

WuKong

Release 0.4

Chinese

Version

WuKong Project

Published
with GitBook



Table of Contents

更新日誌	0
第1章: 悟空中介軟體簡介	1
如何使用這本書	1.1
第2章: 設定伺服器環境	2
第3章: 物聯網開發板	3
設定Intel Edison	3.1
設定Intel Galileo Gen 2	3.2
設定Raspberry Pi Board	3.3
第4章: 範例	4
電腦上的範例 - 音樂播放器	4.1
步驟1. 啓動主控台及通訊閘程式	4.1.1
步驟2. 新增裝置	4.1.2
步驟3. 撰寫資料流程式	4.1.3
步驟4. 執行程式	4.1.4
尋找裝置問題	4.1.5
物聯網開發板 - 控制LED	4.2
步驟1. 啓動主控台及通訊閘程式	4.2.1
步驟2. 新增裝置	4.2.2
步驟3. 撰寫資料流程式	4.2.3
步驟4. 執行程式	4.2.4
第5章: 悟空管理介面	5
網路管理	5.1
裝置管理	5.2
應用程式管理	5.3
應用程式商店	5.4
第6章: 動手寫悟空類別元件	6
悟空屬性架構	6.1
定義	6.2
實作	6.3
範例 - Grove 模組	6.4

第7章: 進階範例	7
TCP伺服器連線	7.1
Web客戶端WuClass	7.2
Web伺服器端WuClass	7.3

Version History

Date	Version	Content
May 6, 2016	1	Initial version

- [第1章: 簡介] 振豪 90%
 - [如何使用這本書]
- [第2章: 設定伺服器環境] 博文 90%
- [第3章: 物聯網開發板] 博文 90%
 - [設定Intel Edison]
 - [設定Intel Galileo Gen 2]
 - [設定Raspberry Pi Board]
- [第4章: 範例] 君哲 90%
 - [電腦上的範例 - 音樂播放器]
 - [步驟1. Start Master and Gateway]
 - [步驟2. Include New Device]
 - [步驟3. Build an FBP]
 - [步驟4. Running the Application]
 - [Discovery Issues]
 - [物聯網開發板 - 控制LED]
 - [步驟1. Start Master and Gateway]
 - [步驟2. Include a New Device]
 - [步驟3. Build and Deploy FBP]
 - [步驟4. Running the Application]
- [第5章: 悟空Web介面] 俊翰 90%
 - [網路管理]
 - [裝置管理]
 - [應用程式管理]
 - [應用程式商店]
- [第6章: 動手寫WuClasses] 景棋 90%
 - [悟空Profile框架]
 - [定義]
 - [實作]

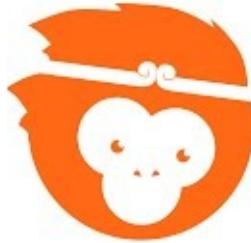
- [範例 - Grove 模組]
- [第7章: 進階範例] 元場 90%
 - [TCP伺服器連線]
 - [Web客戶端WuClass]
 - [Web伺服器端WuClass]

Terminology Translation:

English	中文
access	存取
Add device	新增裝置
API bytecodes	
attribute	元素
cached file	快取檔案
cloud	雲端
configure	設定
Debugger	除錯器
deploy	部署程式
discovery	尋找裝置
event-driven model	事件驅動模型
firmware	韌體
flow-based programming (FBP)	資料流編程
gateway	通訊閘
hardware-independent	不受硬體限制
HTTP	
Intel Edison	
Intel Galileo	
IoT	物聯網
IoT device	物聯網裝置
library	函式庫
link table	資料流表單
local	本地

location	位置
Local Area Network	區域網路
logical services	邏輯服務 / 純運算服務
map	
mapping	配對
master	主控台
middleware	中介軟體
NAT	網路地址轉換 / IP轉換
network prefix	網址前綴
node	裝置
parse	分析
physical board	實體開發版
port forwarding	通訊埠轉發
propagate	傳播
Property	屬性 / 參數
protocol	協定
public IP	公共IP
Raspberry Pi 2	
refresh method	刷新方法
refresh rates	刷新速度
remove device	移除裝置
router	路由器
sensor profile	傳感器資料
server	伺服器
source code	程式碼
subnet	內網
TCP socket	TCP封包
terminal	終端機
toolchains	工具鏈
update method	更新方法
URL	網址
virtual device	虛擬裝置

virtual software module	虛擬軟體元件
Web Client	網頁客戶端
Web Server	網頁伺服器
Web socket	網頁封包
WiFi	
WuClass	悟空類別
WuKong Profile Framework	悟空屬性架構
WuObjects	悟空物件
Zigbee	
ZWave	



悟空(WuKong)計畫是在[Intel-臺大創新研究中心](#)發展的專案，致力於開發物聯網的中介軟體(IoT Middleware)。

悟空計畫的目的是讓使用者設計、開發硬體獨立的物聯網應用，因此使用者能夠更容易的配置並動態部署他們的應用到各式各樣的物聯網平台。

悟空物聯網中介軟體有以下特點：

- 虛擬化物聯網裝置：虛擬化物聯網裝置使得應用程式能獨立於硬體層次，並且能夠簡化物聯網服務在不同裝置的遷移，而不需要重新設計應用程式。因此，使用者部署他們的應用到各式各樣的平台時，他們並不需要寫出與硬體、網路通訊相關的代碼。
- 基於流程的編程環境：悟空提供了資料流編程(flow-based programming)。在此編程環境，使用者可以定義資料流與流程控制來搭建物聯網應用。使用者只需要從悟空組件庫選擇所需的組件，拖曳至編程的畫布上，接著使用單向鏈接串起各組件。應用程式的資料流在部署階段對應與編譯至合適的服務或是硬體設施。
- 異質與虛擬服務：悟空中介軟體在不同的硬體平台上使用了虛擬機器來簡化應用的部署與遷移，也能夠實現虛擬化的物聯網服務(以Python實現)，來支援網路相關的資料服務或者使用者介面，以運行在微型裝置，電腦，或是智能手機。此外，基於Dajeerling的微型Java虛擬機也包含在悟空中介軟體中，因此系統能夠動態新增應用程式碼於微型裝置上。
- 部署時期服務映射：為了支援異質與日新月異的硬體平台，悟空將軟體上的物聯網服務與硬體上的物聯網裝置的綁定延後到了部署時期。因此在開發應用時，平台相關屬性與配置（如埠號分配、針腳配置）都可被簡化。與平台相關的屬性會在註冊至悟空系統時被悟空主服務蒐集，主服務接著會使用這些屬性來生成適當的配置進而生成物聯網服務的可執行代碼。

在這份手冊中，我們使用了指令、範例來展示如何配置一個悟空環境以及開發悟空物聯網應用。這份文件包含了所需的系統工具套件的說明、硬體與韌體配置、主服務器的安裝及操作、感測裝置連接、以及進階物聯網裝置的支援。所有會用到的硬體及軟體都是公開的。透過公開這項計畫，我們希望未來能夠使得物聯網更容易開發及部署。

1.1 如何使用這本書

這本書的設計是要帶領你：

- 配置悟空中介軟體與開發環境。
- 配置物聯網裝置到悟空系統上。
- 在悟空開發新的物聯網應用。

這本書分成七個章節：

- 第一章是悟空中介軟體及系統建置環境的介紹。
- 第二章及第三章展示了建置、配置軟件及硬體環境的步驟。
- 第四章給出兩個簡單的悟空運行的範例，其一沒用到實體上的物聯網裝置，另一則使用了一個物聯網裝置。
- 第五章簡述了悟空的開發及管理設施。
- 第六章與第七章展示了如實現及如何增加硬體設施(悟空類別)到你的專案中。

建議章節

這本書設計給三種讀者：裝置開發者，應用開發者，以及系統安裝者。不同的章節可能會對不同的讀者相對上比較實用。

以下是對於不同讀者所建議的閱讀內容：

- 裝置開發者: 2, 3, 6 (硬體相關悟空類別編寫)
- 應用開發者: 4.*.3, 5.2, 5.3
- 系統安裝者: 4.*.1, 4.*.2, 5.1 (主服務與網路配置)

目標硬體

悟空是個分散式中介軟體包含三個組件：主控台、通訊閘、以及物聯網裝置。各組件可以分別運行在不同或是共同的平台上。

- 主控台必須運行在Linux-Based的電腦，並且須具備網路連線及具有網路伺服器的能力。主控台是用來生成、傳送目標代碼給不同的物聯網裝置。建議以較具有運算能力的電腦運行主控台。
- 通訊閘是用來連接物聯網裝置並使用各式各樣的通訊協定轉發路封包給不同的內網。通訊閘可以運行在Linux-Based的伺服器，或是簡單的通訊閘機器。
- 物聯網裝置配備感測器及致動器，可以運行在不具有操作系統的小型開發板上。有些具有感測、控制能力物聯網開發板能會有開發商提供的實時操作系統或是嵌入式Linux。此外，悟空也能夠定義虛擬裝置運行在個人電腦或是伺服器來蒐集使用者輸入以及顯示使

用者介面或是媒介。

圖表1(a-c)展示悟空系統可能的配置：

- 可將悟空物聯網應用搭建成一個Linux-Based電腦，包含了主控台、通訊閘、虛擬裝置。

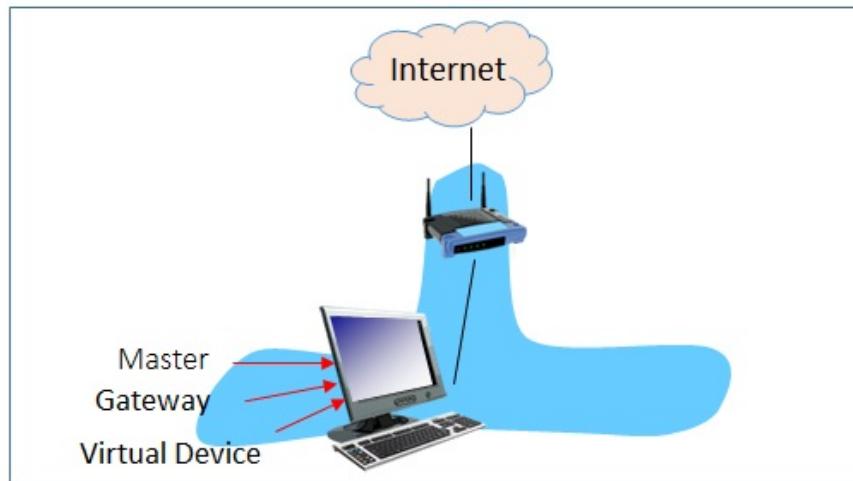


Fig 1a. A WuKong-based IoT application can be run using only a Linux-based computer, with Master, gateway, and virtual devices all on the same machine.

- 也能將應用運行於若干個實體物聯網裝置，並且使用一個Linux-Based電腦運行主控台、通訊閘。

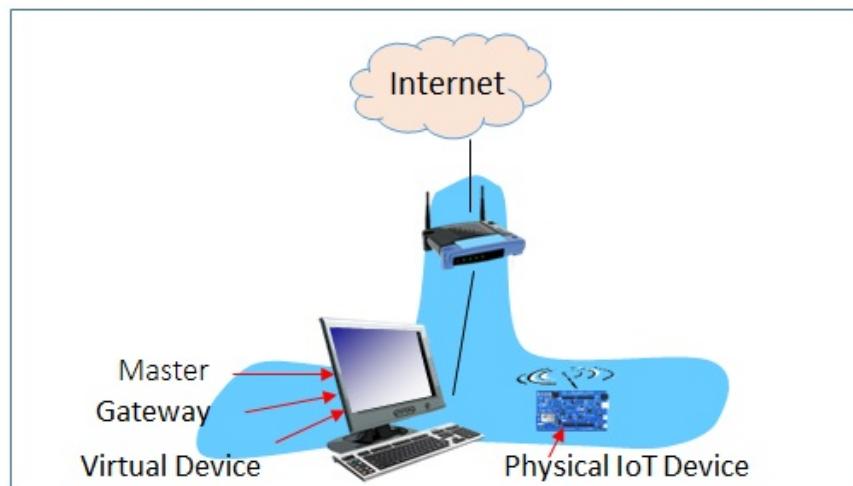


Fig 1b. WuKong-based IoT applications can be run on physical IoT devices connected to a Linux server running Master and gateway.

- 完整規模的悟空系統包含了獨立出來的主控台、通訊閘，通訊閘所負責的內網可以採用各式各樣的通訊協定（如WiFi, ZWave, Zigbee），有許多物聯網裝置運行在實體開發板，也會有軟體的模塊運行在伺服器。

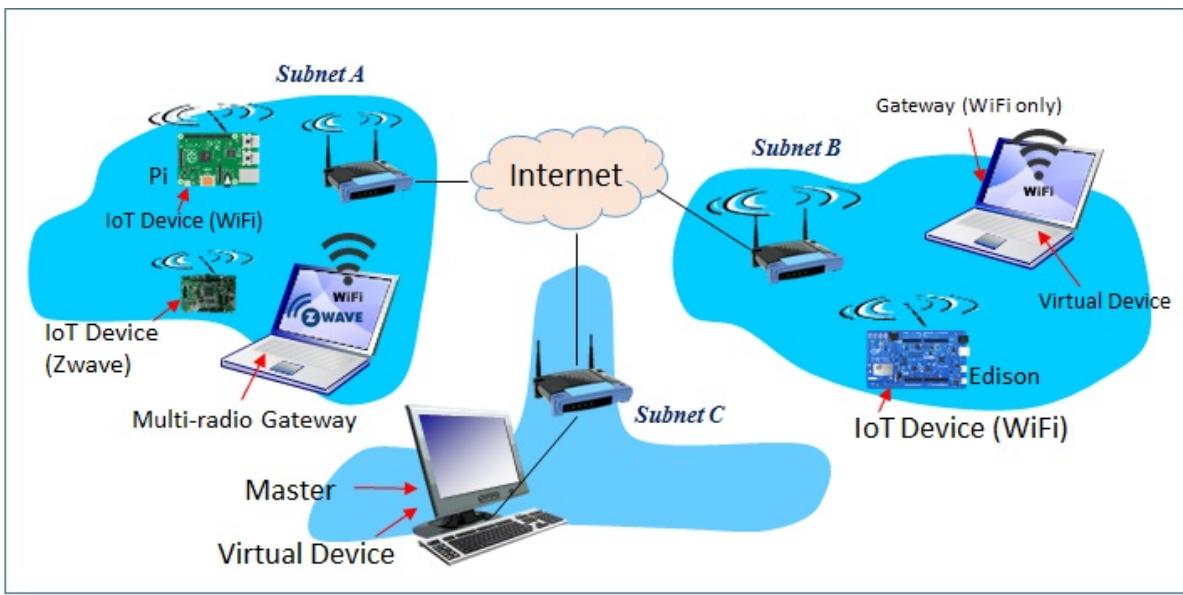


Fig 1c. A full scale WuKong system has a dedicated Master, several gateways each managing devices on different subnets or protocols (e.g. WiFi, ZWave, Zigbee), and IoT devices running on physical and/or virtual machines.

對於只使用Linux-Based電腦來運行悟空物聯網裝置的資訊，請閱讀2, 4.1, 5, 6。

對於使用Linux-Based電腦及物聯網開發板來運行悟空物聯網應用，請閱讀2, 3, 4.2, 5, 6。

第二章：設定伺服器環境

這份章節呈現出在Linux-based的電腦上安裝發展悟空環境所需要的指令，這些指令在Ubuntu 14.04 LTS的電腦上執行，我們計畫在未來提供OSX和Windows的安裝指南。在那之前，對於PC使用者可能的選項就是安裝虛擬機器並執行Ubuntu。

執行悟空需要四個工具，Git可以取得悟空專案的原始碼，Java執行主控台(Master)軟體，Gradle可以建立專案，而Python是用來實作悟空屬性架構(Wukong Profile Framework)，如果你的伺服器已經安裝上述的軟體，你可以跳過任何一步。

- 安裝 Git

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

- 安裝 Java

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

- 安裝 Gradle

下載Gradle2.4（連結）並且解壓縮到一個資料夾，例如：~/gradle-2.4，將Gradle的資料夾路徑加到環境變數，如下表示

```
vim ~/.bashrc
```

加環境變數

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

添加環境變數在（bash startup）檔案中的最後一行，並使用source的指令去執行腳本(script)

```
source ~/.bashrc
```

- 安裝 Python 工具

```
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
```

在下一章節，我們會呈現如何安裝物聯網裝置(例如Intel Edison, Galileo,樹莓派(Raspberry Pi2))，來實作出可以感測和控制環境裝置的應用程式(Application)。如果目前沒有上述的物聯網裝置，只要使用個人電腦和這章節所介紹的工具套件，就可以發展悟空的物聯網應用程式，範例將在[4.1節](#)展示。

第三章：物聯網開發板

現在市面上已經可以買到各式各樣的物聯網開發版以及裝置，每個物聯網裝置或感應器開發版都必須透過悟空中介軟體進行事前的設定，才能讓悟空主控台管理悟空裝置。

我們會在此章節中呈現如何安裝三種常見並容易取得的板子，分別為：

- [Intel Edison](#)
- [Intel Galileo Gen2](#)
- [Raspberry Pi 2](#)

3.1 設定Intel Edison

安裝Intel edison成悟空裝置有四個主要步驟

步驟一 開發版組裝: 在Arduino擴充開發版上組裝Intel edison模組

步驟二 開發版設定: 設定Intel edison的密碼，以便我們可以用SSH指令連線到開發版

步驟三 韌體更新: 獲得相容性的Yocto 映像檔，目前版本為159.devkit，可以支援更大的root 分割容量 以及更好的Python工具

步驟四 Python 工具安裝: 安裝執行悟空所必備的Python函式庫

步驟五 下載悟空原始碼: 從Github上複製悟空的原始碼到Edison開發版

- 步驟一：開發版組裝

Intel提供了兩種擴充開發版給Intel Edison所使用，分別為:Arduino 擴充開發版和mini breakout board，在此章節中，我們使用Arduino擴充開發版，不須焊接就可以很方便的連接組合包中的感應器。

請依照下列網頁的指示來組裝Intel Edison 開發版：

<https://software.intel.com/en-us/assembling-intel-edison-board-with-arduino-expansion-board>

組裝完後，再依照下列網頁的指示來設定序列終端機，以便連線到開發版：

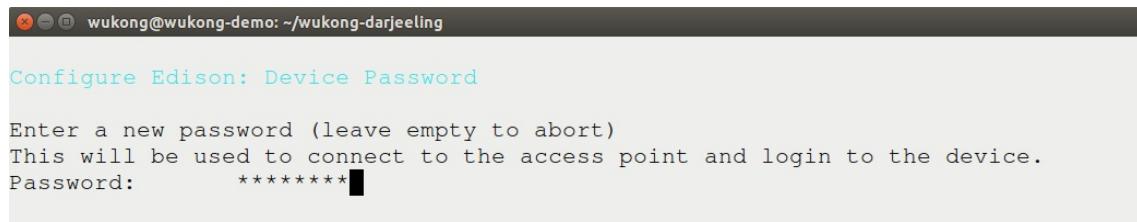
<https://software.intel.com/en-us/setting-up-serial-terminal-intel-edison-board>

- 步驟二：開發版設定

- 在終端機上輸入下方的指令去設定開發版

```
configure_edison --setup
```

- 終端機出現提示時，你必須輸入密碼，以便我們可以透過SSH的指令連線到開發版



The screenshot shows a terminal window titled "wukong@wukong-demo: ~/wukong-darjeeling". It displays the command "Configure Edison: Device Password". The user is prompted to enter a new password, stating "Enter a new password (leave empty to abort)". It also mentions that the password will be used for connecting to the access point and logging into the device. The password is entered as a series of asterisks ("*****") at the prompt.

