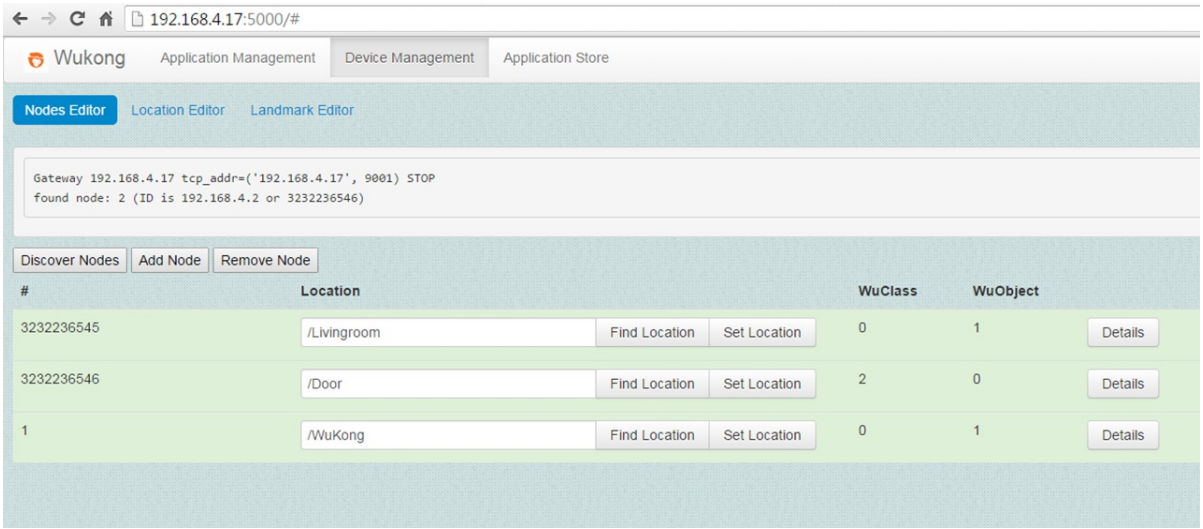
8. The sensor profile of this device can be seen by clicking the Details button.



9. The location of the device can be changed as follows.



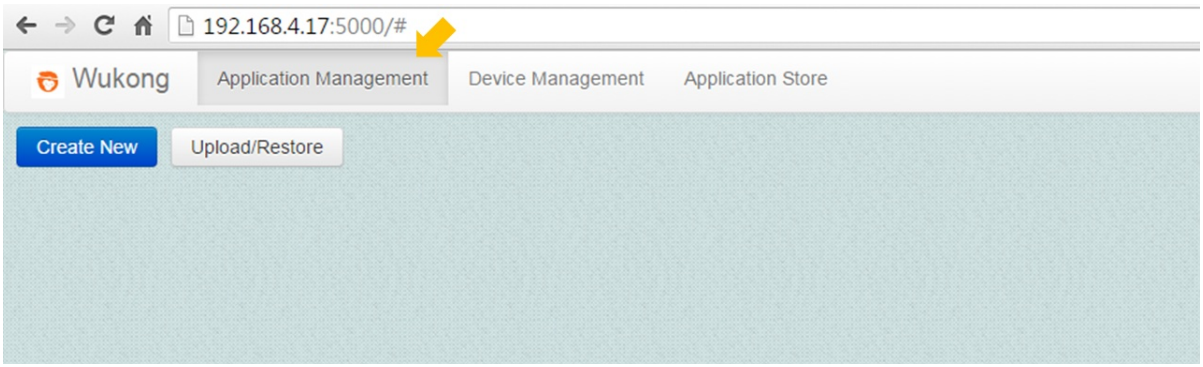
10. Add another device program called udpdevice_logic.py by repeating Step 3 to Step 9. After that, the device management list should be as follows.



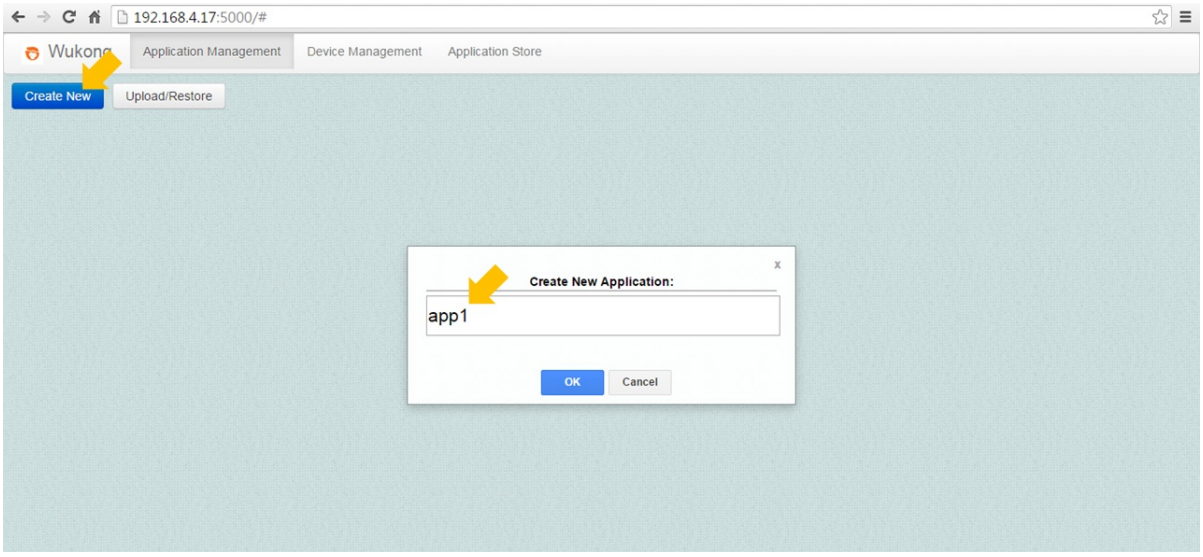
4.1.3 Build an Application FBP

We are now ready to create an application to play the simple theme music.

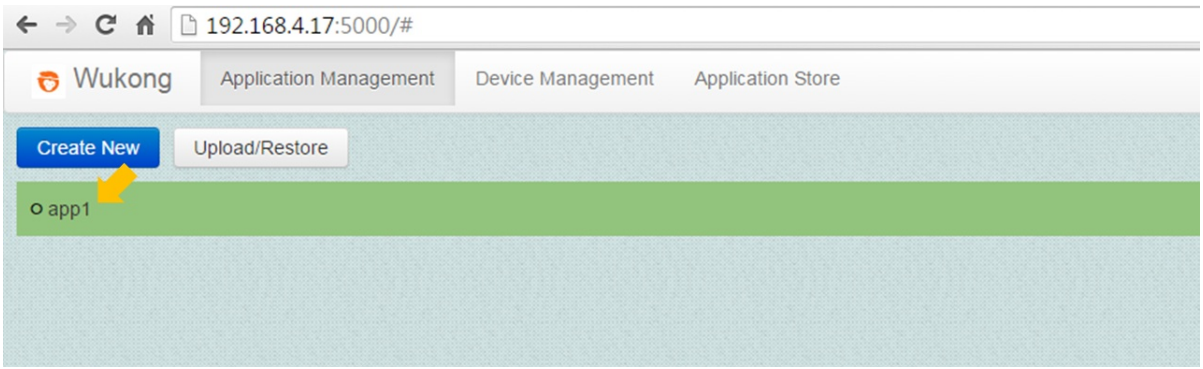
1. Click and enter the **Application Management** page



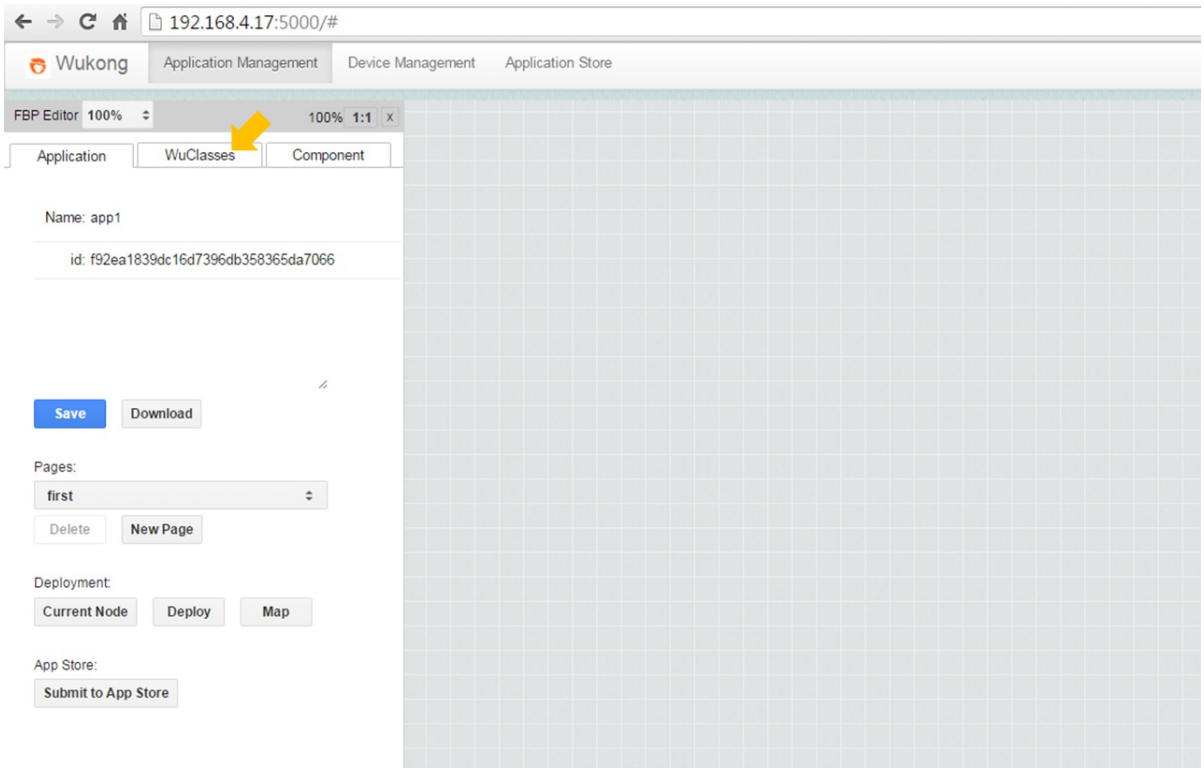
2. Clcik the **Create New** button and then enter the name of the application, "app1" in our example.



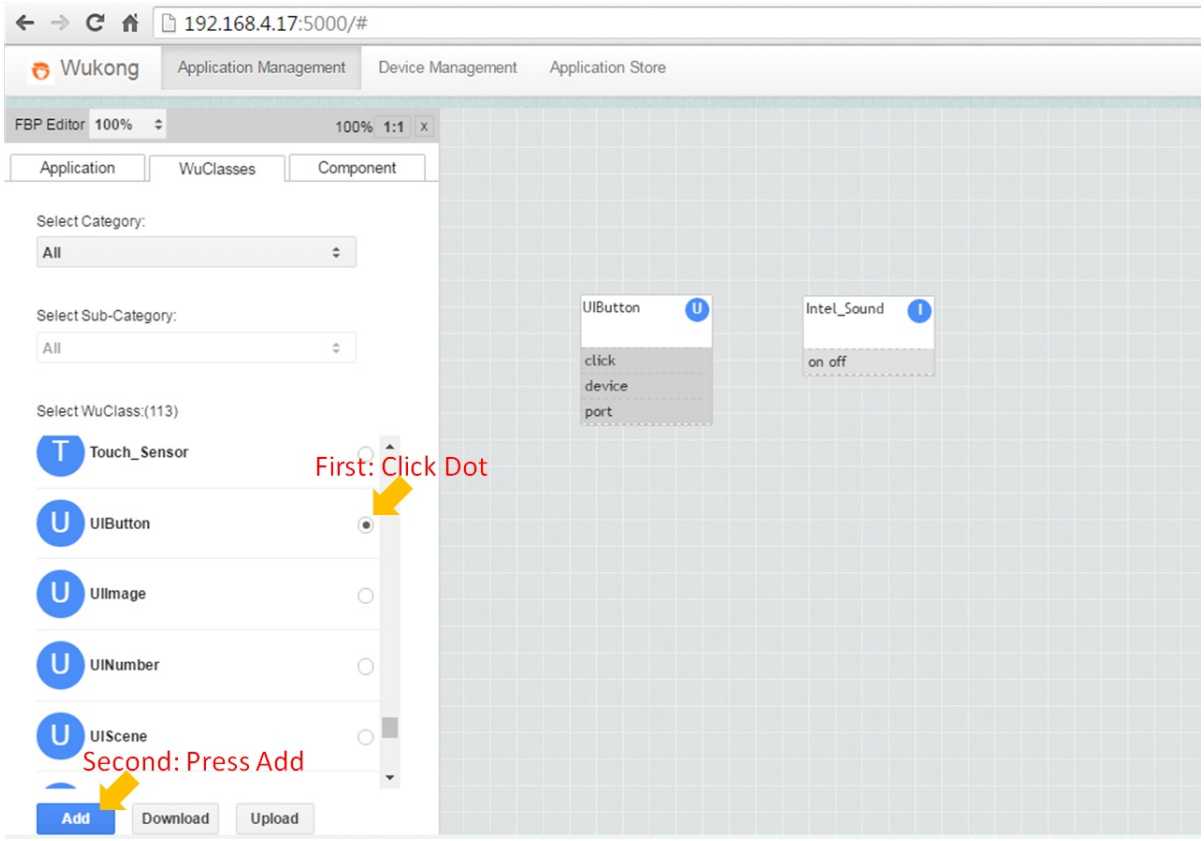
3. Click on the application name just created to open the composition canvas.



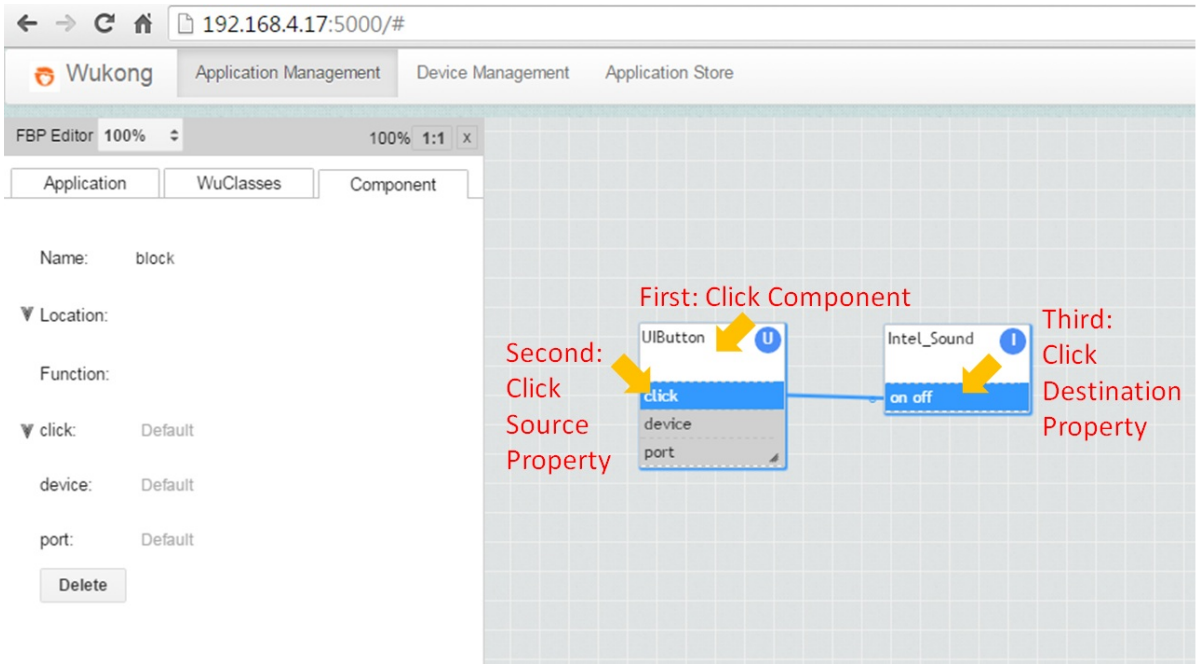
4. Click on the **WuClasses** tab to show all WuClasses available for FBP composition. All existing WuClasses on the Master database will be displayed.



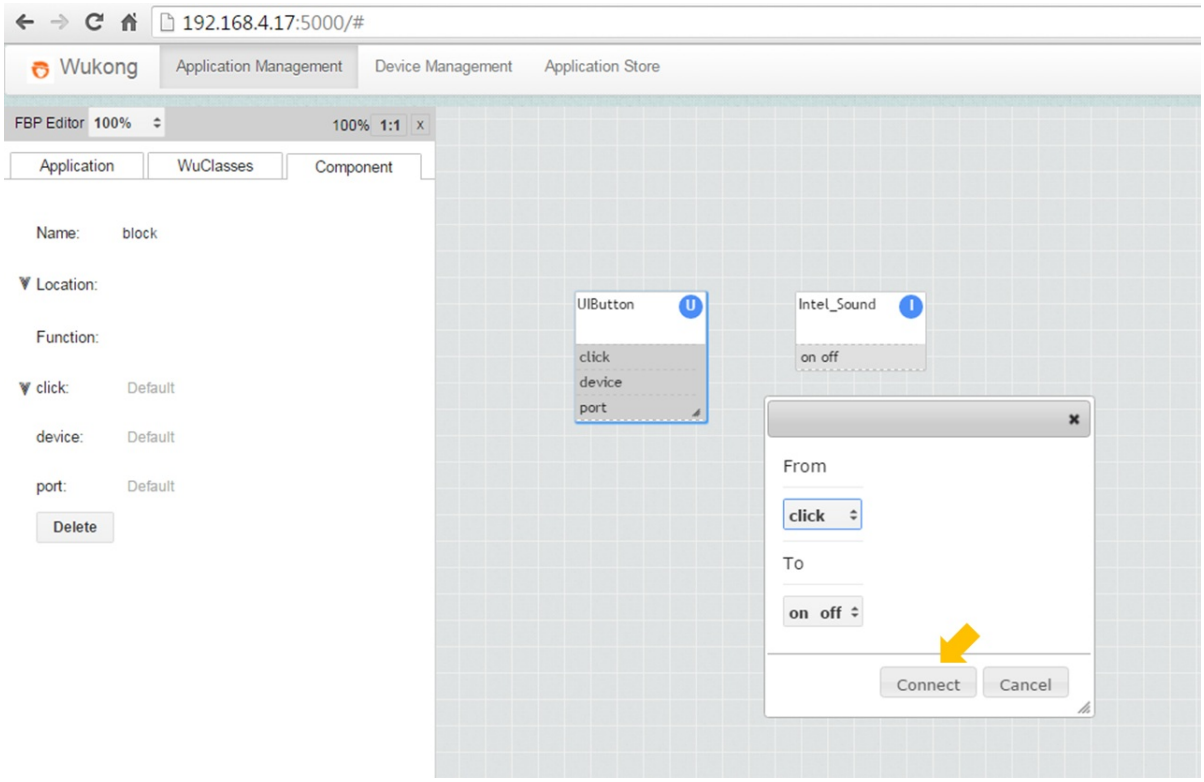
5. Scroll through the WuClass list to find the UIButton component. Click its selection button and the **Add** button at the bottom. Repeat this step for the Intel_Sound component. Two component boxes will be shown on the composition canvas.



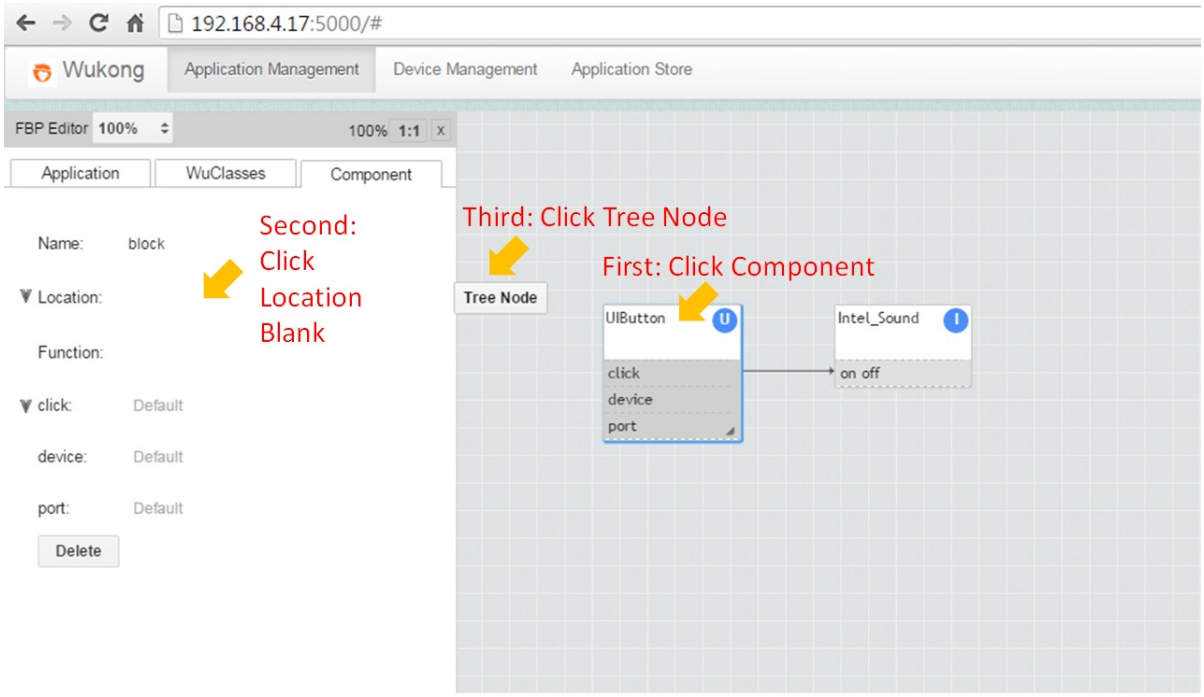
6. To add a link between the two components, first click on the source component. After the source component has been highlighted, click on the source property, the "click" property in this example. Then, move the cursor and click on the destination property of the destination component.

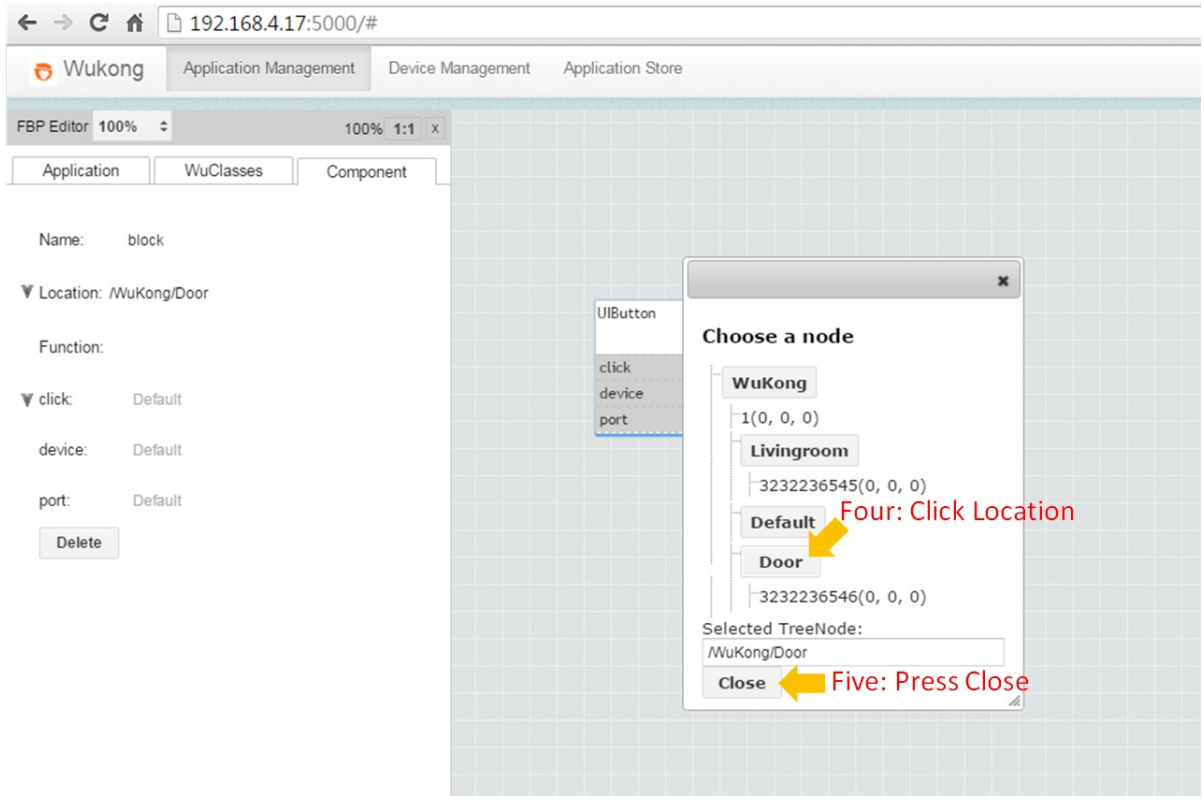


7. Check whether the source and the destination of the data link created are correct. If not, you can change them in the dropout menu.

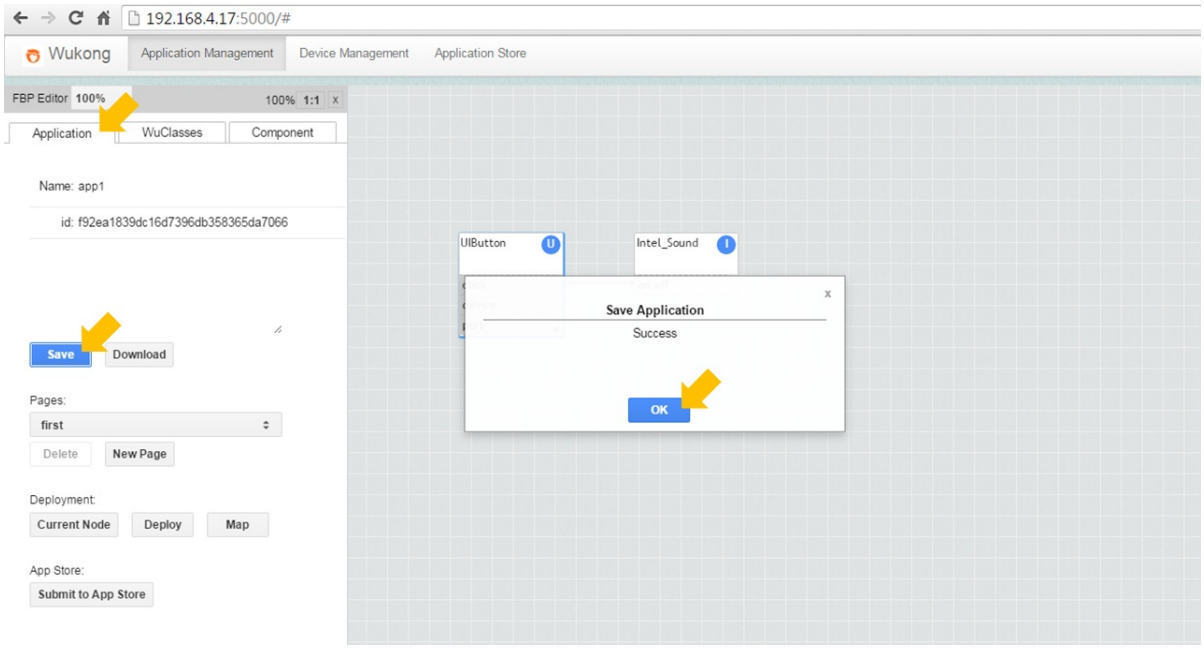


8. (Optional) Enter the location for each component using the 5 steps in the following figures. Note that even if the location of each component is left empty, Master can still choose a suitable device for each component according to the device's sensor profile.

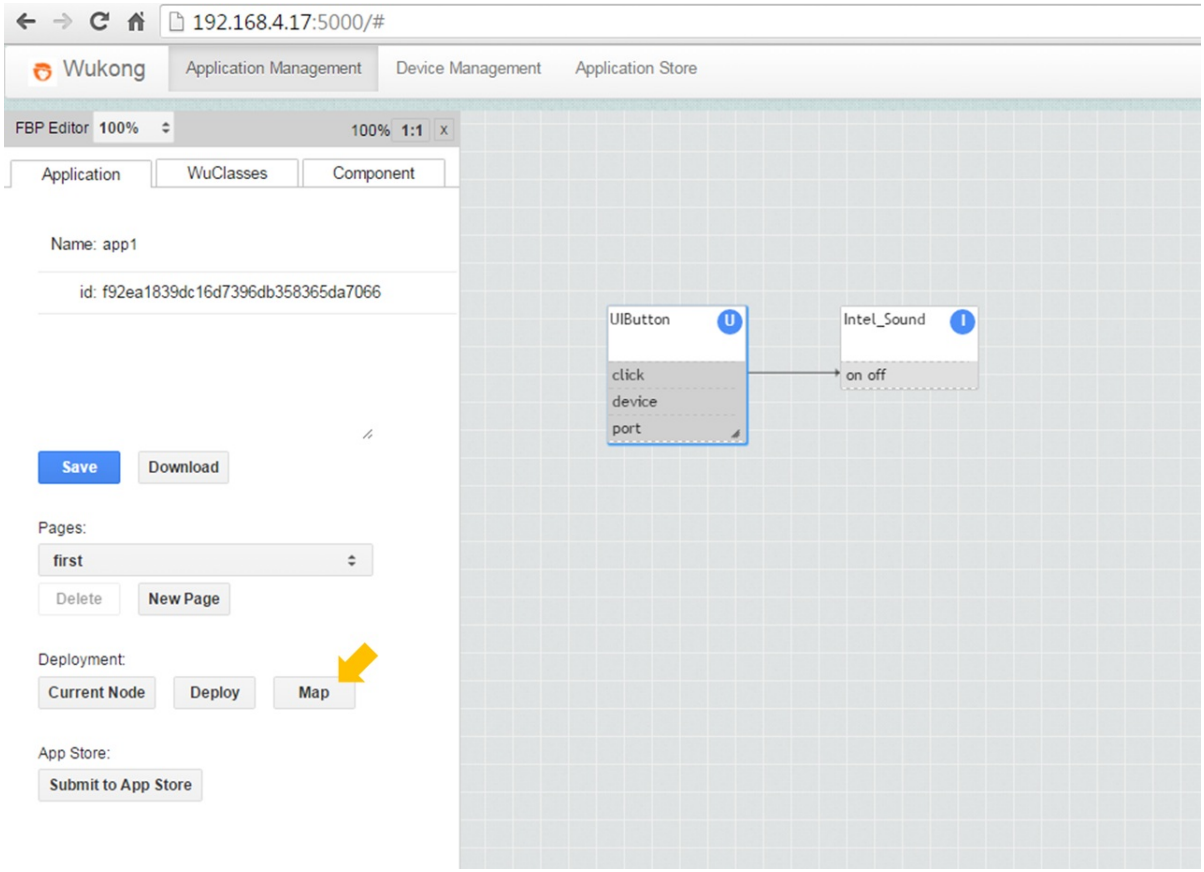




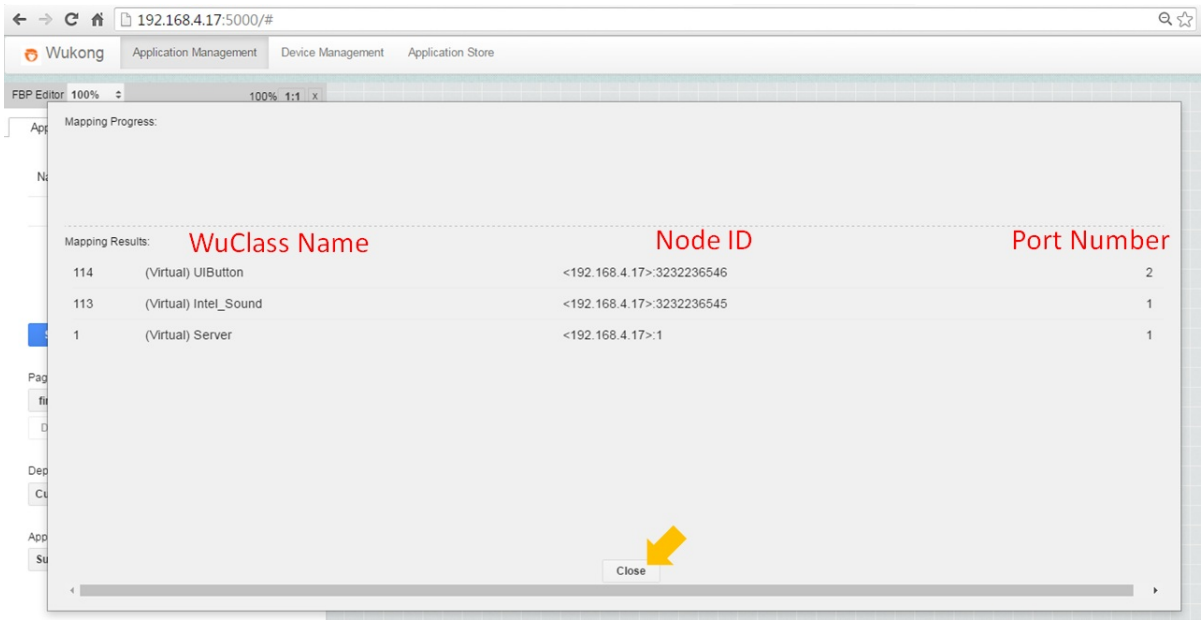
9. Click on the **Application** tab and the **Save** button.



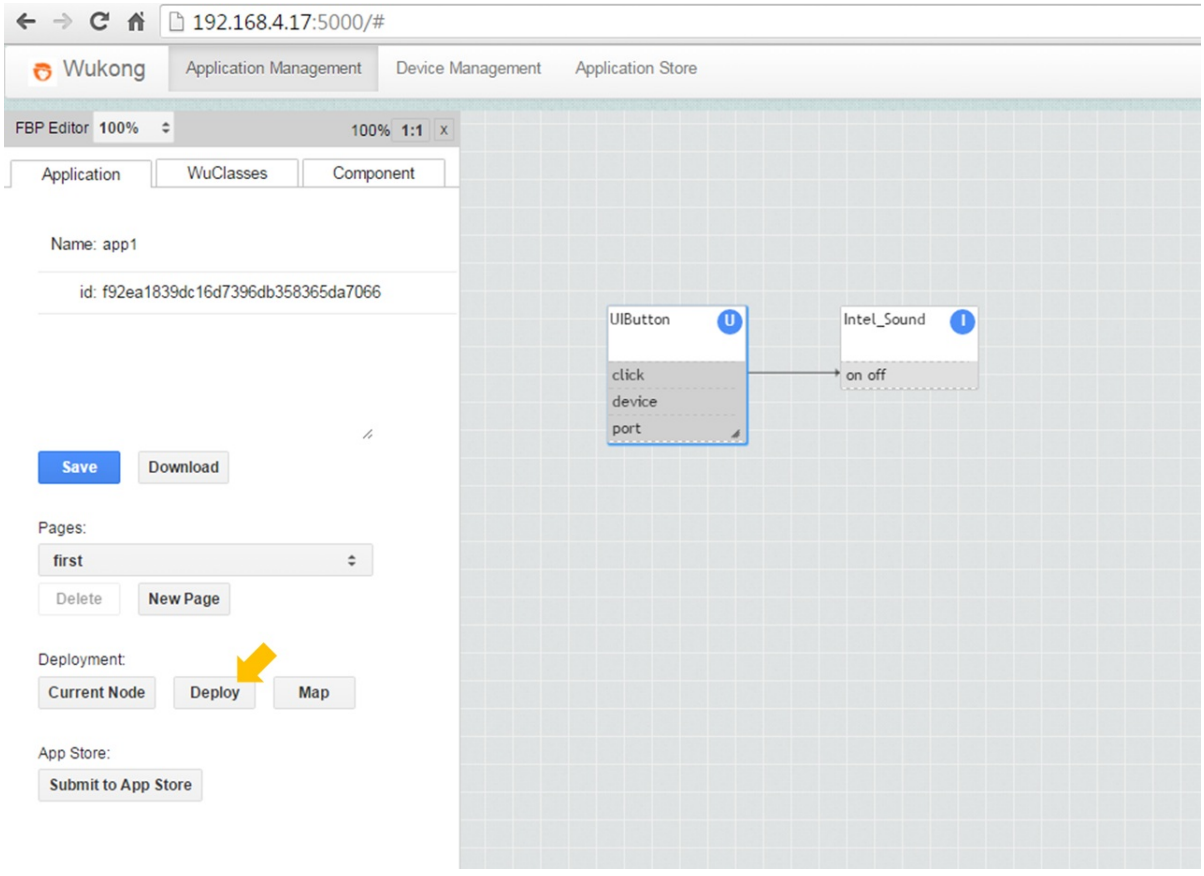
10. Click on the **Map** button to select the device to run each component.