- 終端機出現提示時，你可以替開發版取名字；或者空白，則 "root" 將會成為預設的名字

```
wukong@wukong-demo: ~/wukong-darjeeling
Configure Edison: Device Name
Give this Edison a unique name.
This will be used for the access point SSID and mDNS address.
Make it at least five characters long (leave empty to skip): █
```

- 下一步是設定WiFi，輸入"Y"來設定WiFi，而開發版會開始掃描WiFi熱點，一旦開發版完成掃描，可選擇一個WiFi熱點，並輸入密碼。下方圖片為設定WiFi的例子。

```
wukong@wukong-demo: ~/wukong-darjeeling
8 :      network SSID
9 :      network SSID
10 :     network SSID
11 :     network SSID
12 :     network SSID
13 :     network SSID
14 :     network SSID
15 :     network SSID
16 :     network SSID
17 :     network SSID
18 :     network SSID
19 :     network SSID
20 :     network SSID
21 :     network SSID
22 :     network SSID
23 :     network SSID
24 :     SSID

Enter 0 to rescan for networks.
Enter 1 to exit.
Enter 2 to input a hidden network SSID.
Enter a number between 3 to 24 to choose one of the listed network SSIDs: 24
Is SSID correct? [Y or N]: Y
Password must be between 8 and 63 characters.
What is the network password?: *****
Initiating connection to SSID. Please wait...
Attempting to enable network access, please check 'wpa_cli status' after a minute to confirm.
Done. Please connect your laptop or PC to the same network as this device and go to http://192.168.4.180 or http://edison.local in your browser.
root@edison:~# █
```

- 假如你需要連別的WiFi熱點，你可以使用下方的指令來重新設定WiFi。

```
configure_edison --wifi
```

• 步驟三：韌體更新

請確保開發版的韌體為最新版本，你可以使用下方的指令來確認版本：

```
configure_edison --version
```

如果螢幕秀出 `159.devkit`，則此版本可兼容，反之，請依照下列網頁來更新韌體：
<https://software.intel.com/en-us/iot/hardware/edison/downloads>

- 步驟四：Python 工具安裝

- 複製下方的指令到 `/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  
src/gz core2-32 http://repo.opkg.net/edison/repo/core2-32
```

- 安裝 `pip` 和其他工具

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

- 步驟五 下載悟空原始碼

- 複製原始碼並下載到開發版上，放在你想要的資料夾路徑

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

3.2 設定Intel Galileo Gen2

安裝Intel Galileo Gen2成悟空裝置有四個步驟

步驟一 準備**Linux** 映像檔：準備可燒錄Linux 映像檔的microSD卡

步驟二 開發版組裝：組裝Intel Galileo 開發版

步驟三 開發版設定：設定Intel Galileo的密碼，以便我們可以用SSH指令連線到開發版

步驟四 **Python** 工具安裝：安裝執行悟空所必備的Python函式庫

步驟五 下載悟空原始碼：從Github上複製悟空的原始碼到Galileo開發版

- 步驟一 準備**Linux** 映像檔

Galileo Gen2不像Edison，沒有內建的快閃記憶體，所以我們需要準備可燒錄的microSD卡。

請依照下列網頁的指示來燒錄microSD卡，當成開機的SD卡

<https://software.intel.com/en-us/get-started-galileo-linux-step1>

- 步驟二 開發版組裝

根據Intel 官方物聯網的網頁，需要的硬體有

- Intel Galileo Gen 2,
- 一片SD卡(>2GB),
- 6 pin序列到Type A USB (例如：FTDI #TTL-232R-3V3)
- 一個7-15伏特並至少有1500毫安培的電壓器
- (可選擇)WiFi網路卡 (例如: Intel® Centrino® Wireless-N 135 or Intel® Centrino® Advanced –N 6205)

準備好硬體後，請依照下列網頁的指示來組裝Intel Galileo Gen 2開發版：

<https://software.intel.com/en-us/get-started-galileo-linux-step2>

組裝完後，再依照下列網頁的指示來設定序列終端機，以便連線到開發版：

<https://software.intel.com/en-us/get-started-galileo-linux-step3>

- 步驟三 開發版設定

如果在步驟二你有推薦的WiFi模組，依照下列網頁的指示來安裝WiFi網卡並設定WiFi
<https://software.intel.com/en-us/get-started-galileo-linux-step4>

假如沒有，依照下列網頁的指示來設定乙太網路連線，不需要序列終端機，可以使用Arduino IDE來和開發版溝通。

<https://software.intel.com/en-us/articles/intel-galileo-getting-started-ethernet>

- 步驟四：Python 工具安裝

- 複製下方的指令到 `/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
src/gz core2-32 http://repo.opkg.net/edison/repo/core2-32
```

- 安裝 **pip** 和其他工具

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

- 步驟五 下載悟空原始碼

- 複製原始碼並下載到開發版上，放在你想要的資料夾路徑

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

3.3 設定Raspberry Pi Board

安裝Raspberry Pi成悟空裝置有四個主要步驟

步驟一 準備 **Raspbian Linux** 映像檔：準備可燒錄Raspbian Linux 映像檔的microSD卡

步驟二 開發版組裝：組裝Raspberry Pi board及相關的配件

步驟三 開發版設定：設定Raspberry Pi的密碼

步驟四 **Python** 工具安裝：安裝執行悟空所必備的Python函式庫

步驟五 下載悟空原始碼：從Github上複製悟空的原始碼到Raspberry Pi Board

- 步驟一 準備**Raspbian Linux**映像檔

請依照下列網頁的指示來安裝NOOBS:

<https://www.raspberrypi.org/help/noobs-setup/>

我們也推薦到官方網站下載映像檔，請依照下列網頁的指示來燒錄映像檔到SD卡，變成可開機的SD卡。

<https://www.raspberrypi.org/documentation/installation/installing-images/README.md>

預設放入映像檔的SD卡容量為4GB，如果你需要更多的空間，請依照官方網頁的解決方式。

<https://www.raspberrypi.org/documentation/configuration/raspi-config.md>

- **Step 2.** 開發版組裝

需要的硬體包含：

- Raspberry Pi Board
- 一片 SD 卡 (>4GB)
- HDMI cable
- 一個7-15伏特並至少有1500毫安培的電壓器
- (可選擇) USB WiFi 網卡
- (可許澤) Grove Pi 擴充開發板

你可以觀看下列影片來組裝Raspberry Pi

<https://www.youtube.com/watch?v=JiTNdwD1fS0>

- **Step 3.** 開發版設定

一旦RPi啓動後，你可以使用圖形化介面或透過SSH指令登入，帳號為 `pi`，密碼為 `raspberry`

- 在終端機輸入指令來改變鍵盤佈局

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

在檔案中找到下列這行：

```
XKBLAYOUT="gb"
```

將`gb`改成你所在的國家代碼，為兩個字母，例如 "`us`"。輸入`Ctrl+X` 儲存並關閉檔案

假如你不知道你的國家代碼，可以到下列的維基百科網頁搜尋目前的國家代碼（使用第二行:`alpha-2` 的代碼）

[Here](#)

- 變更Raspbian的更新套件來源

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

請從下方的官方網站上，

<https://www.raspbian.org/RaspbianMirrors>

找到最近的伺服器連結，去取代檔案中的下方連結

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

並使用下列指令進行更新。

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

- 步驟四：**Python** 工具安裝

- Python工具安裝：安裝 `pip` 和其他工具

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

- (可選)在終端機輸入指令，安裝grovepi函式庫來啓動I2C對Grove pi擴充開發版的介面

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

一旦表單出現，使用方向鍵和Enter鍵選擇**Advanced Options**，以及**I2C**，選擇**YES**去啓動**I2C**介面，以及**YES**去啓動預設模組，最後重開RPi

```
sudo reboot
```

重開後，如果你有Pi Model A, B+ 或 2 B(記憶體：512 MB)，在終端機上使用下列指令去檢查I2C是否有開啓

```
sudo i2cdetect -y 1
```

如果為Pi 1 B (記憶體：256MB)，則執行下列指令

```
sudo i2cdetect -y 0
```

在表格中應該會顯示一些位址如20,40 或70

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

- 步驟五 下載悟空原始碼

- 複製原始碼並下載到開發版上，放在你想要的資料夾路徑。

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

Chapter 4: 悟空部署範例

此章節會用兩個範例來說明如何在悟空系統下部署物聯網的環境。第一個範例的應用程式只需使用個人電腦，而不會用到任何開發版。第二個範例則會包含如何在 Intel Edison 及 Raspberry Pi 開發版上建立悟空的應用程式。兩個範例中的應用程式在運行階段皆只需要在區域網路內執行，而不需要連接到網際網路。

4.1 音樂撥放器

因為悟空的主控台、通訊閘、裝置程式皆可以在個人電腦上執行，所以我們的第一個範例只會使用到電腦上的軟體元件。

此範例包含四個步驟：

步驟 1. 啓動主控台及通訊閘程式

步驟 2. 新增裝置

步驟 3. 撰寫資料流程式

步驟 4. 程式執行

4.1.1 啓動主控台及通訊閘程式

1. 請先輸入下列指令從 `github` 下載程式碼到您的電腦

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

2. 建立 `infuser`

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

```
wukong@wukong-demo:~$ cd wukong-darjeeling/src/infuser/
wukong@wukong-demo:~/wukong-darjeeling/src/infuser$ gradle
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE

BUILD SUCCESSFUL

Total time: 5.114 secs

This build could be faster, please consider using the Gradle Daemon: http://gradle.org/docs/2.4/userguide/gradle_daemon.html
wukong@wukong-demo:~/wukong-darjeeling/src/infuser$ █
```

3. 複製主控台設定

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

4. 執行主控台程式

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

```
wukong@wukong-demo:~$ cd ~/wukong-darjeeling/wukong/master/
wukong@wukong-demo:~/wukong-darjeeling/wukong/master$ python master_server.py
[I 160430 12:06:53 master_server:1659] Starting up...
[I 160430 12:06:53 master_server:107] setting up signal handler
Scanning /home/wukong/wukong-darjeeling/wukong/master/../../wukong/ComponentDefinitions/WuKongStandardLibrary.xml
creating wutypes
Scanning classes & properties
[I 160430 12:06:53 master_server:186] updating applications:
[I 160430 12:06:53 master_server:195] scanning d70c1e5d44de8a9150eb91ecff563578:
[I 160430 12:06:53 master_server:197] d70c1e5d44de8a9150eb91ecff563578
[I 160430 12:06:53 master_server:195] scanning 7d5c009e4eb8bbc78647caeca308e61b:
[I 160430 12:06:53 master_server:197] 7d5c009e4eb8bbc78647caeca308e61b
[I 160430 12:06:53 master_server:195] scanning 8b04d5e3775d298e78455efc5ca404d5:
[I 160430 12:06:53 master_server:197] 8b04d5e3775d298e78455efc5ca404d5
[I 160430 12:06:53 master_server:195] scanning 72ab8af56bddab33b269c5964b26620a:
[I 160430 12:06:53 master_server:197] 72ab8af56bddab33b269c5964b26620a
[I 160430 12:06:53 make_js:33] make_js_complete
[I 160430 12:06:53 make_js:38] make manifest to /home/wukong/wukong-darjeeling/wukong/master/static/js/components_list.txt
[I 160430 12:06:53 make_fbp:149] make_fbp_complete
[transport] BrokerAgent init
[I 160430 12:06:54 transportv3:471] TCP server listens on ('0.0.0.0', 9010)
```

5. 開一個新的終端機，並且複製通訊閘程式的設定

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

6. 編輯 gtwconfig.py 檔案

```
# 使用 ifconfig 指令來獲得網路資訊
ifconfig
```

```
wukong@wukong-demo:~/wukong-darjeeling/wukong/gateway$ ifconfig
eth0      Link encap:Ethernet HWaddr c0:3f:d5:b3:e5:41
          inet addr:192.168.4.17 Bcast:192.168.4.255 Mask:255.255.255.0
          inet6 addr: fe80::c23f:d5ff:feb3:e541/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:6644608 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4567148 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3264710795 (3.2 GB) TX bytes:2711422594 (2.7 GB)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:1924113 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1924113 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:7603118323 (7.6 GB) TX bytes:7603118323 (7.6 GB)

wukong@wukong-demo:~/wukong-darjeeling/wukong/gateway$
```

```
vim gtwconfig.py
# 在 MASTER_IP 位置填入主控台的IP
# 在 TRANSPORT_INTERFACE_ADDR 填入根據上面方法取得的網路介面卡編號
```

```
# 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
```

7. 執行通訊閘程式

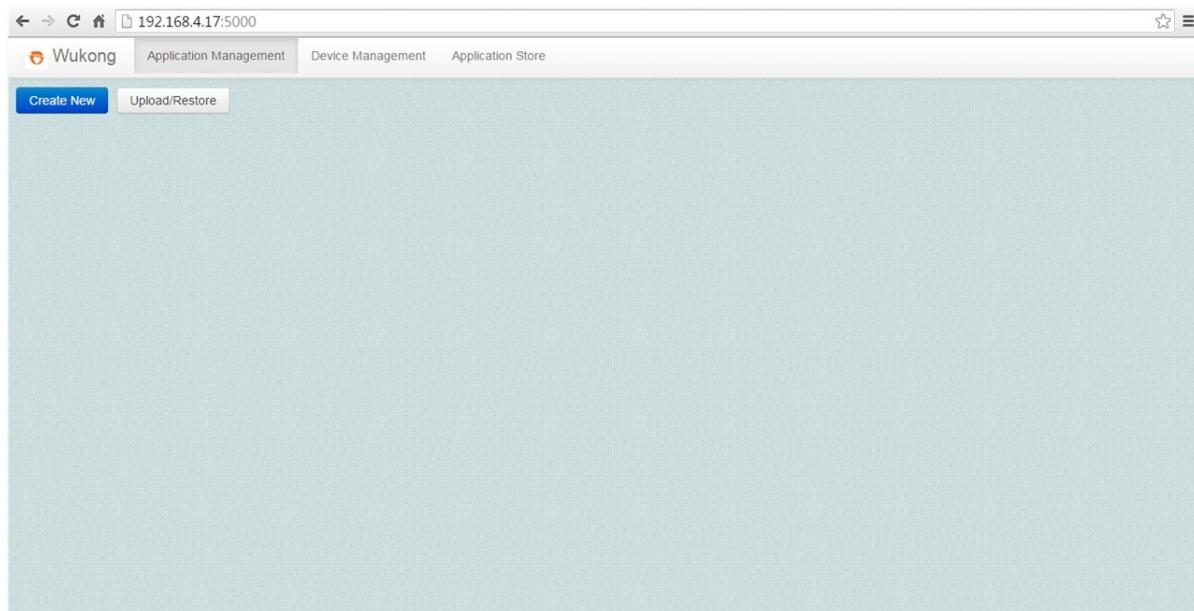
```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/
python start_gateway.py
```

```
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) peer
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 新增裝置

在我們利用悟空部署物聯網程式前，主控台必須先找到能夠執行此一程式的裝置。在這一個章節，我們會說明如何使用主控台上的裝置管理功能。

1. 使用 Chrome 瀏覽器開啟主控台介面：<http://localhost:5000>



2. 點選 **Device Management** 分頁並點擊 **Discover Node** 按鈕來獲得初始狀態。

Gateway 192.168.4.17 tcp_addr=('192.168.4.17', 9001) STOP

#	Location	WuClass	WuObject
1	/WuKong	0	1

3. 點擊主控台上的 **Add Node** 按鈕來新增裝置。

#	Location	WuClass	WuObject
1	/WuKong	0	1

注意：如果沒有顯示出 **ready to ADD** 的訊息的話請點擊 **Stop to complete operation** 按鈕並重試一次。

#	Location	WuClass	WuObject
1	/WuKong	0	1

4. 開啓一個新的終端機並且移至裝置程式所在的資料夾下。

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

5. 執行裝置程式 `udpdevice_intel_sound.py`。這個程式會啓動一個包含 `Intel_Sound` 哲空類別的裝置並且從通訊閘取得一個裝置識別碼。

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

注意：當裝置程式啓動後，裝置會從主控台取的一個裝置識別碼。

```
wukong@wukong-demo:~/wukong-darjeeling/wukong/gateway/udpwkpf$ python udpdevice_intel_sound.py 192.168.4.17 192.168.4.17:3000
Your ID is 3232236545 of which dotted format is 192.168.4.1
```

6. 點擊 **Stop to complete operation** 按鈕。

#	Location	WuClass	WuObject
1	/WuKong	0	1

7. 點擊 **Discover Nodes** 按鈕以刷新裝置列表。

#	Location	WuClass	WuObject
3232236545	Default	0	1
1	/WuKong	0	1

8. 點擊 **Details** 按鈕即可查看此裝置得傳感器資料。

Node Info

WuClasses:

WuObjects:

- Class Name: Intel_Sound

Port Number: 1

WuObject

WuObject
1

9. 裝置的位置可以用下列方法設定。

The screenshot shows the Wukong Node Editor interface. At the top, there are tabs for Application Management, Device Management, and Application Store. Below that, there are sub-tabs: Nodes Editor (which is selected and highlighted in blue), Location Editor, and Landmark Editor. A message box at the top displays: "Gateway 192.168.4.17 tcp_addr=('192.168.4.17', 9001) STOP found node: 1 (ID is 192.168.4.1 or 3232236545)". Below the tabs, there is a header with buttons: Discover Nodes, Add Node, Remove Node, Change Location Name, Click to Save Location, Find Location, and Set Location. The main table has columns: #, Location, WuClass, and WuObject. There are two rows: one for node ID 3232236545 with location '/Livingroom' and another for node ID 1 with location '/WuKong'. Red boxes highlight the 'Location' input field for the first row and the 'Set Location' button. Yellow arrows point from the text 'Change Location Name' and 'Click to Save Location' to these highlighted areas.

#	Location	WuClass	WuObject			
3232236545	/Livingroom	Find Location	Set Location	0	1	<button>Details</button>
1	/WuKong	Find Location	Set Location	0	1	<button>Details</button>

10. 重複步驟 3 至步驟 9 執行另一個名為 `upddevice_logic.py` 的裝置程式。之後您會在裝置管理介面看到下面的畫面。

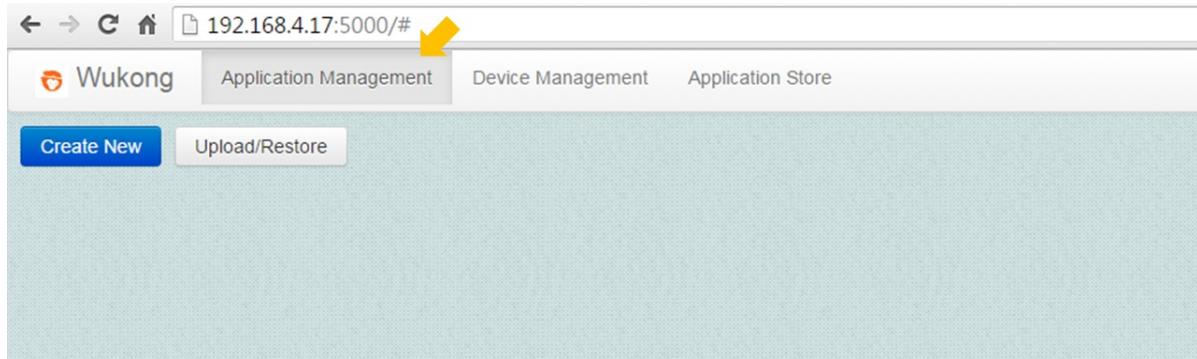
This screenshot shows the Wukong Node Editor after adding a new device. The interface is identical to the previous one, but the table now includes a third row for the newly added device. The table columns are: #, Location, WuClass, and WuObject. The new row for node ID 3232236546 has a location of '/Door'. The 'Set Location' button for this row is also highlighted with a red box and a yellow arrow.