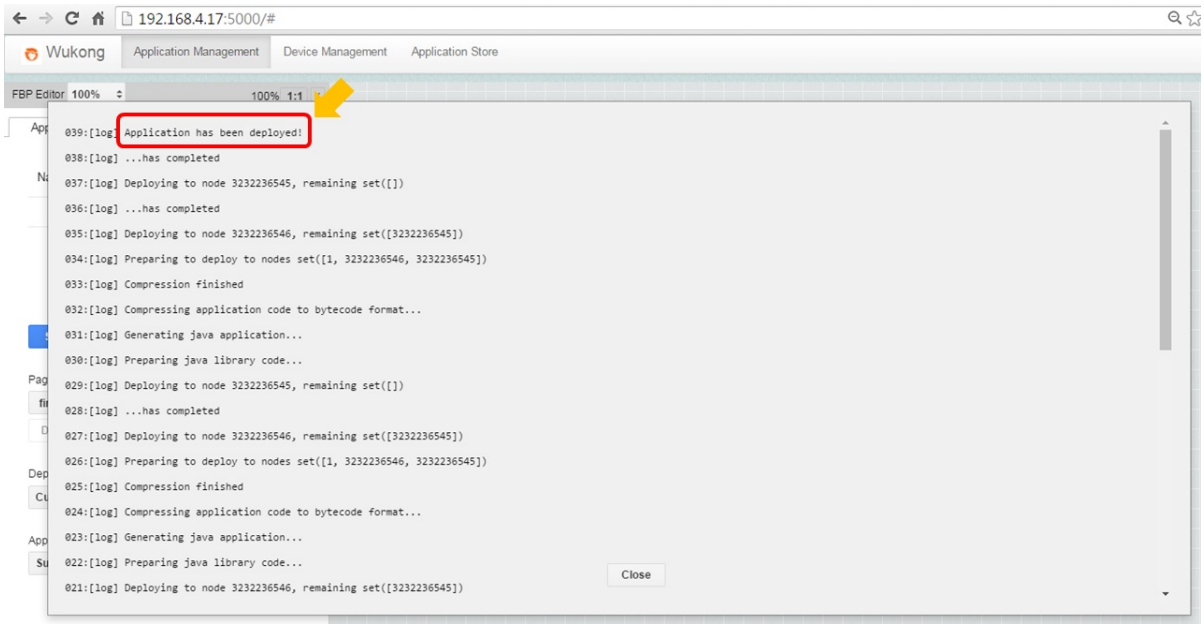
11. After the mapping procedure, the mapping results will be displayed as follows.



12. If everything is successful, you can now click on the **Deploy** button to run the application.

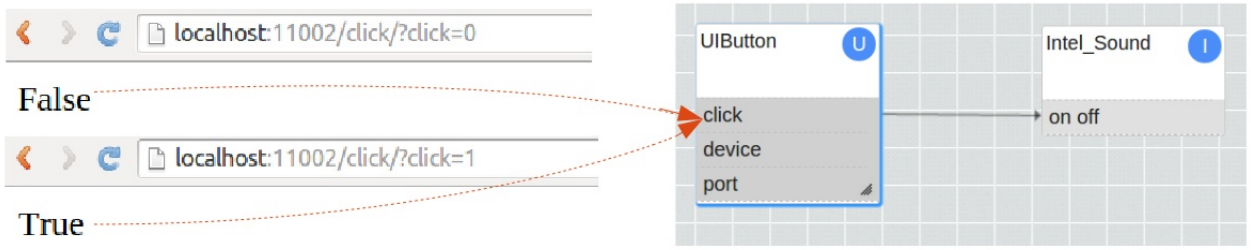


13. Messages will be displayed if WuKong is able to deploy the application successfully.


192.168.4.17:5000/#
Wukong
Application Management
Device Management
Application Store
FBP Editor 100%
100% 1:1
039:[log] Application has been deployed!
038:[log] ...has completed
037:[log] Deploying to node 3232236545, remaining set([])
036:[log] ...has completed
035:[log] Deploying to node 3232236546, remaining set([3232236545])
034:[log] Preparing to deploy to nodes set([1, 3232236546, 3232236545])
033:[log] Compression finished
032:[log] Compressing application code to bytecode format...
031:[log] Generating java application...
030:[log] Preparing java library code...
029:[log] Deploying to node 3232236545, remaining set([])
028:[log] ...has completed
027:[log] Deploying to node 3232236546, remaining set([3232236545])
026:[log] Preparing to deploy to nodes set([1, 3232236546, 3232236545])
025:[log] Compression finished
024:[log] Compressing application code to bytecode format...
023:[log] Generating java application...
022:[log] Preparing java library code...
021:[log] Deploying to node 3232236546, remaining set([3232236545])
Close

4.1.4 Running the Application

After the deployment, we can use the http requests to send 0 or 1 to UIButton and check if the theme music is played. Since WuKong adopts the even-driven model, data will propagate along a link only when its value is changed; therefore, we must send 0 before sending 1.



4.1.5. Discovery Issue

One common issue for **node discovery** happens when computers or IoT devices are moved to another network environment. Although they obtain new IP addresses, Master, gateways, or devices may still use old cached files that have old IP addresses.

To solve the problem, whenever the WuKong network environment is changed, you must follow the instructions below to reset the WuKong system to its default settings. Moreover, even if some nodes are in the same network but still cannot be discovered, use the instructions to reset the system as well.

```
cd <path_source_code>/wukong-darjeeling/wukong/master
rm *.pyc *.json change*

# Go to the WuKong Master directory
# Remove the compiled Python files, JSON files, and also the changeset.
# changeset is a history record of deployment.
```

```
cd <path_source_code>/wukong-darjeeling/wukong/gateway
rm *.pyc *.json device*

# Go to the WuKong gateway directory
# Remove the compiled Python files, JSON files, and the devices.pkl.
# The device.pkl is a file that includes the discovery result for gateway program.
```

```
cd <path_source_code>/wukong-darjeeling/wukong/gateway/udpwkpf
rm *.pyc *.json

# Go to the device program directory
# Remove the compiled Python files and the JSON
# The *.json are files that include detailed information of each device.
```

After resetting a WuKong system to its default settings, you should restart Master and gateway, and then add devices to the WuKong system again.

4.2 LED Control on Devices

In this section, we show how to run Master on a computer as in the last section, but the gateway and FBP components on an IoT board.

Similar to the last section, the instructions are divided into four steps:

Step 1. Start Master and gateway

Step 2. Include a New Device

Step 3. Build and Deploy an FBP.

Step 4. Running the application

4.2.1 Start Master and Gateway

The instructions in this section are slightly different from Section 4.1.1 since the gateway software in run on an IoT board (either Edison or Raspberry Pi). Therefore, after starting Master, we will access the IoT board remotely to start the gateway program.

Note: Before entering this section, if you have run the example in the last section, you need to follow the instructions in the Section 4.1.5 to flush the cache files.

1. On the computer, download the source code from github as follows.

```
git clone -b release0.4 http://github.com/wukong-m2m/wukong-darjeeling
```

2. Build infuser

```
cd <path_of_source_code>/wukong-darjeeling/src/infuser/
gradle
```

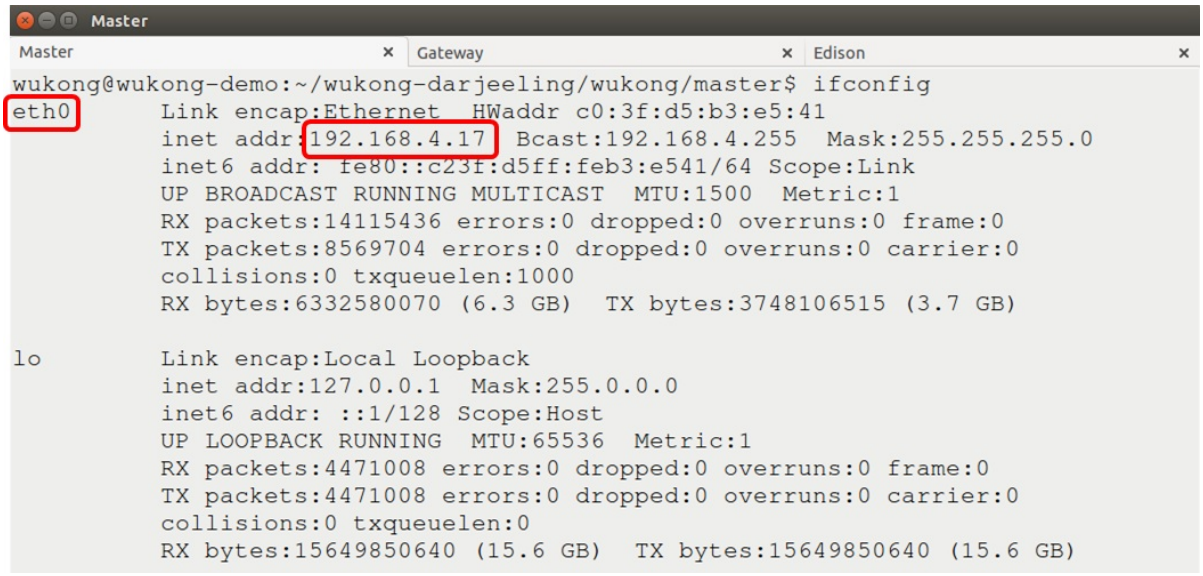
3. Copy the configuration file for Master

```
cd <path_of_source_code>/wukong-darjeeling/wukong/config/
cp master.cfg.dist master.cfg
```

4. Check the network information for Master.

```
ifconfig
# In this case, the network interface of Master is eth0.
# The IP address of Master is 192.168.4.17
```

Note: This IP address will be the MASTER_IP at step 9.



5. Start Master

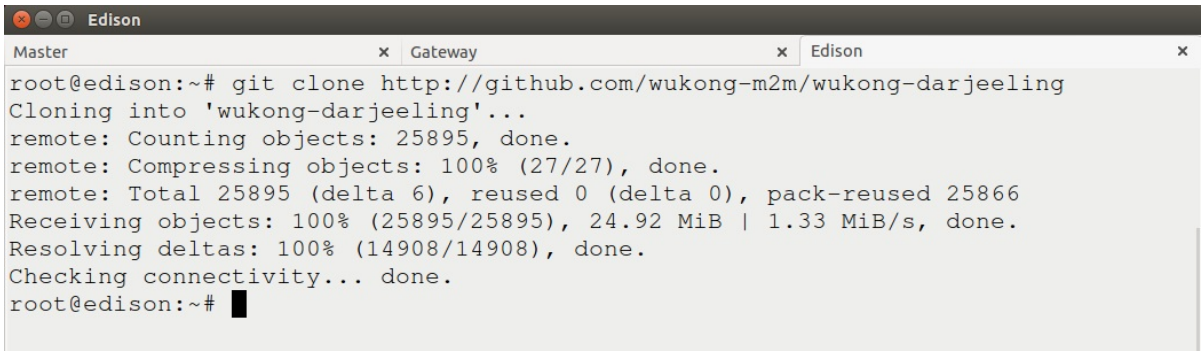
```
cd <path_of_source_code>/wukong-darjeeling/wukong/master/
python master_server.py
```

6. From the computer, open a new terminal and use the SSH command to connect to the IoT board. Although we use Edison in this section, the same operations can be used on Raspberry Pi.

```
ssh root@<IP address of Intel Edison board>
```

7. Download the source code to the IoT board after SSH is successful started.

```
git clone -b release0.4 http://github.com/wukong-m2m/wukong-darjeeling
```

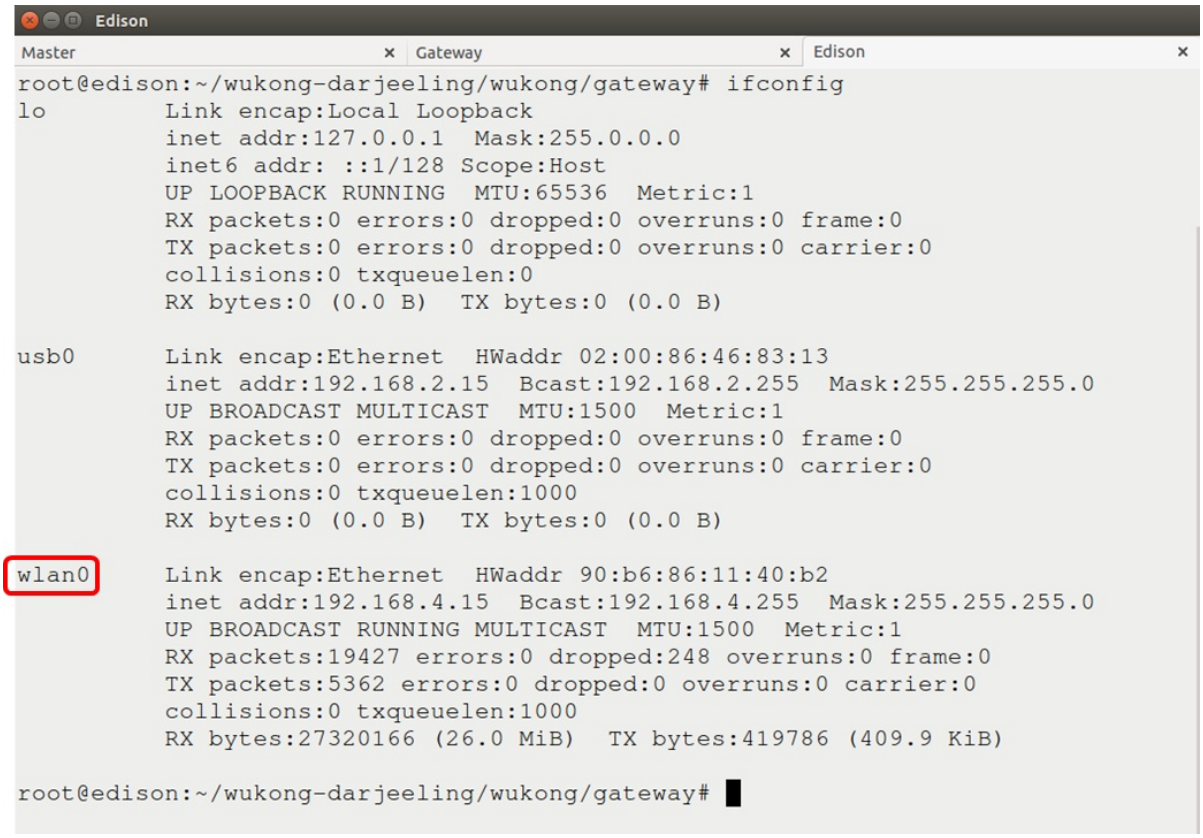


8. Copy the configuration file for gateway.

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
cp gtwconfig.py.dist gtwconfig.py
```

9. Configure gtwconfig.py according to the network information.

```
ifconfig
# Use this command to check network interface of IoT board.
# In this case, the network interface of Edison is wlan0.
```




```
vim gtwconfig.py
# if you cannot use vim, please install it with "opkg install vim"
# Change MASTER_IP to IP address of your computer
# Change TRANSPORT_INTERFACE_ADDR to network interface of IoT board
```

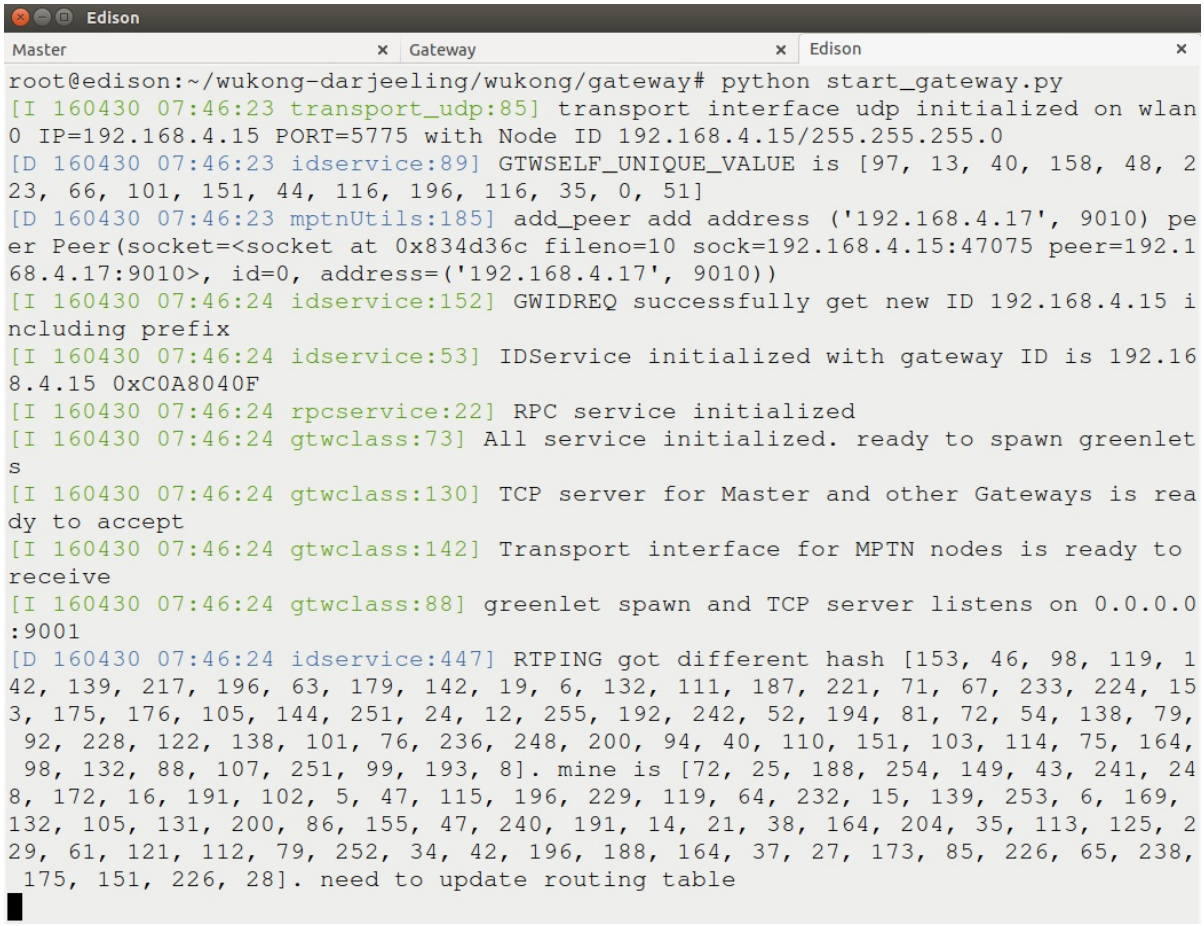
Edison

Master | Gateway | Edison

```
# import os
import logging

# from configobj import ConfigObj

# ROOT_PATH = os.path.join(os.path.dirname(os.path.abspath(__file__)), '..', '..
')
# CONFIG_PATH = os.path.join(ROOT_PATH, 'wukong', 'config', 'gateway.cfg')
# config = ConfigObj(CONFIG_PATH)

LOG_LEVEL = logging.ERROR

MASTER_IP = '192.168.4.17'
MASTER_TCP_PORT = 9010
MASTER_ADDRESS = (MASTER_IP, MASTER_TCP_PORT)

SELF_TCP_SERVER_PORT = 9001

# TRANSPORT_INTERFACE_TYPE = 'zwave'
# TRANSPORT_INTERFACE_TYPE = 'zigbee'
TRANSPORT_INTERFACE_TYPE = 'udp'
# TRANSPORT_INTERFACE_ADDR = '/dev/ttyACM0'
# TRANSPORT_INTERFACE_ADDR = '/dev/cu.usbmodem1421' # for Zwave on MacOSX
TRANSPORT_INTERFACE_ADDR = 'wlan0' # for UDP interface
# TRANSPORT_INTERFACE_ADDR = 'lo' # for UDP interface
# TRANSPORT_INTERFACE_ADDR = 'eth0' # for UDP interface on MacOSX
```

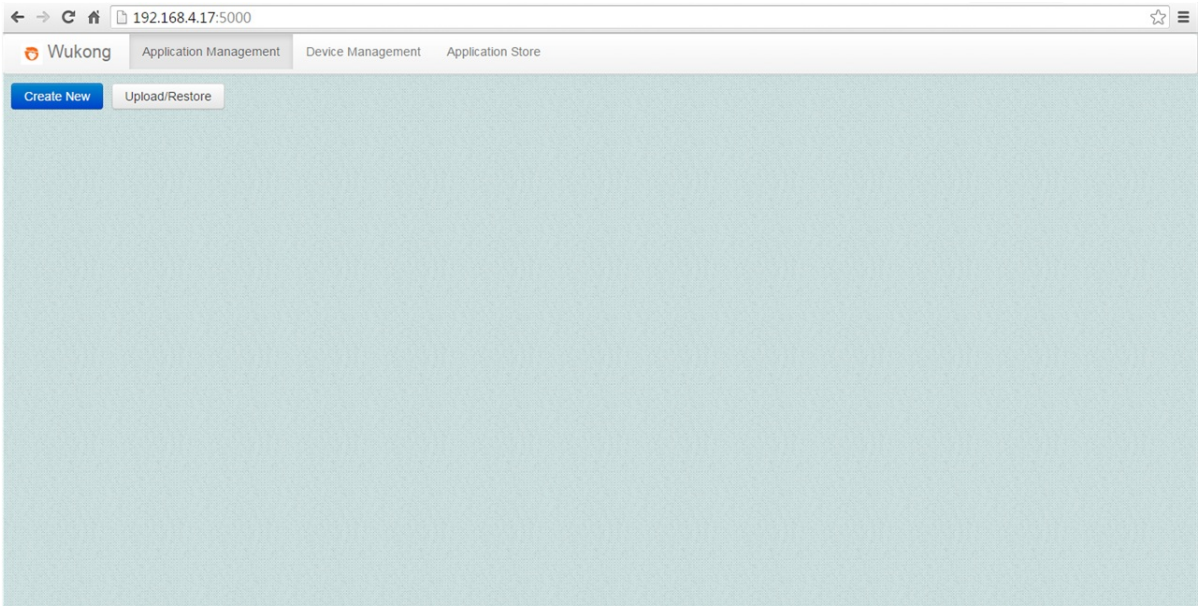
10. Start gateway program on the IoT board

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
python start_gateway.py
```

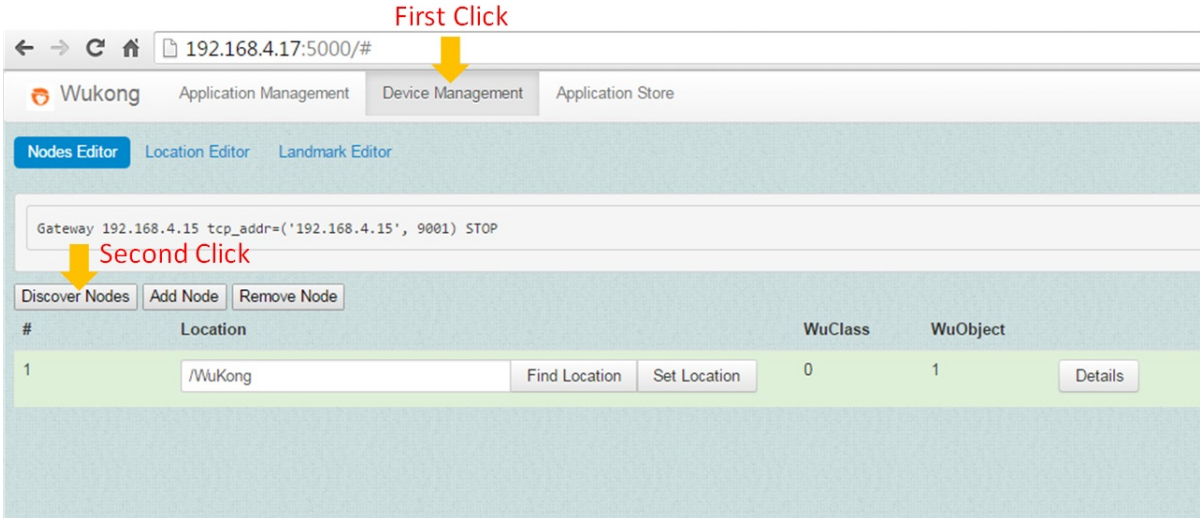

Edison
Master
Gateway
Edison
root@edison:~/wukong-darjeeling/wukong/gateway# python start_gateway.py
[I 160430 07:46:23 transport_udp:85] transport interface udp initialized on wlan0 IP=192.168.4.15 PORT=5775 with Node ID 192.168.4.15/255.255.255.0
[D 160430 07:46:23 idservice:89] GTWSELF_UNIQUE_VALUE is [97, 13, 40, 158, 48, 223, 66, 101, 151, 44, 116, 196, 116, 35, 0, 51]
[D 160430 07:46:23 mptnUtils:185] add_peer add address ('192.168.4.17', 9010) peer Peer(socket=<socket at 0x834d36c fileno=10 sock=192.168.4.15:47075 peer=192.168.4.17:9010>, id=0, address=('192.168.4.17', 9010))
[I 160430 07:46:24 idservice:152] GWIDREQ successfully get new ID 192.168.4.15 including prefix
[I 160430 07:46:24 idservice:53] IDService initialized with gateway ID is 192.168.4.15 0xC0A8040F
[I 160430 07:46:24 rpcservice:22] RPC service initialized
[I 160430 07:46:24 gtwclass:73] All service initialized. ready to spawn greenlets
[I 160430 07:46:24 gtwclass:130] TCP server for Master and other Gateways is ready to accept
[I 160430 07:46:24 gtwclass:142] Transport interface for MPTN nodes is ready to receive
[I 160430 07:46:24 gtwclass:88] greenlet spawn and TCP server listens on 0.0.0.0:9001
[D 160430 07:46:24 idservice:447] RTPING got different hash [153, 46, 98, 119, 142, 139, 217, 196, 63, 179, 142, 19, 6, 132, 111, 187, 221, 71, 67, 233, 224, 153, 175, 176, 105, 144, 251, 24, 12, 255, 192, 242, 52, 194, 81, 72, 54, 138, 79, 92, 228, 122, 138, 101, 76, 236, 248, 200, 94, 40, 110, 151, 103, 114, 75, 164, 98, 132, 88, 107, 251, 99, 193, 8]. mine is [72, 25, 188, 254, 149, 43, 241, 248, 172, 16, 191, 102, 5, 47, 115, 196, 229, 119, 64, 232, 15, 139, 253, 6, 169, 132, 105, 131, 200, 86, 155, 47, 240, 191, 14, 21, 38, 164, 204, 35, 113, 125, 229, 61, 121, 112, 79, 252, 34, 42, 196, 188, 164, 37, 27, 173, 85, 226, 65, 238, 175, 151, 226, 28]. need to update routing table

4.2.2 Include a New Device

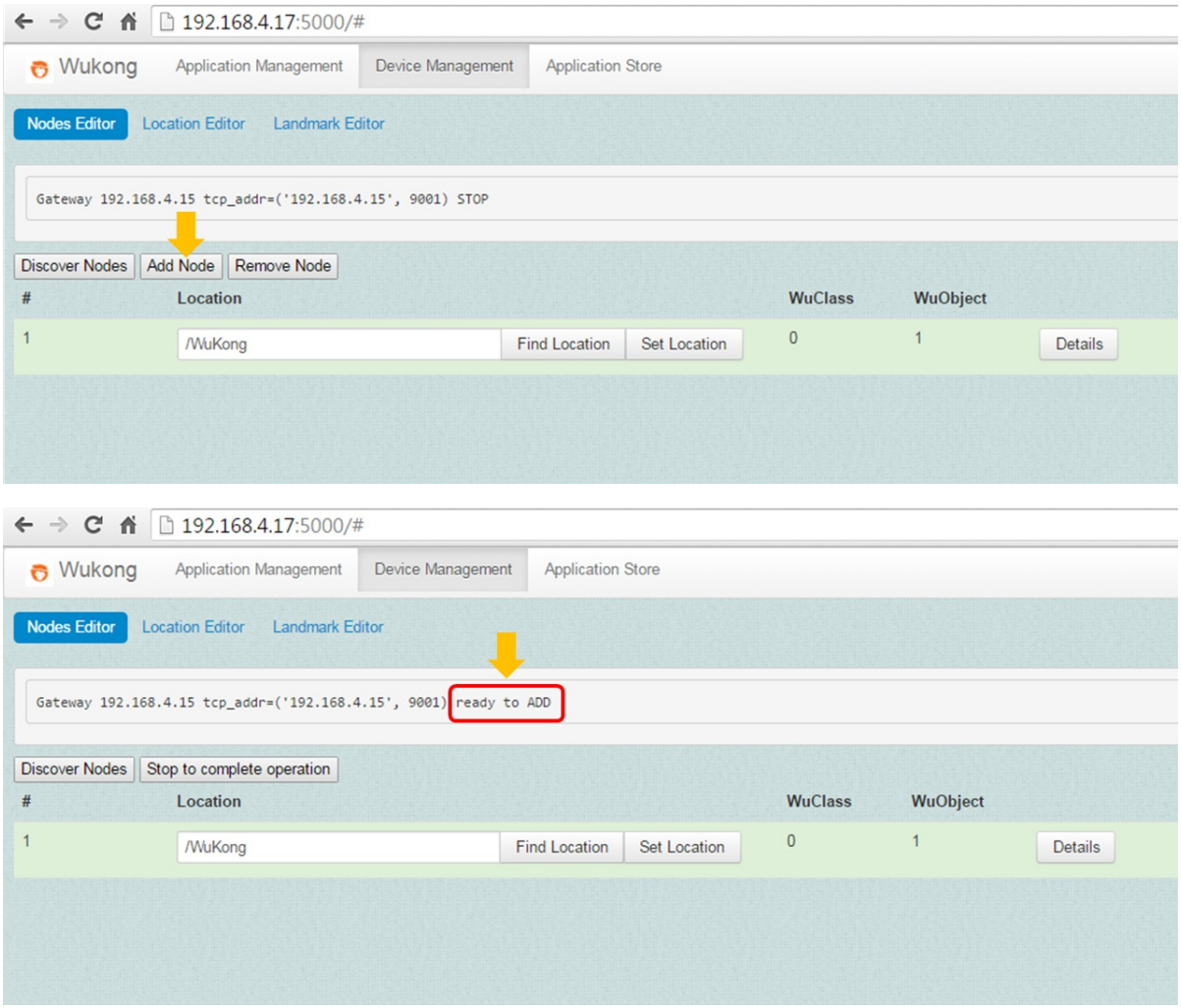
1. Use the Chrome browser to open FBP editor: http://localhost:5000



2. Enter the Device Management page to check the initial state



3. Click the **"Add Nodes"** button to start including the IoT board for Master management. You should see the message stating Master is "ready to ADD".



4. Open another terminal and use SSH command to connect to the IoT board.

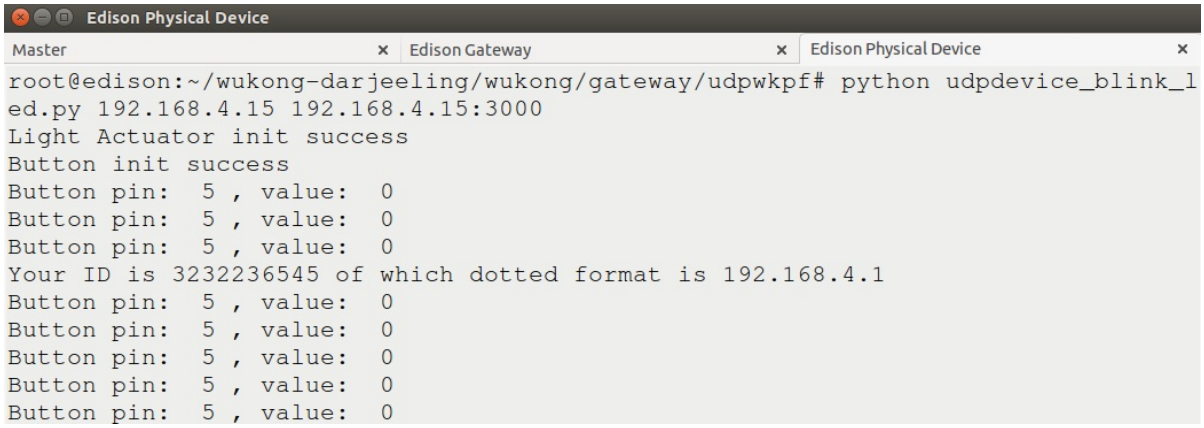
```
ssh root@<IP address of IoT board>
```

5. After SSH is established, go to the folder of the Python WuClass on the IoT board.

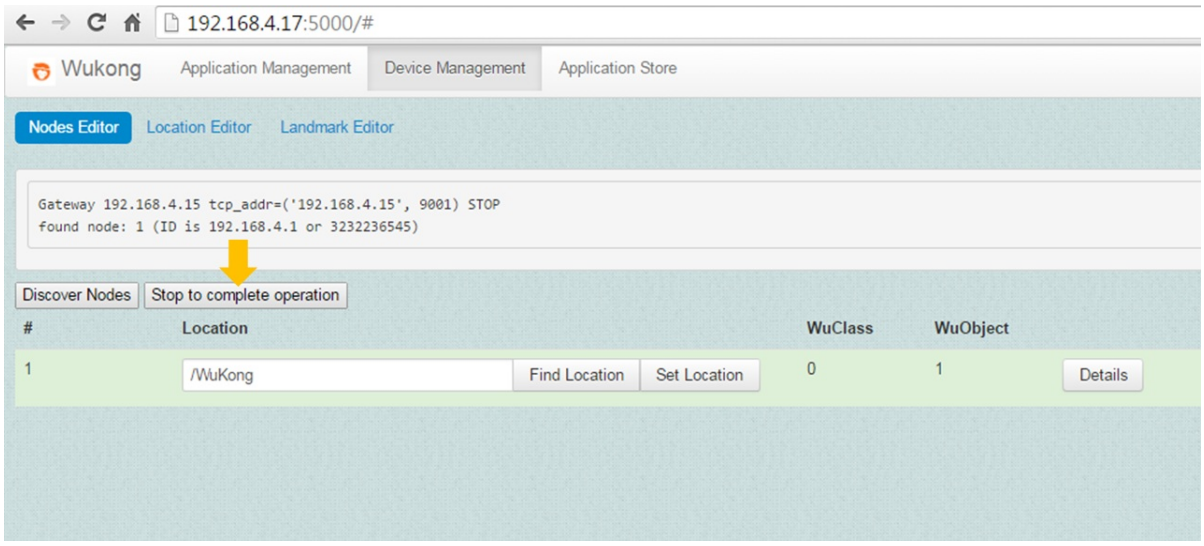
```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/udpwkpf/
```

6. Run the device program udpdevice_blink_led.py. This Python program will start the device to receive a node ID from the WuKong gateway, and then the two WuObjects, Button and Light Actuator, residing on the device.