#	Location	WuClass	WuObject			
3232236545	/Livingroom	Find Location	Set Location	0	1	<button>Details</button>
3232236546	/Door	Find Location	Set Location	2	0	<button>Details</button>
1	/WuKong	Find Location	Set Location	0	1	<button>Details</button>

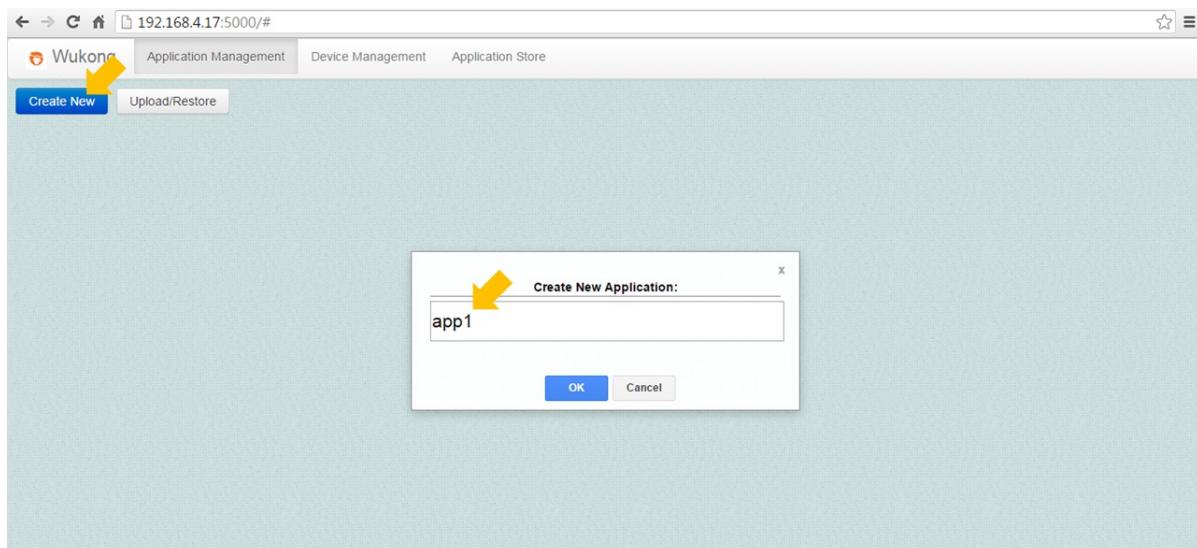
4.1.3 Build an Application FBP

現在我們準備好可以撰寫一個撥放音樂的程式了。

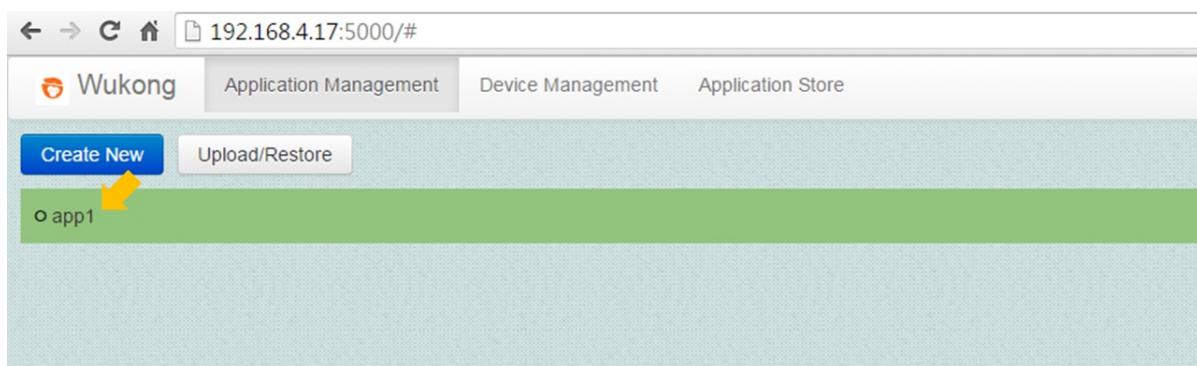
- 點擊 **Application Management** 標籤。



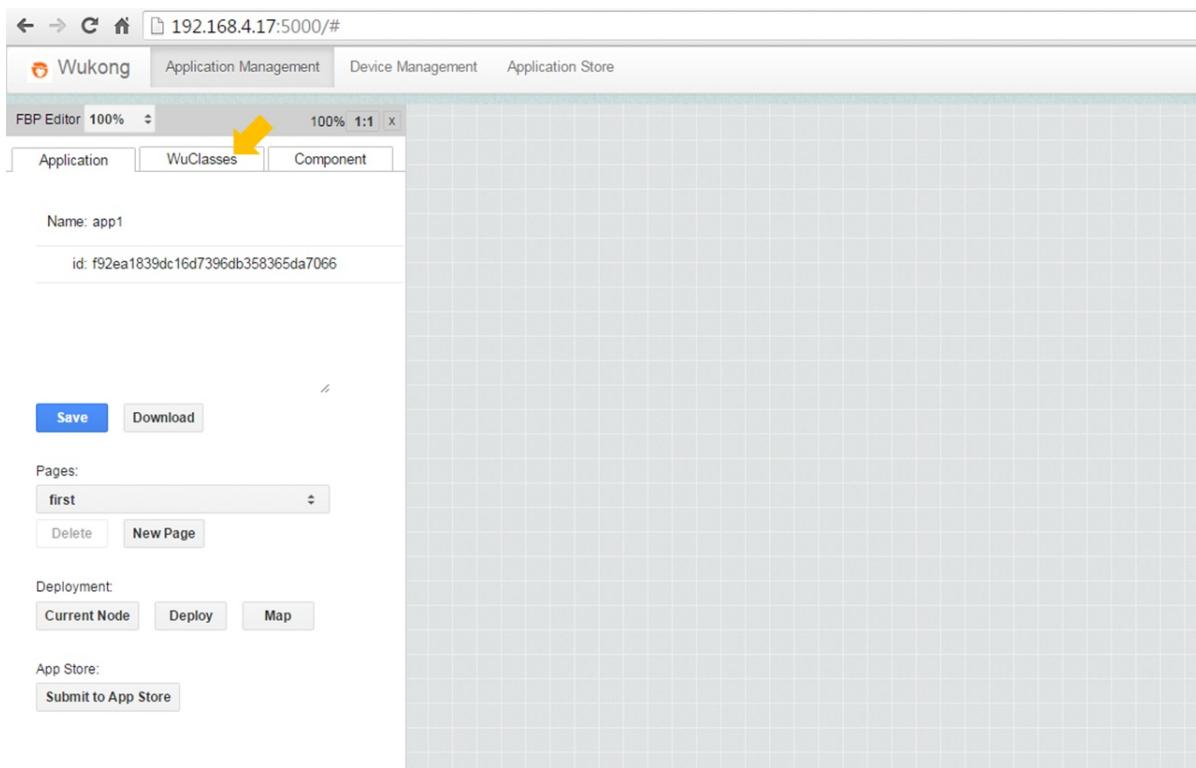
- 點擊 **Create New** 按鈕並且輸入任意的程式名稱，我們的範例使用 "app1" 為例子。



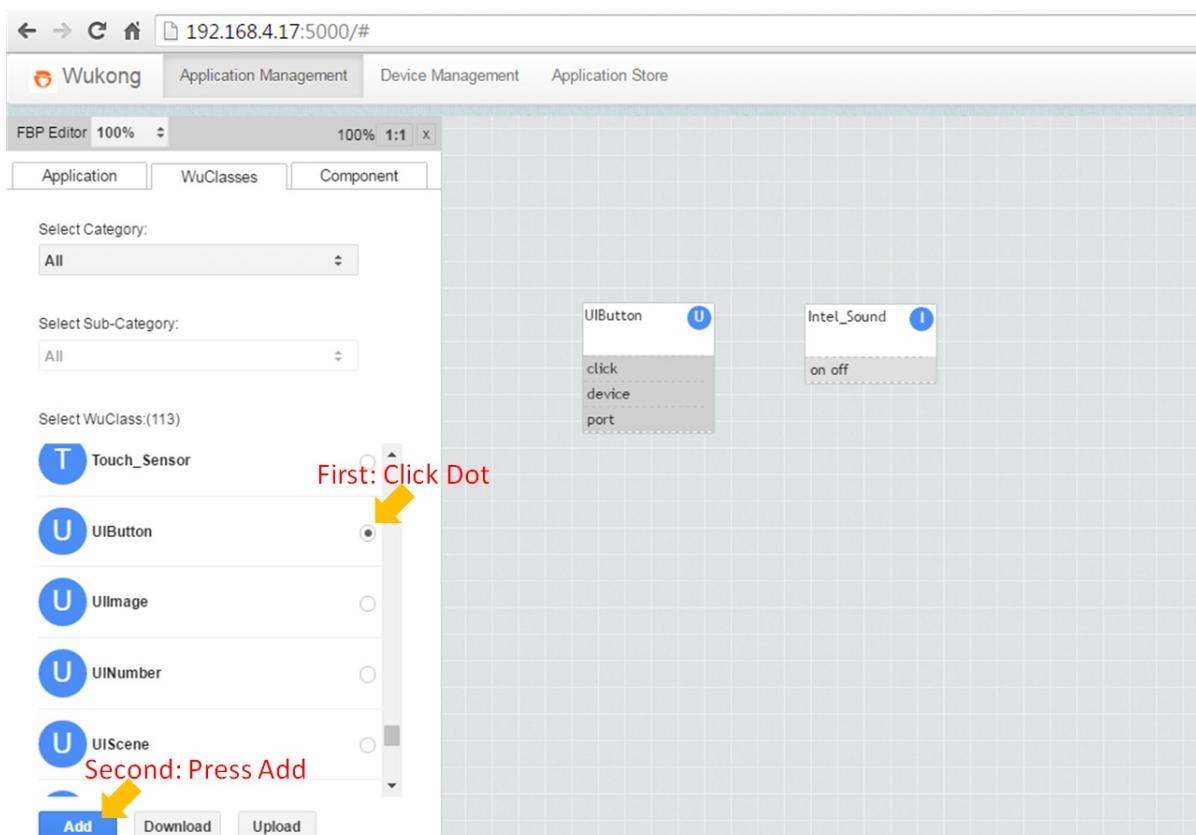
- 點擊剛才建立的程式名稱，進入編輯畫面。



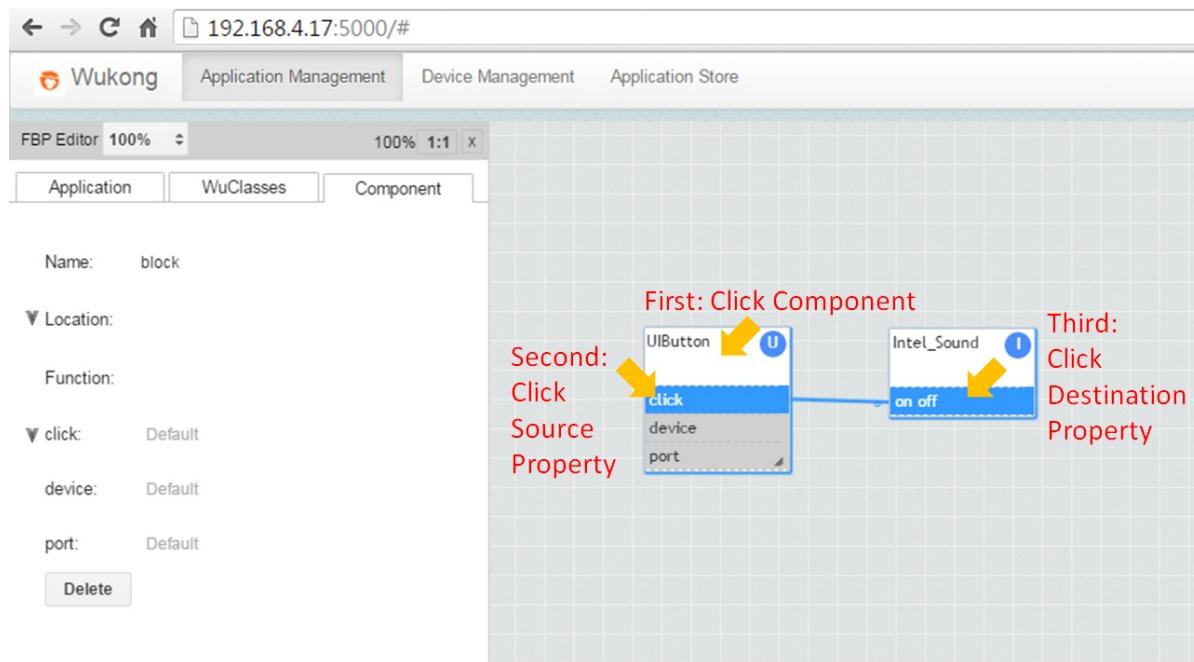
- 點擊 **WuClasses** 標籤，此時會顯示出所有可以在物件流編程時所使用的悟空類別。



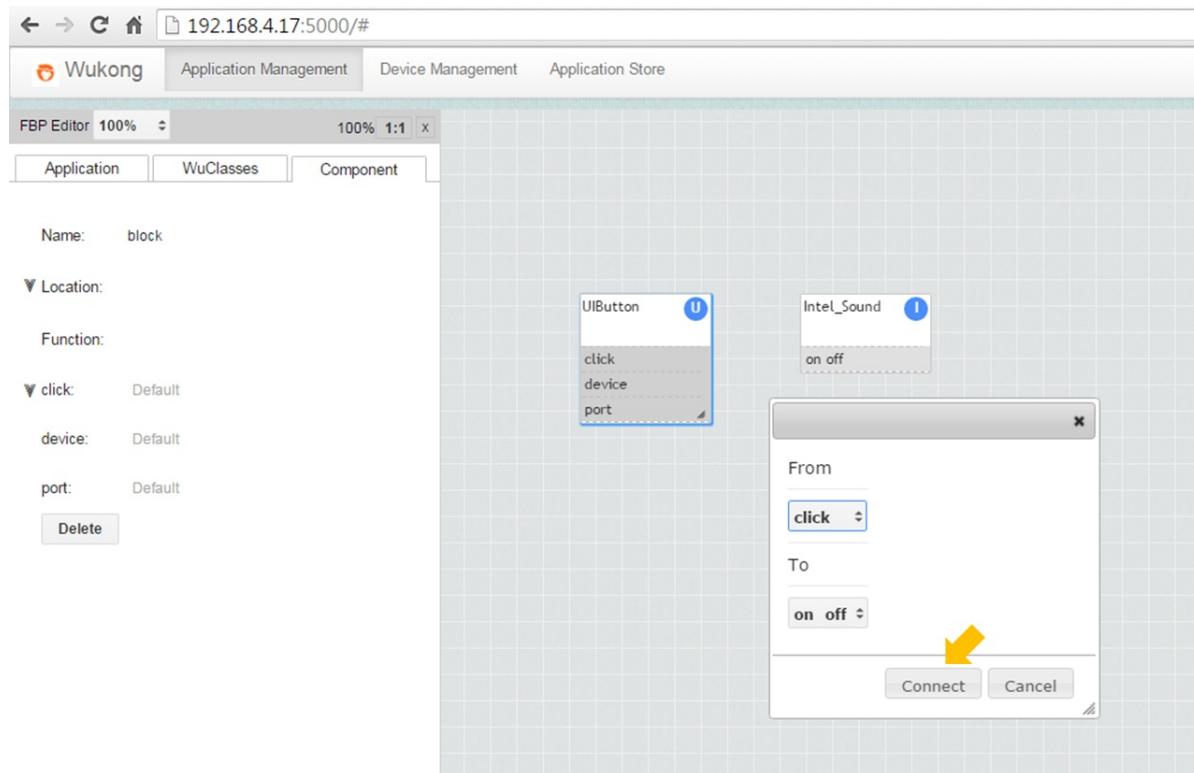
5. 滾動此列表直到找到名為 `UIButton` 的元件。點擊該元件即可選取，之後點擊下方的 **Add** 按鈕。重複此一步驟加入 `Intel_Sound` 元件。現在，編輯畫面中會出現兩個元件方塊。



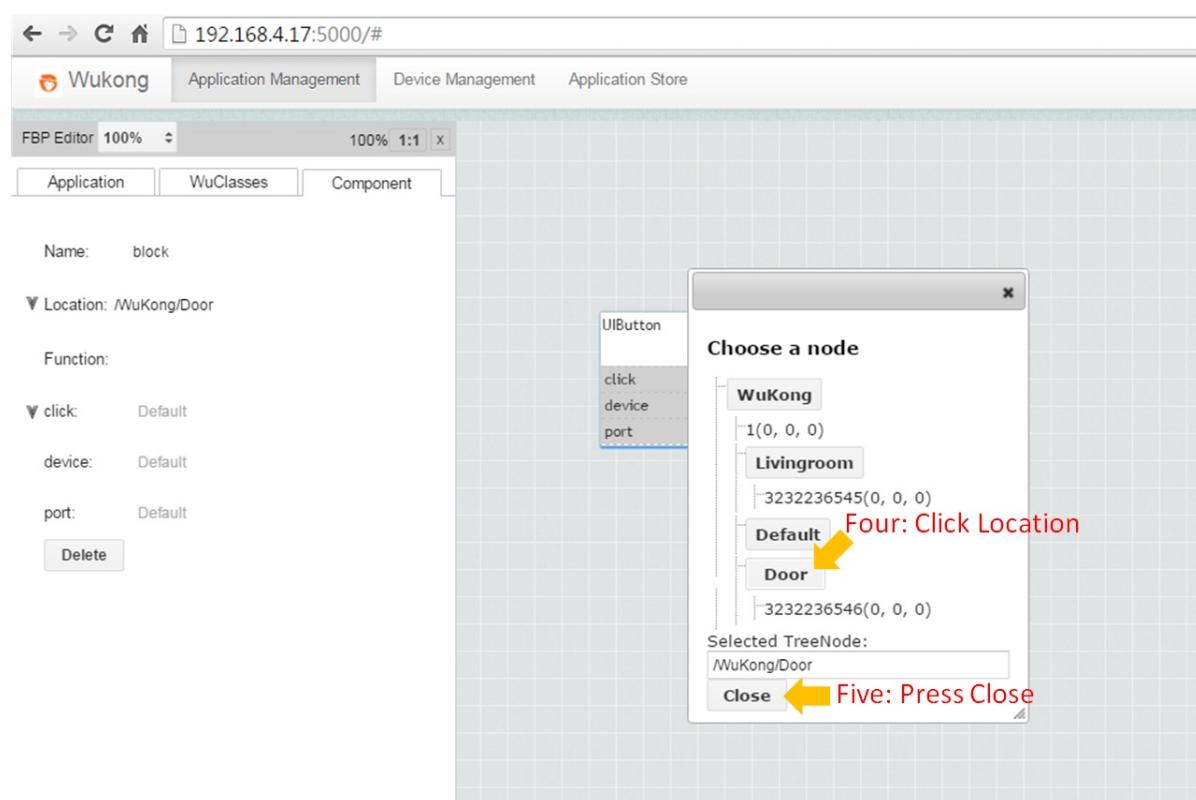
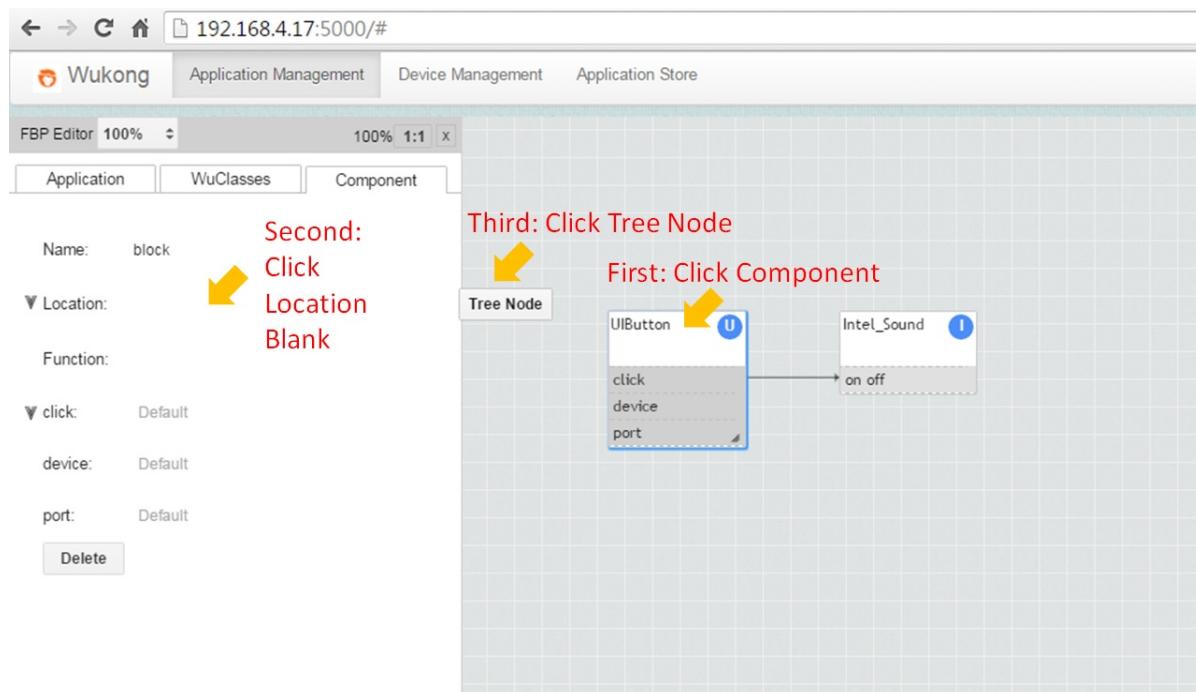
6. 在兩個元件之間新增資料流。首先，點擊來源元件，該元件會被框起。之後，點擊被選取元件的目標參數，如範例中的 "click" 參數。最後，移動游標到目標元件，並點擊。



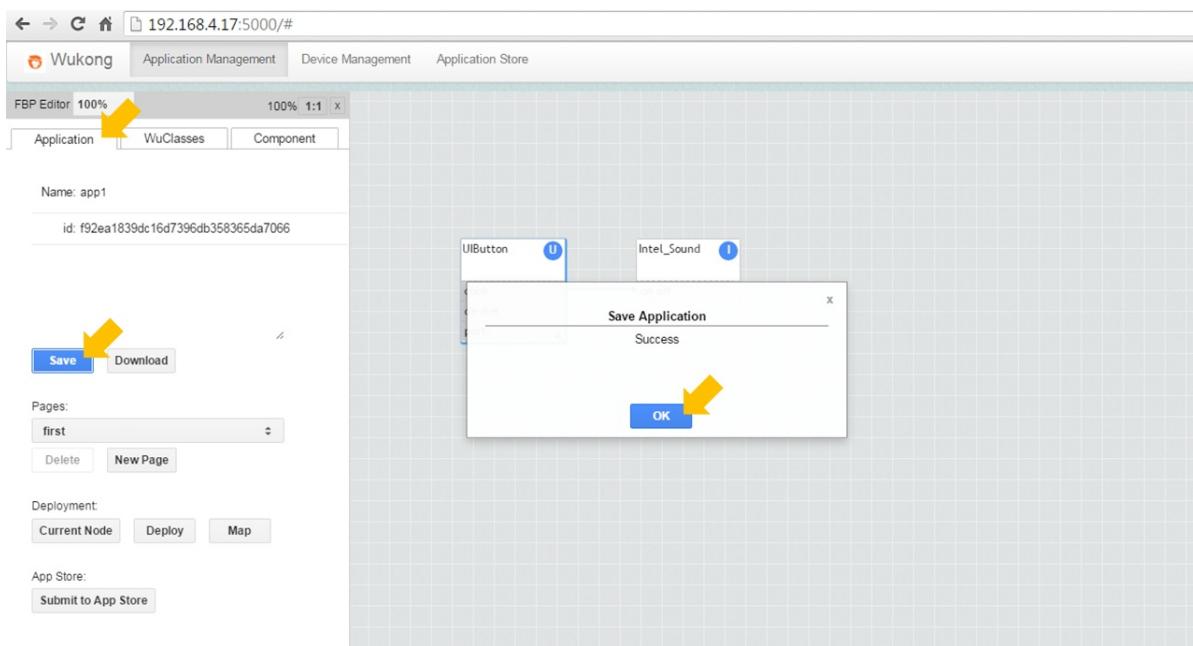
7. 此時，你可以確認所新增的資料流的來源與目標是否正確。如果有錯誤，可以在下拉式選單中做修正。



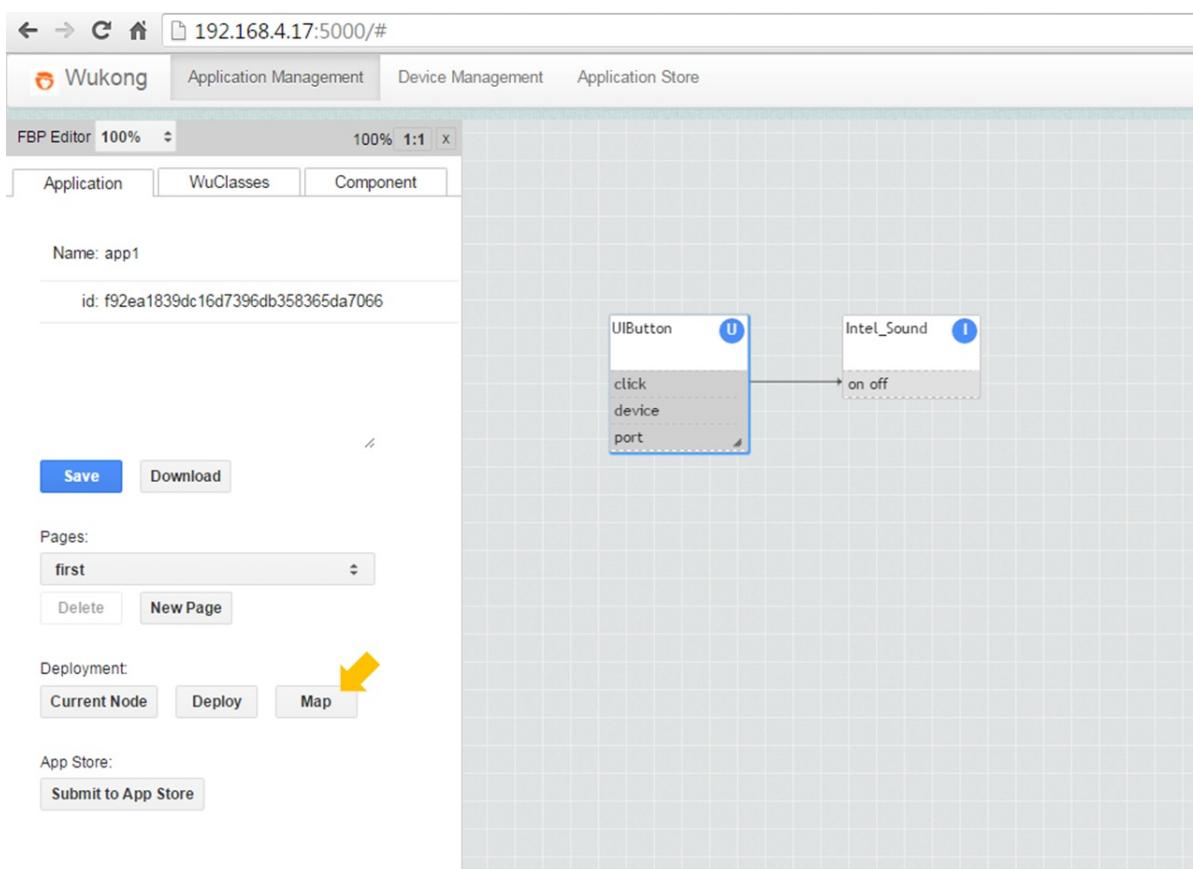
8. (非必須) 依照下列圖片中的5個步驟輸入元件所在位置。即使位置為空白，主控台仍然會依照每個裝置的傳感器資料來挑選合適的裝置。



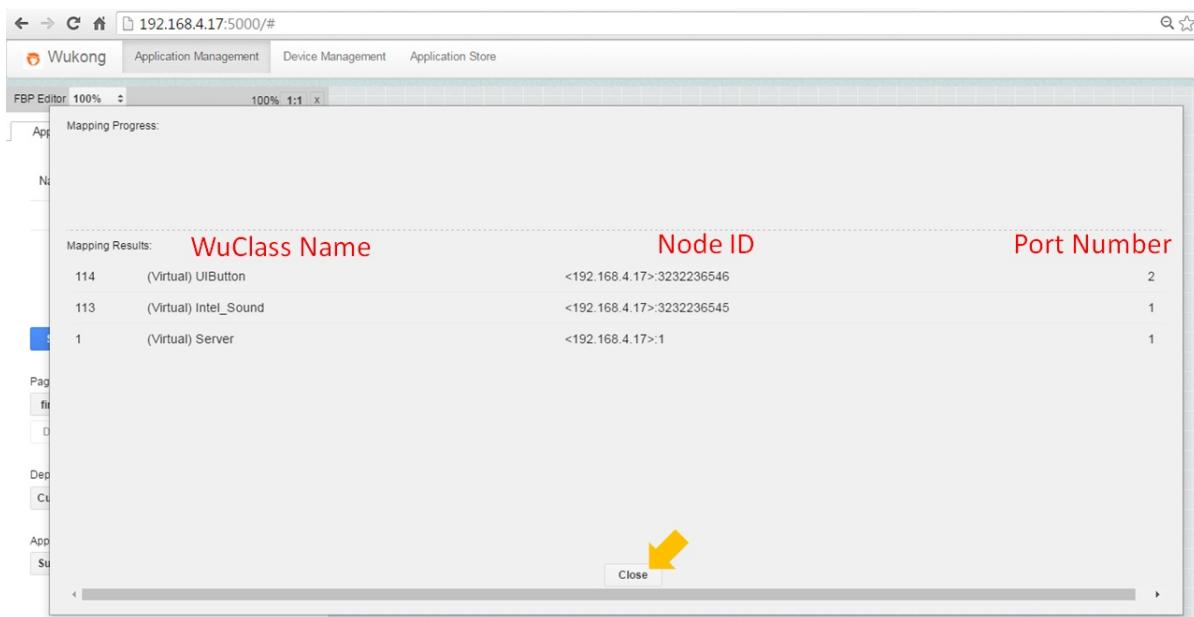
9. 點擊 Application 標籤頁中的 Save 按鈕。



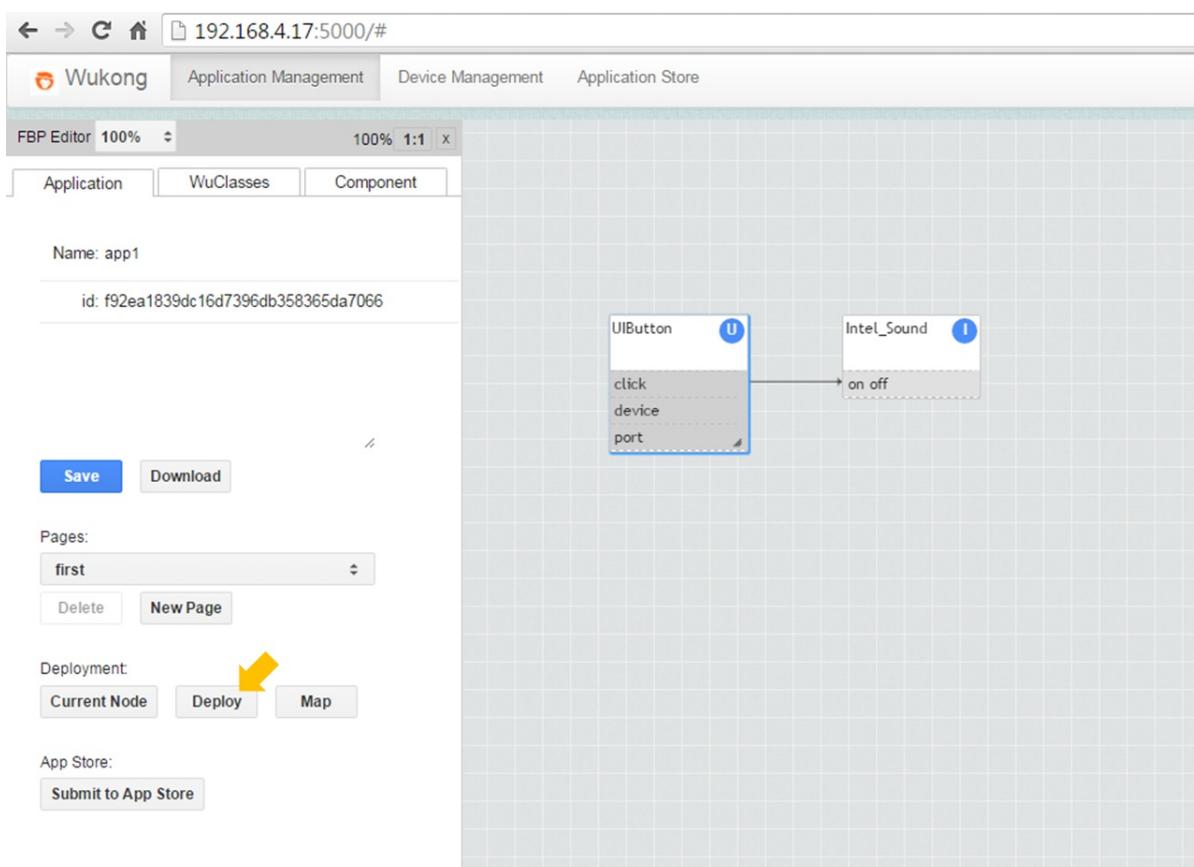
10. 點擊 **Map** 按鈕以為每個元件挑選適合的執行裝置。



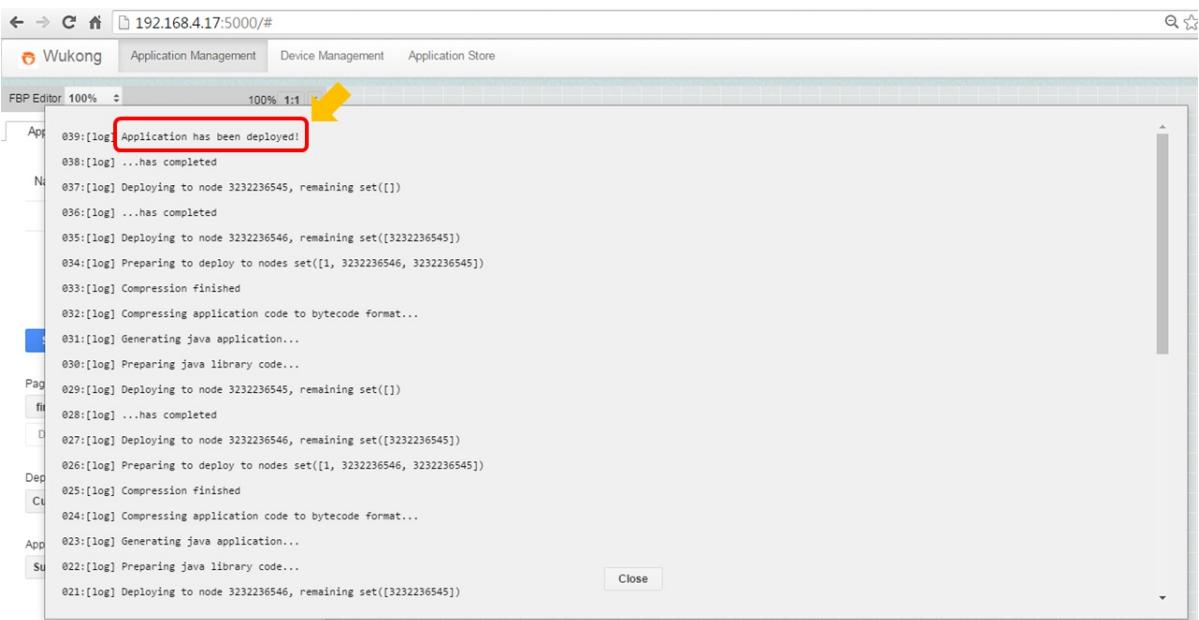
11. 等待配對完成後，配對結果會顯示在畫面上。



12. 如果配對結果沒有錯誤，你可以點擊 Deploy 按鈕以開始部署程式。

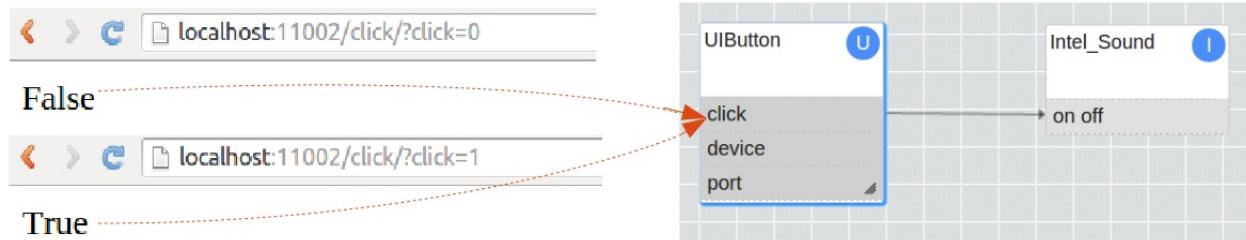


13. 您可以透過顯示在主控台上的訊息得知部署是否成功。



4.1.4 執行程式

等待部署完成後，我們可以透過發送 HTTP 封包給 UIButton 的方式來改變其參數，並檢查音樂是否有撥放。因為悟空是使用事件觸發模型，所以如果數值沒有改變，便不會透過資料流傳送出去，所以我們必須交替傳送 0 與 1。



4.1.5. 尋找裝置問題

在尋找裝置時，一個很常見的問題是當移動到不同的網路環境下，並且獲取了新的 IP 位置。然而在主控台、通訊閘或裝置們，依舊保存了之前舊的 IP 位置，而沒有更新。為了解決這個問題，你必須依照下列步驟來重設悟空系統。此外，當有部分裝置在同一網域內，但卻無法被尋找到時也可以用此方法重設整個系統。