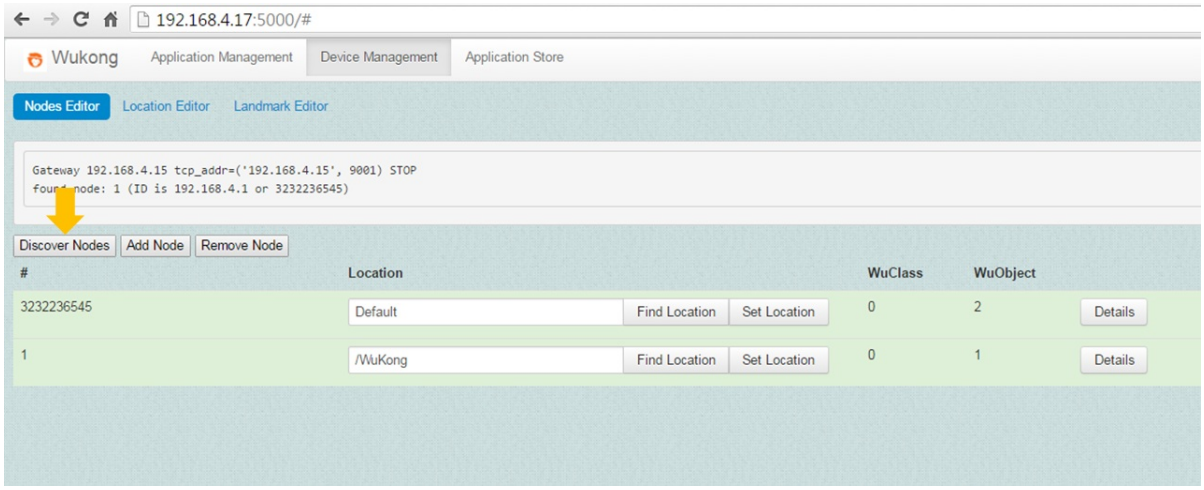
```
python udpdevice_blink_led.py <IP address of Gateway> \
<IP address of IoT board>:<arbitrary port number>
```



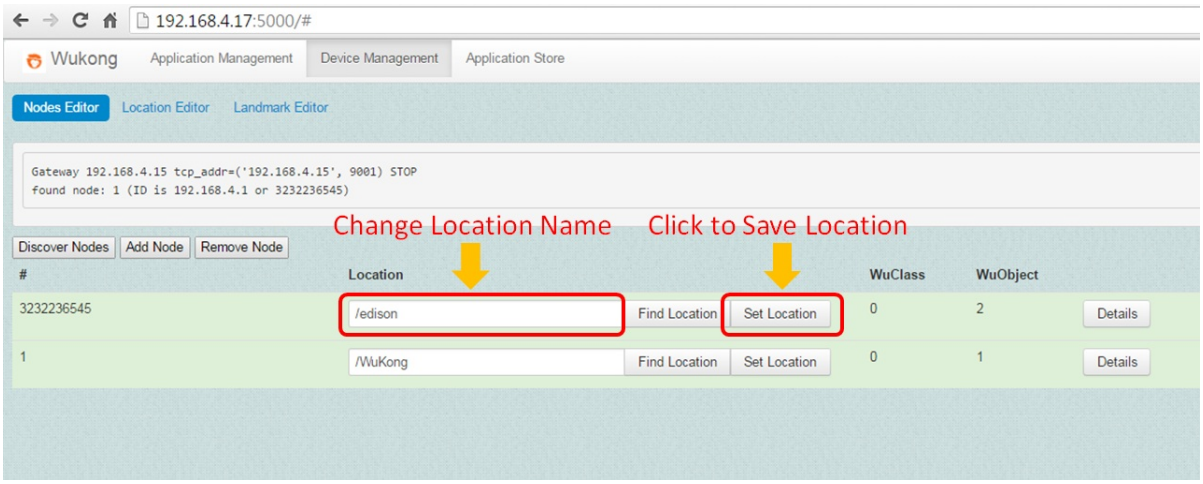
7. Once Master has found the board (displaying the message as shown), stop the inclusion process.



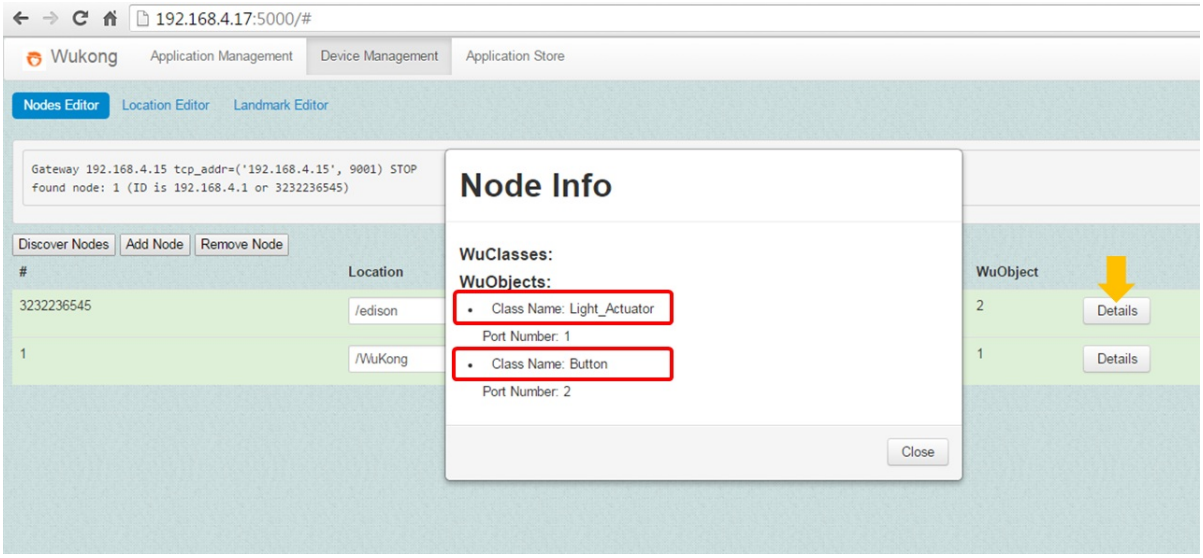
8. Check the device management list again by clicking the Discover Nodes button.



9. Modify the device location and press the Set Location button to save it.



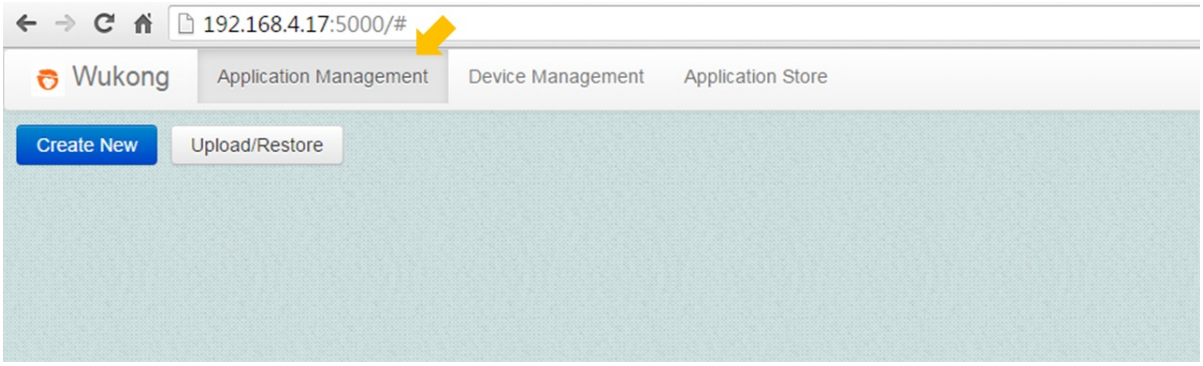
10. Press the Details button to check the sensor profile of the Python device.



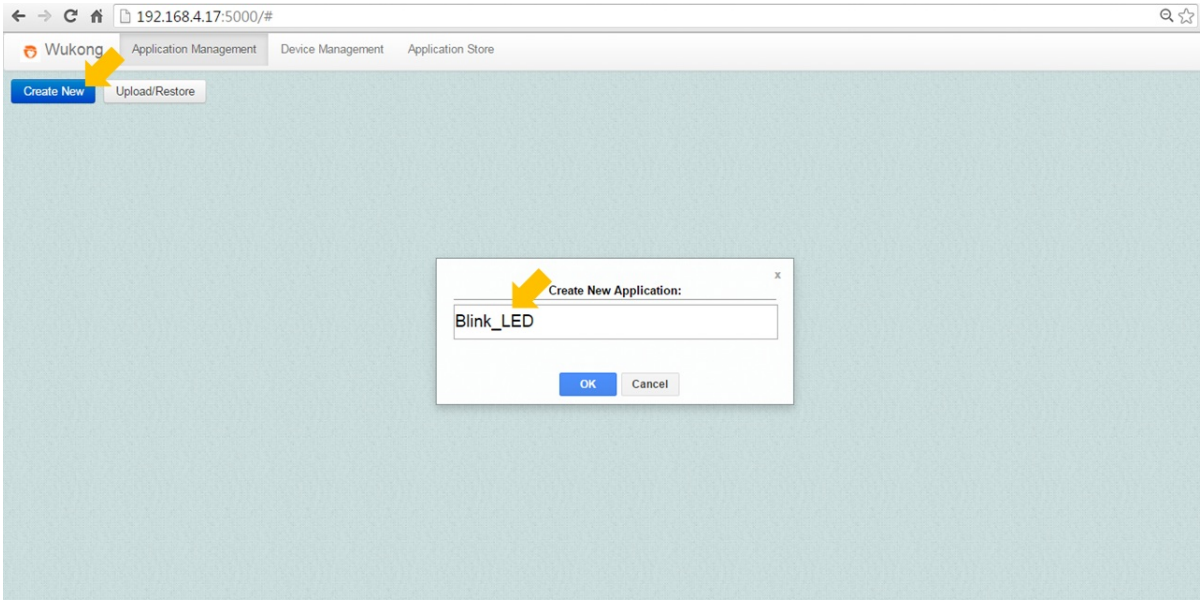
4.2.3 Build and Deploy an FBP Application

We will now define a LED control application using the FBP editor and deploy this application to the IoT board.

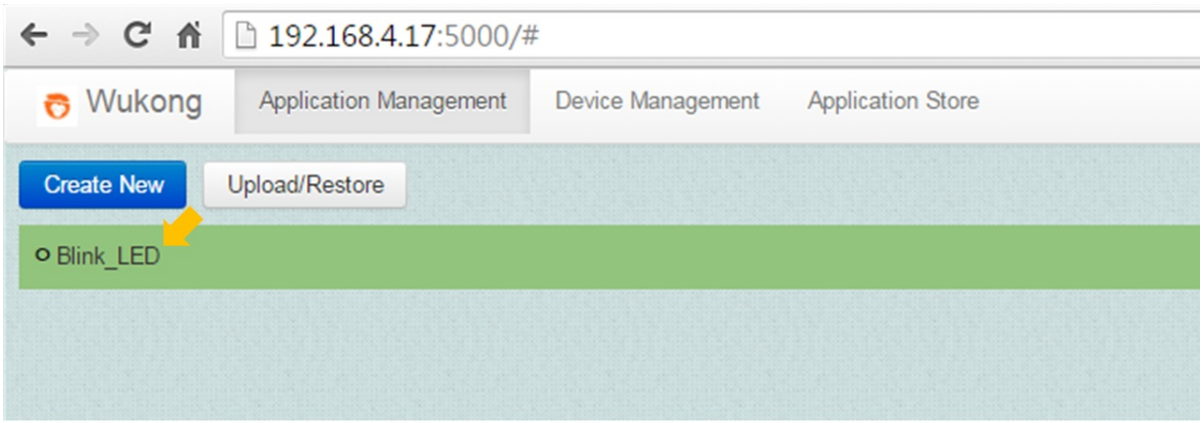
1. Click and enter the **Application Management** page.



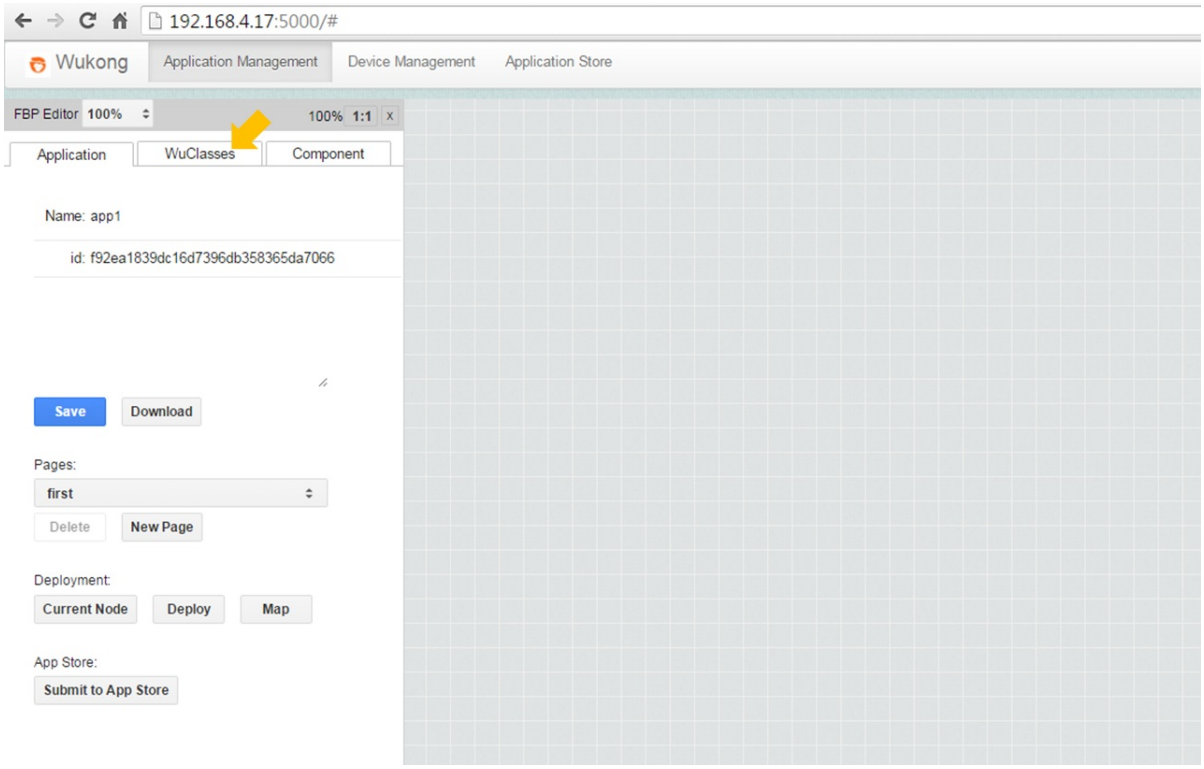
2. Click the **Create New** button and then enter the name for application.



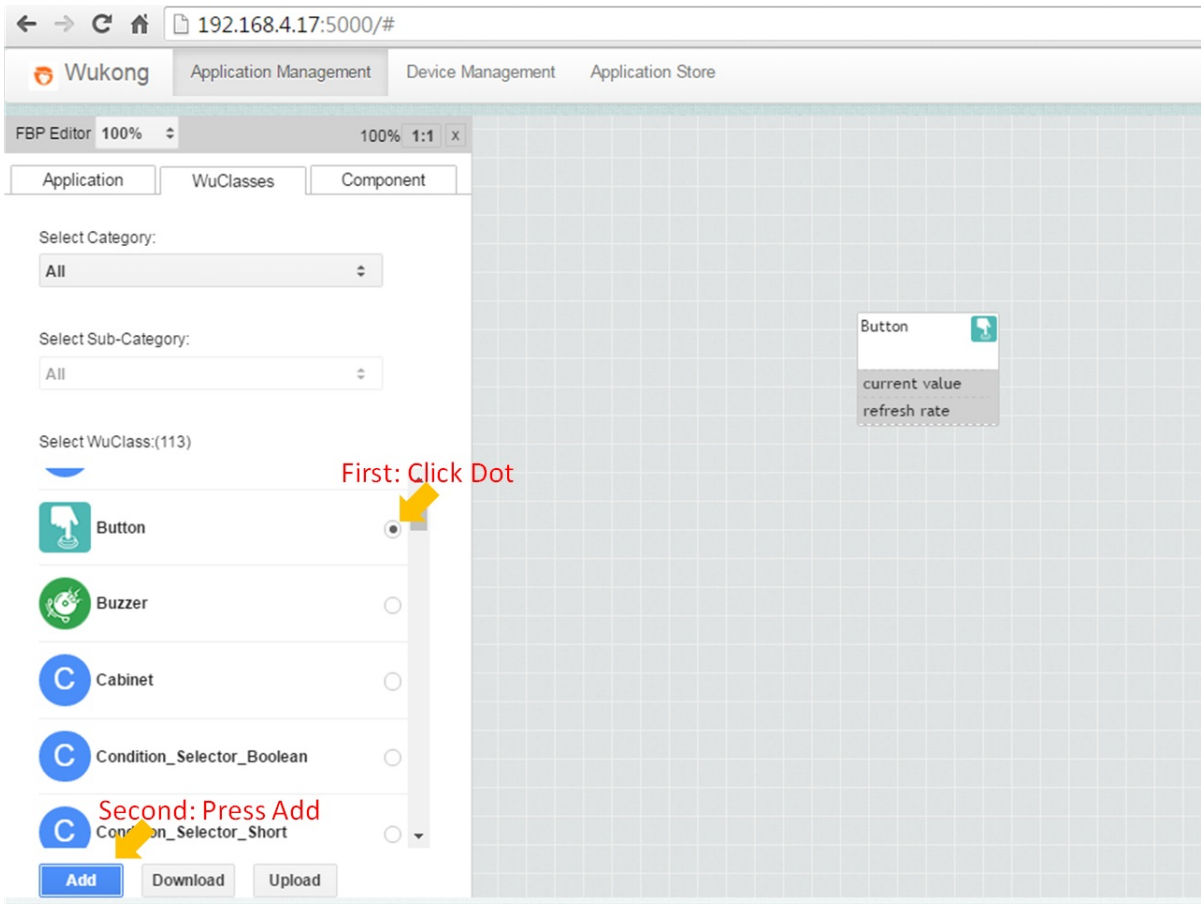
3. Click on the FBP name to enter the FBP composition canvas.



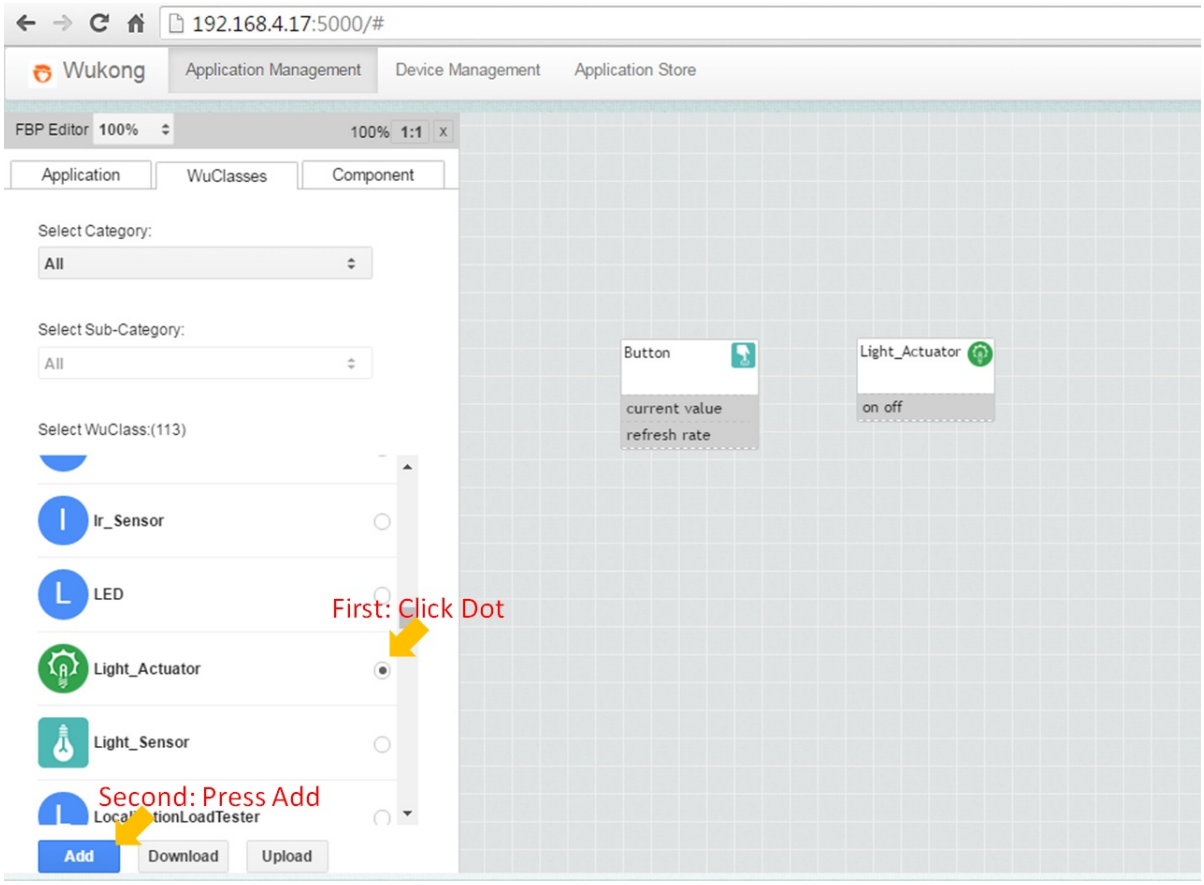
4. Click the **WuClasses** tab.



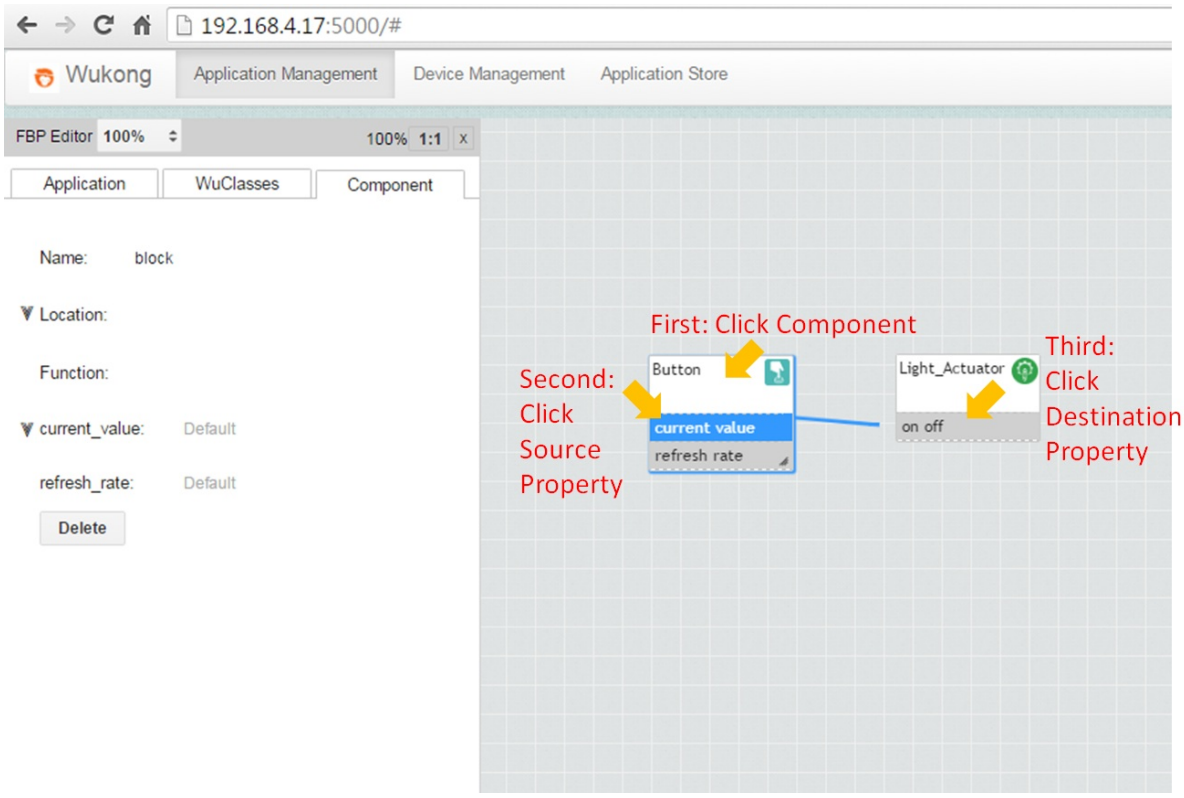
5. Scroll through the WuClass list to find the Button component. Select it and click the **Add** button. A Button component will appear on the composition canvas.



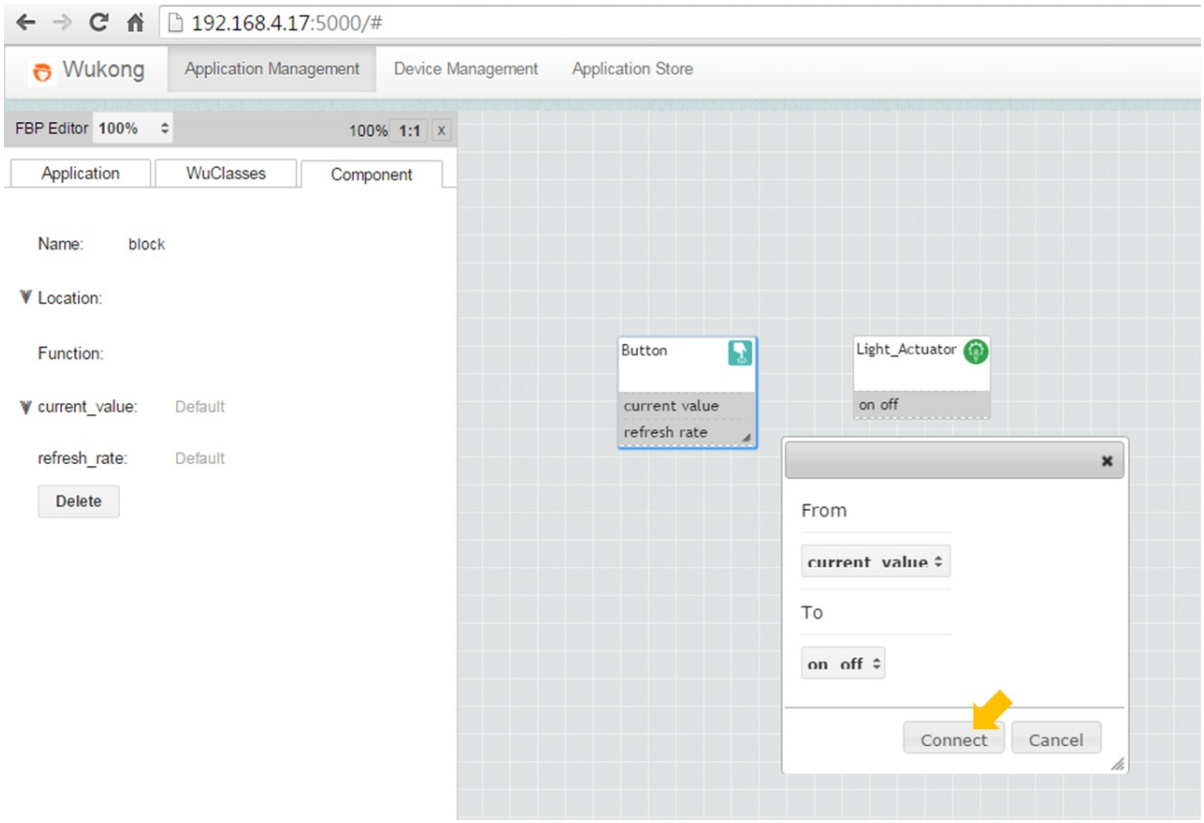
6. Repeat the last step to add the Light_Actuator component. Please be noted that the new added component will overlap the previous one. You have to draw them apart.



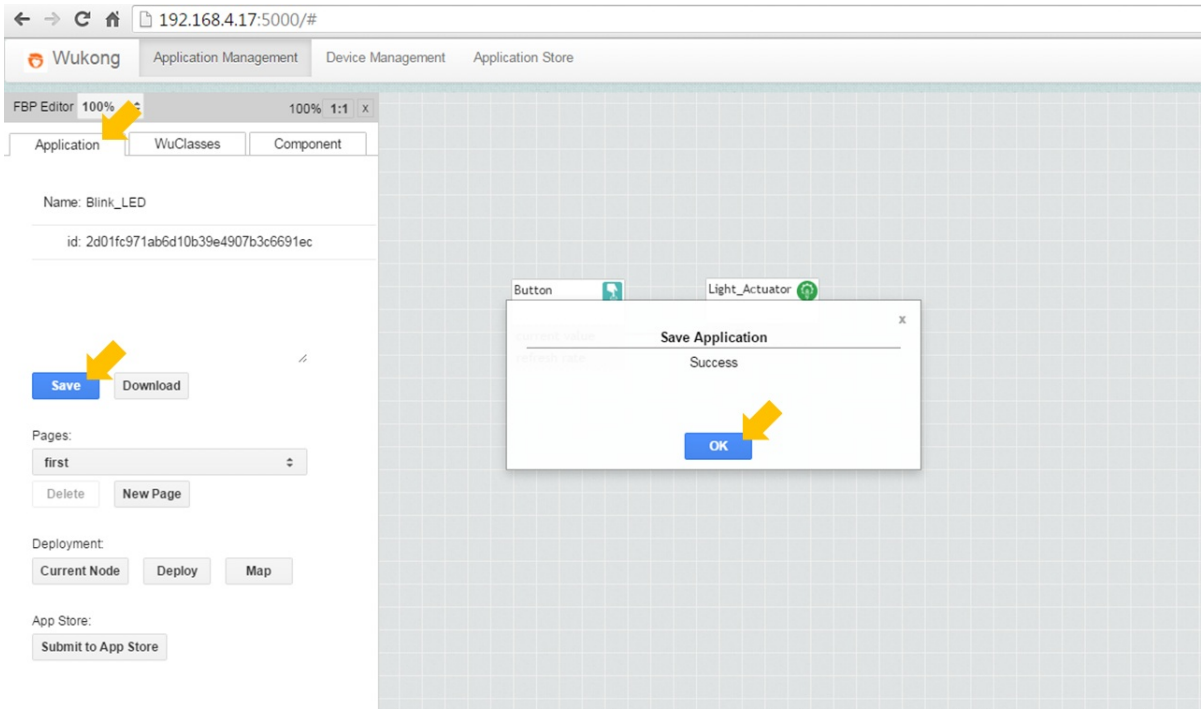
7. To create a link between two components, first click on the source component. After the source component has been highlighted, select the source property. Then move the cursor and click the property on the destination component.



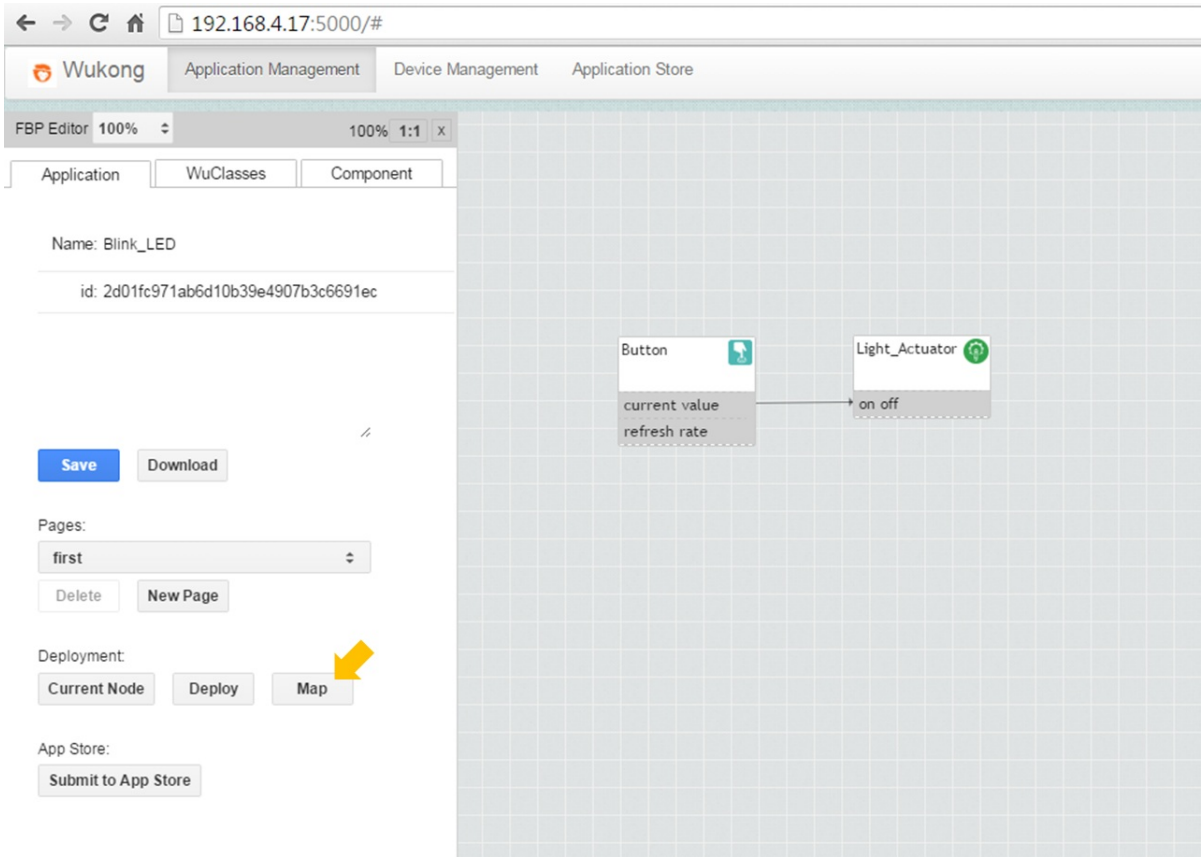
8. Check whether the source and destination of the data link is correct. If not, change the property in the dropout menu.



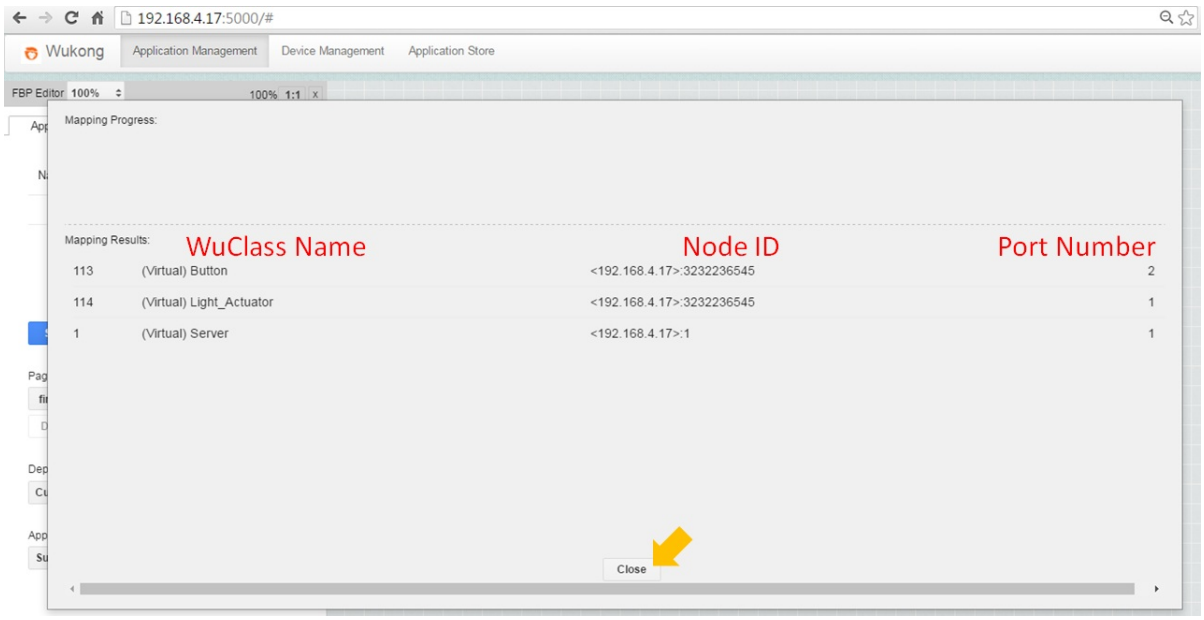
9. Press the **Application** tab and click the **Save** button. The FBP has now been created.



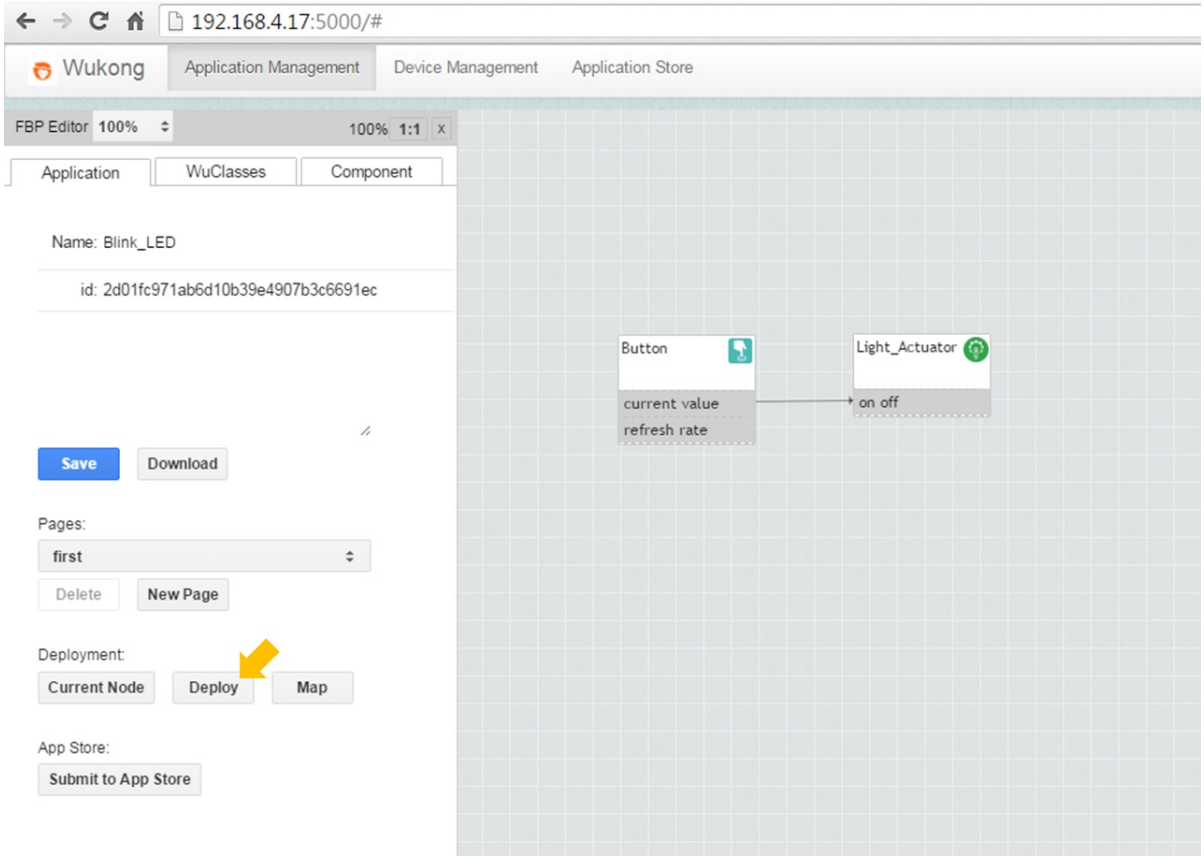
10. We will now start the deployment process. First click the **Map** button to start the Master FBP mapping selection process.



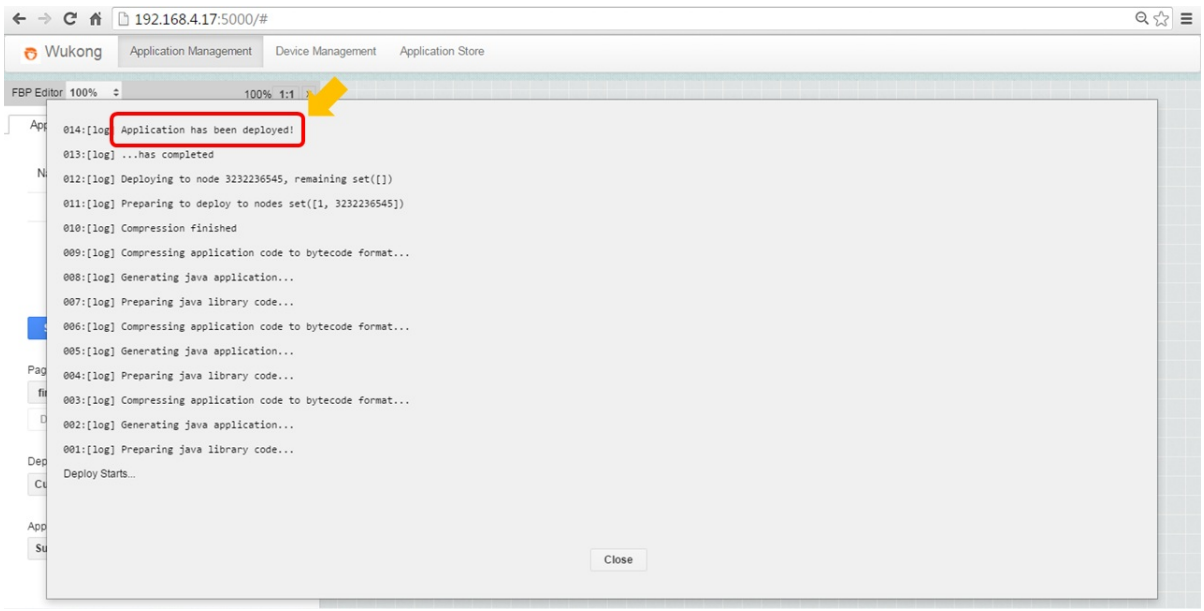
11. After Master has made its selection, the mapping result will be displayed as below. If the mapping is successful, each FBP component should have a device ID and a port number.



12. Click the **Deploy** button to start the FBP execution.

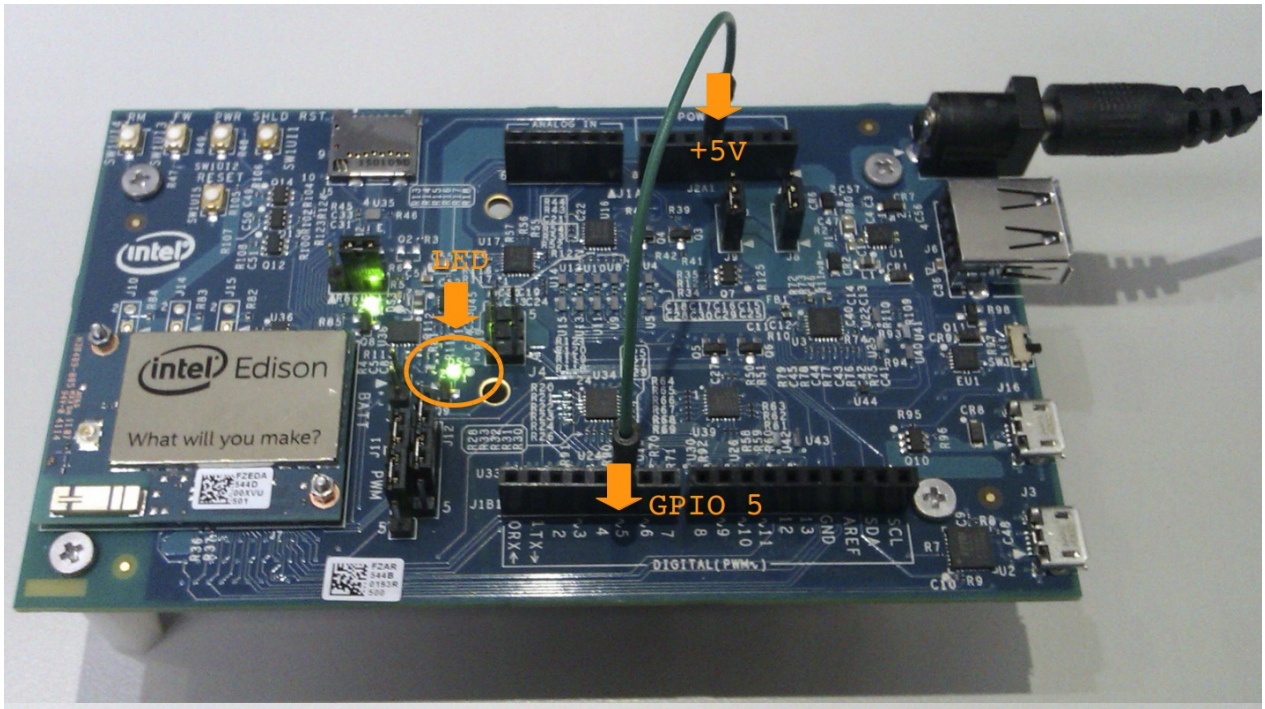


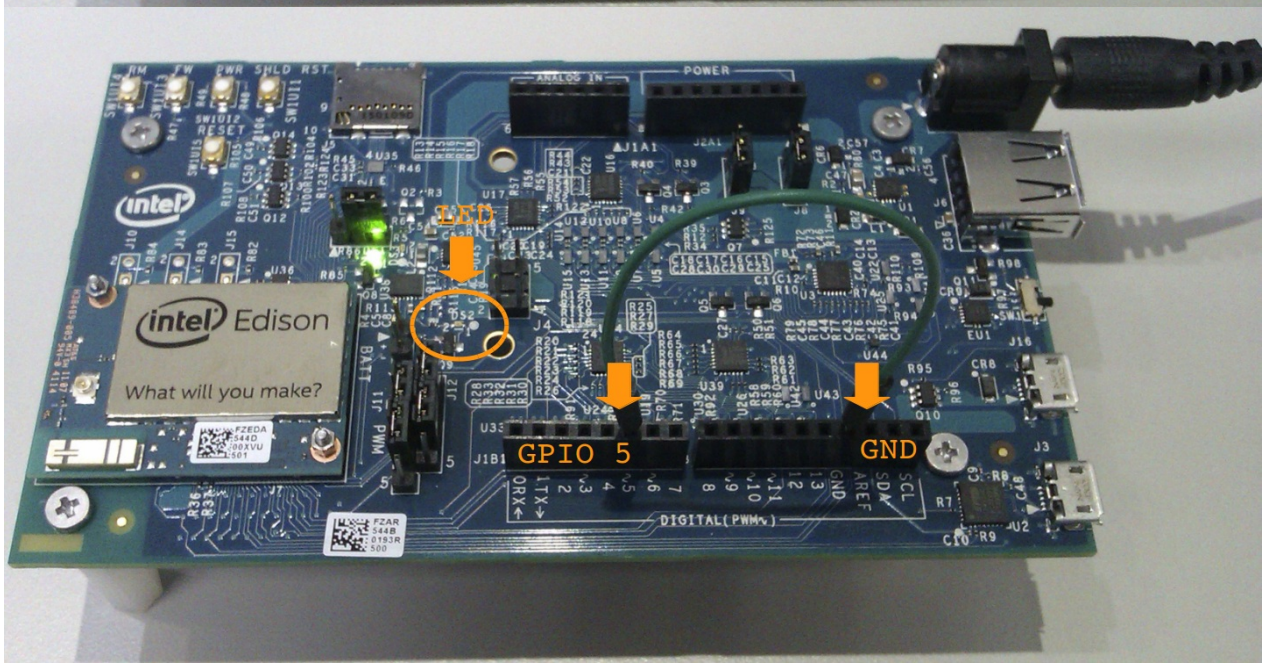
13. Master will display some messages and show if the deployment is successful.


192.168.4.17:5000/#
Wukong
Application Management
Device Management
Application Store
FBP Editor
100%
014:[log] Application has been deployed!
013:[log] ...has completed
012:[log] Deploying to node 3232236545, remaining set([])
011:[log] Preparing to deploy to nodes set([1, 3232236545])
010:[log] Compression finished
009:[log] Compressing application code to bytecode format...
008:[log] Generating java application...
007:[log] Preparing java library code...
006:[log] Compressing application code to bytecode format...
005:[log] Generating java application...
004:[log] Preparing java library code...
003:[log] Compressing application code to bytecode format...
002:[log] Generating java application...
001:[log] Preparing java library code...
Deploy Starts...
Close

4.2.4 Running the Application

After the FBP is deployed, you should find a male-to-male Grove cable and plug one end point of the cable to DIGITAL pin5 of the Edison board, and toggle the other end between power 5V and GND. This action is used to emulate pressing a physical button. You should see the LED on Edison blinking as shown below. In addition, since both gatway and device are running on Edison, the deployed application will continue to run even when the computer is not running.

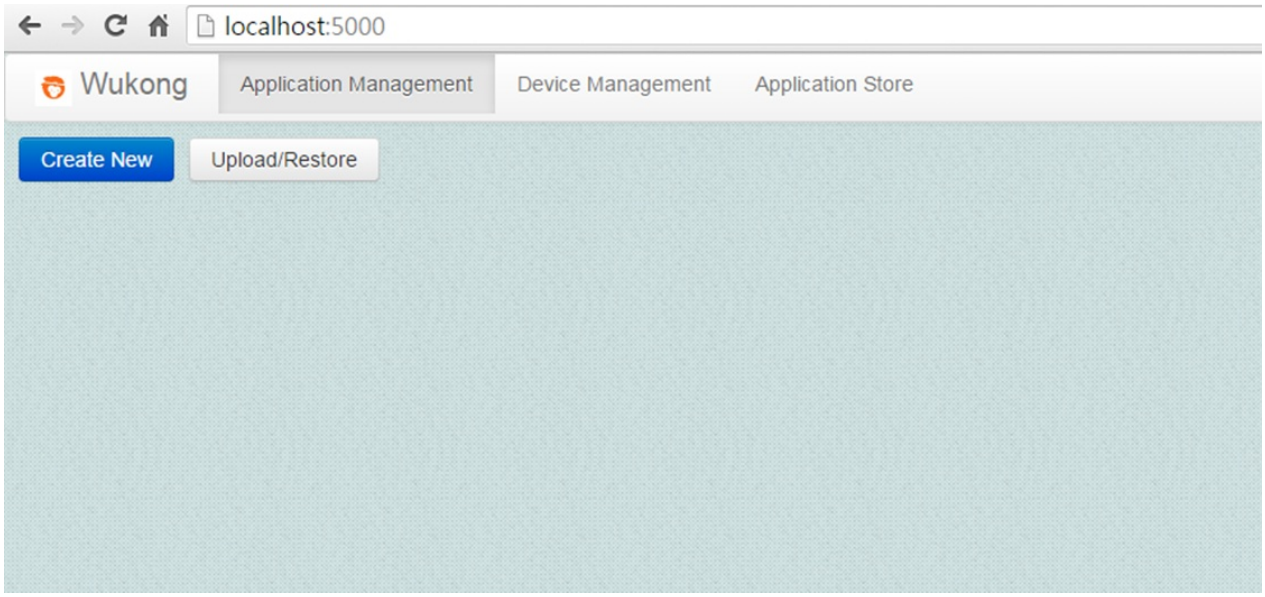




Chapter 5: Using the WuKong Management Interface

We now show how to use the WuKong management in the following sections.

1. Network Management: To present the network architecture of WuKong, and explain how to configure master and gateway.
2. Device Management: To add devices to a WuKong system, configure their locations, and discover their availability.
3. Application Management: To create a flow based program, map the components to sensor profiles, and deploy the application on each device.
4. Application Store: To create the collection of your FBP applications and make them available for sharing.

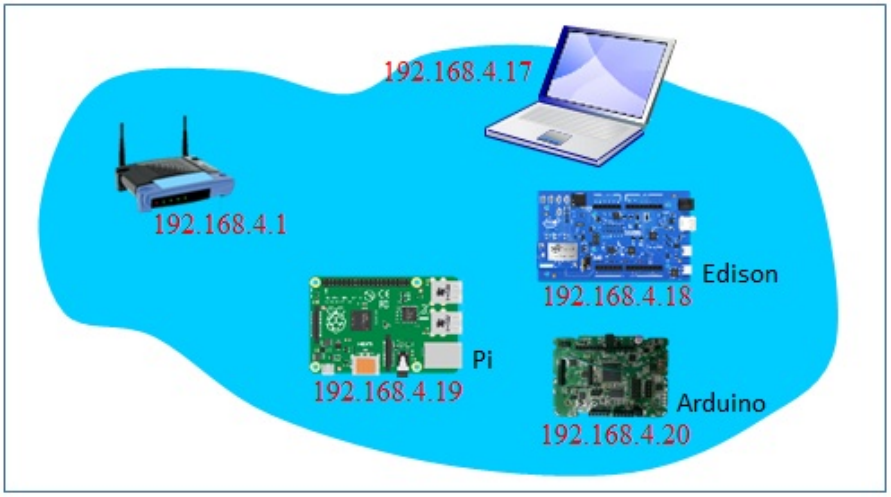


5.1 Network Management

Before a WuKong application can be deployed, we must set up the network connections among Master, gateways and devices so that they can communicate with each other correctly. As we have discussed in Section 1.1, a WuKong system has three types of system components: Master, gateway, and node (physical or virtual device). Each component may be run on a specific hardware platform in a **local area network** or on **different subnets**. In this section, we first present system components running in the same local area network. Later, we will include system components running on different subnets.

All Components in the Same Local Area Network

This network setting is shown in the following figure. In this setting, there are laptop, IoT boards with embedded Linux, and IoT board without OS support (Arduino). The laptop is used to run all the system components as discussed in Section 4.1. The IoT board with embedded Linux can be used to run both gateway and devices as shown in Section 4.2. The IoT board without OS support can only be used to run FBP components.



The following steps are used to start master and gateway on the same local area network. They are the same as in Section 4.2.1.

1. Make sure the laptop is connected to the router in the local area network. If the network environment has been changed, please follow the instructions in the section 4.1.5 to restore default settings.
2. Make sure the infuser has been compiled. If not, please do the following:

```
cd <path_of_source_code>/wukong-darjeeling/src/infuser/
gradle
```

3. Make sure the config file for Master has been created. If not, please do the following:

```
cd <path_of_source_code>/wukong-darjeeling/wukong/config/
cp master.cfg.dist master.cfg
```

4. Start Master.

```
cd <path_of_source_code>/wukong-darjeeling/wukong/master/
python master_server.py
```

5. After starting Master, open a terminal to start the gateway. You can either run the gateway on an IoT board or on the laptop. Notethat the order of starting Master and gateway must be followed on the first run.

6. Make sure the config file for gateway has been created. If not, please do the following:

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
cp gtwconfig.py.dist gtwconfig.py

# Note: For running the gateway program on a IoT board
# the source code has to be downloaded first with next line
# git clone http://github.com/wukong-m2m/wukong-darjeeling
```

7. Check the network interface of the platform running the gateway.

8. Configure gtwconfig.py according to the gateway network information.

```
vim gtwconfig.py

# Change MASTER_IP to the IP address of the master program.
# Change TRANSPORT_INTERFACE_ADDR to network interface of the gateway program
```

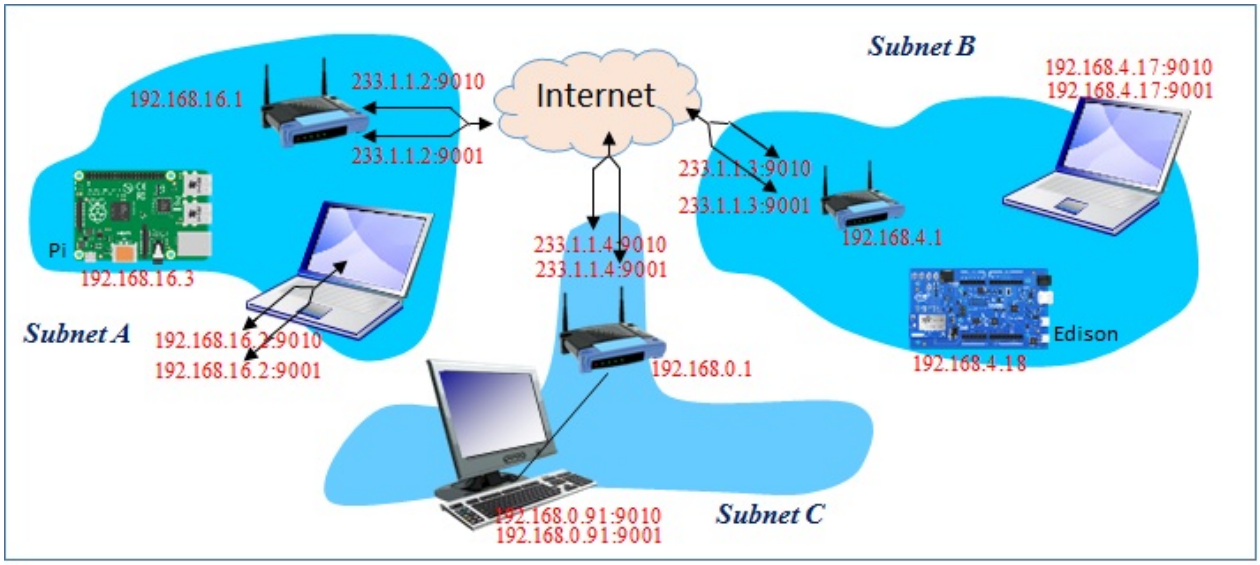
9. Start the gateway.

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
python start_gateway.py
```

10. Use a Chrome browser and connect to http://<*MASTER_IP*>:5000.

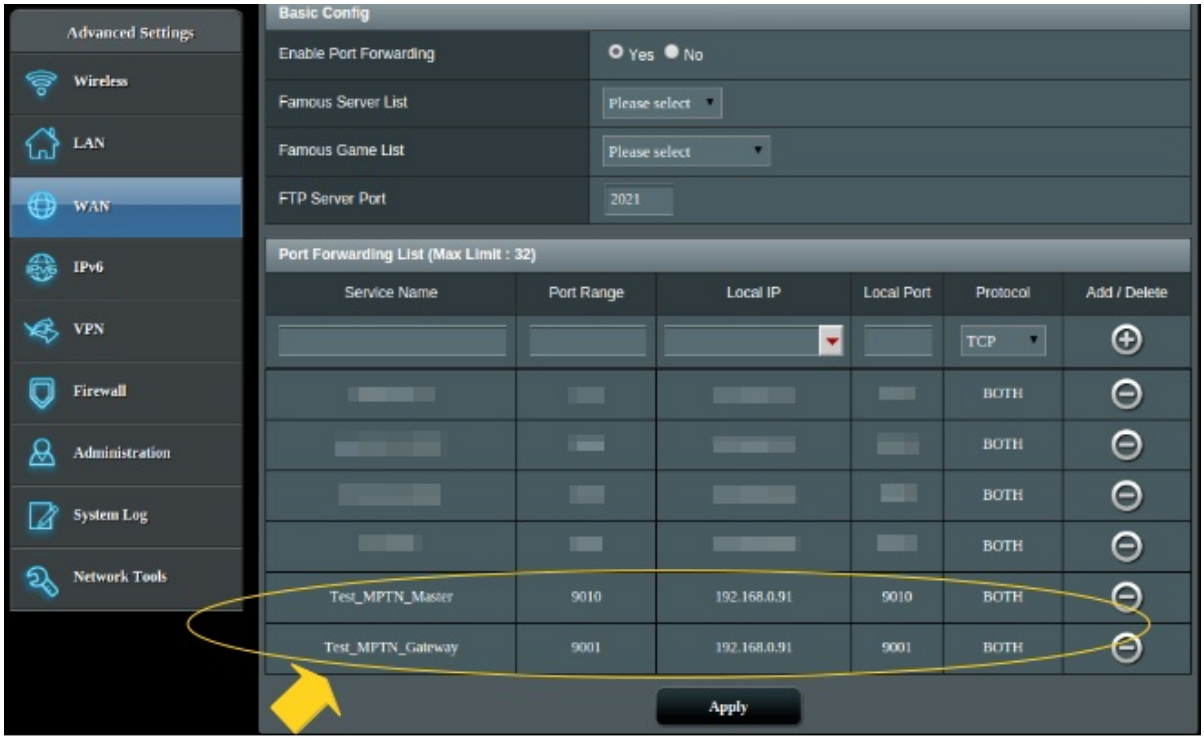
On Different Subnets

An example network setting is shown in the following figure. In this figure, there are three subnets. Subnet A has a Raspberry Pi with a local IP of 192.168.16.3 and Subnet B has an Intel Edison with a local IP 192.168.4.18. To build a system in this environment, we need to run a WUKong gateway on each subnet. In addition, since we don't handle NAT punch-through between different subnets, we must run Master on a third subnet. In this example, Master is run on Subnet C.

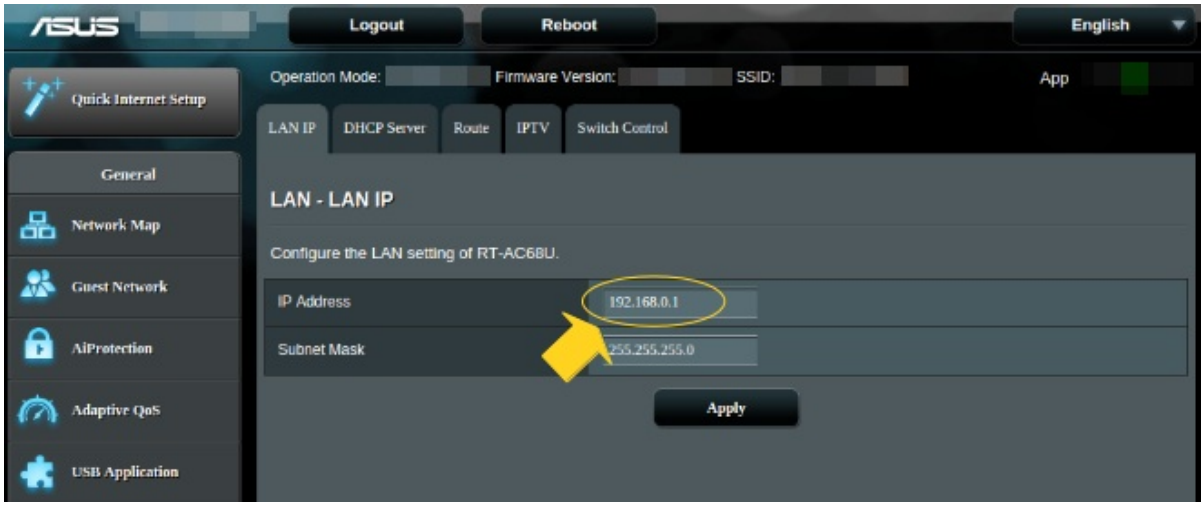


The following steps are used to start Master and gateways.

1. Add port forwarding at 9010 and 9001 of **each** subnet router. Port 9010 is used by Master to receive packet from a gateway. Port 9001 is used by the gateway to support RPC requests from Master. The following screenshot shows the interface of ASUS RT-AC68 WiFi router. Please refer to your router manual to set port forwarding.



2. Configure **each** router to use a different network prefix. The reason is that if two subnets use the same network prefix, their messages may have conflicts when received by Master. Therefore, in our example, the network prefix of each subnet is different. Please refer to your router manual to change the network prefix.



3. For the computer on Subnet C, follow steps 1 to 4 of the instructions for devices on the same Local Area Network to start Master.

4. After starting Master, open a terminal to remotely access to Raspberry Pi on Subnet A.

5. Make sure the source code has been downloaded on the laptop or Raspberry Pi on Subnet A. If not, please use the following command to download the source code:

```
git clone http://github.com/wukong-m2m/wukong-darjeeling
```

6. Make sure the config file for gateway has been created. If not, please do the following commands:

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
cp gtwconfig.py.dist gtwconfig.py
```

7. Find the public IP or WAN IP of each router. In the figure, the public IP of each router are 233.1.1.2 (subnet A), 233.1.1.3 (subnet B) and 233.1.1.4 (subbet C).

8. Configure gtwconfig.py.

```
vim gtwconfig.py

# Change MASTER_IP to the public IP of WiFi router on the subnet C.
# In this example, the MASTER_IP is 233.1.1.4
# Change TRANSPORT_INTERFACE_ADDR to network interface of the laptop or Pi
```

9. Run gateway on the laptop or Raspberry Pi on Subnet A

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
python start_gateway.py
```

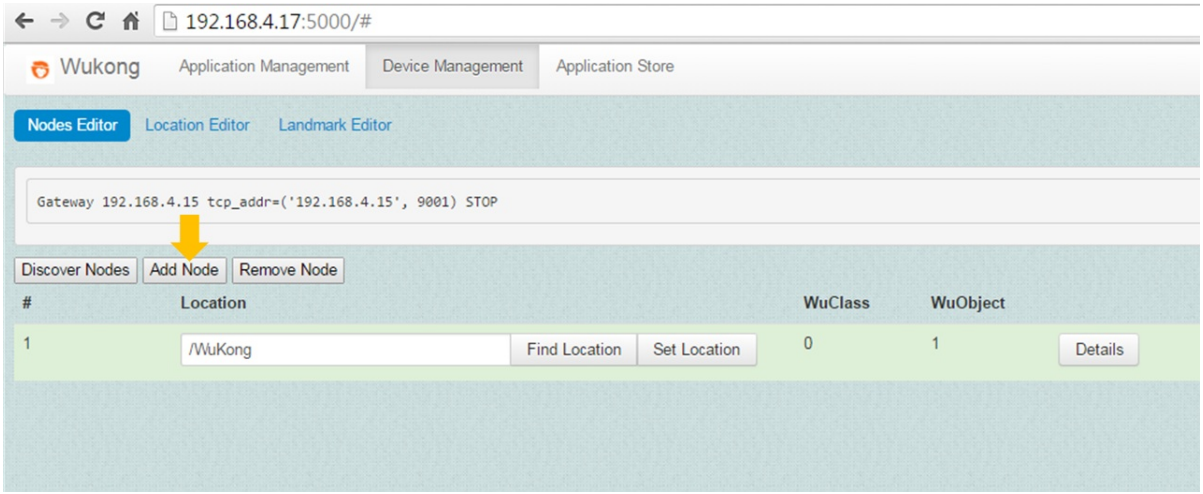
10. Repeat Steps 4 to 9 on the laptop or Edison on Subnet B.

11. Open a Chrome browser on **subnet C** and enter http://<*MASTER_IP*>:5000. Note: This webpage cannot be seen on the subnet A or B unless you have set port forwarding for 5000 on subnet A or B.

5.2 Device Management

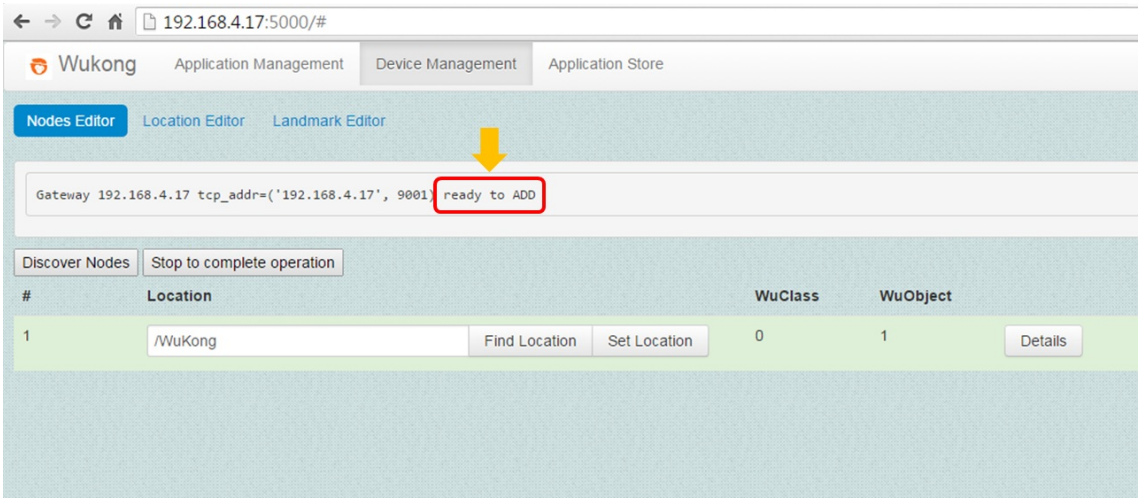
In this section, we will show how to use the WuKong Master to manage devices.

- Click on the **Device Management** top menu



- Adding devices to a WuKong system

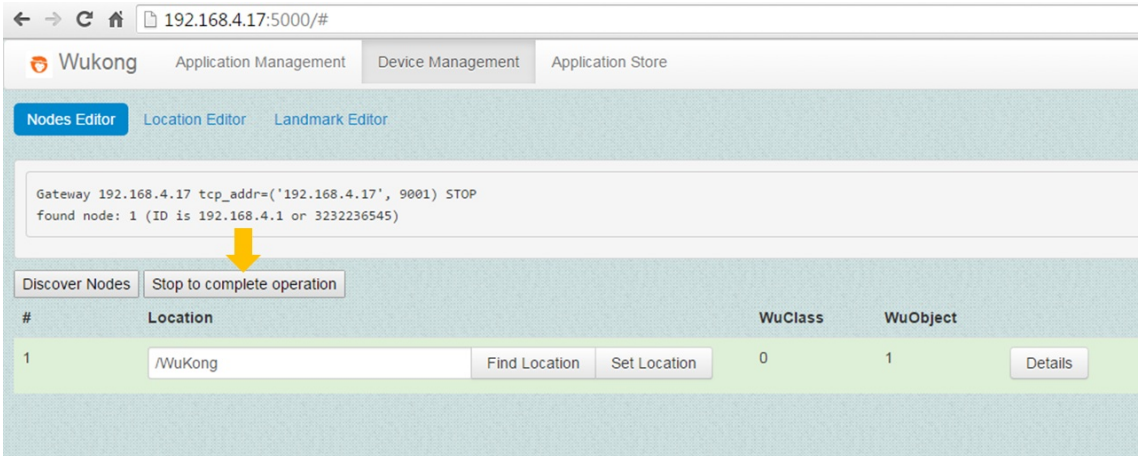
  1. To add a new device, click the **Add Node** button. This action will put Master in the **ADD** mode as the message shows "ready to ADD".