```
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.
```

當重設完成後，你必須重新啓動主控台、通訊閘，並一一把裝置加回。

4.2 控制 LED

在這個章節，我們會跟上一個章節一樣在個人電腦上執行主控台程式，但是通訊閘程式將會執行在物聯網開發板上。

如同上個章節一樣，此章節一樣分為四個步驟。

步驟 1. 啓動主控台及通訊閘程式

步驟 2. 新增裝置

步驟 3. 撰寫資料流程式

步驟 4. 程式執行

4.2.1 啓動主控台及通訊閘程式

此章節的步驟與章節 4.1.1 相比有些許的不同。因為通訊閘程式將執行在物聯網開發版上，所以當我們執行主控台程式後，需遠端連線到物聯網開發版上，並執行訊閘程式。

注意: 在進入此章節前，如果您已執行過上一個章節的範例，您必須依照 [章節 4.1.5](#) 的步驟清除快取資料。

1. 請先輸入下列指令從 `github` 下載程式碼到您的電腦。

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

2. 建立 `infuser`。

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

3. 複製主控台設定。

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

4. 確認主控臺的網路介面資訊

```
ifconfig  
# 此範例中，主控臺的網路介面是 wlan0  
# 主控臺的網路位址是192.168.4.17
```

Note: 這個網路位址為步驟9的MASTER_IP。

```
wukong@wukong-demo:~/wukong-darjeeling/wukong/master$ ifconfig
eth0      Link encap:Ethernet HWaddr c0:3f:d5:b3:e5:41
          inet addr:192.168.4.17 Bcast:192.168.4.255 Mask:255.255.255.0
          inet6 addr: fe80::c23f:d5ff:feb3:e541/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:14115436 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8569704 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6332580070 (6.3 GB) TX bytes:3748106515 (3.7 GB)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:4471008 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4471008 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:15649850640 (15.6 GB) TX bytes:15649850640 (15.6 GB)
```

1. 執行主控台程式。

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

2. 開一個新的終端機，並且使用 SSH 指令連結到物聯網開發版。此章節，我們使用 Edison 當作範例，Raspberry Pi 也可以執行相同的指令。

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

3. 當SSH成功登入後，下載程式碼到物聯網開發版。

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

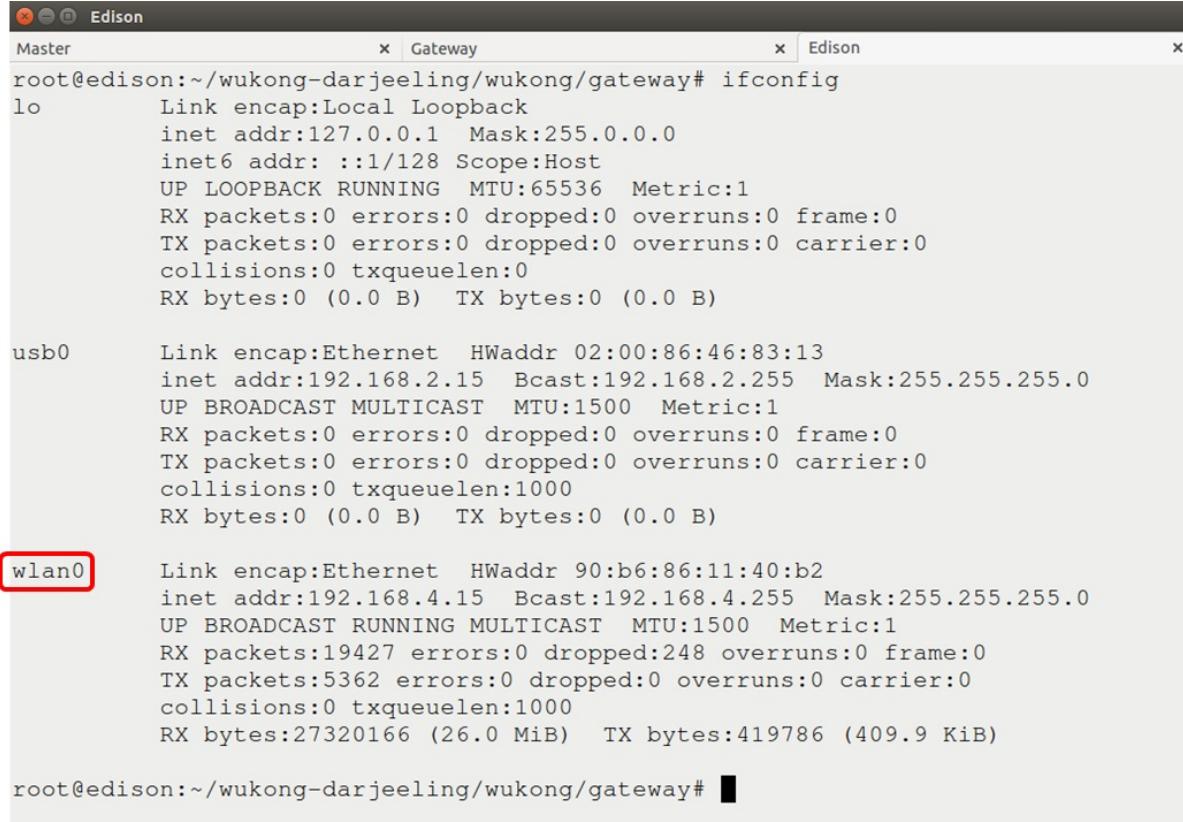
```
root@edison:~# git clone http://github.com/wukong-m2m/wukong-darjeeling
Cloning into 'wukong-darjeeling'...
remote: Counting objects: 25895, done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 25895 (delta 6), reused 0 (delta 0), pack-reused 25866
Receiving objects: 100% (25895/25895), 24.92 MiB / 1.33 MiB/s, done.
Resolving deltas: 100% (14908/14908), done.
Checking connectivity... done.
root@edison:~#
```

4. 複製通訊閘程式的設定。

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

5. 編輯 gtwconfig.py 檔案。

```
ifconfig
# 使用此指令來獲得物聯網開發版的網路資訊
# 此範例中，Edison 的網路介面是 wlan0
```



```
root@edison:~/wukong-darjeeling/wukong/gateway# ifconfig
lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
              inet6 addr: ::1/128 Scope:Host
                     UP LOOPBACK RUNNING MTU:65536 Metric:1
                     RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                     TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
                     collisions:0 txqueuelen:0
                     RX bytes:0 (0.0 B)   TX bytes:0 (0.0 B)

usb0    Link encap:Ethernet HWaddr 02:00:86:46:83:13
        inet addr:192.168.2.15 Bcast:192.168.2.255 Mask:255.255.255.0
              UP BROADCAST MULTICAST MTU:1500 Metric:1
              RX packets:0 errors:0 dropped:0 overruns:0 frame:0
              TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:1000
              RX bytes:0 (0.0 B)   TX bytes:0 (0.0 B)

wlan0  Link encap:Ethernet HWaddr 90:b6:86:11:40:b2
        inet addr:192.168.4.15 Bcast:192.168.4.255 Mask:255.255.255.0
              UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
              RX packets:19427 errors:0 dropped:248 overruns:0 frame:0
              TX packets:5362 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:1000
              RX bytes:27320166 (26.0 MiB)   TX bytes:419786 (409.9 KiB)

root@edison:~/wukong-darjeeling/wukong/gateway#
```

```
vim gtwconfig.py
# 如果你無法使用 vim , 請用 "opkg install vim" 指令來安裝
# 在 MASTER_IP 位置填入主控台所在電腦的IP
# 在 TRANSPORT_INTERFACE_ADDR 填入物聯網開發版的網路介面
```

```

# 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

```

6. 執行通訊閘程式

```

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

```

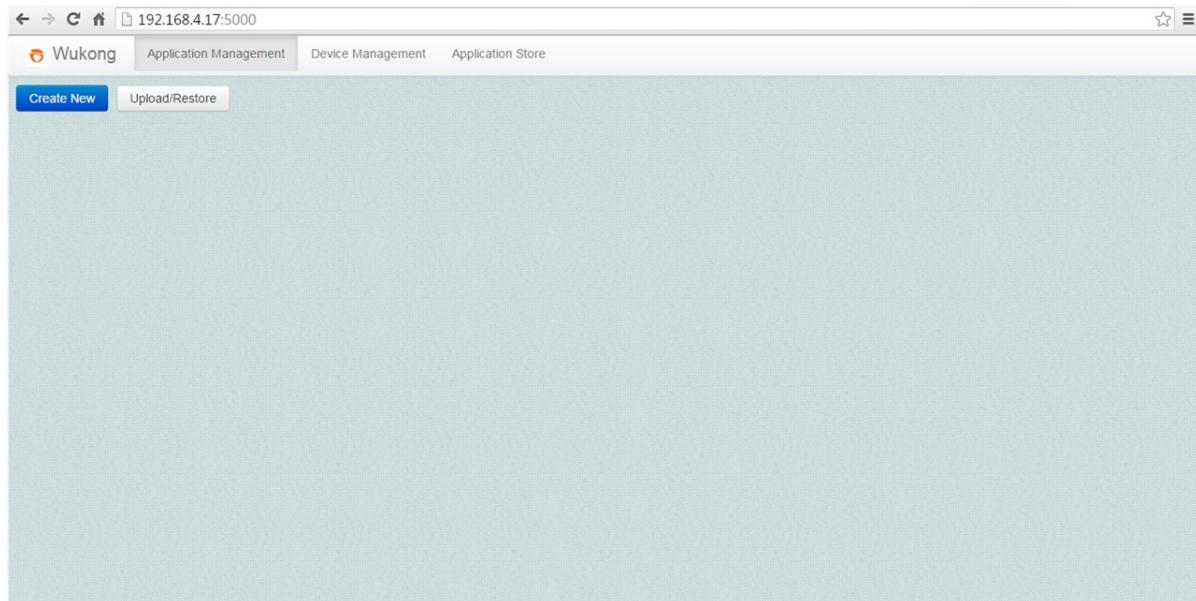
```

root@edison:~/wukong-darjeeling/wukong/gateway# python start_gateway.py
[I 160430 07:46:23 transport_udp:85] transport interface udp initialized on wlan
0 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, 2
23, 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) pe
er Peer(socket=<socket at 0x834d36c fileno=10 sock=192.168.4.15:47075 peer=192.1
68.4.17:9010>, id=0, address=('192.168.4.17', 9010))
[I 160430 07:46:24 idservice:152] GWIDREQ successfully get new ID 192.168.4.15 i
ncluding prefix
[I 160430 07:46:24 idservice:53] IDService initialized with gateway ID is 192.16
8.4.15 0xC0A8040F
[I 160430 07:46:24 rpcservice:22] RPC service initialized
[I 160430 07:46:24 gtwclass:73] All service initialized. ready to spawn greenlet
s
[I 160430 07:46:24 gtwclass:130] TCP server for Master and other Gateways is rea
dy 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, 1
42, 139, 217, 196, 63, 179, 142, 19, 6, 132, 111, 187, 221, 71, 67, 233, 224, 15
3, 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, 24
8, 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, 2
29, 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 新增裝置

1. 使用 Chrome 瀏覽器開啟主控台介面：<http://localhost:5000>



2. 點選 **Device Management** 分頁並點擊 **Discover Node** 按鈕來獲得初始狀態。

First Click

Second Click

#	Location	WuClass	WuObject			
1	/WuKong	Find Location	Set Location	0	1	Details

3. 點擊主控台上的 **Add Node** 按鈕來新增物聯網裝置。

Gateway 192.168.4.15 tcp_addr=('192.168.4.15', 9001) STOP

Discover Nodes Add Node Remove Node

#	Location	WuClass	WuObject
1	/WuKong	Find Location Set Location	0 1

Gateway 192.168.4.15 tcp_addr=('192.168.4.15', 9001) ready to ADD

Discover Nodes Stop to complete operation

#	Location	WuClass	WuObject
1	/WuKong	Find Location Set Location	0 1

4. 開啓一個新的終端機並且 SSH 登入至物聯網開發版。

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

5. 當 SSH 登入成功後，移至物聯網開發版上的 Python 悟空類別的資料夾。

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

6. 執行裝置程式 `udpdevice_blink_led.py`。這個程式會啓動一個裝置並且從通訊閘取得一個裝置識別碼，之後在此裝置上產生兩個悟空物件，分別是 Button 跟 Light Actuator。

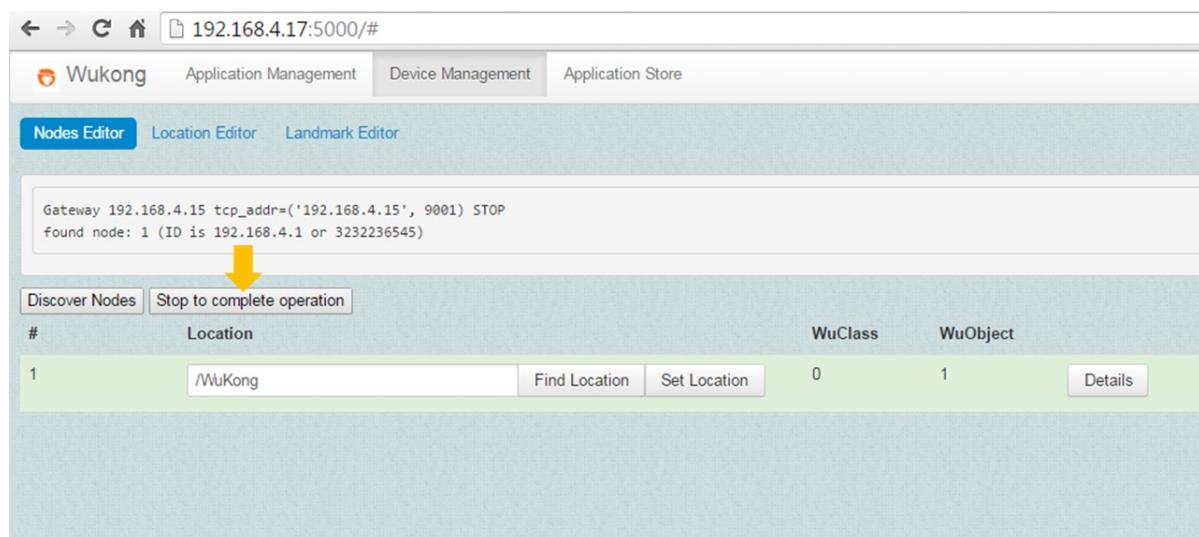
```
python udpdevice_blink_led.py <IP address of Gateway> \
<IP address of IoT board>:<arbitrary port number>
```

```

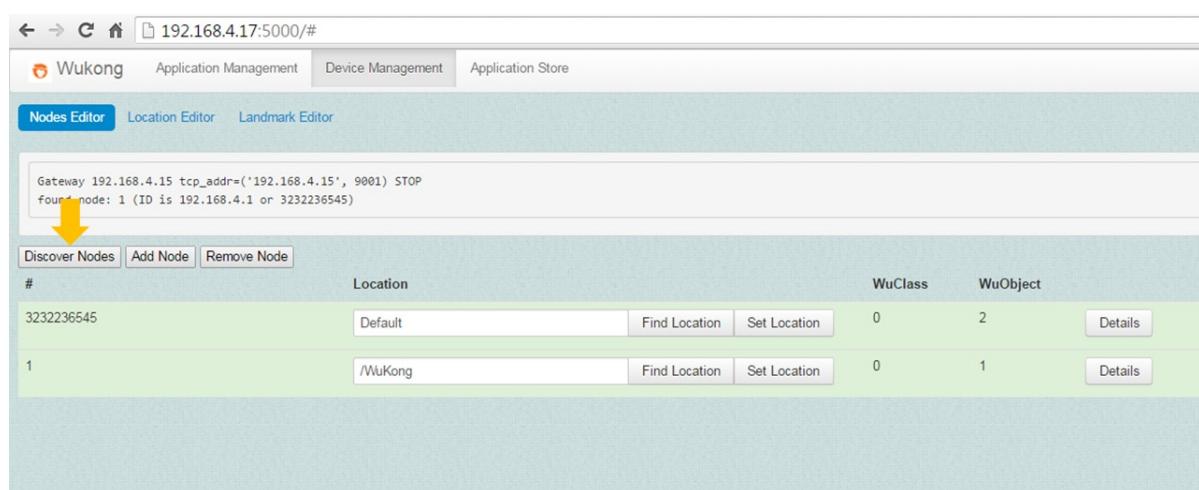
root@edison:~/wukong-darjeeling/wukong/gateway/udpwkpf# python udpdevice_blink_led.py 192.168.4.15 192.168.4.15:3000
Light Actuator init success
Button init success
Button pin: 5 , value: 0
Button pin: 5 , value: 0
Button pin: 5 , value: 0
Your ID is 3232236545 of which dotted format is 192.168.4.1
Button pin: 5 , value: 0

```

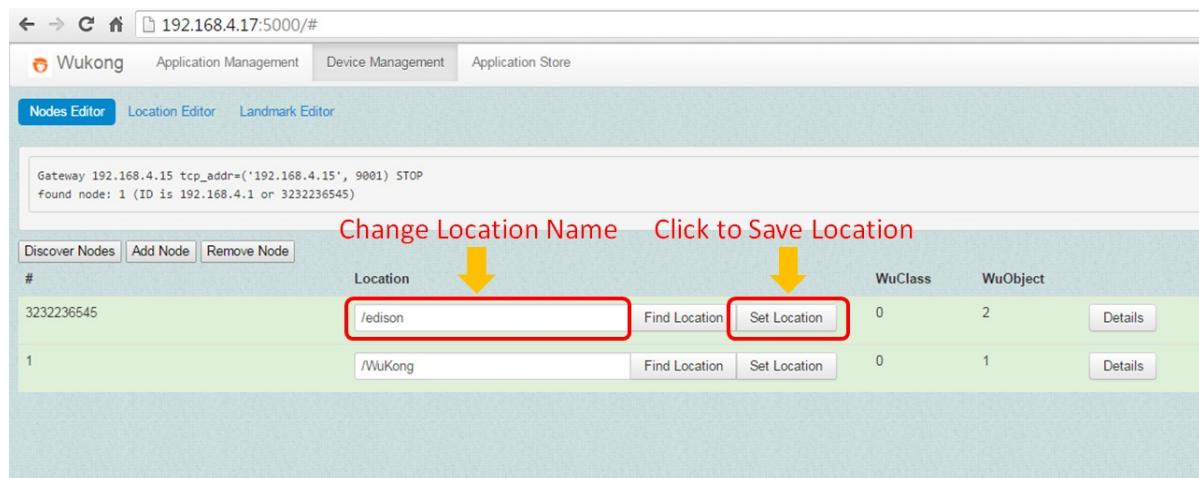
7. 當主控台出現下列文字時，點擊 **Stop to complete operation** 按鈕。



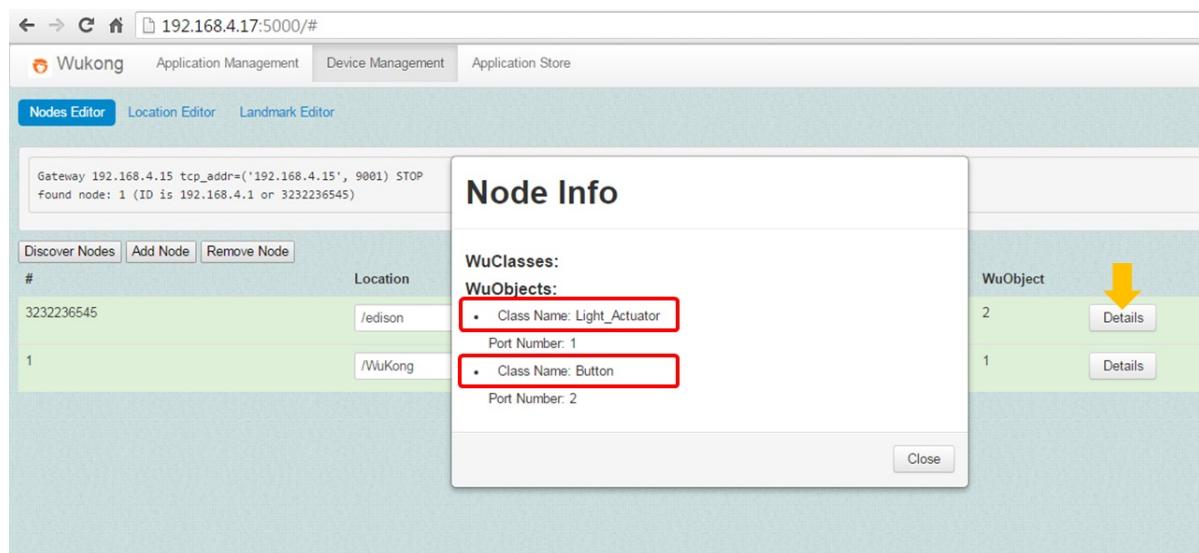
8. 點擊 **Discover Nodes** 按鈕以刷新裝置列表。



9. 更改裝置的位置並點擊 **Set Location** 按鈕以儲存。



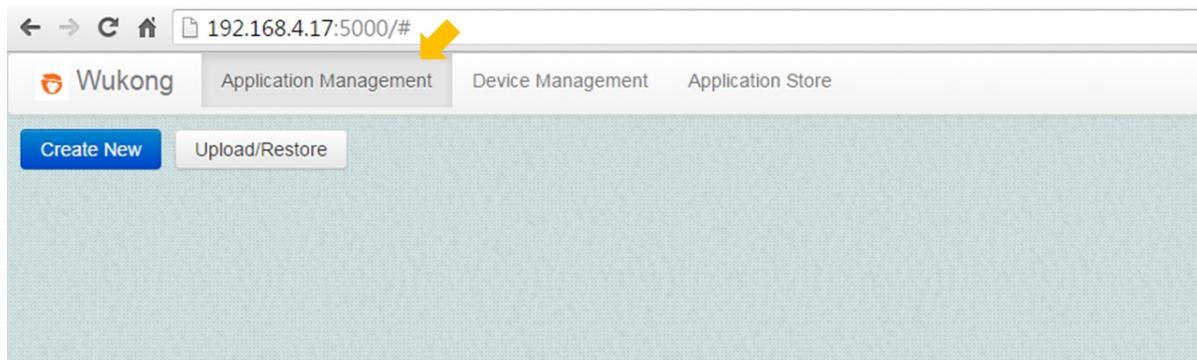
10. 點擊 **Details** 按鈕即可查看此 Python 裝置的傳感器資料。