  2. Start the device program and let it enter the learning mode.

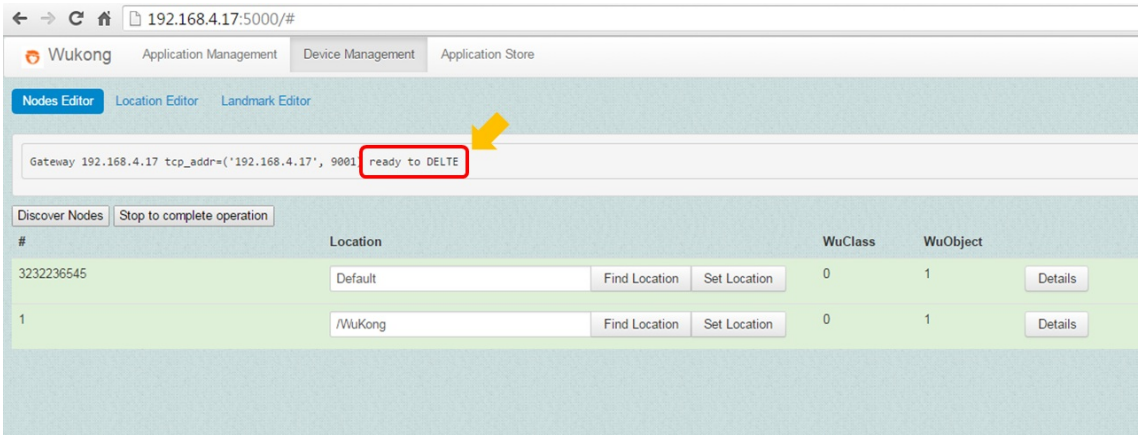
```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/udpwkpf/
python udpdevice_blink_led.py <gip> <dip>:<port>
#<gip> is IP address of gateway program
#<dip> is IP address of device program>
#<port> is an unique port number for this device program>

#Go to the folder of Python WuClass
#Execute Python device to be included to WuKong system
```

3. Press **Stop to complete operation** to let Master return to the **STOP** mode.



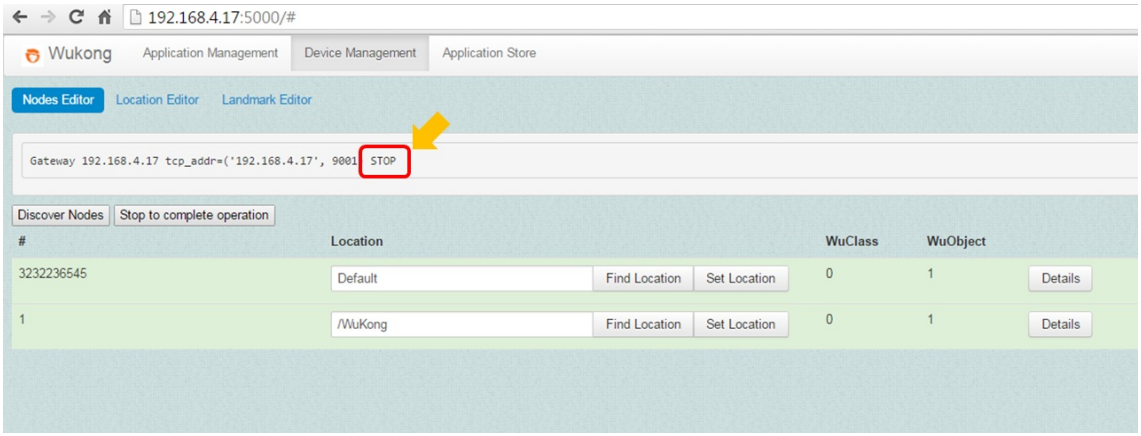
- Removing devices from a WuKong system

1. To remove a device, click the **Remove Node** button. This action will put the master program in the **DELETE** mode with a message showing "ready to DELETE"

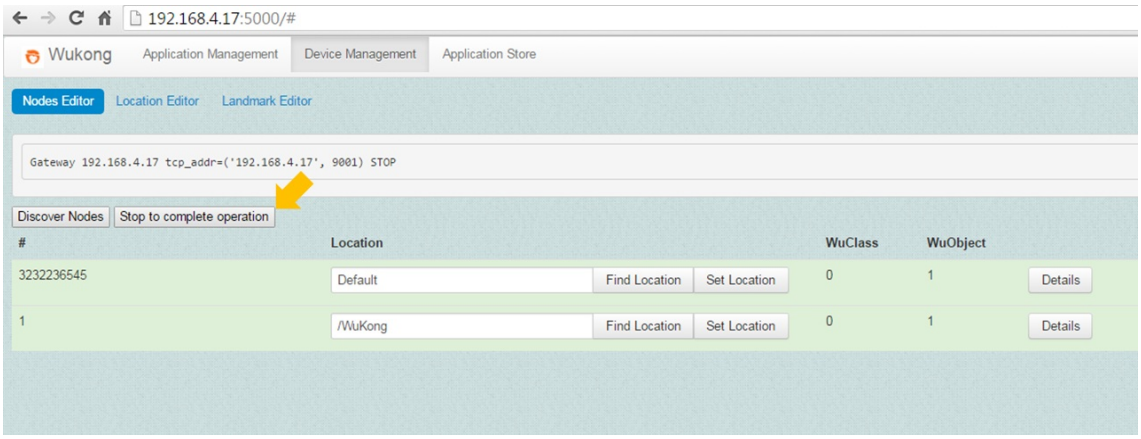


2. Restart the device program and let it to enter the learning mode.

```
ctrl + c
python udpdevice_blink_led.py <gip> <dip>:<port>
#<gip> is IP address of gateway program
#<dip> is IP address of device program>
#<port> is the identical port number as adding this device program>
ctrl + c

#Go to the terminal of device program which is going to be removed
#Restart Python device program to exclude it from WuKong system
#Wait for the message showing "STOP" to terminate the program
```

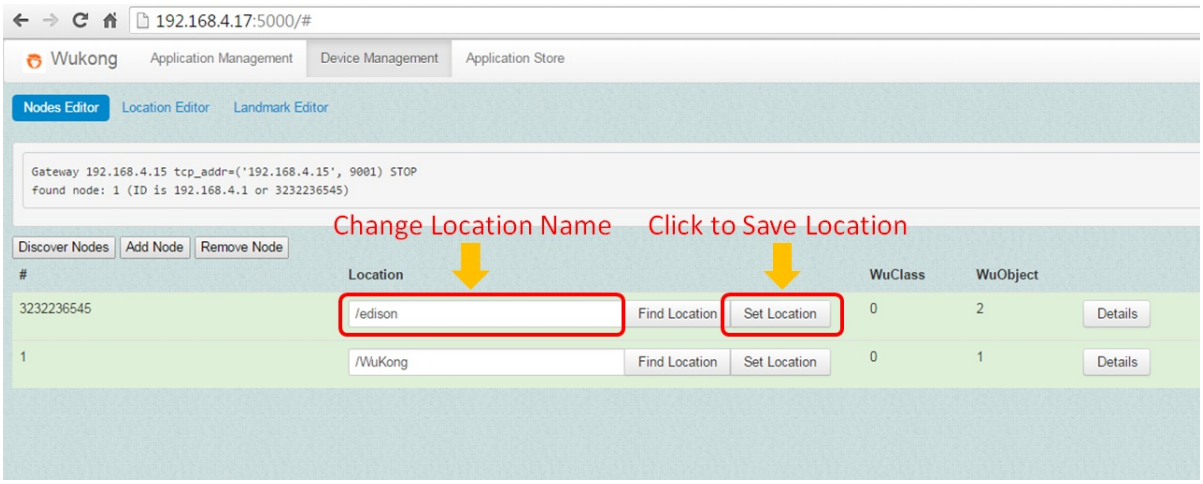


3. Click **Stop to complete operation** to let Master go back to the **STOP** mode.

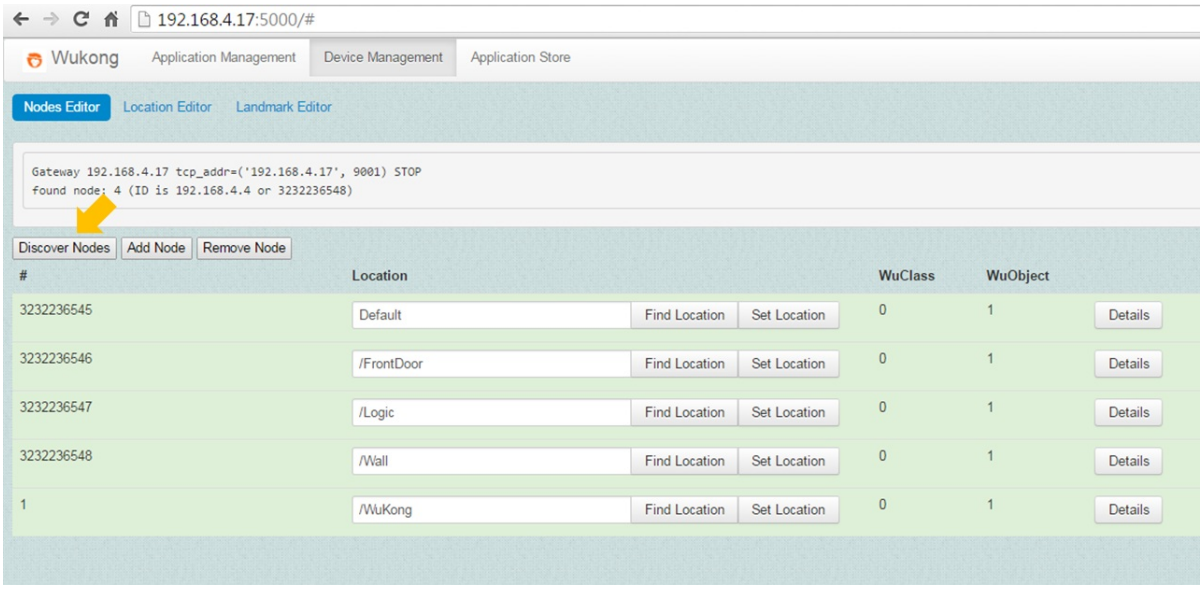


- How to set location for each device

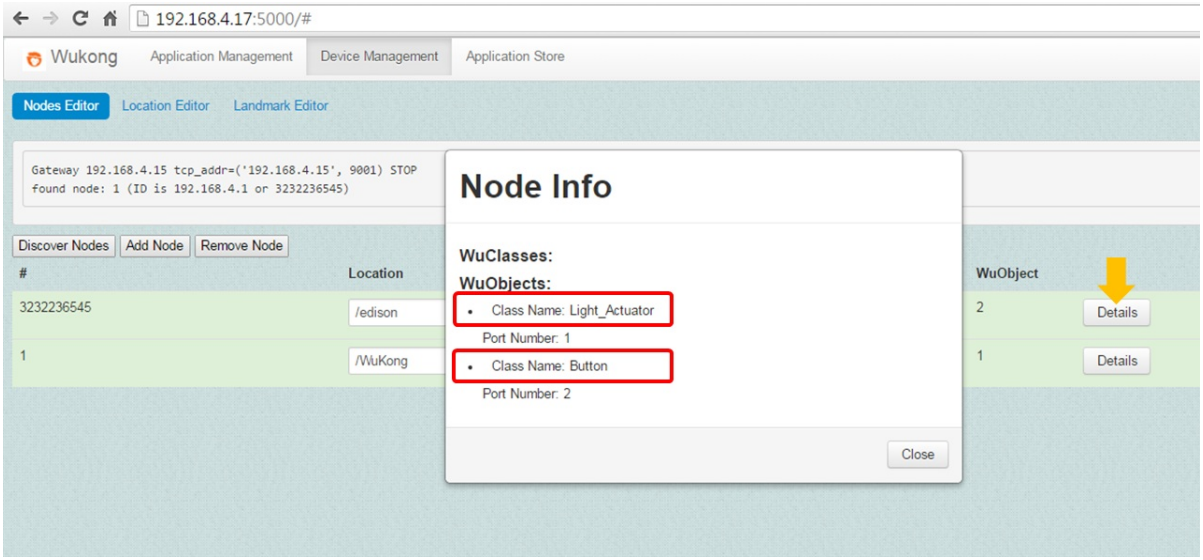
  Modify the device location and click the Set Location button to save it.



- The **Discover Nodes** button can be used to show the current devices known to Master.



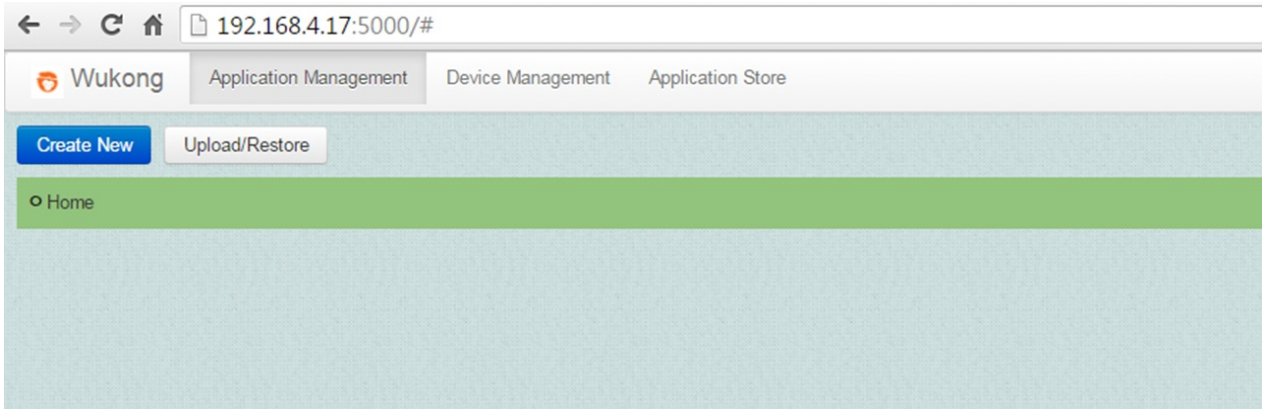
- The **Details** button can be used to check the sensor profile on each device.



5.3 Application Management

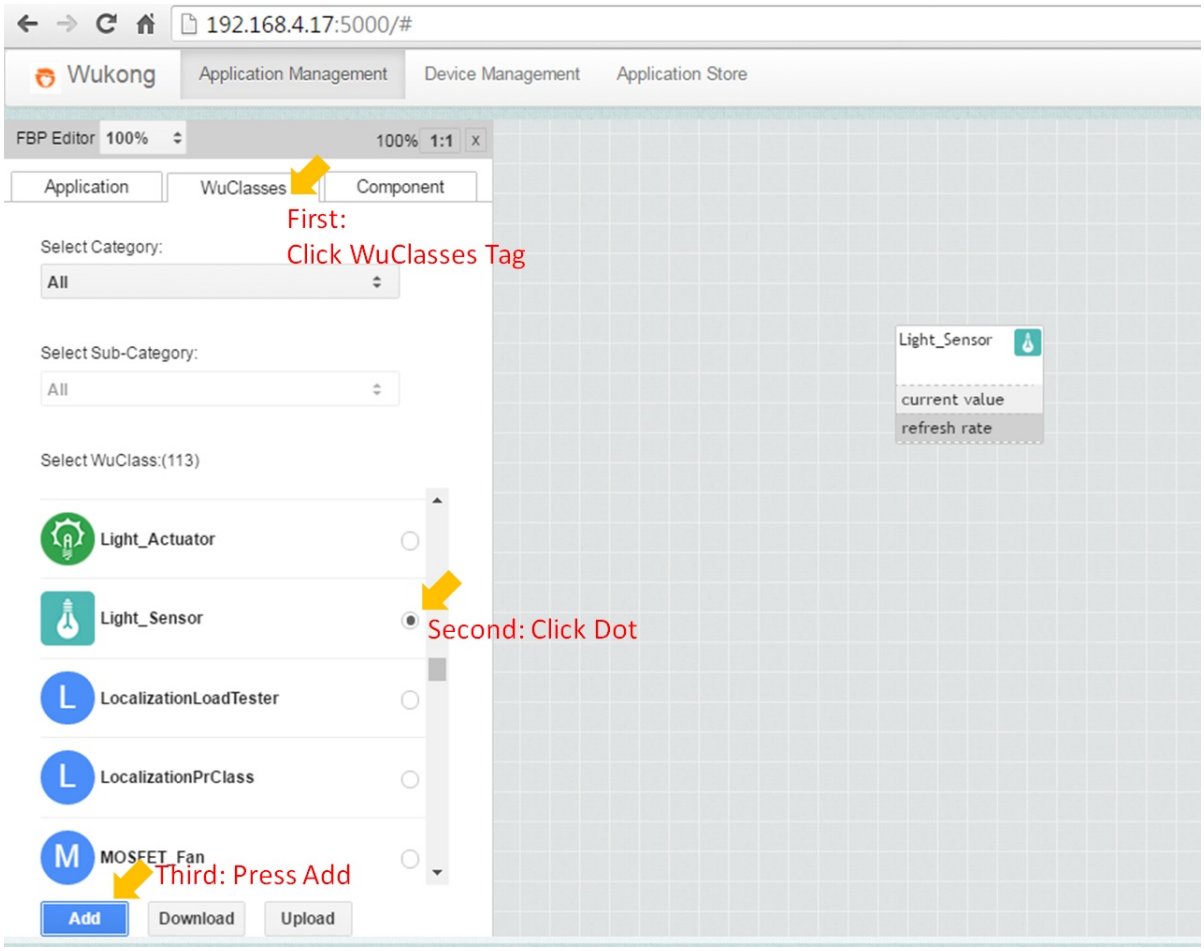
Createing an FBP

Click the **Application Management** top menu. It will show a list of existing applications. We can add a new application using the **Create New** button, or we can delete an application by clicking the  symbol on the right-hand side. After creating a new application, we can click on its name to edit the application.



- How to Add Components

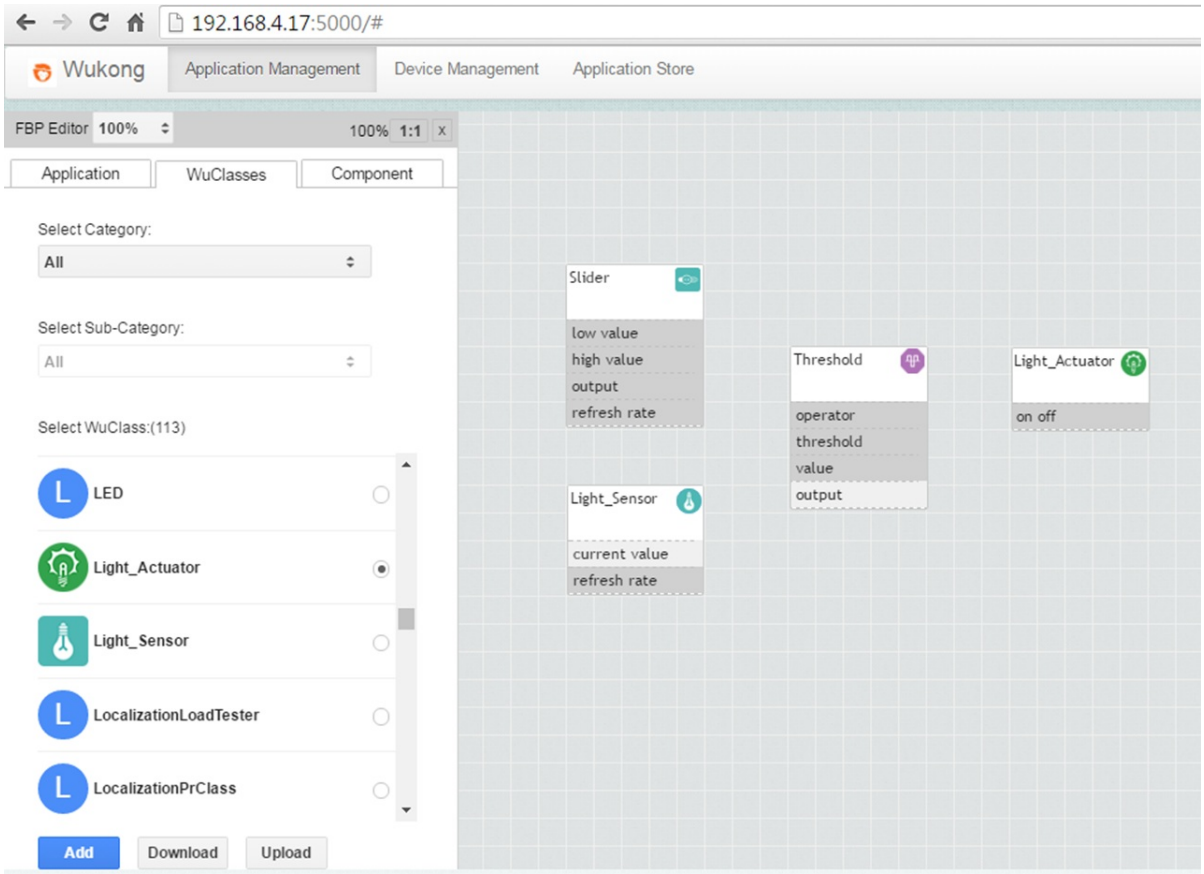
  To add components to an FBP, first click on the WuClasses tab at the left-hand side menu to list all available components in the WuKongStandardLibrary.xml. Next, scroll down the list to find the right component and press the **Add** button to generate a new instance of the component on the canvas. The figure below shows the canvas after adding a Light_Sensor component.



The Light_Sensor component has two properties. Input properties are displayed with gray color; in this case, a “refresh rate” property specifies how often the sensor needs to take a new measurement (in milliseconds). Output properties are displayed with white color; in this case, the current sensor value represents the measured light sensor data.

In the following example we will create a simple application to turn on a light if the light sensor’s value drops below a certain value. For this application we need 4 components:

1. A Light Sensor component: This will return a value in the range of 0 to 255.
2. A Threshold component: This will compare the light sensor’s value to the desired minimum light value
3. A Slider component: This will provide the Threshold component with it’s threshold value.
4. A Light Actuator: This represents a lamp.

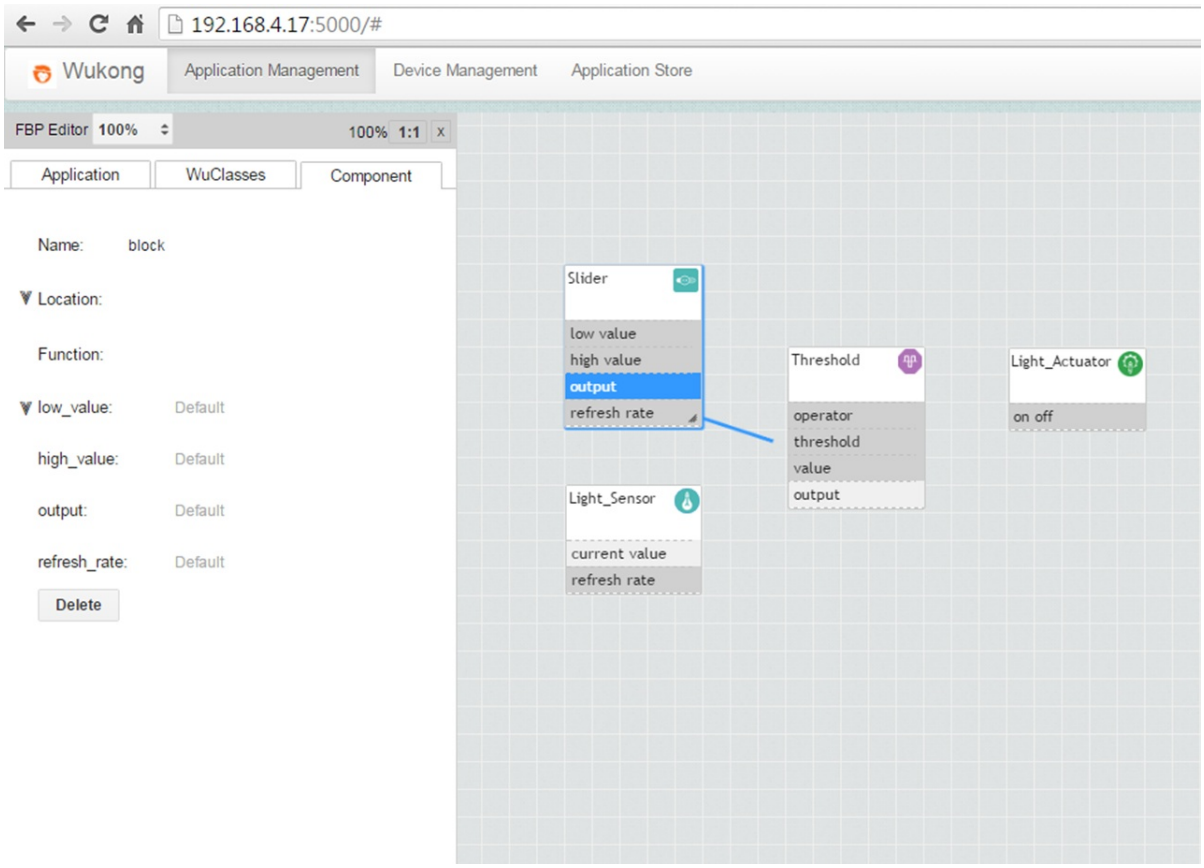


- How to Create Links

  After placing all required components on the canvas, the properties can be connected using the link.

  1. First, a component has to be selected by clicking on it.
  2. Next, click a source property, and a link will be generated from that property.
  3. When a link is generated, draw the cursor to the target property and click on it.
  4. Last, a pop-up screen will be displayed to confirm which properties of the source and target component are going to be linked.

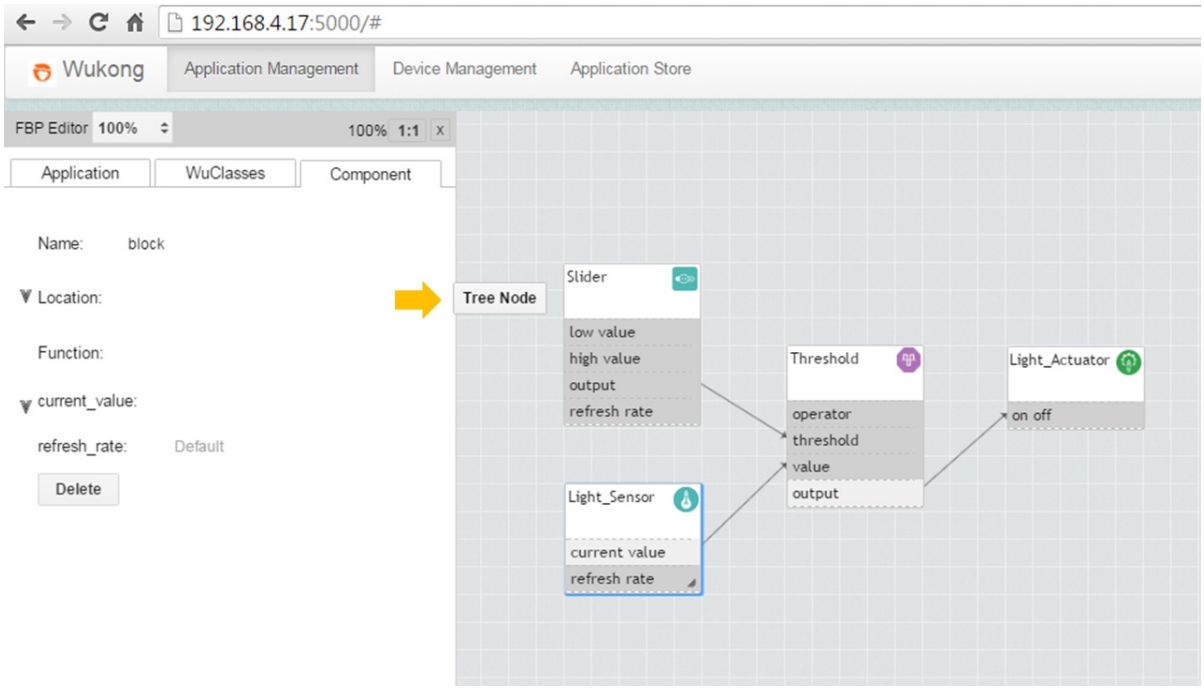
  The picture below shows the canvas after adding all the required components, and the process of creating the link between the Slider's output value property and the Threshold's threshold property.



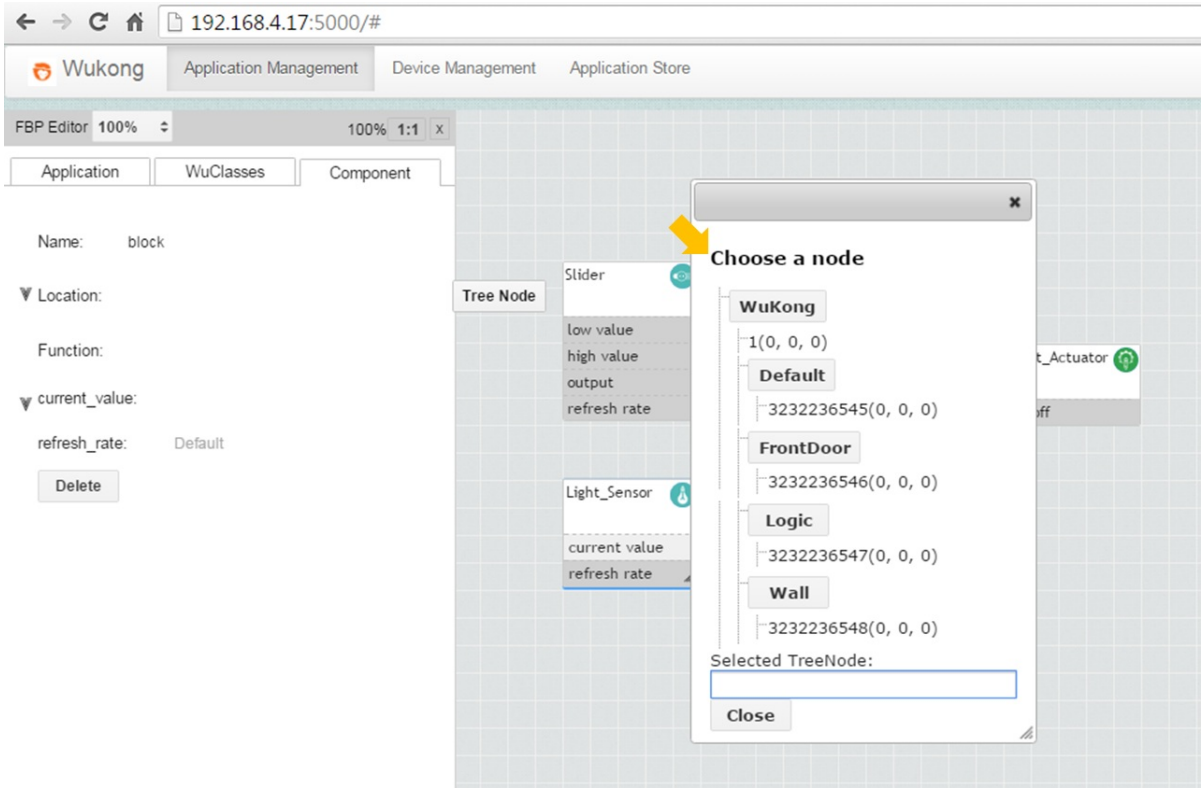
- How to Set the Location

  When deploying the application, the Master will try to find suitable nodes in the network to run each component on. This is the mapping step discussed below. Since there might be multiple sensors in different locations, we need to specify the location where we want to measure the light value and turn on the lamp.

  To set a component's location, move the cursor around the **Location** blank until a **Tree Node** button is shown up as below.



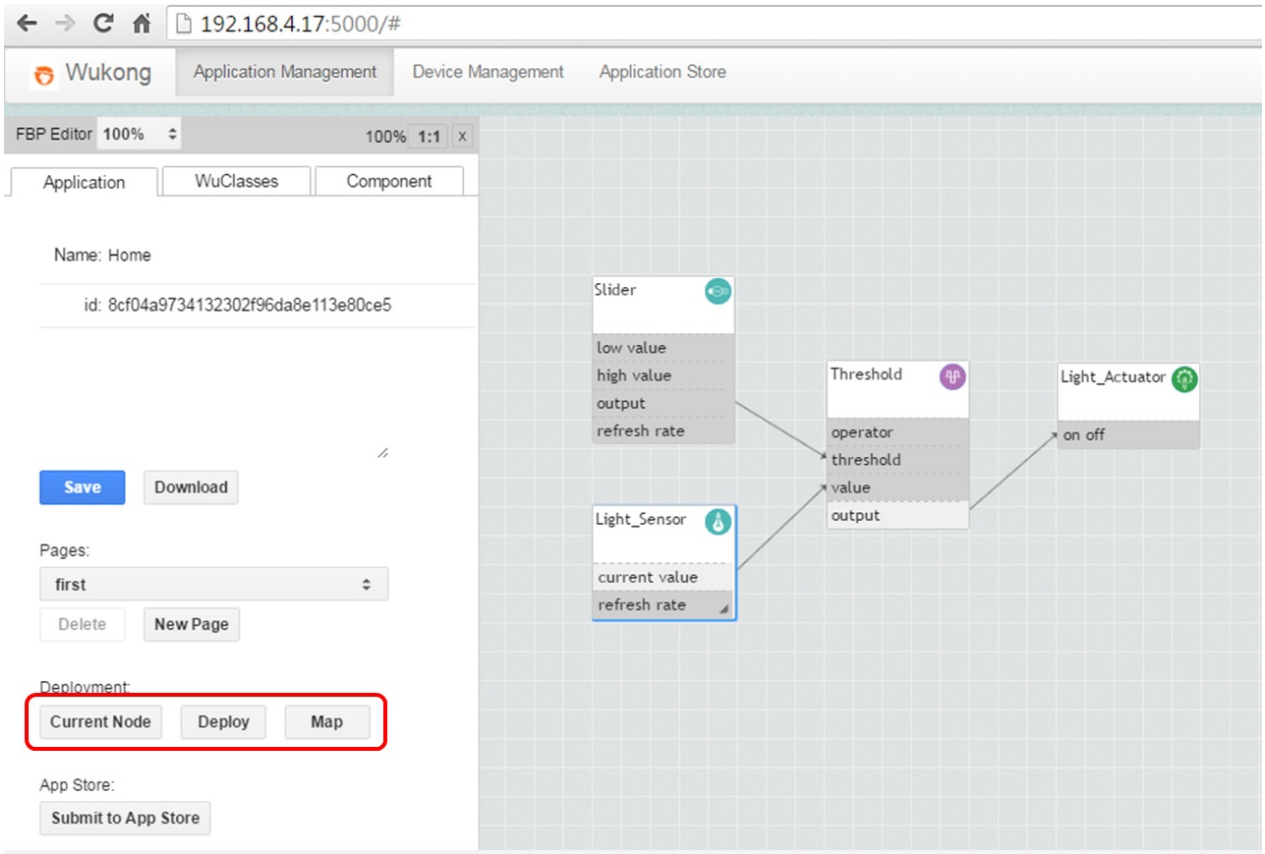
Click on the **Tree Node** button and a pop-up screen will be prompt to let you select location for each component.



Deploying an Application

After an application has been defined, a user may want to deploy it in his network. This is done in the **Application** tab at the left-hand side menu.

Deploying an application consists of three steps that will be done by the master automatically in the future, but are currently separated to allow for more control over the process and to show the intermediate results. These three steps are shown as three buttons in the **Application** tab.

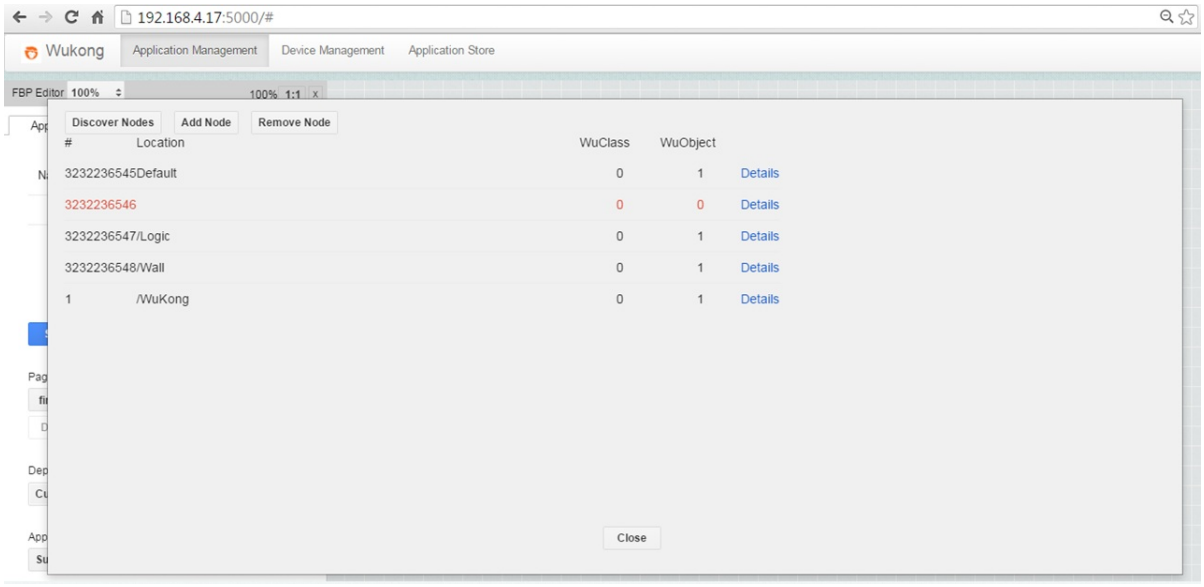


- First Step - Current Node

  The first button “Current Node” will show the discovery result. When we press this for the first time, the Master will probe the known nodes in the network for their capabilities, which may take some time. After this the cached result is used.

  Each node is represented in a row in the table, when the row is filled in red it means this node has not responded to Wukong Master’s query and is considered dead at the moment of discovery; on the other hand, it means this node has responded and is considered alive.

  If one suspects that the discovery result does not reflect the current device setup, then one could go back to the **Device Management** page to redo the discovery.

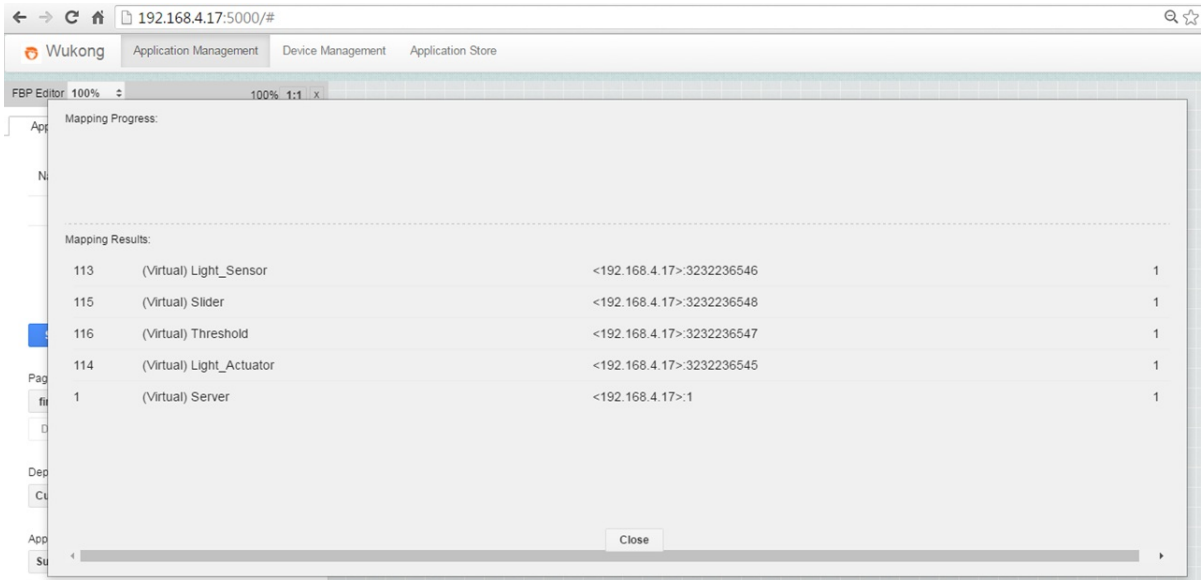


- How to Map

  When the master knows the resources available in the network, the next step is to map each component in the flow based programme onto the nodes in the network.

  Pressing the **“Map”** button will cause the master to attempt to create a mapping. The result is shown as a table with multiple columns:

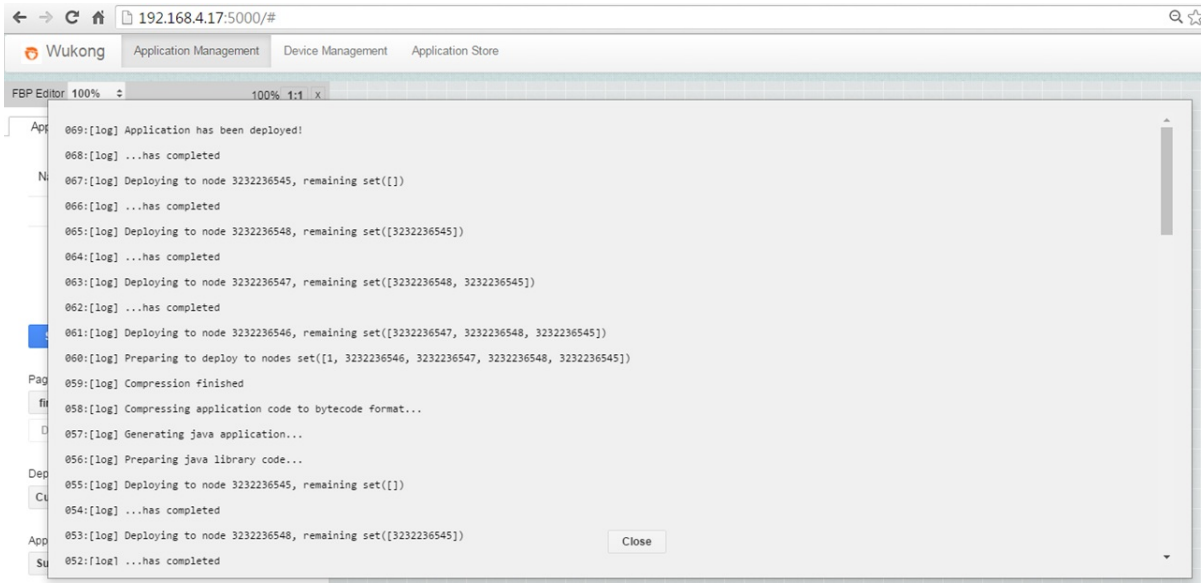
  1. WuClass Name contains the name of the wuobject in the FBP application
  2. Node Id contains the destination node id that the corresponding component will be deployed to
  3. Port Number is the port number the component will take on the destination node.



- How to Deploy

  After a mapping has been made, the application can be deployed by pressing the **"Deploy"** button. This may take up to half a minute. When the process is finished, a log will be printed showing all the internal steps the master takes to compile the application

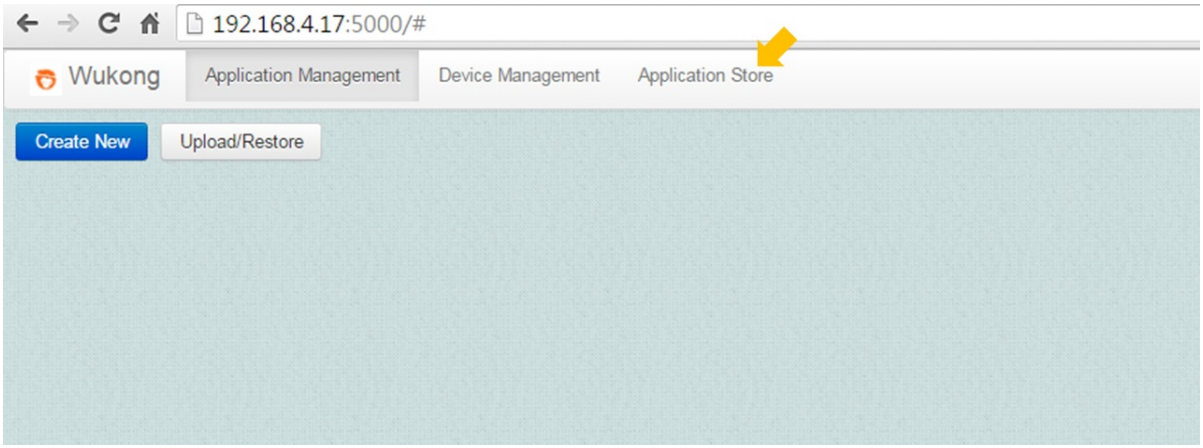
and upload it to the nodes.



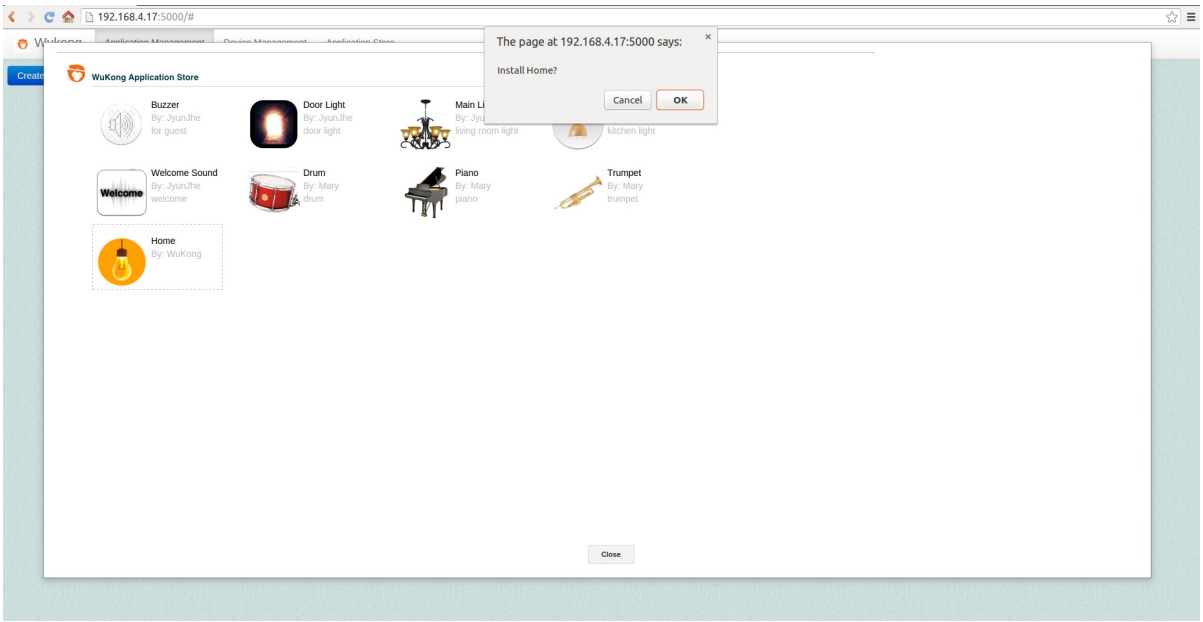
5.4 Application Store

In addition to draw a FBP, you can use sample applications in the **Application Store** as well. There are some general purposed FBP samples for your reference.

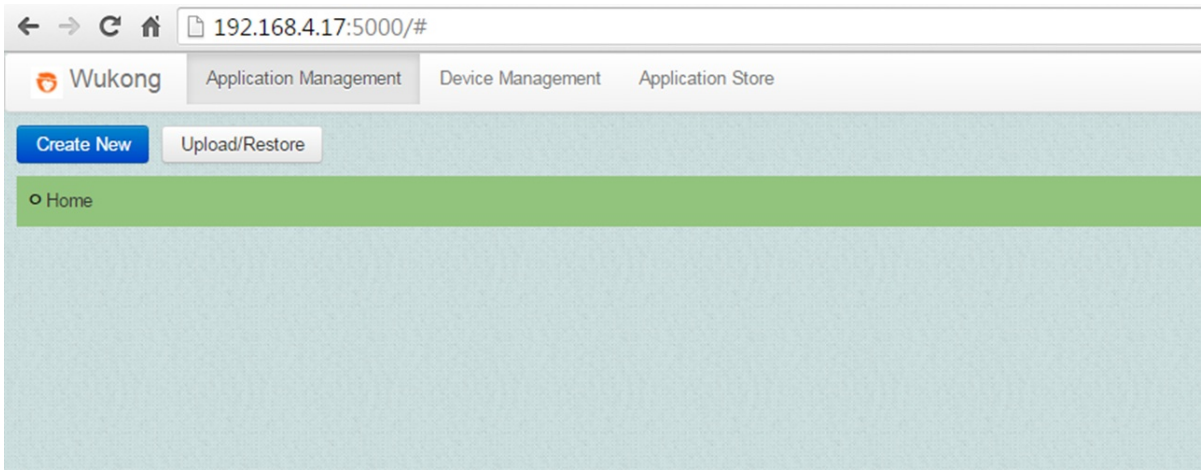
1. Click **Application Store** page



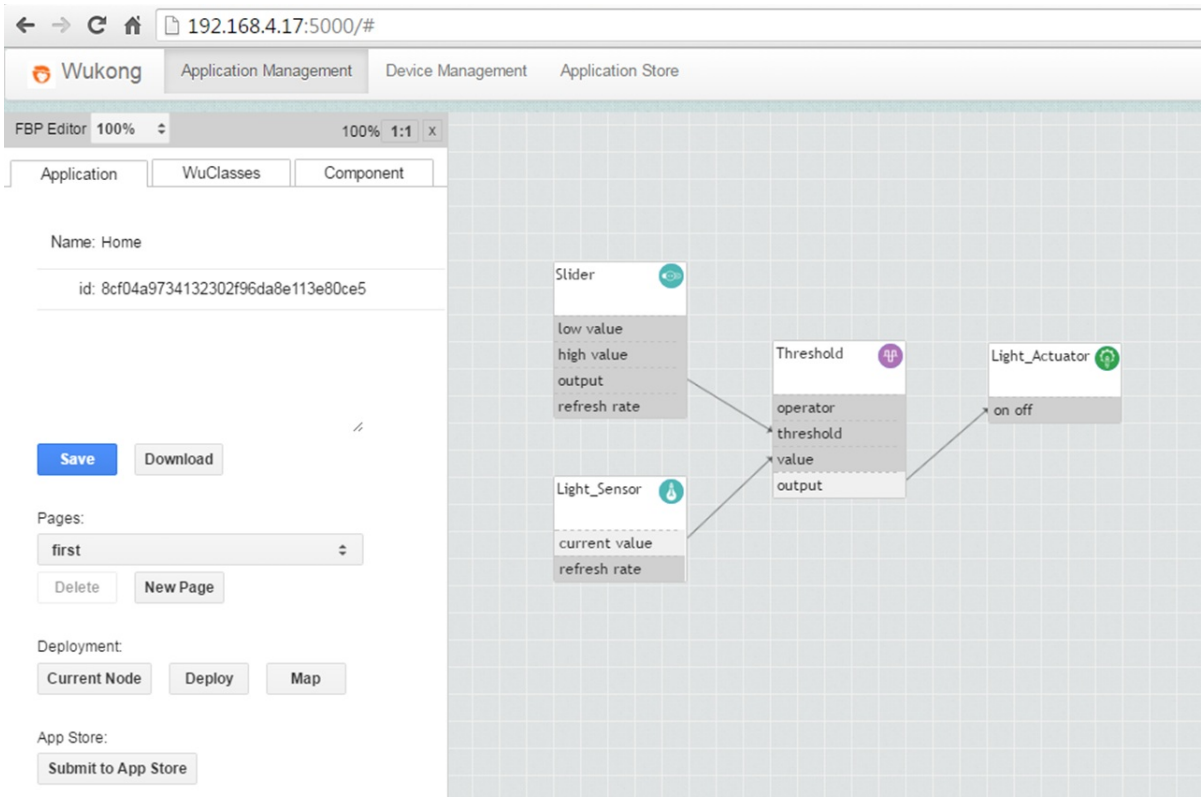
2. Choose an application. **"Home"** is the application of the last example.



3. After selecting an application, the application will be loaded into the canvas.



4. Click into the **"Home"** application, the last example appears again. You can do the same for other applications in the Application Store.



However, this application store is in its initial stage. we **Need more discussion on how to save an application in the store**, and hope that this will be a channel for people to share their IoT applications.

Chapter 6: Building New WuClasses

IoT applcation developers may create new WuClasses in WuKong to enrich their application design and to connect to new hardware. In this chapter, we show how to create new WuClasses in four sections:

1. In the WuKong Profile Framework section, we explain the system architecture of the WuKong Profile Framework running on each device node.
2. Adding a New WuClass Definition shows how to add a new WuClass definition in the WuKong WuClass library.

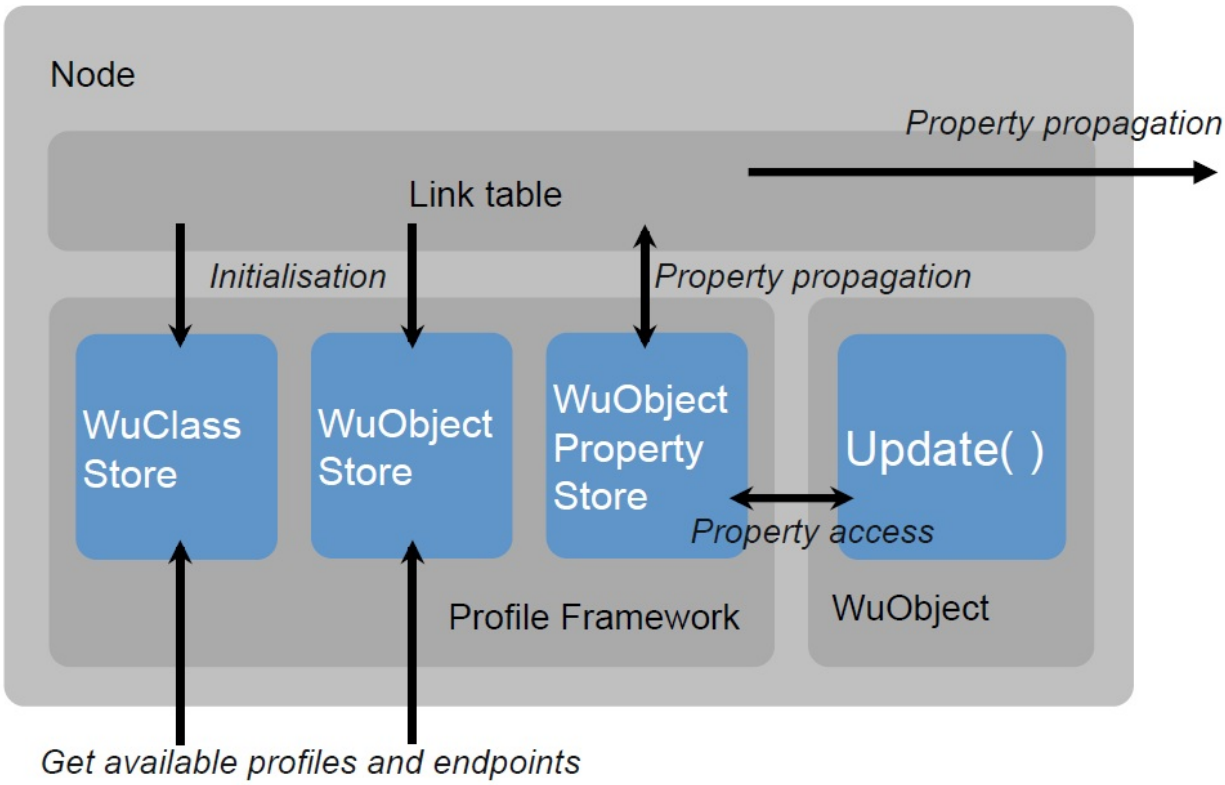
   Once a new WuClass is defined, a new entry will be created in the available WuClasses panel when restarting Master's FBP editor.
3. In the Implement a WuClass from Definition, we go through the template of WuClass implementation.

   Using the template, developers can implement specific functionalities for the WuKong components defined earlier.
4. We show some examples on WuClasses built for Grove Sensor Modules. These modules are the most commonly used in many IoT applications.

6.1 The WuKong Profile Framework (WKPF)

Before we show how to implement a WuClass, we give a brief introduction on how a WuKong device runs the application using WuClasses. This presentation is helpful for understanding the WuClass implementation.

The following figure shows a view on the software structure of a WuKong node. The most important module is the Profile Framework (WKPF) which manages the WuKong classes and objects running on the node. The node also keeps a Link Table which is created by WuKong Master to specify where a new local property value should be propagated to a remote node. Finally, a node may have many WuObjects, each with an update function running periodically (based on individual refresh rates) or running responsively (based on property propagation). Each WuObject runs the code defined by its WuClass, including the property types and the update method.



Profile Framework

The WuKong Profile Framework (WKPF) is an application framework for WuKong devices. While an FBP defines the logical view of the application, WKPF tracks the physical resources on the networks and manages the communication between them.

The framework has four main responsibilities:

1. Load WuClasses and create new WuObject instances on a node.
2. Allow Master to discover which WuClasses and WuObjects are available on a node.
3. Trigger executions within WuObjects, either periodically or as a result of changing data (from sensing hardware or network).
4. Propagate changes between linked properties of WuObjects, which may be hosted locally, or on a remote node.

Property

Property is the basic data unit of WKPF. Properties of an WuClass are the primary way to control an Wuobject. As we will see the definition of WuClass in the next section, each property has four attributes including name, access, datatype and default value.

WuClass and WuObject

As its name, WuClass is a class similar to the class in the object oriented programming. It is used to define a set of data and methods in order to create some "objects" for the wukong. Those objects are called WuObject in wukong. WuObjects are the main units of processing in an application and are hosted on the nodes.

WuObject Property Store

The properties of an WuObject are managed by the profile framework in a common property store. The framework provides functions for an WuObject to access its data within the update() method. This allows the framework to monitor the changes that an object makes to its properties, and to propagate them to connected destination WuObjects if necessary.

Update Method

update() method is a function which implements the class' behaviour. The function will be called whenever a property changes value, or periodically according to the refresh rate schedule. Once be called, WuObject will respond to a change in one of its inputs according to the behaviour defined in the update method. Therefore, writing a wuclass is all about writing update method.

An important point to remember is that an WuObject does not store its properties by itself. The properties are stored by WKPF which is responsible for monitoring and propagating changes. Therefore, an WuObject has to communicate with WKPF to read and update its properties.

Basically each update function has three phases:

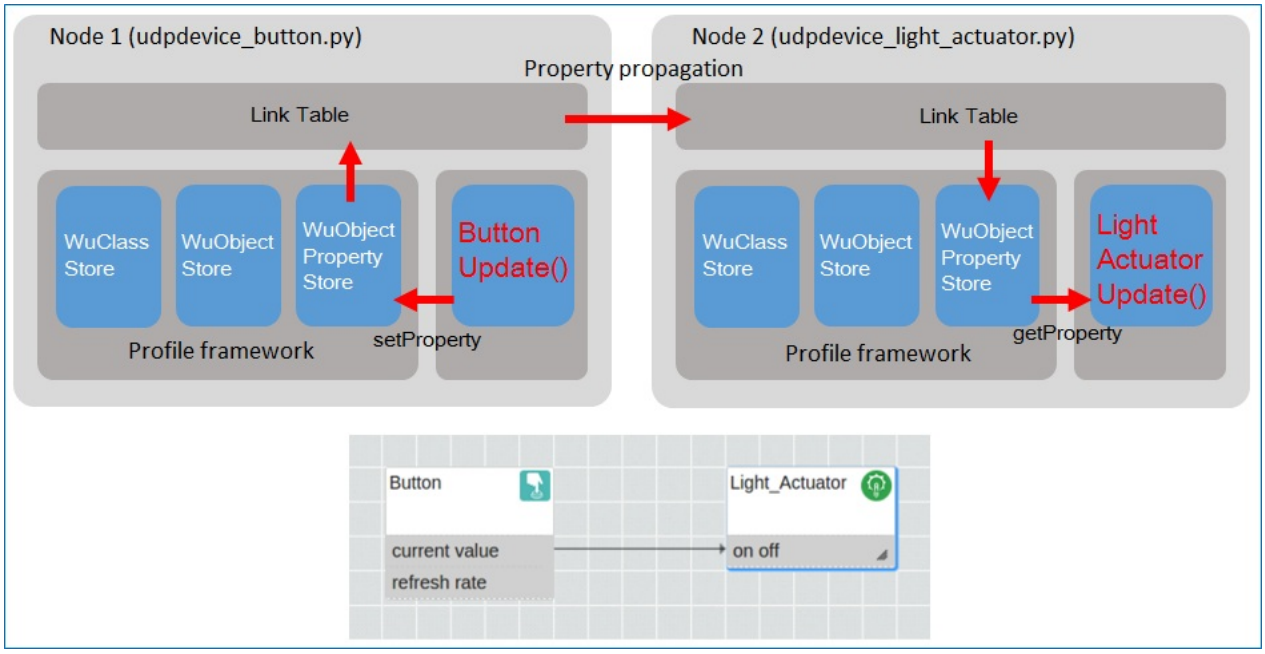
1. read input properties from the profile framework
2. do some processing
3. write output properties to the profile framework

For example, a simple sensor like a button skips the first step and just do the processing (take a measurement) and write the result to its output property. An actuator like a light actuator does not have any output, so it skips the third step and only reads its input and turns the light on or off accordingly.

Link Table

This table stores the source property and the destination property of every link on an FBP. The node will obtain this table after deployment.

The LED Control FBP Example



To explore the LED control FBP of Section 4.2, we will go through the steps from "add node" to "testing", and see how those steps relate to the above diagram.

1. Add node
   When we add a node to WuKong as in Section 4.2.2, the node will be initialized. That is, WuClasses are loaded and new WuObject instances are created on a node.

2. Discovery
   When we press "Discover node" button, Master will collect the information of WuClasses and WuObjects on every node.

3. Map
   According to the information of WuClasses and WuObjects, Master will map all FBP components to the WuObjects of some nodes.

4. Deploy When we press "Deploy" button, a node will get a link table from Master so that it will propagate its value according to the link table. The profile framework will also get available profiles and endpoints on the field from master.

5. Testing
   After we press the physical button, the periodic update method of button WuObject will transmit data to the WuObject property store. And next, the profile framework will propagate changes between linked properties of the WuObjects, which may be hosted locally, or on a remote node.

   Finally, the light actuator WuObject will be triggered by the profile framework and get the data from the WuObject property store. Therefore, the light will be turned on or off accordingly.

6. Function to access property from the update() method
   The functions which will be used to access property in the next section are:

```
    def setProperty(self,pID,val):
        self.cls.setProperty(self.port,pID,val)
    def getProperty(self,pID):
        return self.cls.getProperty(self.port,pID)
```

setProperty will write output properties to the profile framework.
getPropert will read input properties from the profile framework.

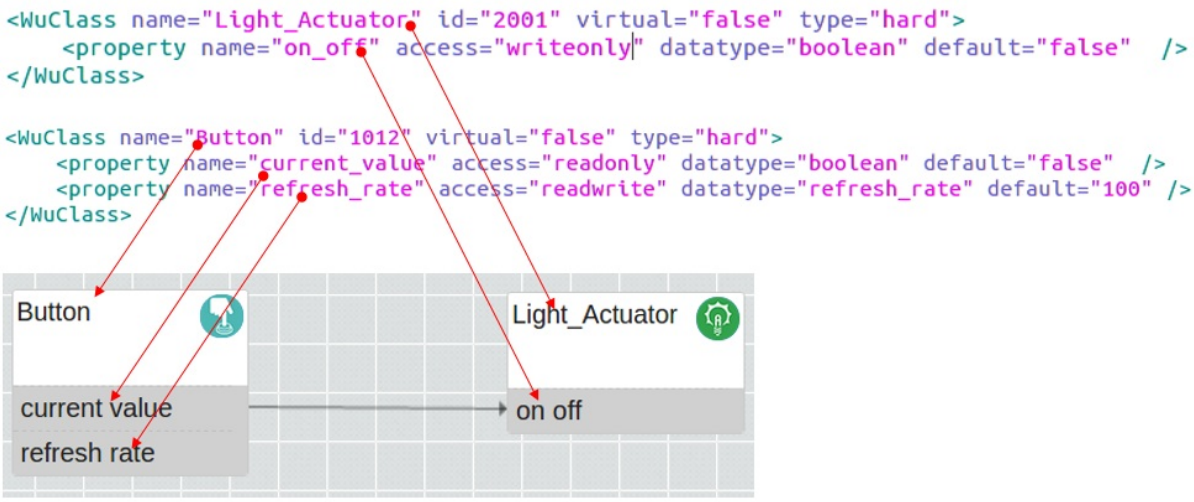
These are two methods of the WuObject defined in the /wukong/gateway/udpwkpf/udpwkpf.py.

6.2 Adding a New WuClass Definition in Master

Before we implement a WuClass, we need to specify the interface of the WuClass. The definition of WuClass is straightforward. All definitions can be found in the <*path_of_source_code*>wukong/ComponentDefinitions/WuKongStandardLibrary.xml. Writing a definition requires the following steps.

- **Define WuClass and property names**

  Comparing our first FBP application to its WuClass definition as below, we can see that the name attribute of WuClass and property are identical to the names of the components in the FBP editor. In addition, the number of property is also defined by this xml file. For example, since there are two properties defined in the button WuClass, the component of the button in the FBP editor has two equivalent properties. In contrast, light actuator has only one property both in the WuClass definition and the FBP editor.



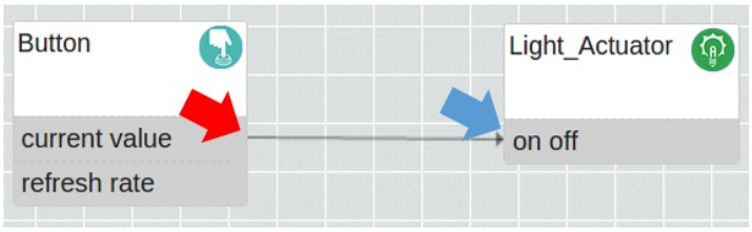
- **Define identifier attribute**

  Assign an unique identifier for a new WuClass definition. In our convention, the software WuClass ID starts from 1; the sensor WuClass ID starts from 1001; the actuator WuClass ID starts from 2001.

- **Define access attribute**

  The access attribute defines the source and the destination of data flow link. For example, the access of the current value for a button is read-only because it is a source of data for reading. The access of on_off for a light actuator is write-only because it is a destination of data for controlling LED. If the access is set as read-write, it can be both a source and a destination of a data flow link.

```
<WuClass name="Light_Actuator" id="2001" virtual="false" type="hard">
    <property name="on_off" access="writeonly" datatype="boolean" default="false"  />
</WuClass>

<WuClass name="Button" id="1012" virtual="false" type="hard">
    <property name="current_value" access="readonly" datatype="boolean" default="false"  />
    <property name="refresh_rate" access="readwrite" datatype="refresh_rate" default="100" />
</WuClass>
```



- **Define datatype attribute**

  Each property needs to have a datatype. Currently, the framework supports four datatypes: short (16 bit integer), boolean, enum, refresh rate.

  The first two should be clear. Enum types can be used to make the code, both the FBP and the WuClass implementation more readable. Currently, this type is used primarily in logic components such as threshold and math operation.

  Since a sensor should take a new measurement periodically, the refresh rate datatype is designed to specify how often measurements will be taken. It is specified in **milliseconds** and is internally a 16 bit unsigned integer.

  Therefore, since button is a sensor, when we define WuClass in the WuKongStandardLibrary.xml, we have to add refresh_rate property or else the value of button will not be propagated to the data link periodically.

- **Define default attribute**

  A default value can be specified for properties. For example, the refresh rate of button is set as 100 milliseconds. This means the current value of button will be checked per 100 msec.

- **Define virtual and type attribute**

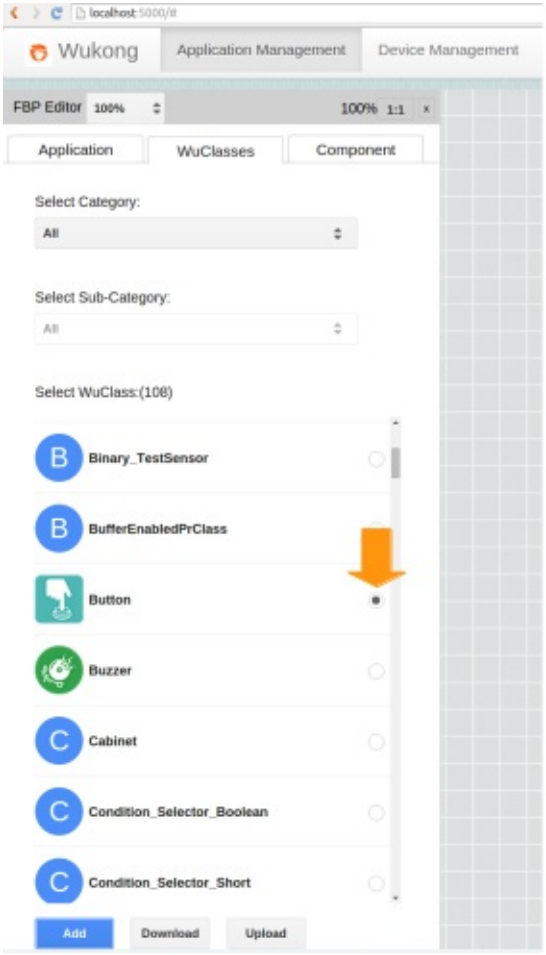
  This is an advanced attribute to be covered in future documents to define whether the component can be created dynamically and can be in software implementation. For now, it's sufficient to know that for most cases, the virtual attribute is set as false, and the type attribute is set as hard.