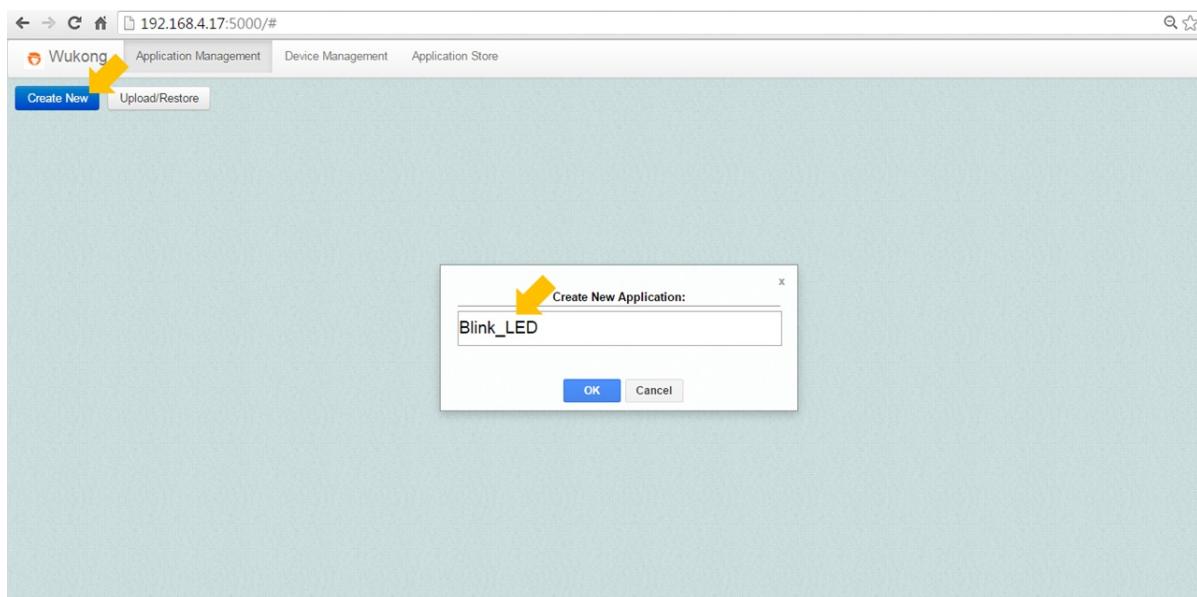
4.2.3 撰寫物件流程式

現在，我們要撰寫一個控制 LED 燈的程式，並且部署到物聯網開發板上

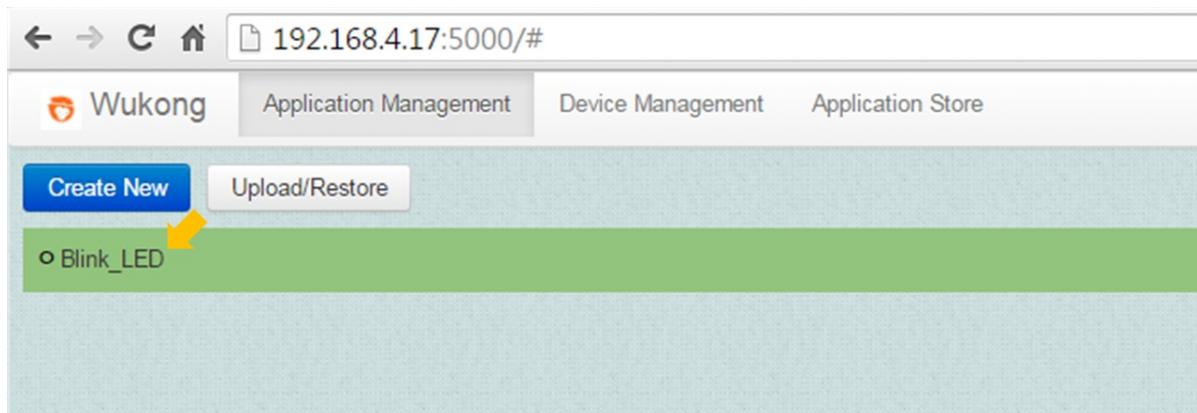
- 點擊 **Application Management** 標籤。



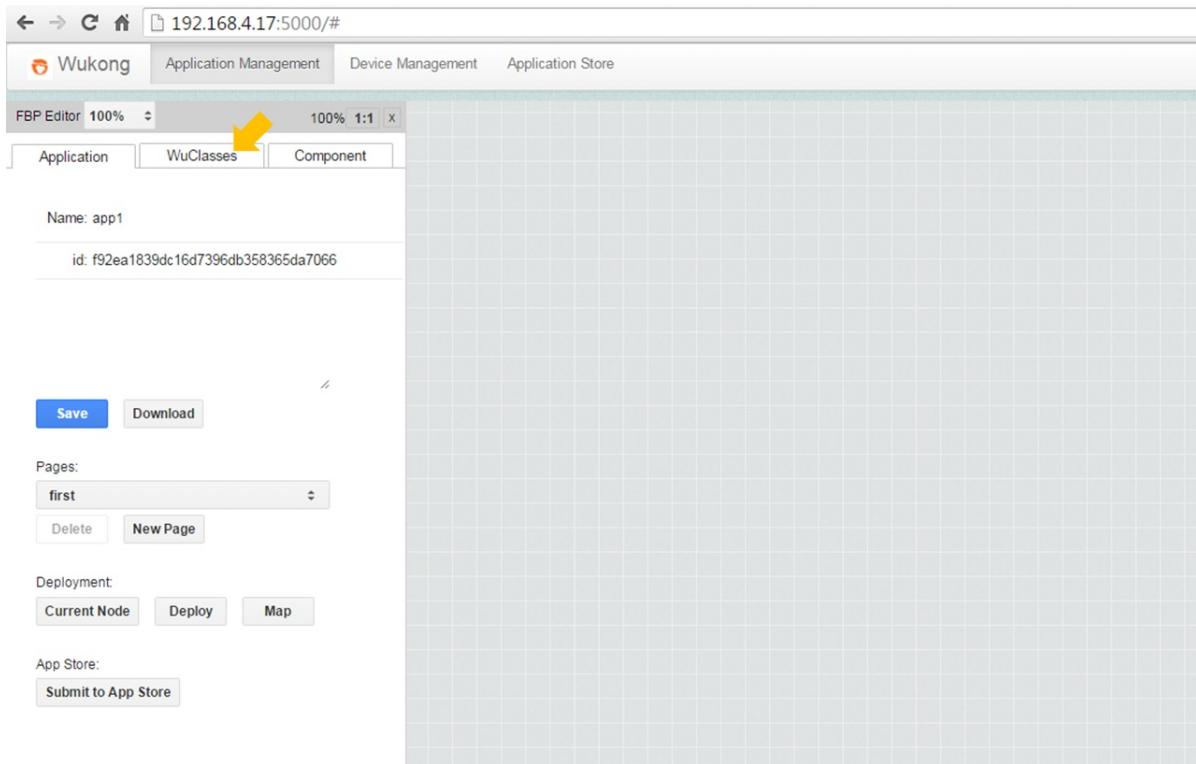
- 點擊 **Create New** 按鈕並且輸入任意的程式名稱。



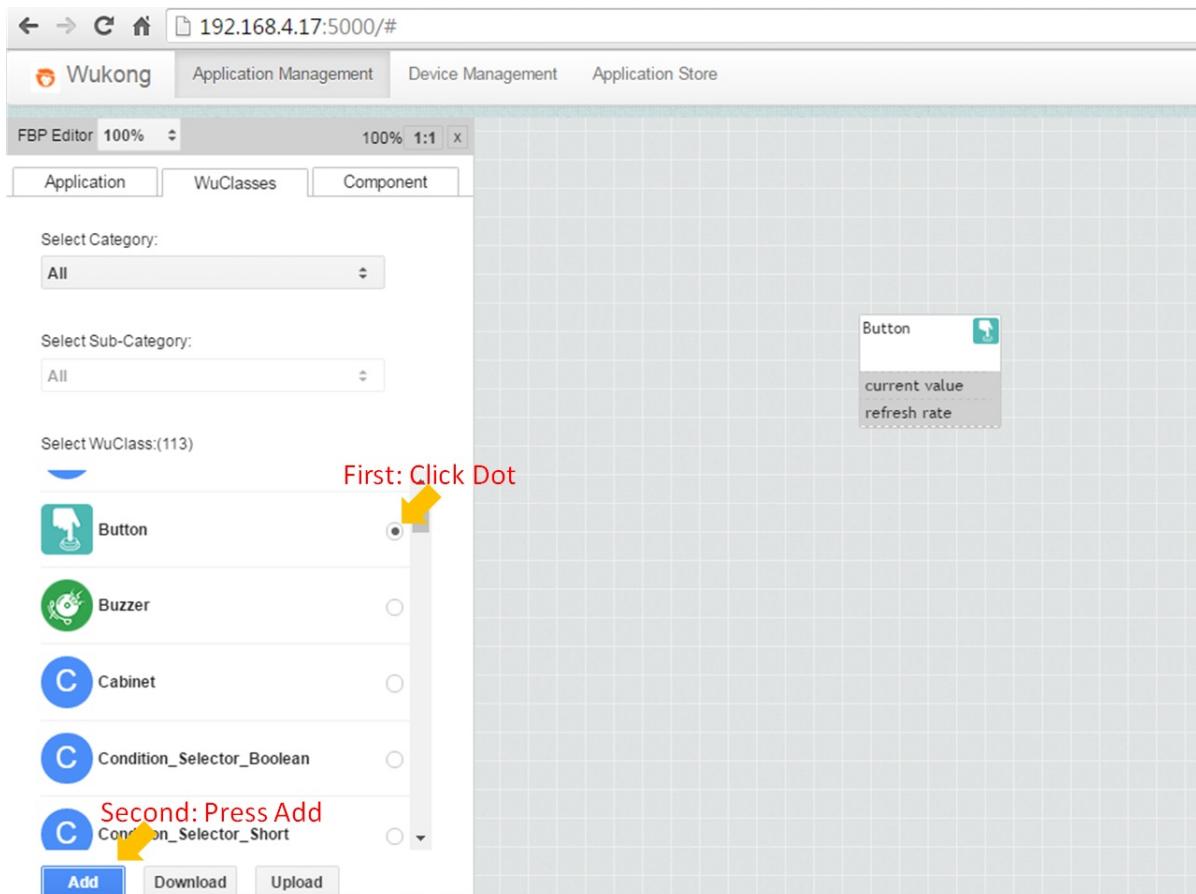
- 點擊剛才建立的程式名稱，進入編輯畫面。



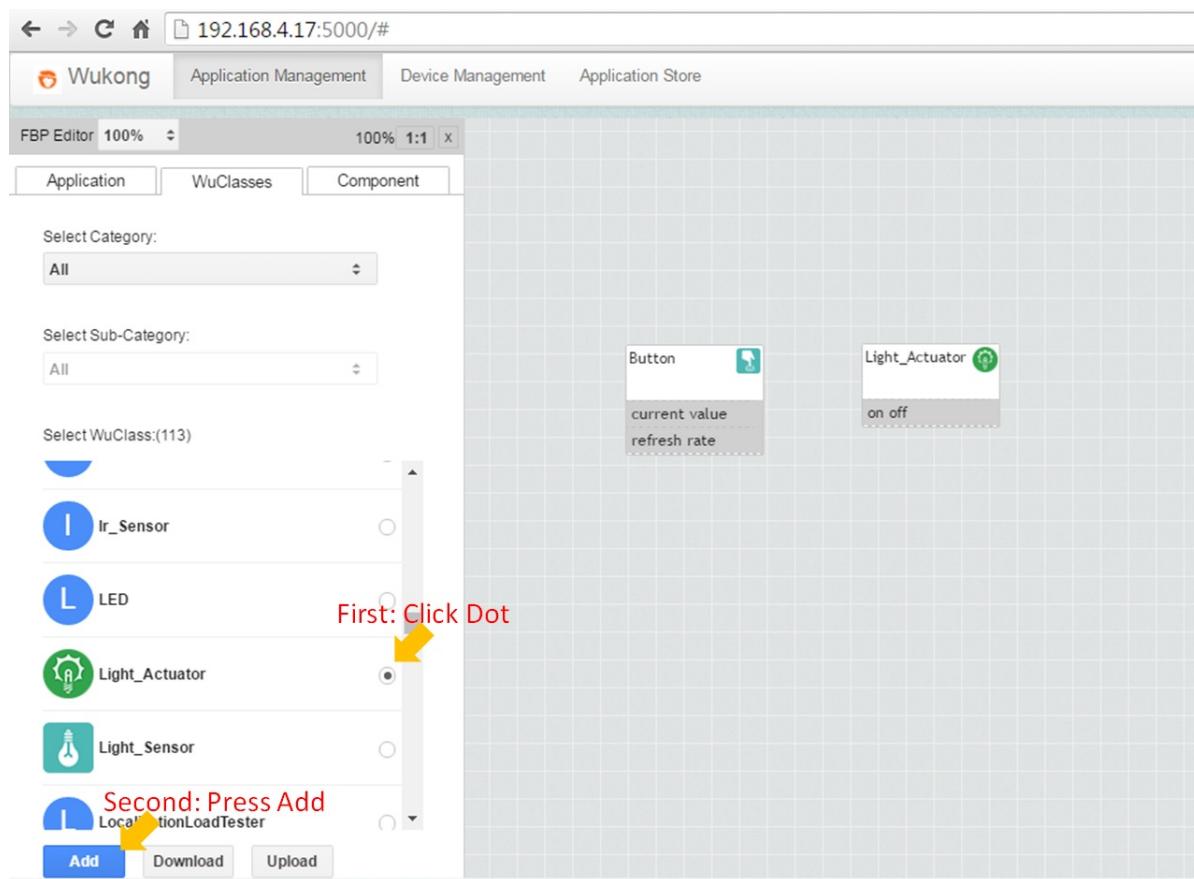
4. 點擊 **WuClasses** 標籤。



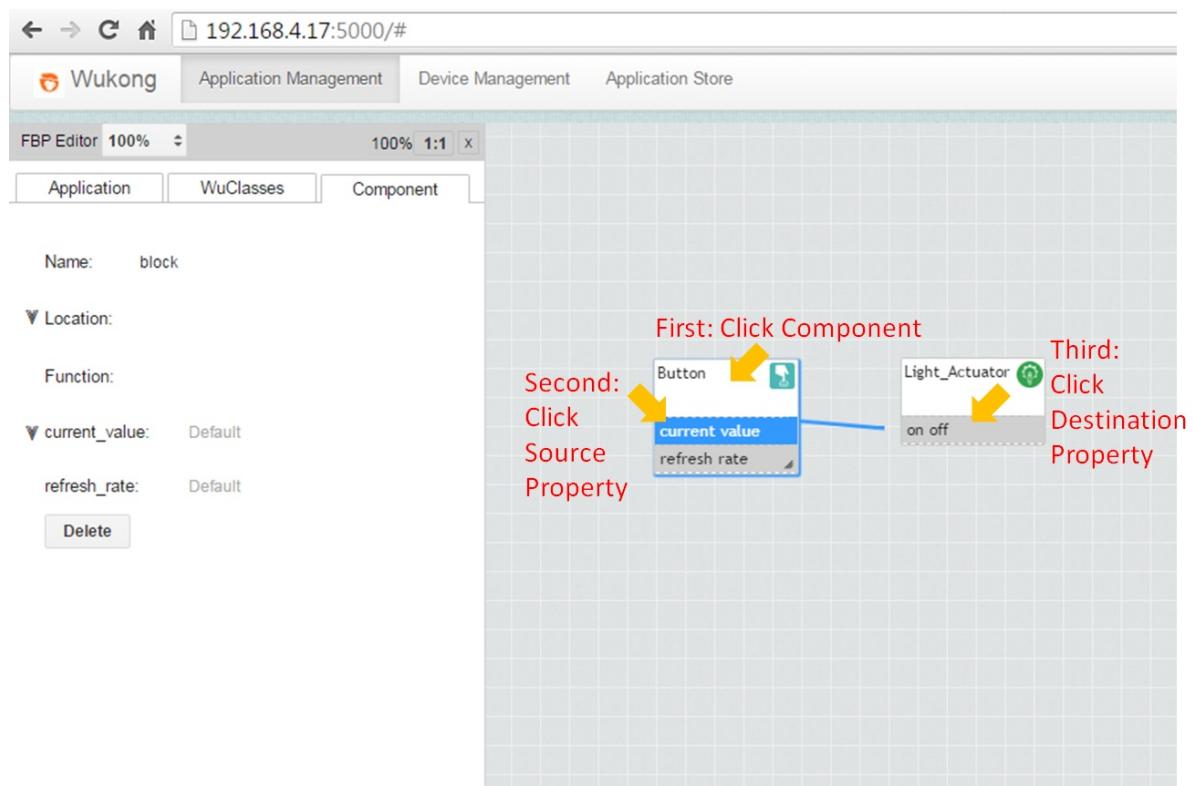
5. 滾動此列表直到找到名為 **Button** 的元件。點擊該元件即可選取，之後點擊下方的 **Add** 按鈕。現在，編輯畫面中會出現一個 **Button** 元件方塊。