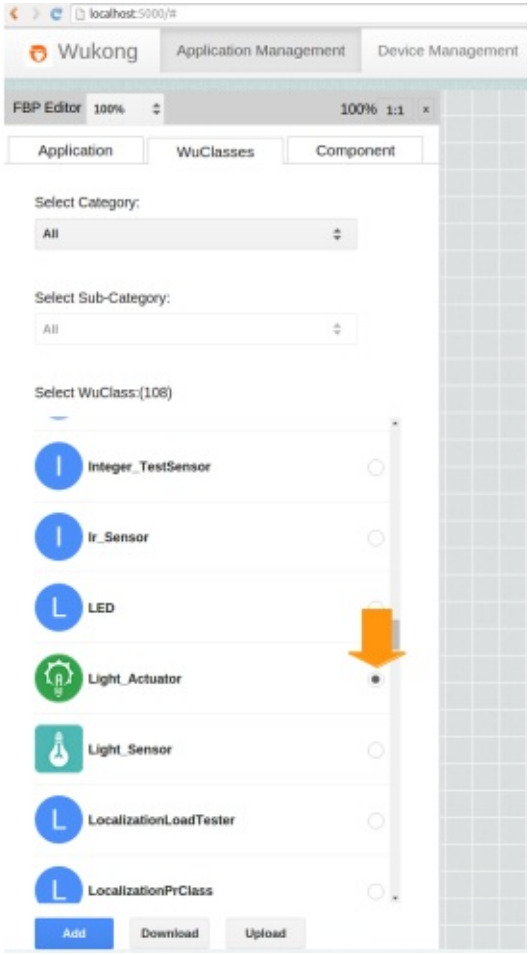
- **Delete cache files and restart master_server**

After a new WuClass is defined in the WuKongStandardLibrary.xml, you should restart Master to read the new definition. Before that, you should remove the cache file to make sure Master will create a new list of WuClasses.

```
cd <path_of_source_code>/wukong-darjeeling/wukong/master/static/js/
rm __comp__*
# next, clean browser history of http://localhost:5000
# restart master server
```

After following the above steps, you will see a new WuClass in the WuClasses list of the FBP editor just like the button and light actuator WuClasses shown below.





6.3 Implementing a WuClass from Definition

In this section, we use the LED-controlled FBP defined in Section 4.2 as an example to explain the basic template of the Python program for each device. This template can be applied to Edison, Galileo and Raspberry Pi, to include some WuClass definitions of WuKongStandardLibrary.xml as mentioned in the previous section. Once an WuClass has been implemented according to the template, we can add this WuClass as a device in the same manner as Section 5.2.

- ## WuKong Device Python Template

  All Python samples of WuKong device can be found in the following directory: *<path_of_source_code>*/wukong/gateway/udpwkpf/udpdevice*.py. All WuClass codes to be used by WuObjects on a device are included in a device template as shown below:

```
from twisted.internet import reactor
from udpwkpf import WuClass, Device
import sys

if __name__ == "__main__":
    class XXX(WuClass):
        def __init__(self):
            WuClass.__init__(self)
            self.loadClass('XXX')

        def update(self,obj,pID=None,val=None):
            pass

    class MyDevice(Device):
        def __init__(self,addr,localaddr):
            Device.__init__(self,addr,localaddr)

        def init(self):
            cls = XXX()
            self.addClass(cls,0)
            self.addObject(cls.ID)

    d = MyDevice(sys.argv[1],sys.argv[2])
    reactor.run()
```

- ## LED-control example code

Based on this device template, we have added two WuClasses, Button (Line 10) and Light-Actuator (Line 25), to realize a LED-control application in the following example code. If you have deployed that application on your device as in Section 4.2, this code should make more sense to you. On lines 6 and 7, the pins for the sensors are defined. Each WuClass has the *init*( ) method to initialize some data values, states and hardware parameters, and update( ) to accept WKPF's action invocation periodically or responsively. The code also defines MyDevice( ) which activates the device by starting all WuClasses on the device.

```
1  from twisted.internet import reactor
2  from udpwkpf import WuClass, Device
3  import sys
4  from udpwkpf_io_interface import *          ❶
5
6  Light_Actuator_Pin = 13
7  Button_Pin = 5
8
9  if __name__ == "__main__":
10     class Button(WuClass):
11         def __init__(self):
12             WuClass.__init__(self)
13             self.loadClass('Button')        ❷
14             self.button_gpio = pin_mode(Button_Pin, PIN_TYPE_DIGITAL, PIN_MODE_INPUT)  ❸
15             print "Button init success"
16
17         def update(self,obj,pID=None,val=None):                                       ❹
18             try:
19                 current_value = digital_read(self.button_gpio)
20                 obj.setProperty(0, current_value)                                     ❺
21                 print "Button pin: ", Button_Pin, ", value: ", current_value
22             except IOError:
23                 print "Error"
25     class Light_Actuator(WuClass):
26         def __init__(self):
27             WuClass.__init__(self)
28             self.loadClass('Light_Actuator')
29             self.light_actuator_gpio = pin_mode(Light_Actuator_Pin, PIN_TYPE_DIGITAL, PIN_MODE_OUTPUT)
30             print "Light Actuator init success"
31
32         def update(self,obj,pID=None,val=None):    ❻
33             try:
34                 if pID == 0:
35                     if val == True:
36                         digital_write(self.light_actuator_gpio, 1)
37                         print "Light Actuator On"
38                     else:
39                         digital_write(self.light_actuator_gpio, 0)
40                         print "Light Actuator Off"
41             except IOError:
42                 print ("Error")
```

```
44      class MyDevice(Device):
45          def __init__(self,addr,localaddr):
46              Device.__init__(self,addr,localaddr)
47
48          def init(self):
49              m1 = Light_Actuator()
50              self.addClass(m1,0)    ❼
51              self.obj_light_actuator = self.addObject(m1.ID)
52
53              m2 = Button()
54              self.addClass(m2,0)
55              self.obj_button = self.addObject(m2.ID)
56
57      if len(sys.argv) <= 2:
58          print 'python %s <gip> <dip>:<port>' % sys.argv[0]
59          print '       <gip>: IP addrees of gateway'
60          print '       <dip>: IP address of Python device'
61          print '       <port>: An unique port number'
62          print ' ex. python %s 192.168.4.7 127.0.0.1:3000' % sys.argv[0]
63          sys.exit(-1)
64
65      d = MyDevice(sys.argv[1],sys.argv[2])
66      reactor.run()  ❽
67      device_cleanup()
```

❶Since we control button and LED through gpio, we need to import the gpio library for the IoT board. To make this template work with various IoT boards, we create an interface to define how to use gpio. We will see more about this interface in the next section.

❷The input argument of self.loadClass function must be a WuClass name defined in the WuKongStandardLibrary.xml. With WuClass name, this function will parse all its attributes from WuKongStandardLibrary.xml file.

❸Use the function of udpwkpf_io_interface.py to specify which gpio pin will be used. We also specify whether this pin is digital or analog, and whether its direction is input or output.

❹Since button is a sensor, the update function should be run periodically to read sensing data from gpio. For this reason, we have defined a property with refresh_rate datatype for the button WuClass, so the update function of the button WuClass will be called by WKPF regularly.

❺The **setProperty** function of button will send its value to WKPF which then propagates the value according to the data link of FBP. The first parameter of setProperty function is pID. In this example, pID=0 indicates the first property of button, which is current_value (you can confirm with the WuKongStandardLibrary.xml). That is, the current_value is going to be propagated.

❻This update function will be called whenever there is a new data propagated to the light actuator. And because we doesn't define a property with refresh_rate for the light actuator, its update function will only run responsively. For example, in the FBP of

Ch4.2, when button is pressed, its value will propagate through data link to light actuator. And then, the update function of light actuator will be called by WKPF. WKPF also passes three parameters including **obj**(WuObject of light actuator), **pID**(which property of WuObject going to be updated), and **value**. After being called, the update function will turn on or off the light actutator responsively.

❼Add WuClass to the WuClass list of the WuKong device so that this WuClass will share the same WKPF as other WuClasses on a node. About the second parameter of addClass function, this value indicates whether this WuClass can be used to create objects. "0" means this WuClass can't create objects. That's why we use addObject function in the next line to create an object(self.obj_button) for the device. In the Ch7.3, we will use a WuClass which can create objects.

❽Don't forget to add **reactor.run()** or else the value of button will neither be refreshed periodically nor be propagated to light actuator through data link.

- ## Summary

In this section, we show the steps to implement the Python code for an IoT device:

- Copy the template to a new file as <*path_of_source_code*>/wukong/gateway/udpwkpf/udpdevice_XXX.py
- Write a WuClass for each of the sensors according to the pattern of Button, or write a WuClass for actuator according to the pattern of Light_Actuator.
- Add WuClass to MyDevice and use this WuClass to create an object on the device.

6.4 WuClass Examples for Grove Modules

Having introduced the template of writing a WuClass in Python, we will show several WuClass examples so that you can use and expand them according to your needs.

We use a sensor kit called Grove modules (http://www.seeedstudio.com/wiki/Grove_System). These modules can be assembled easily on Edison, Galileo and Raspberry Pi using individual shields. Also, they provide abundant sensors, actuators and libraries for IoT developers.

For more information of Grove starter kit, please refer to the following websites:
Using Intel Edison or Galileo: https://software.intel.com/iot/hardware/devkit
Using Raspberry Pi: http://www.dexterindustries.com/grovepi/

IO Interface

As we mentioned in the last section, in order to be compatible to all libraries of Edison, Galileo and Raspberry Pi, we have added an io interface to bridge individual library. For Edison and Galileo, we have to import mraa library; for Raspberry Pi, we have to import rpi or grovepi library. To achieve this, we add a device_type definition on line 12 of udpwkpf_io_interface.py. Currently, there are three options: DEVICE_TYPE_MRAA, DEVICE_TYPE_GPI and DEVICE_TYPE_RPI. You need to configure this definition before running a WuClass of Grove modules on the IoT board.

```
1  DEVICE_TYPE_MRAA = 0 # Edison, Galileo
2  DEVICE_TYPE_RPI  = 1 # Raspberry Pi
3  DEVICE_TYPE_GPI  = 2 # Grove Pi
4
5  PIN_TYPE_DIGITAL = 0
6  PIN_TYPE_ANALOG  = 1
7  PIN_TYPE_I2C     = 2
8
9  PIN_MODE_INPUT   = 0
10 PIN_MODE_OUTPUT  = 1
11
12 device_type = DEVICE_TYPE_MRAA
13
14 if device_type == DEVICE_TYPE_MRAA:
15     import mraa
16     import pyupm_grove
17 elif device_type == DEVICE_TYPE_RPI:
18     import RPi.GPIO as GPIO
19     GPIO.setmode(GPIO.BOARD)
20 elif device_type == DEVICE_TYPE_GPI:
21     import grovepi
22 else:
23     raise NotImplementedError
```

GPIO USAGE

After specifying which library we want to import for the WuClass, we also use this type definition to manage how the code uses the gpio function. For the time being, we have defined six common functions for gpio usage. These functions are as follows.

```
pin_mode(pin, pin_type, pin_mode, **kwargs)
# this function declares whether the pin is digital/analog, input/output, etc.

digital_read(pin_obj)
# if  pin_obj is digital and input,  we can use this to retrieve data from gpio

digital_write(pin_obj, val)
# the usage is opposite to digital_read.

analog_read(pin_obj)
# since rpi library doesn't support anaglog reading,
# we can only use this function when we import mraa or grovepi library.

analog_write(pin_obj, val)
# only grovepi supports analog writing,
# so we cannot use this function when we import mraa or rpi library.

temp_read(pin_obj)
# this function is specific for reading value from temperature sensor
# of Grove starter kit.
```

We have used the above functions throughout the WuClasses for the Grove kit. However, before you use these functions, you need to check updwkpf_io_interface.py to see whether it supports the "device type" you have selected.

Examples

According to the connection types of Grove modules, we can divide them into following categories: Digital IO, Analog IO, PWM, I2C, SPI, and UART. Sometimes, one device handles certain connection type using individual library which is quite different than others. In this case, the function of io interface could be difficult to be defined. Therefore, for the time being, we only apply io interface to WuClass of Grove starter kit.

- **Sensor with digital input**

```
from udpwkpf_io_interface import *

Button_Pin = 5

class Button(WuClass):
    def __init__(self):
        WuClass.__init__(self)
        self.loadClass('Button')
        self.IO = pin_mode(Button_Pin, PIN_TYPE_DIGITAL, PIN_MODE_INPUT)

    def update(self,obj,pID=None,val=None):
        try:
            current_value = digital_read(self.IO)
            print "Button value: %d" % current_value
            obj.setProperty(0, current_value)
        except IOError:
            print "Error"
```

- **Actuator with digital output**

```
    class Relay(WuClass):
        def __init__(self):
            WuClass.__init__(self)
            self.loadClass('Relay')
            self.relay_gpio = pin_mode(Relay_Pin, PIN_TYPE_DIGITAL, PIN_MODE_OUTPUT)
            print "Relay init success"

        def update(self,obj,pID=None,val=None):
            try:
                if pID == 0:
                    if val == True:
                        digital_write(self.relay_gpio, 1)
                        print "Relay On"
                    else:
                        digital_write(self.relay_gpio, 0)
                        print "Relay Off"
                else:
                    print "Relay garbage"
            except IOError:
                print ("Error")
```

- **Sensor with Analog input**

```
    class Slider(WuClass):
        def __init__(self):
            WuClass.__init__(self)
            self.loadClass('Slider')
            self.slider_aio = pin_mode(Slider_Pin, PIN_TYPE_ANALOG)
            print "Slider init success"

        def update(self,obj,pID=None,val=None):
            try:
                current_value = analog_read(self.slider_aio)
                obj.setProperty(2, current_value)
                print "Slider analog pin: ", Slider_Pin, ", value: ", current_value
            except IOError:
                print ("Error")
```

- **Sensor with individual library**

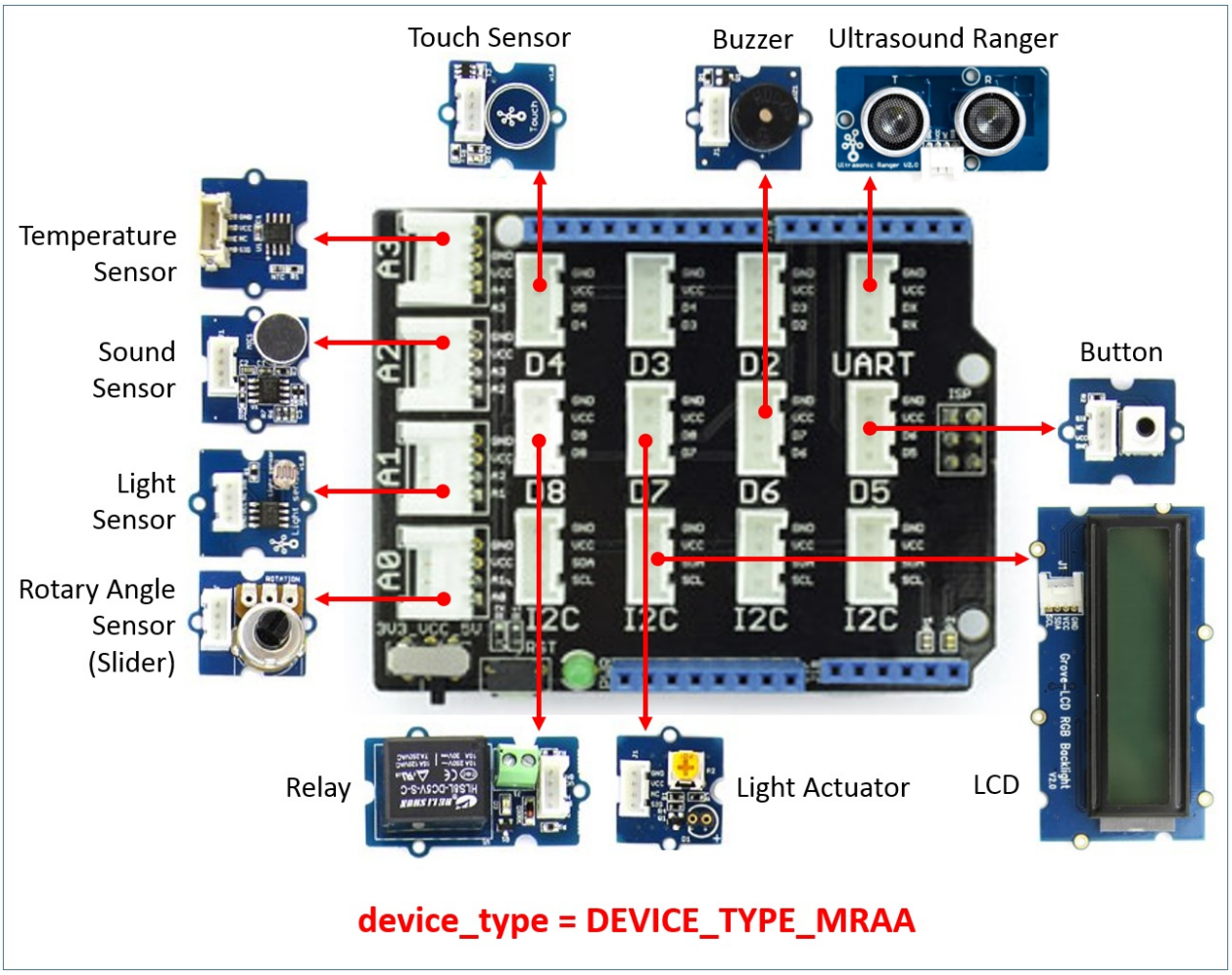
```
from udpwkpf_io_interface import *

PIN = 3 #Analog pin 0

class Temperature_sensor(WuClass):
    def __init__(self):
        WuClass.__init__(self)
        self.loadClass('Temperature_Sensor')
        print "temperature sensor init!"

    def update(self,obj,pID=None,val=None):
        try:
            current_value = temp_read(PIN)
            obj.setProperty(0, current_value)
            print "WKPFUPDATE(Temperature): %d degrees Celsius" % current_value
        except IOError:
            print ("Error")
```

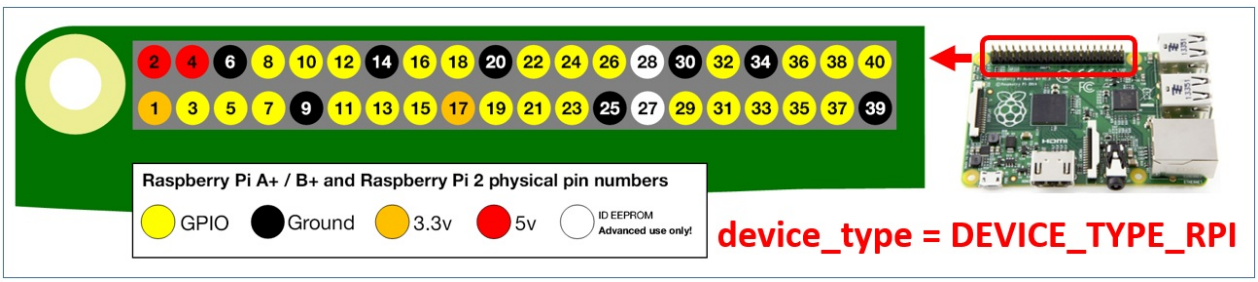
All of these WuClasses for Grove starter kit can be found in

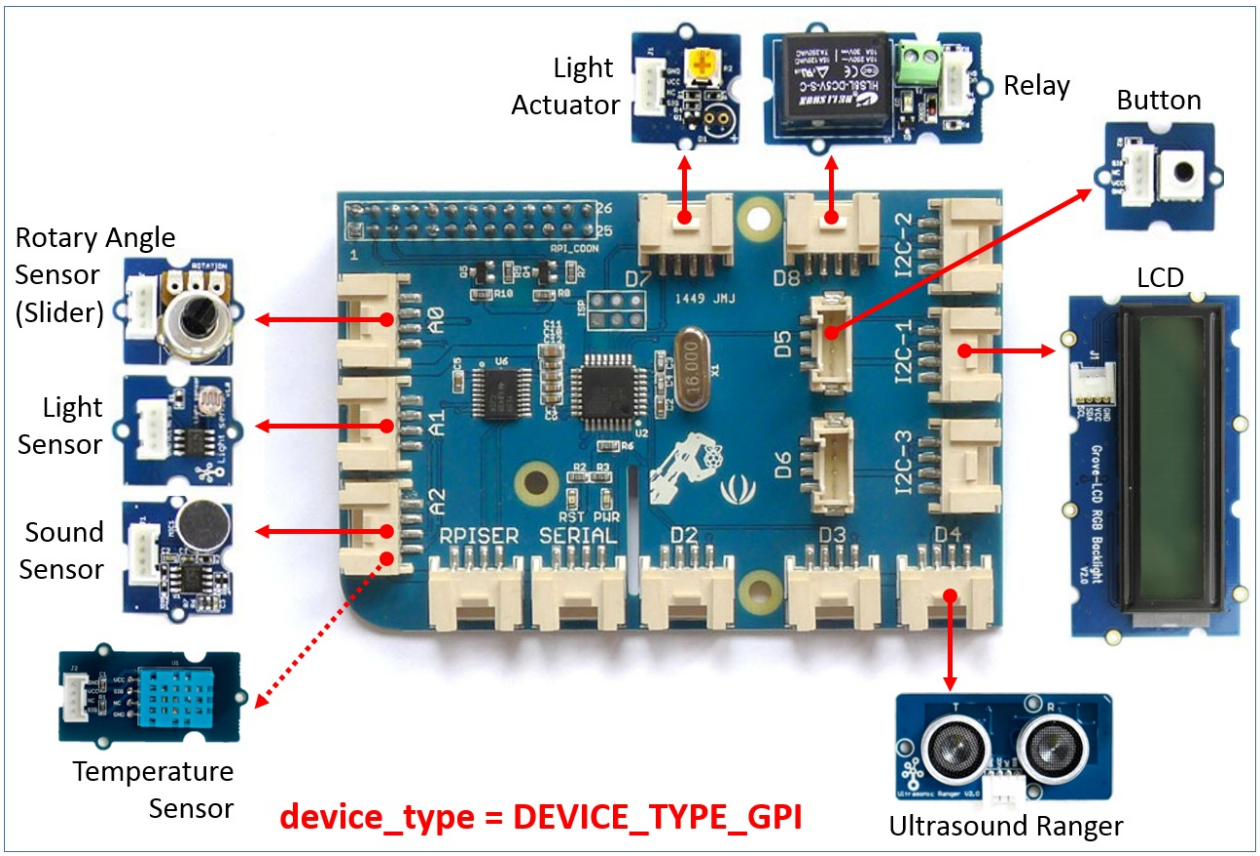
```
https://github.com/wukong-m2m/wukong-darjeeling/tree/release0.4/wukong/gateway/udpwkpf
```

Pin Numbering

While selecting a device type, we have to define the pin numbering for each sensor and actuator. The following figures are the pin assignments for grove starter kit on Edison base board, Raspberry Pi and GrovePi shield. In our code, the pin number used in each WuClass is defined at the beginning of the code so that you can change it easily.


Touch Sensor
Buzzer
Ultrasound Ranger
Temperature Sensor
Sound Sensor
Light Sensor
Rotary Angle Sensor (Slider)
Button
Relay
Light Actuator
LCD
A3
A2
A1
A0
D4
D3
D2
UART
D8
D7
D6
D5
I2C
I2C
I2C
I2C
device_type = DEVICE_TYPE_MRAA


2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
Raspberry Pi A+ / B+ and Raspberry Pi 2 physical pin numbers
GPIO
Ground
3.3v
5v
ID EEPROM
Advanced use only!
device_type = DEVICE_TYPE_RPI

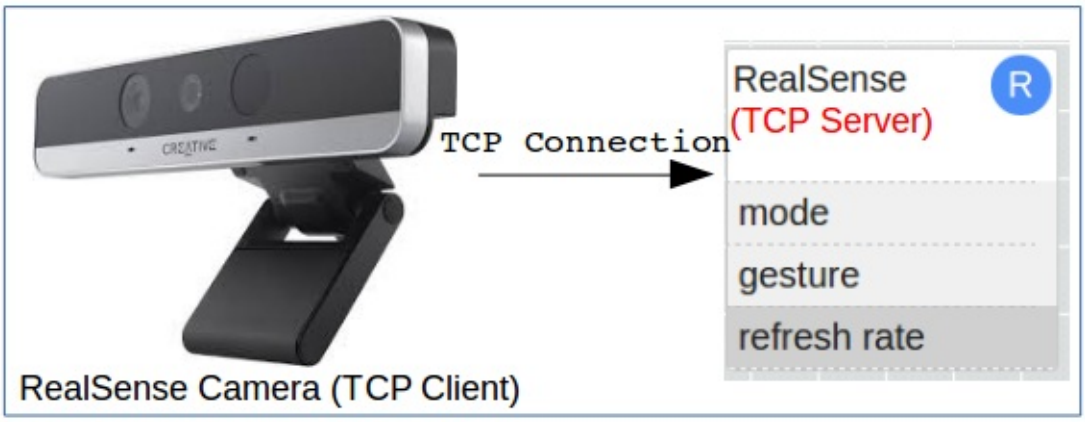

Light
Actuator
Relay
Button
Rotary Angle
Sensor
(Slider)
LCD
Light
Sensor
Sound
Sensor
Temperature
Sensor
device_type = DEVICE_TYPE_GPI
Ultrasound Ranger

Chapter 7: Building Connections to Non-WuKong Services

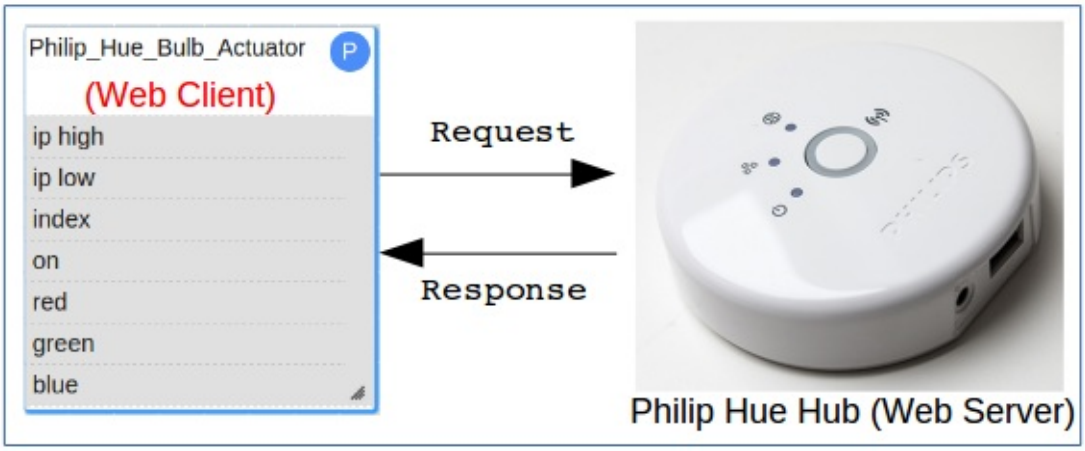
In this chapter, we show some advanced examples on how to create an external data flow in WuClass using protocols such as TCP and Web sockets. With these types of non-WuKong data flow, we can connect WuKong FBP components to external Web-based services and commercial IoT products. Such connections allow users to implement flexible and powerful distributed IoT applications by integrating WuKong and Web.

We show three examples that uses TCP server and Web connections respectively. The WuClass examples in this chapter are more complex than the ones in the previous chapter. Some of these WuClass examples use device-specific APIs to invoke their services. Other examples show how to use various web-access protocols in IoT applications. To keep examples simple and easy to understand, some non-essential details may be omitted. The presentation is focused on how to create data flows with different types of products.

- TCP Server WuClass
  In this example, we use Twisted library to establish a TCP connection between Intel RealSense Camera and FBP components so that whenever the camera captures a new gesture, it will send the reading to WuKong. This design can apply to other external sensing devices as well.

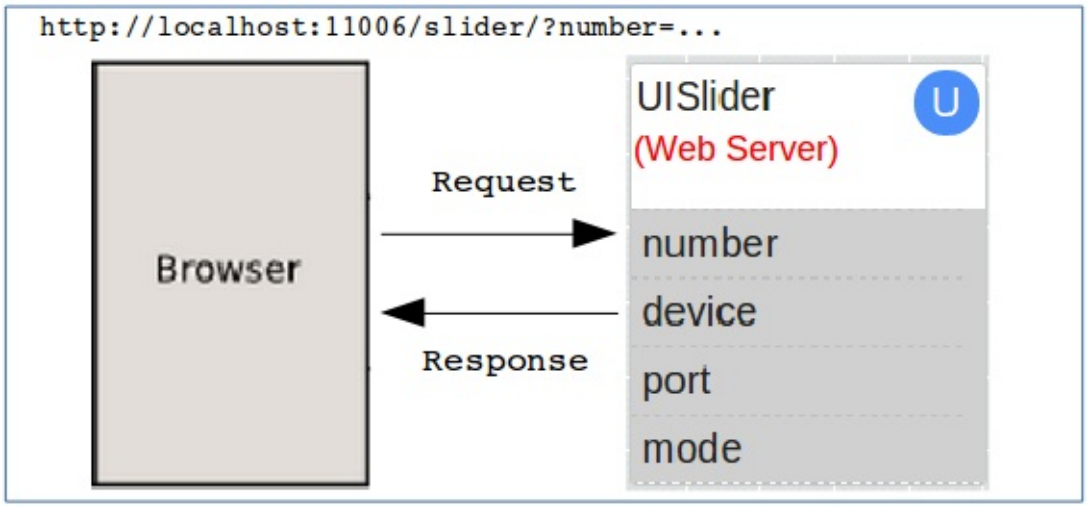


- Web Client WuClass
  In this example, we use the Twisted Web library to send HTTP requests to a Philip Hue hub. When the Hue's hub server receives messages, it will control Hue lights accordingly and return a response.



- Web Server WuClass

  In this example, we use the Twisted Web library to implement a Web server to receive URL requests from a browser application. The browser application provides a UI for users to control the output value of a UISlider component. In this way, we can provide HTTP APIs for developers to control WuKong applications. As long as they implement a Web client on their platform, they can use the HTTP API to make a connection to WuKong FBPs.



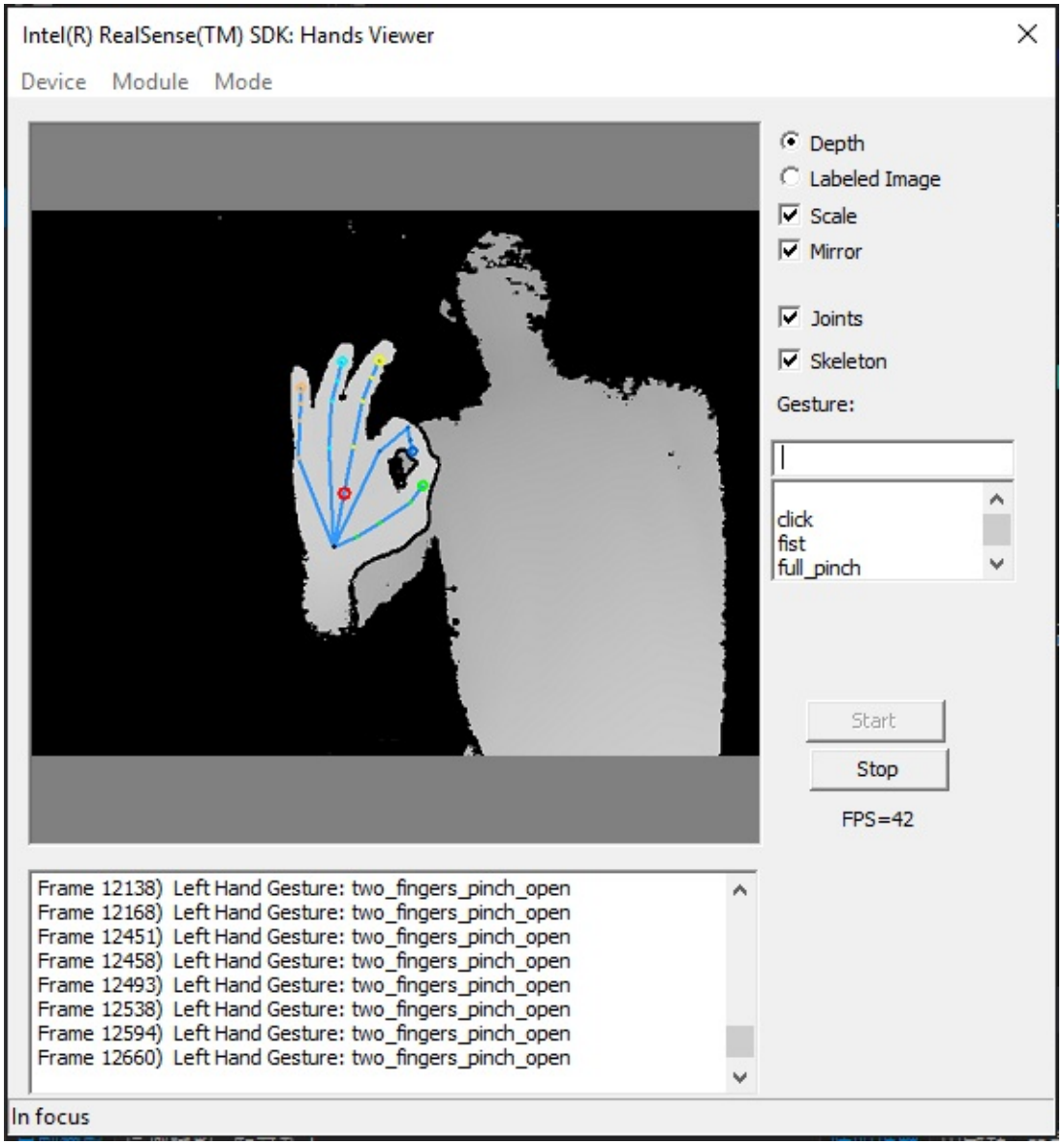
7.1 TCP Server Connection

In this section, we explain how to create a TCP connection between the Intel RealSense SDK and the WuKong FBP components. We also show an FBP application on how to adopt a RealSense camera in an IoT application.

- ## Connecting Intel RealSense Camera with WuKong

1. The Intel RealSense camera provide many features for facial analysis, hand and finger tracking, sound processing, and augmented reality. In this example, we use only the feature of finger tracking to show how to add a TCP client in WuKong FBPs. Since RealSense has provided its SDK on the website https://software.intel.com/en-us/realsense/home, we simply use its HandsViewer code. The example is under the Intel folder, Intel/RSSDK/sample/FF_HandsViewer/.
2. Our modified HandsViewer example code can be found from https://drive.google.com/open?id=0B31WAUDq3a3uZWZzb21iR0NwMDQ where we have included a socketclient file and add a function to define how to send signals. Using this code to replace the original source code under Intel/RSSDK/sample/FF_HandsViewer/src folder, you can use Microsoft Visual Studio to open the solution file called FF_HandsViewer_vs201*. When a RealSense camera is plugged in a USB port, an interactive window can be seen as below. This window shows the skeleton of the user's fingers and identify the

gesture by showing results on the log output. In this picture, it has identified the gesture as two_fingers_pinch_open.



In the HandsViewer.cpp file, a socket client is created on line 535 and passed to the DisplayGesture function which is responsible to identify gestures. After getting the gesture, the SendMessage function will send different pre-defined number to there TCP server. In that way, WuKong can respond to each gesture accordingly.