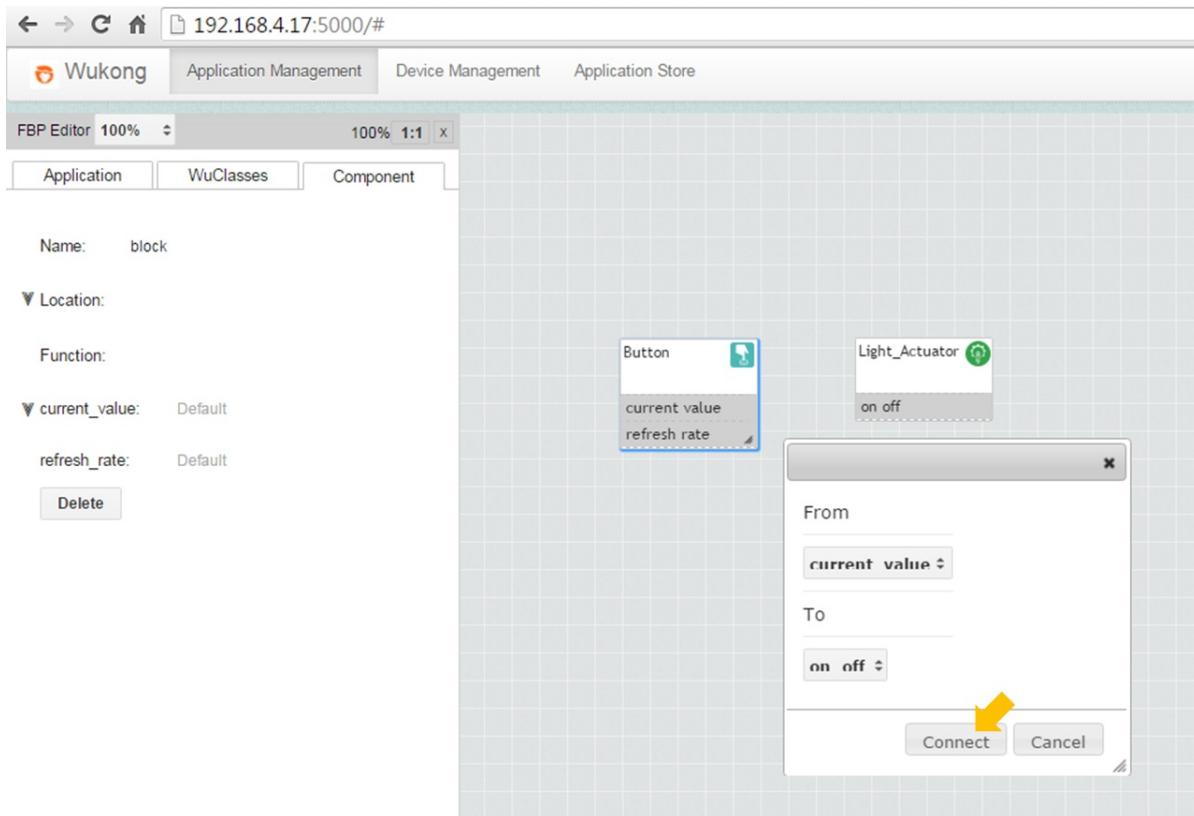
6. 重複上一步驟加入 **Light_Actuator** 元件。因為新加進來的元件可能會疊在上一個元件上，請把他們拉開。



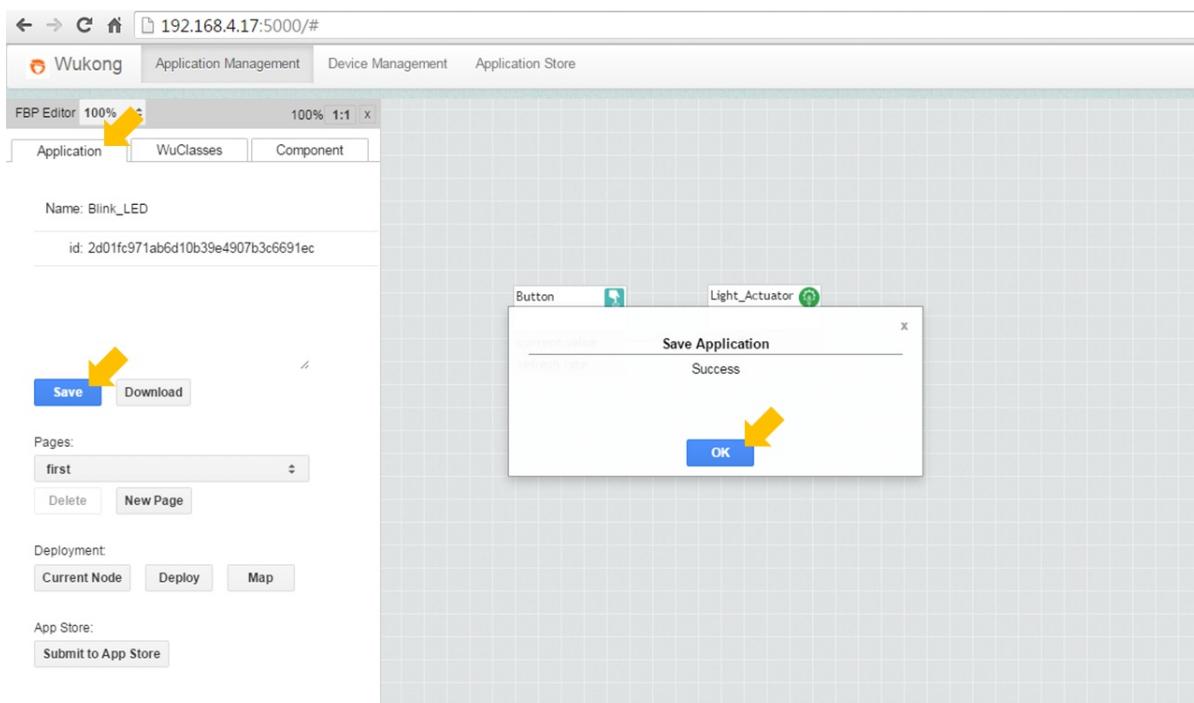
7. 在兩個元件之間新增資料流。首先，點擊來源元件，該元件會被框起。之後，點擊被選取元件的目標參數。最後，移動游標到目標元件，並點擊。



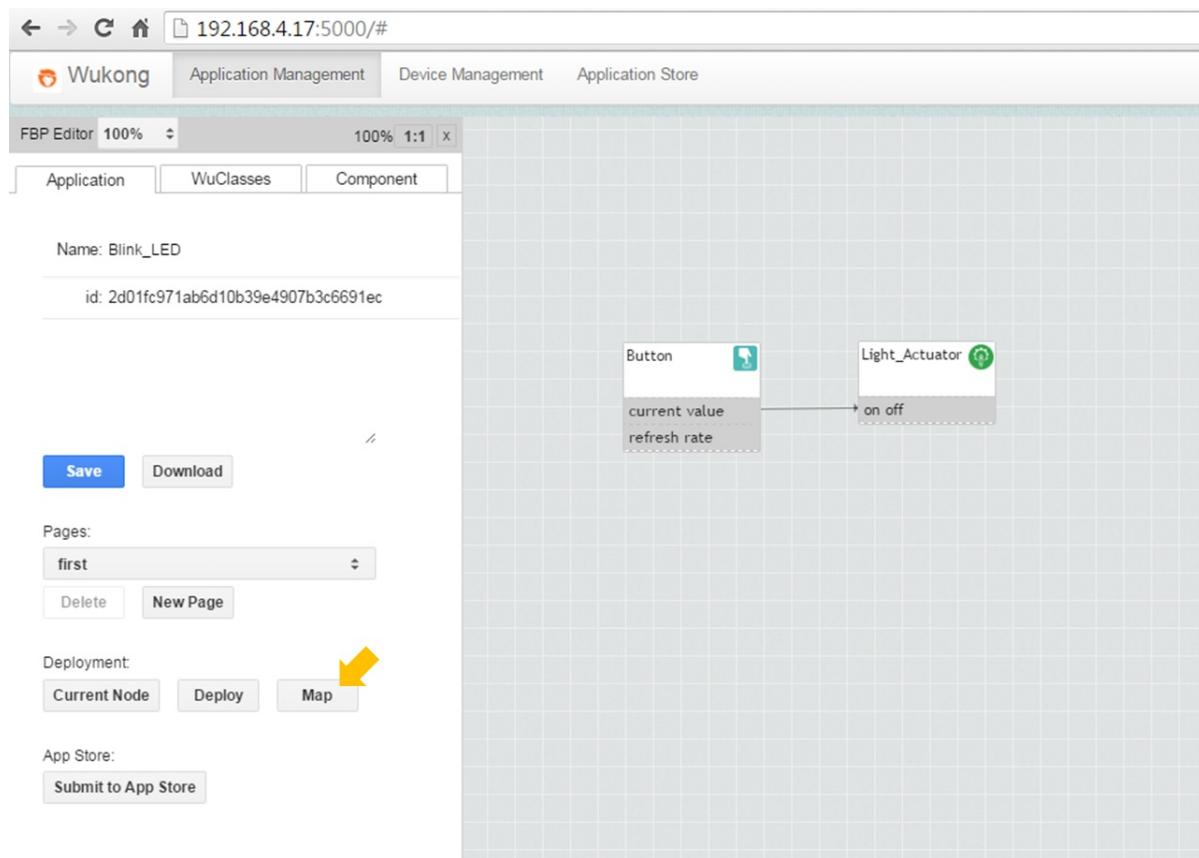
8. 此時，你可以確認所新增的資料流的來源與目標是否正確。如果有錯誤，可以在下拉式選單中做修正。



9. 點擊 Application 標籤頁中的 Save 按鈕。



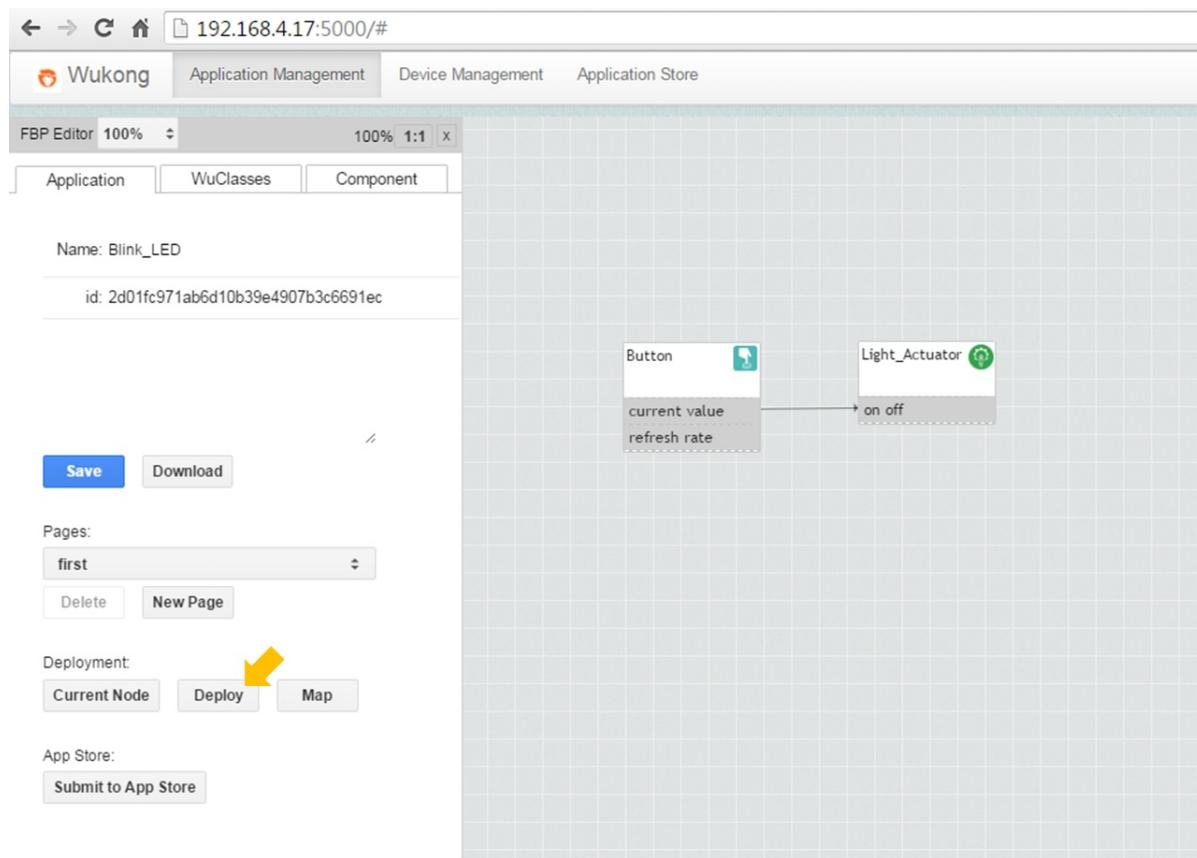
10. 然後我們就可以開始部署我們的程式。首先，點擊 Map 按鈕，開始配對程序。



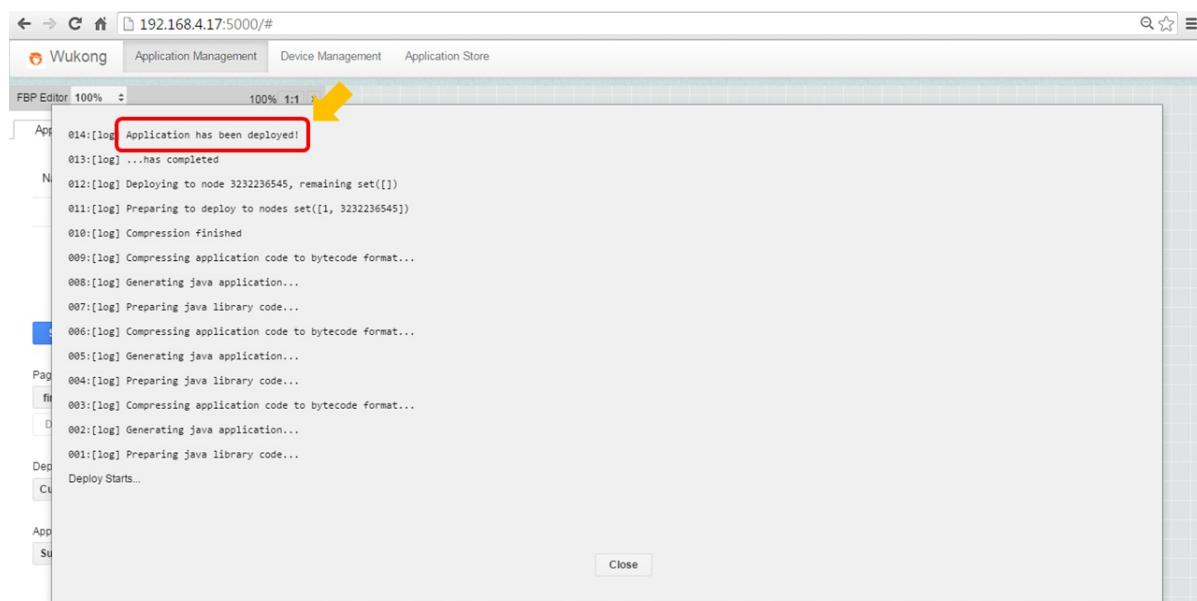
11. 之後在畫面上就會顯示配對的結果。如果配對成功了，每一個物件流程式上的元件都會對應到一個裝置 ID 和連接埠(port)。

WuClass Name	Node ID	Port Number
113 (Virtual) Button	<192.168.4.17>:3232236545	2
114 (Virtual) Light_Actuator	<192.168.4.17>:3232236545	1
1 (Virtual) Server	<192.168.4.17>:1	1

12. 點擊 Deploy 按鈕以開始部署程式。

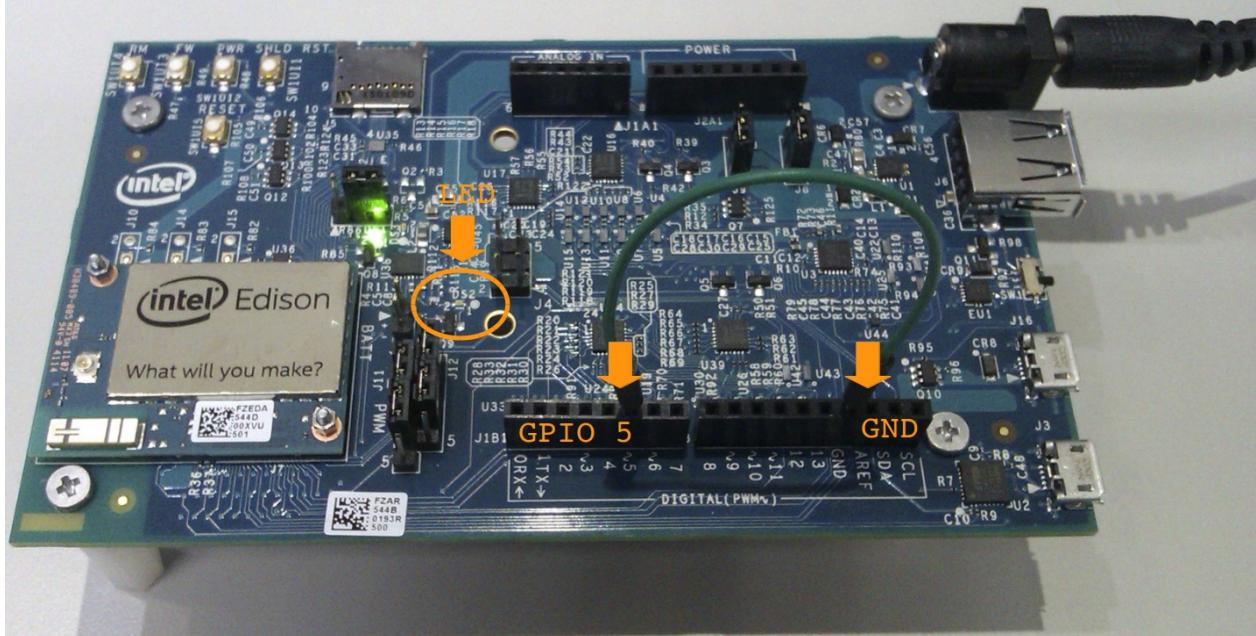
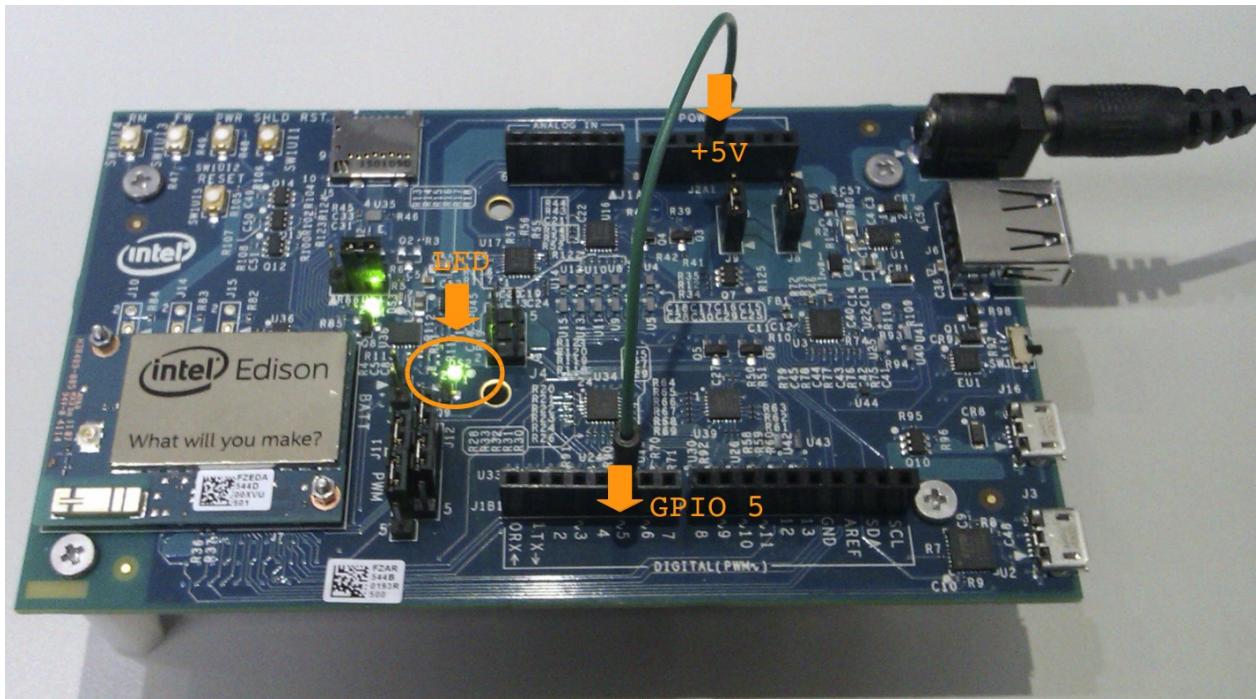