3. In WuKong, the TCP server is implemented in the udpdevice_mux_gesture_control.py, under the RealSenseProtocol class as shown below. This class implements the user-defined network protocol parsing and handling for TCP servers, as an example of explaining how we can support different protocols in WuKong.

```
43      class RealSenseProtocol(basic.LineReceiver):  ❶
44          def connectionMade(self):
45              print "Got new client!"
46              self.factory.clients.append(self)
47
48          def connectionLost(self, reason):
49              print "Lost a client!", str(reason)
50              self.factory.clients.remove(self)
51
52          def lineReceived(self, data):    ❷
53              data = data[data.find("#"):data.find("@")]
54              if len(data) < 3: return
55
56              print "received", repr(data), map(ord, data)      ❸
57              mode = int(data[1])
58              gesture = int(data[2])
59
60              self.factory.obj_realsense.setProperty(0, False) ❹
61              self.factory.obj_realsense.setProperty(1, 7)
62
63              if self.factory.obj_realsense.cls.prev_mode != mode:    ❺
64                  self.factory.obj_realsense.cls.prev_mode = mode
65                  self.factory.obj_realsense.setProperty(0, True)
66
67              if 0 <= gesture <= 7:
68                  print 'received gesture', gesture
69                  self.factory.obj_realsense.setProperty(1, gesture) ❻
```

❶ This class inherits basic.LineReceiver which is a protocol that receives lines and/or raw data, depending on the mode. For further information, please refer to https://twistedmatrix.com/documents/9.0.0/api/twisted.protocols.basic.LineReceiver.html

❷ In the lineReceived mode, each line received becomes a callback to lineReceived. In other words, data is the packet received from other TCP client. So data is a message sent by the SendMessage function in the HandsViewer.cpp file.

❸ Once received a message, the code can split and parse the message according to the design. In this WuClass, we define the mode and gesture properties. The meaning of gesture is straightforward, but the meaning of mode requires some explanation. Since RealSense doesn't provide the sample code to identify the number gesture, we cannot change mode simply by using the number gesture. One alternative is using True or False to indicate whether a user wants to change mode or not. The changing mode property corresponds to the "tap" gesture. When a user posts a tap gesture (similar to pressing a virtual button), the mode property will propagate a signal to next component.

❹ Every time we receive new data, we have to reset mode and gesture before calling the setProperty function because WuKong adopts the even-driven model. If

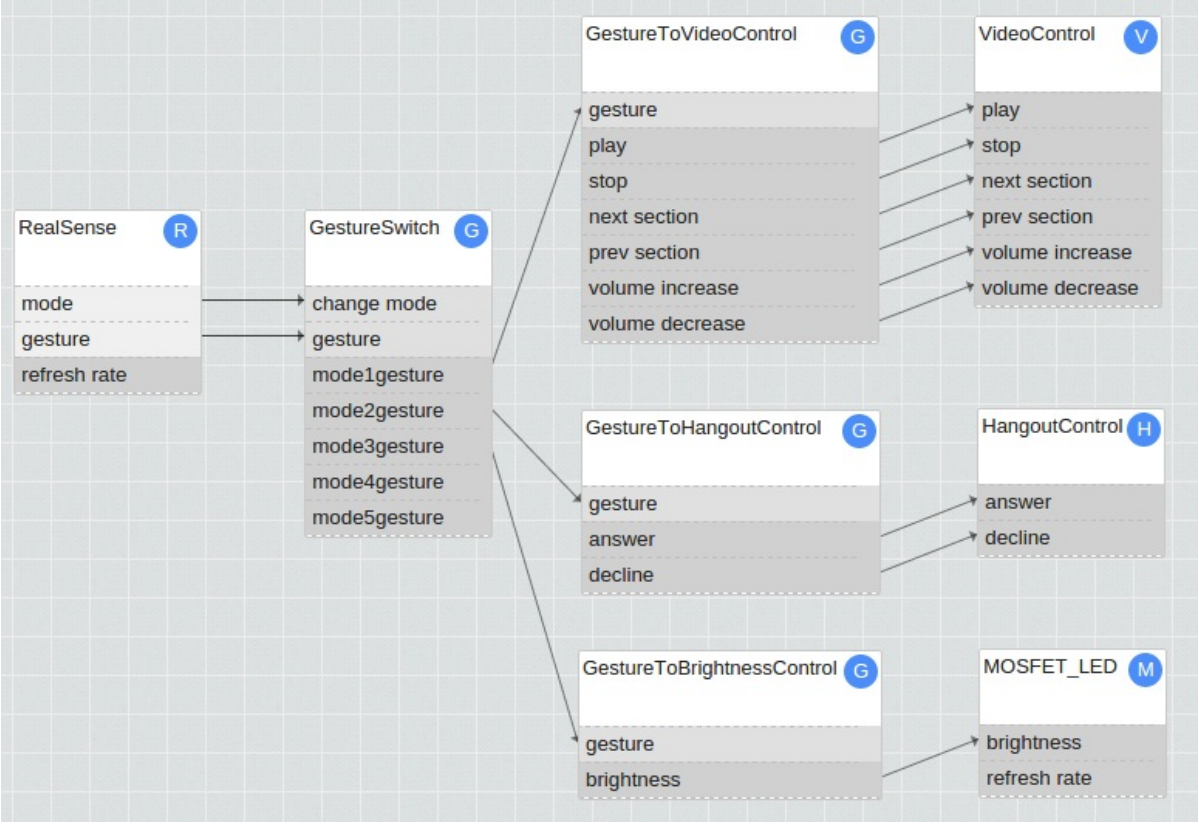
we don't reset the property, the same continual mode and gesture signal will be propagated only once.

❺ The previous state of mode property will be stored and compared to the new mode property, or otherwise we will count mode change incorrectly.

❻ Finally, the gesture value is propagated to the next component.

- ## FBP Using Intel RealSense Camera

We now show a simple FBP that uses RealSense Gesture output to control several applications running on a computer. In the following figure, the RealSense WuClass produces two output properties, mode and gesture, that will be linked to a switch component. The switch component then uses the gesture readings to issue control commands to one of the three applications, video player, Hangout communication, and LED light.



7.2 Web Client WuClass for Philip Hue

In this section, we show how to create and use the Philip Hue WuClass in FBP.

- ## Controlling Philip Hue Light in WuKong

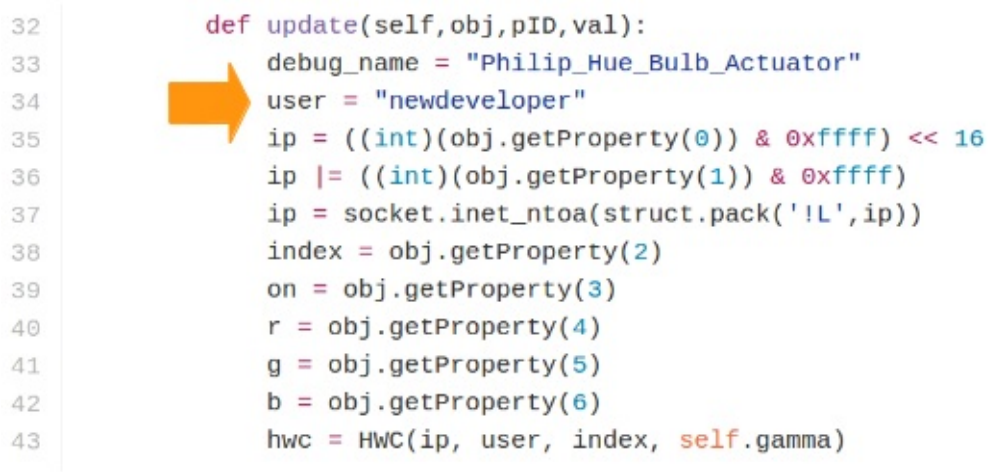
1. To allow users build their own apps, Philip Hue provides the HTTP API to get information and control Hue systems. Before using HTTP API, we must get the URL address of the Hue hub. Please follow http://www.developers.meethue.com/documentation/getting-started to find the URL address by API Debugger Tool. The URL address will be something like:

   ```
   /api/1028d66426293e821ecfd9ef1a0731df/lights
   ```

   The long number above is the username of the Philip Hue Bridge, which is created if you follow the instructions of the above website.

2. Go to the directory of Philip Hue WuClass and change the URL address definition in a Philip Hue utility file according to your bridge.

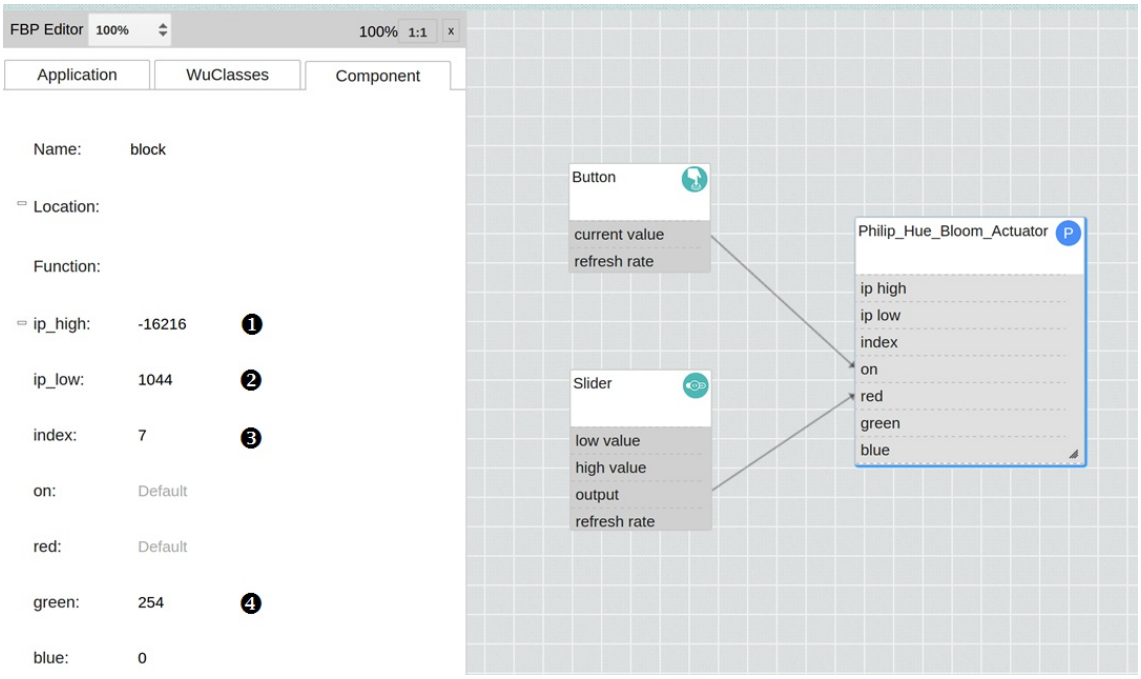
   ```
   cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/udpwkpf/
   vim udpdevice_philip_hue_*.py
   # change the user around line34 according to user name of your hue hub.
   ```

3. Add udpdevice*philip_hue**.py to WuKong according to Section 5.2.

4. Create the FBP and set the initial value

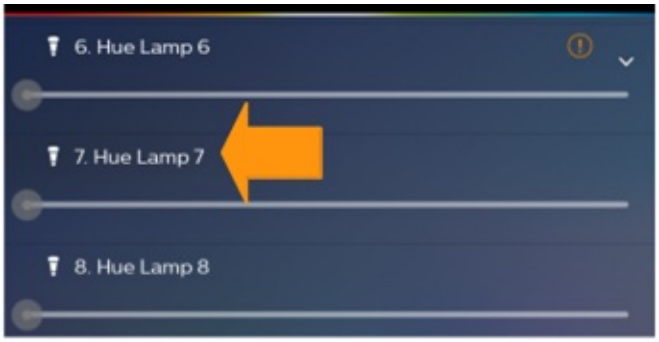
An example Philip Hue FBP can be seen as below. Button is used to turn on/off Philip Hue Bulb, and slider is used to adjust the intensity of the red color. The rest of properties can be filled in the left menu, which will pop up once the Philip Hue component is clicked.



❶❷Currently, WuKong has not supported string type, so we have to convert ip address to integer first. Since the integer length in the WuKong is only 2 bytes, we have to use two properties ip_high and ip_low to indicate the ip address of Philip Hue bridge. Here is an example:

```
Philip Hue Bridge IP = 192.168.4.20
ip_high = (192*256 + 168) - 65536 = -16216
ip_low = 4*256 + 20 = 1044
```

❸index is the id of each Philip Hue lamp, which shows on the Hue app as below.



❹The range of rgb light intensity is from 0 to 254. The number filled in this blank will be set as the default value for the FBP component.

5. Deploy this FBP according to Section 5.3

- ## Implementing Web Client WuClass

  In the following, we're going to see how web client WuClass is implemented to control the color and on/off state of Philip Hue. Through web client wuclass, we can also connect to other IoT products which provide HTTP API as Philip Hue. Since Hue has a series of products and all of them can be controlled by the same HTTP API, we create an utility file with commonly used classes to avoid repetition codes. This utility file is philip_hue_utils.py as below.

```
14  class BeginningPrinter(protocol.Protocol):
15      def __init__(self, finished, gamma):
16          self.finished = finished
17          self.remaining = 1024 * 10
18          self.gamma = gamma
19
20      def dataReceived(self, bytes):                              ❺
21          if self.remaining:
22              display = bytes[:self.remaining]
23              print 'Some data received:'
24              print display
25              try:
26                  display_parse = json.loads(display)             ❻
27                  self.gamma.x = display_parse['state']['xy'][0]
28                  self.gamma.y = display_parse['state']['xy'][1]
29                  self.gamma.bri = display_parse['state']['bri']
30                  modelid = display_parse['modelid']
31                  strcmp = lambda modelid_list: modelid in modelid_list
32                  modelid_list_1 = ['LCT001', 'LCT002', 'LCT003', 'LCT007', 'LLM001']
33                  modelid_list_2 = ['LLC006', 'LLC007', 'LLC010', 'LLC011', 'LLC012', 'LLC013', 'LST001']
34                  modelid_list_3 = ['LLC020', 'LST002']
35                  if strcmp(modelid_list_1):
36                      self.gamma.gamma = 1
37                  elif strcmp(modelid_list_2):
38                      self.gamma.gamma = 0
39                  elif strcmp(modelid_list_3):
40                      self.gamma.gamma = 2
41                  else:
42                      self.gamma.gamma = -99
```

```
            except Exception as e:
                self.gamma.message = e.message
                self.gamma.gamma = -101

            self.remaining -= len(display)

    def connectionLost(self, reason):                                    ❼
        print 'Finished receiving body:', reason.getErrorMessage()
        self.finished.callback(None)
class hue_web_client():
    def __init__(self, ip, user, index, obj):                            ❶
        self.http_get_path = 'http://'+ip+'/api/'+user+'/lights/'+str(index)
        self.http_put_path = 'http://'+ip+'/api/'+user+'/lights/'+str(index)+'/state/'
        self.gamma = obj

    def get_gamma(self):                                                 ❸
        agent = Agent(reactor)
        defer = agent.request(
            'GET', self.http_get_path,
            Headers({'User-Agent': ['Twisted Web Client Example'],
                     'Content-Type': ['text/x-greeting']}),
            None)
        defer.addCallback(self.cbRequest)
        # defer.addBoth(self.cbShutdown)

    def put_command(self, body):                                         ❷
        agent = Agent(reactor)
        defer = agent.request(
            'PUT', self.http_put_path,
            Headers({'User-Agent': ['Twisted Web Client Example'],
                     'Content-Type': ['text/x-greeting']}),
            body)
        # defer.addBoth(self.cbShutdown)

    def cbRequest(self, response):
        #print 'Response version:', response.version
        #print 'Response code:', response.code
        #print 'Response phrase:', response.phrase
        #print 'Response headers:'
        #print pformat(list(response.headers.getAllRawHeaders()))
        finished = Deferred()
        response.deliverBody(BeginningPrinter(finished, self.gamma))     ❹
        return finished

    def cbShutdown(self, ignored):
        reactor.stop()
```

Note: please refer to Twisted official website for more detailed information about how to use Twisted wevb client. (http://twistedmatrix.com/documents/current/web/howto/client.html)

❶http_get_path and http_put_path are defined according to HTTP API of Philip Hue.

❷put_command is a method used to send HTTP PUT request. The parameters include a request method, a request URI, the request headers, and an object which can produce the request body. The agent is responsible for resolving the hostname into an IP address and connecting to it.

❸get_gamma is used to send HTTP GET request to retrieve gamma value from Hue. Since the gamut of Hue products fall into three different areas and not all of the RGB values can be mapped to those gamuts, we need to calibrate each RGB value with gamma value.

❹The Response object of HTTP GET request has a deliverBody method which makes the response body available.

❺The BeginningPrinter protocol in this example is passed to Response.deliverBody and the response body is then delivered to its dataReceived method as it arrives.

❻After the content of HTTP body is parsed, the gamma value can be determined.

❼When the body has been completely delivered, the protocol's connectionLost method is called.

Next, the WuClass below uses the web client in the utility file to control Philip Hue.

```
import philip_hue_utils; HWC = philip_hue_utils.hue_web_client    ❶
import philip_hue_utils; HC = philip_hue_utils.hue_calculation


    class Philip_Hue_Bulb_Actuator(WuClass):
        def __init__(self):
            WuClass.__init__(self)
            self.loadClass('Philip_Hue_Bulb_Actuator')
            self.lasttime = int(round(time.time() * 1000))
            self.loop_rate = 500
            self.gamma = Gamma()
            print "Hue Bulb Actuator init success"

        def update(self,obj,pID=None,val=None):
            debug_name = "Philip_Hue_Bulb_Actuator"
            user = "newdeveloper"
            ip = ((int)(obj.getProperty(0)) & 0xffff) << 16
            ip |= ((int)(obj.getProperty(1)) & 0xffff)
            ip = socket.inet_ntoa(struct.pack('!L',ip))
            index = obj.getProperty(2)
            on = obj.getProperty(3)
            r = obj.getProperty(4)
            g = obj.getProperty(5)
            b = obj.getProperty(6)
            hwc = HWC(ip, user, index, self.gamma)                 ❷


            currenttime = int(round(time.time() * 1000))

            if (currenttime - self.lasttime > self.loop_rate):     ❸
                if(self.gamma.gamma < 0):
                    hwc.get_gamma()                                ❹
                    if (self.gamma.gamma < 0):
                        print "\n_____%s_____GET gamma error:%d\n" % (debug_name, self.gamma.gamma)
                        if (self.gamma.gamma < -99):
                            print "\n_____%s_____JSON error:%s\n" % (debug_name, self.gamma.message)
                        else:
                            print "\n_____%s____Error!ip:%s,index:%d\n" % (debug_name, ip, index)
                    self.lasttime = currenttime
                    return

                hc = HC()
                hc.RGBtoXY(self.gamma, r, g, b)                    ❺

                if(on):                                            ❻
                    command = ("{\"on\":true, \"xy\":[%.2f,%.2f], \"bri\":%d}" % (self.gamma.x, self.gamma.y
                else:                                                                          , (int)(self.gamma.bri*255)))
                    command = ("{\"on\":false}")

                time.sleep(0.05)
                print "\n_____%s_____PUT command:%s\n" % (debug_name, command)

                body = FileBodyProducer(StringIO(command))         ❼
                hwc.put_command(body)                              ❽
                self.lasttime = currenttime
```

❶Import web client from the utility file.

❷With the property value of Hue bridge ip, username, and index, an object of web client can be created.

❸Too frequent requests will cause Hue bridge to postpone reponse or drop message,

so we have to set a minimum request period to ensure the functionality. In this case, the minimum period is set as 0.55 sec.

❹ Use the method of web client to get gamma value.

❺ Use gamma value to calibrate the RGB value.

❻ These two commands will be sent to the Hue bridge as a HTTP body.

❼ FileBodyProducer is responsible to produce a HTTP body object.

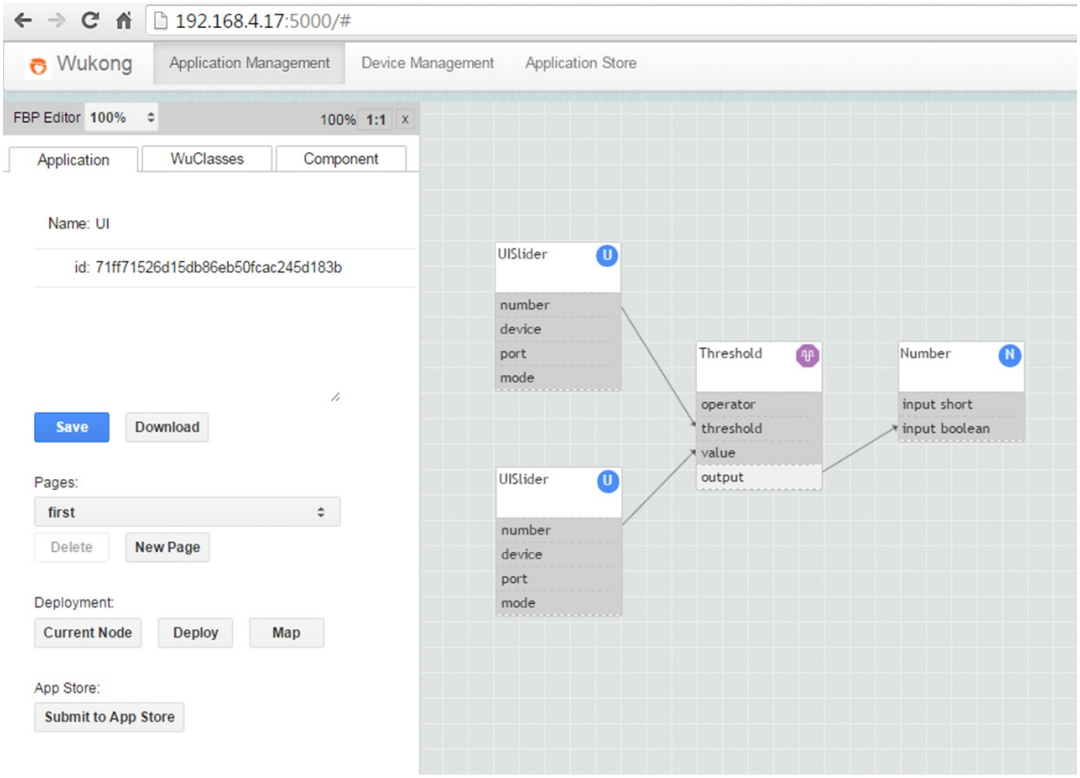
❽ Use put_command to send HTTP request to the Hue bridge.

7.3 Web Server WuClass for UISlider

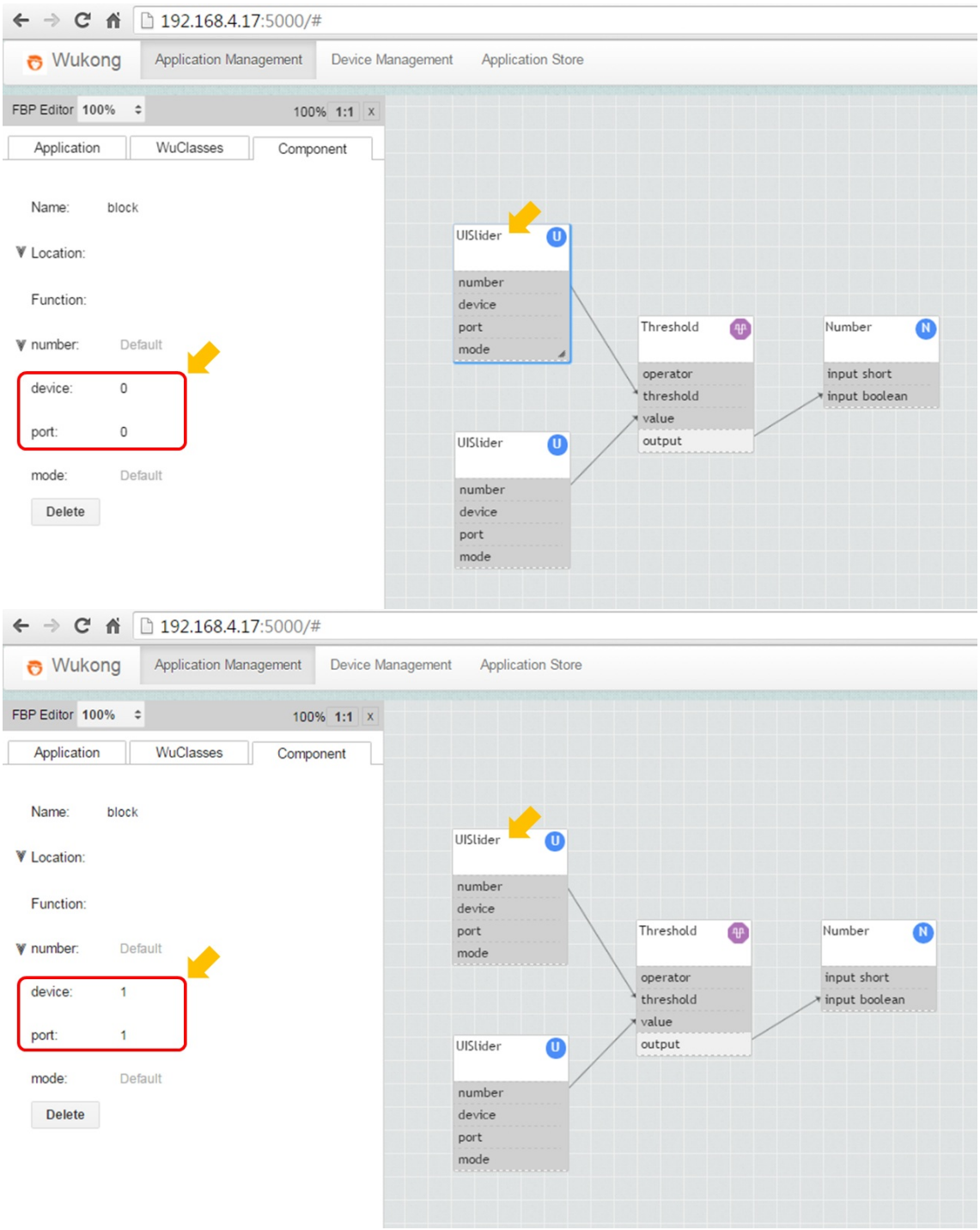
In this section, we first show how to use UISlider in a WuKong FBP. We also present how to use the Twisted Web server to build the UISlider WuClass.

• FBP Using UISlider

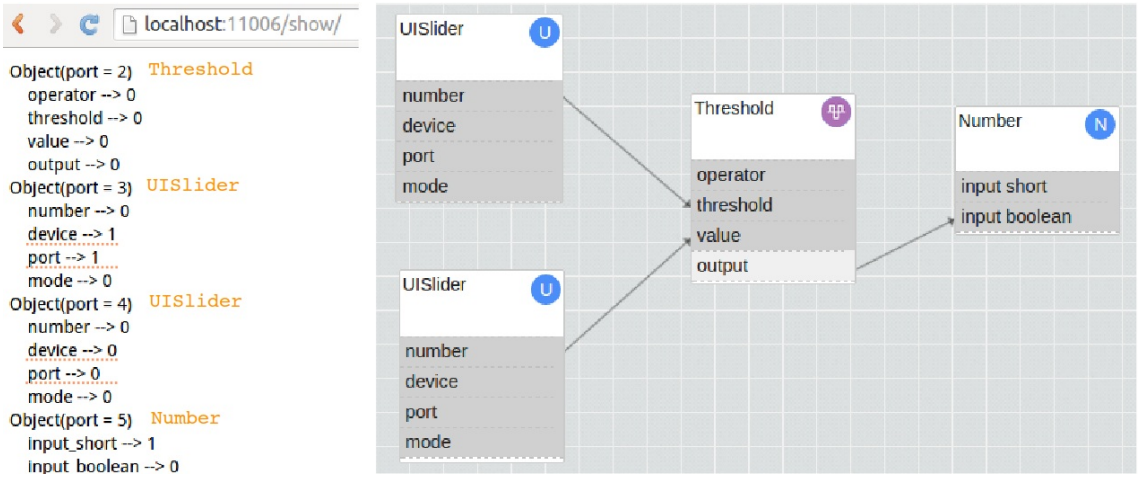
1. This FBP has three components:
   - UISlider
     Just as the output of a slide potentiometer has a range from 0 to vcc volt, a UIslider receives a number from the browser and sends it to the next component. In this FBP, the next component of UISlider is a Threshold.
   - Threshold
     This component uses the output from the upper UISlider as a threshold and compares this value with the output of another UISlider. The operator is used to decide the comparison rule which can be LT, LTE, GT, and GTE. The default operator is LT. The output property will propagate **True** to the next component.
   - Number
     The Number component is mainly for debug purposes. It will display the received value on the console screen.



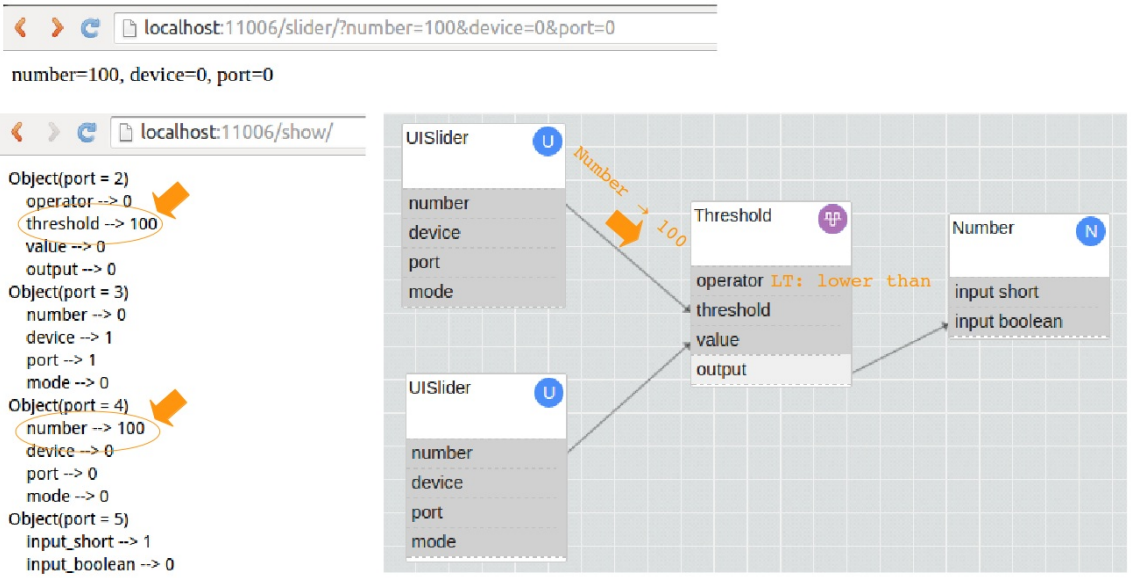
2. Initial value setting
   To specify which UISlider is to be used, each UISlider will be given a different pair (device, port) of initial (device, port) values. The values for different components must be different.



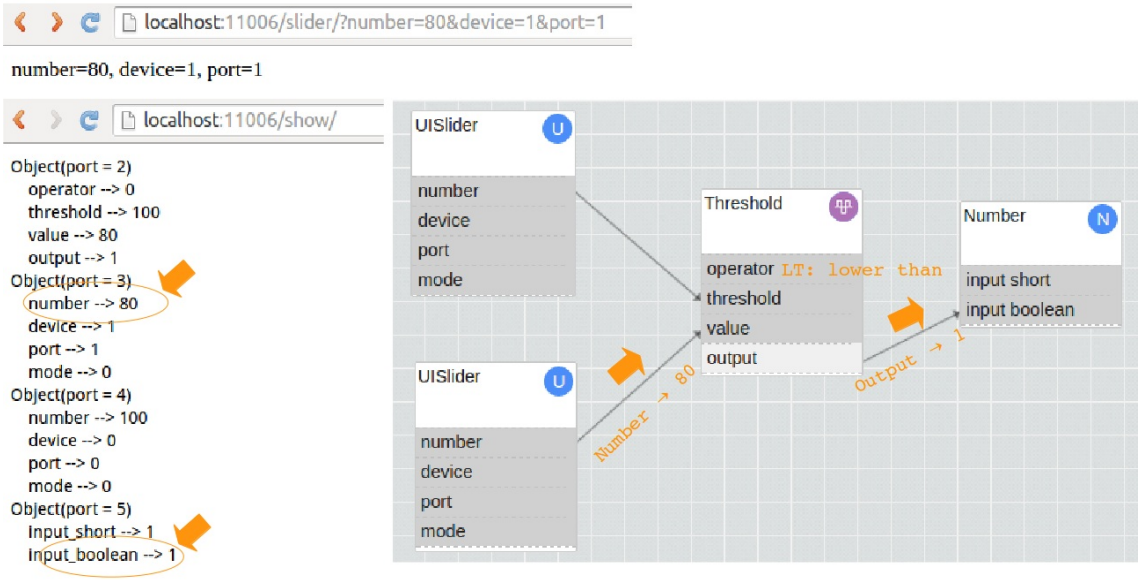
3. To see how the data flow between components, we add a ShowAll class which, when it receives a request from a browser, shows all current values of all component on the browser (the left side of the figure below).



4. When a user uses the url to send a request to change the output number of the upper UISlider, we can see the data propagation from the ShowAll request just mentioned. In this case, it sends a request to set the threshold property to 100.



5. Use a request to change the output value of the lower UISlider to 80, we can see the data propagation as above. In this case, it sends a request to set the value property to 80, and the ouput of the Threshold component will become True.



- **UISlider WuClass**

  The Slider WuClass is shown below. The details are explained on certain lines as footnotes at the end of this section.

```
19  class Slide(resource.Resource):
20      isLeaf = True
21      def __init__(self,device):
22          self.device = device
23          self.UISlider_ID = 11006
24
25      def render_GET(self,request):    ❶
26          try:
27              n = (int)(request.args.get('number')[0])    ❷
28              d = (int)(request.args.get('device')[0])
29              p = (int)(request.args.get('port')[0])
30              print self.device.objects
31              for i in range(0,len(self.device.objects)):
32                  obj = self.device.objects[i]            ❹
33                  if obj.getID() == self.UISlider_ID:
34                      if obj.getProperty(1) == d and obj.getProperty(2) == p:  ❺
35                          obj.setProperty(0,n)
36                          print "(device, port) = (%d, %d) set property" %  (obj.getProperty(1), obj.getProperty(2))
37                      else:
38                          print "(device, port) = (%d, %d) doesn't set property" %  (obj.getProperty(1), obj.getProperty(2))
39          except:
40              pass
41          return 'number=%d, device=%d, port=%d' % (n, d, p)
43  class MyDevice(Device):
44      def __init__(self,addr,localaddr):
45          Device.__init__(self,addr,localaddr)
46      def init(self):
47          cls = UISlider()
48          self.addClass(cls, self.FLAG_APP_CAN_CREATE_INSTANCE| self.FLAG_VIRTUAL)    ❸
49
50  if len(sys.argv) <= 2:
51      print 'python udpwkpf.py <ip> <port>'
52      print '      <ip>: IP of the interface'
53      print '      <port>: The unique port number in the interface'
54      print ' ex. python <filename> <gateway ip> <local ip>:<any given port number>'
55      print ' ex. python udpdevice_eeg_server.py 192.168.4.7 127.0.0.1:3000'
56      sys.exit(-1)
57
58  d = MyDevice(sys.argv[1],sys.argv[2])
59  root=static.File("./www")
60  root.putChild("slider",Slide(d))        ❻
61  site=server.Site(root)
62  site.device = d
63  reactor.listenTCP(11006,site)
64  reactor.run()                           ❼
```

❶ render_GET is a method to be implemented to support GET requests from clients. And render_GET returns string according to our implementation as a response.

❷ We can get the values of each http request by parsing the arguments.

❸ When we use these flags for the addClass function, the class will be used to create new WuObjects. That's why when we add only one UISlider WuClass, we still can have two UISliders in an FBP.

❹ Since we can have several identical FBP components in one FBP, we must decide which component is requested by the client.

❺ The device property and port property of UISlider are used to find the requested one. That is why we have to fill the initial values when drawing a FBP.

❻ putChild can be used to create the URL name for a Resource instance.

❼ Specify the port to serve http get request. In this case, http://localhost:11006/slider is the base to make the http GET request.