13. 您可以透過顯示在主控台上的訊息得知部署是否成功。



4.2.4 Running the Application

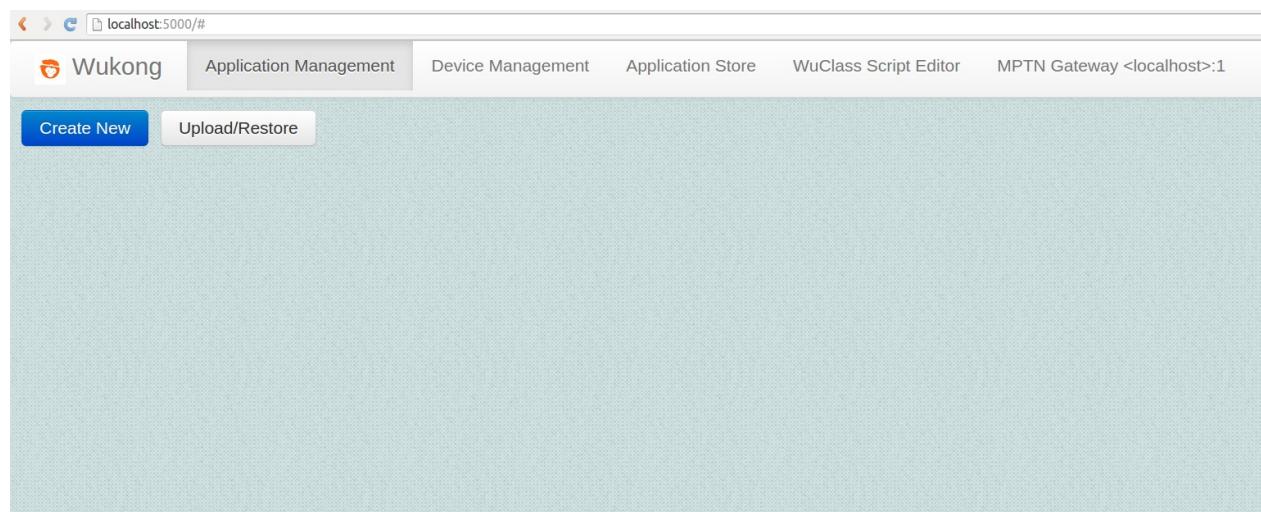
等待部署完成後，你可以拿一個公對公的針頭，一頭接到數位訊號(digital)腳位 5 的地方，另一頭接到 5V 或接地(GND)的位置。這個動作可以模擬實際按鈕的按跟放。你應該可以看到 Edison 開發版上的 LED 燈因此亮暗變化，如同下圖所示。因為通訊開與裝置都是跑在 Edison 開發版上，所以即使個人電腦關閉後，部署下去的程式仍可繼續運行。



Chapter 5: 悟空管理介面

接著，我們會示範如何使用悟空管理介面：

1. 網路管理(network management)：展示悟空的網路架構和解釋如何設定主控台和通訊閘。
2. 裝置管理(device management)：如何將裝置加入悟空系統、設定其位置(location)和獲取裝置可使用的悟空物件(WuObject)。
3. 應用程式管理(application management)：建立一個簡單的資料流編程，建立與資料流相對應的傳感器表格，部署應用程式到各個裝置上。
4. 應用程式商店(application store)：一個可以儲存和分享使用者的應用程式的介面。

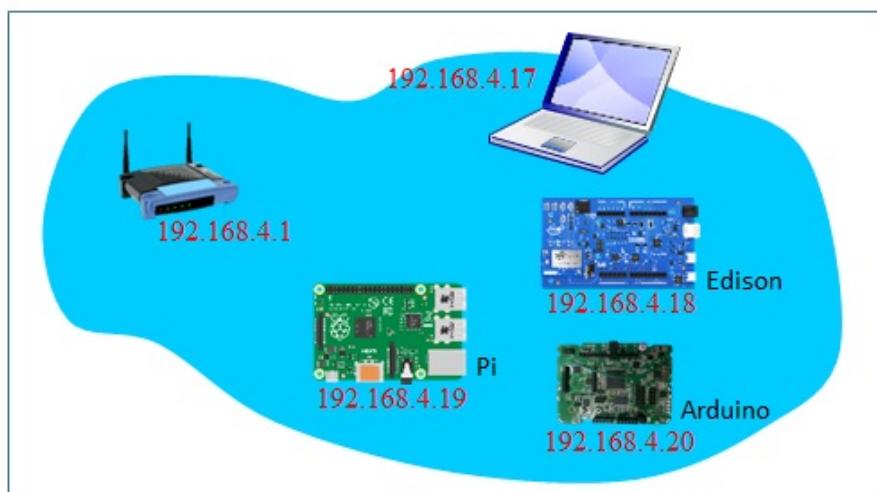


5.1 網路管理(Network management)

在悟空的應用程式(application)部署(deploy)之前，我們必須設定好網路，主控台(master)、通訊閘(gateway)和裝置(devices)之間的連線。如同 [Section 1.1](#) 提到，一個悟空系統由三個部分組成：主控台、通訊閘和裝置(實體裝置或虛擬裝置)。各個部分可以都在同一個網域內執行或都在不同網域內執行。在此章節，我們會先展示三個部分皆於同網域內執行，之後會展示於不同網域上執行。

同個網域

網路設定如下列附圖。此網路設定包含：一台筆電、兩個含有作業系統的物聯網版子(edison & raspberry Pi)和一個不含作業系統的物聯網版子(Arduino)。筆電可執行章節4.1提到的系統程式。含有作業系統的物聯網版子可執行章節4.2提到的通訊閘和裝置，不含作業系統的物聯網版子只可以當作一般裝置使用。



下列步驟是用來啓動同個網域下的主控台和通訊閘。跟章節4.2.1完全相同。

1. 確認筆電連接到同個網域。如果網路環境改變，請依照章節4.1.5回復原始設定。
2. 確認infuser已編譯。如果未編譯，請執行以下步驟：

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

3. 確認設定檔案(config file)是否創建。如果未建立，請執行以下步驟。

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

4. 啓動主控台(master)。

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

5. 在啓動主控台後，開啓終端機(terminal)並啓動通訊閘(gateway)。通訊閘可以選擇在筆電或是含有作業系統的物聯網版子上執行。注意：在首次啓動時，必須依照主控台，通訊閘的順序執行。

6. 確認通訊閘的設定檔案是否創建。如果未建立，請執行以下步驟：

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

# 如果要在物聯網版子上執行通訊閘
# 需在物聯網版子中有程式碼，可經由git下載
# git clone http://github.com/wukong-m2m/wukong-darjeeling
```

7. 確認通訊閘的網路介面(network interfaces)，例如：eth0, wlan0...。

8. 根據通訊閘的網路介面設定 gtwconfig.py。

```
vim gtwconfig.py

# 將 MASTER_IP 改為主控台程式的IP
# 將 TRANSPORT_INTERFACE_ADDR 改為通訊閘使用的網路介面
```

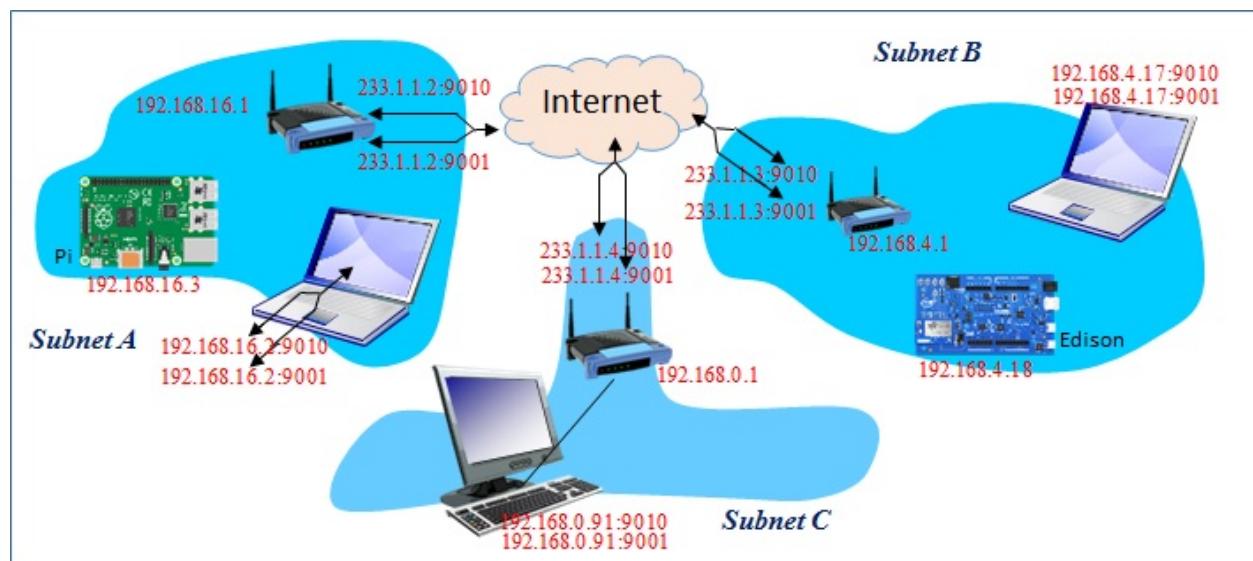
9. 啓動通訊閘。

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

10. 使用Chrome瀏覽器連入Web介面，預設為：<http://<主控台ip>:5000>.

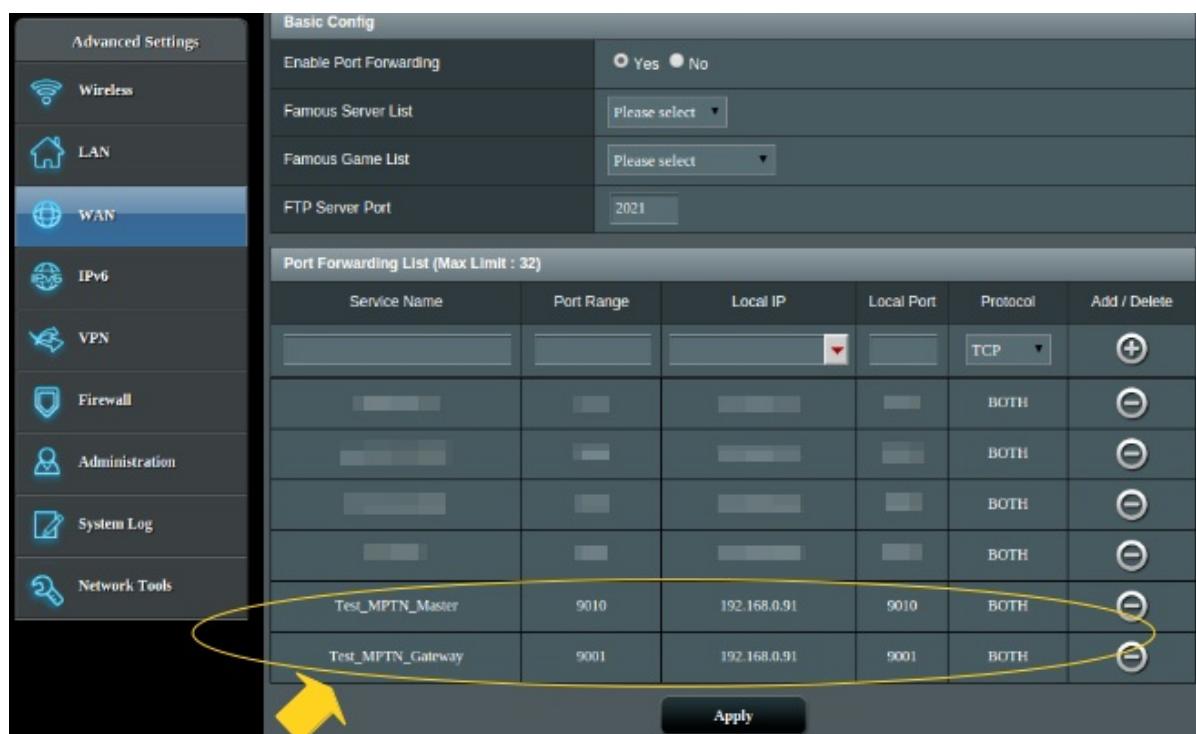
不同網域

網路設定如附圖。在此範例中，有三個子網域(subnet)：A,B,C。子網域A中有raspberry Pi且Pi的ip為192.168.16.3，子網域B中有edison且edison的ip為192.168.4.18。在各個子網域必須各有一個通訊閘，由於目前我們未處理網路地址轉換(NAT)的問題，主控台(master)必須在子網域C執行。

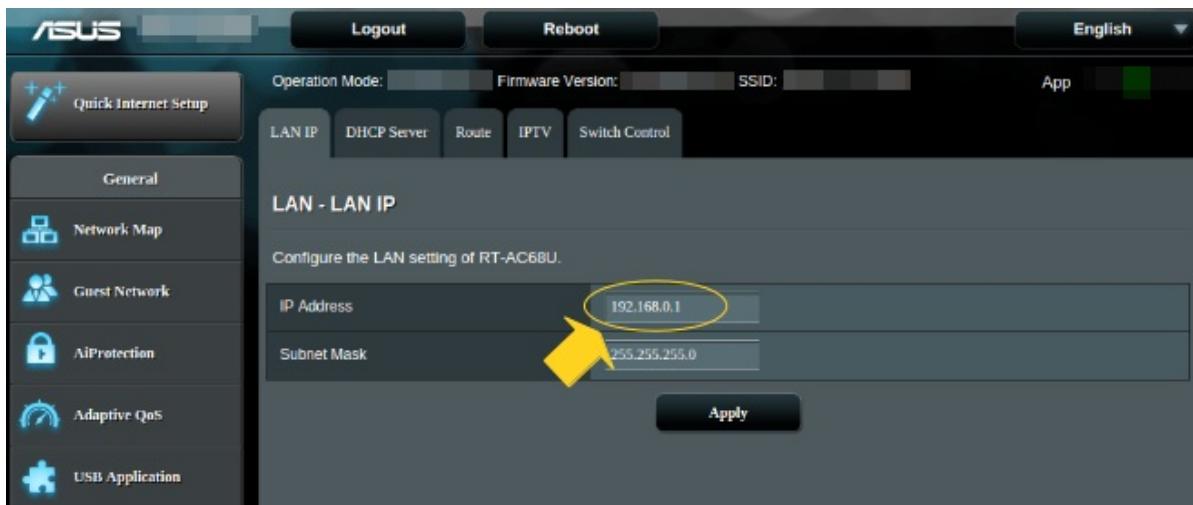


下列步驟用來啓動主控台和通訊閘

- 在每個路由器中開啓9010和9001的通訊埠轉發。通訊埠9010用於主控台從通訊閘傳接封包，通訊埠9001用於通訊閘與主控台溝通。下列圖片是ASUS RT-AC68 WiFi router的設定畫面。



- 在每個路由器中設定不同的網址前綴(network prefix)。原因是如果兩個子網域使用相同的網址前綴，主控台無法辨認其封包(packet)是從哪個子網域傳出。因此，在範例中，我們使用不同的網址前綴。



3. 在子網域中的主控台，設定如同單一網域，請依照單一網域中，步驟1到4設定並啓動。
4. 在啓動主控台後，使用終端機登入Raspberry Pi。
5. 確定Raspberry Pi中有程式碼，如果沒有請依照下列步驟下載

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

6. 確認通訊閘的設定檔案是否創建。如果未建立，請執行以下步驟：

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

7. 從每個路由器中找到路由器公共IP或WAN IP。在此範例中，子網域A的路由器IP為233.1.1.2，子網域B的路由器IP為233.1.1.3，子網域C的路由器IP為233.1.1.4。
8. 設定gtwconfig.py.

```
vim gtwconfig.py

# 將 MASTER_IP 改為主控台網域的WIFI路由器的公共IP
# 此範例中，主控台IP為233.1.14
# 將 TRANSPORT_INTERFACE_ADDR 改為筆電或Pi使用的網路介面
```

9. 在筆電或Raspberry Pi上啓動通訊閘

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

10. 重複步驟4到9設定子網域B的筆電或Edison。
11. 使用Chrome連入(預設)`http://<主控台_IP>:5000`。如要在子網域A,B中存取，需在子網域C的路由器上，設定通訊埠轉發(port forwarding)。

5.2 裝置管理(Device Management)

在此章節，我們將會解解如何使用悟空主控台去管理裝置。

- 點選裝置管理(Device Management)

The screenshot shows the Wukong Device Management interface. At the top, there are tabs: Wukong, Application Management, Device Management (which is selected), and Application Store. Below the tabs, there are three sub-tabs: Nodes Editor (selected), Location Editor, and Landmark Editor. The main area displays a table of nodes. The first row contains the header columns: #, Location, WuClass, and WuObject. There is one data row, which includes a 'Details' button. At the bottom of the table, there are buttons for Discover Nodes, Stop to complete operation, Find Location, and Set Location. A yellow arrow points to the 'Add Node' button.

- 將裝置加入悟空系統

- 點選加入裝置(Add Node)按鈕，主控台會進入加入(ADD)模式，並提示"準備好加入裝置(ready to ADD)"

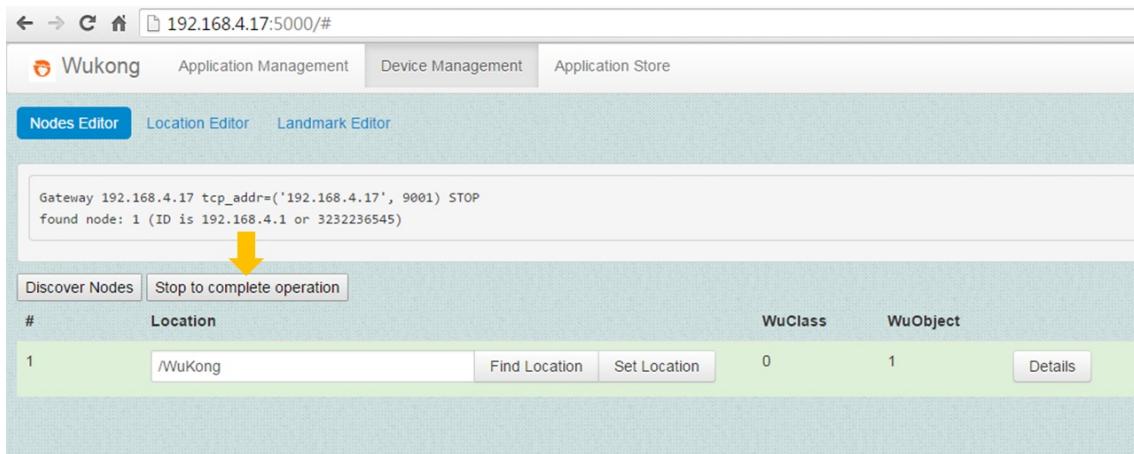
The screenshot shows the Wukong Device Management interface. The 'Nodes Editor' tab is selected. The main area displays a table of nodes. The first row contains the header columns: #, Location, WuClass, and WuObject. There is one data row, which includes a 'Details' button. At the bottom of the table, there are buttons for Discover Nodes, Stop to complete operation, Find Location, and Set Location. A yellow arrow points to the 'Stop to complete operation' button, which is highlighted with a red box and labeled 'ready to ADD'.

- 啓動裝置程式，並令其進入認知模式(learing mode)

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/udpkpf/
python udpdevice_blink_led.py <此網域的通訊閘的IP> \
<此裝置使用的網路介面的 IP>:<任意的通訊埠>

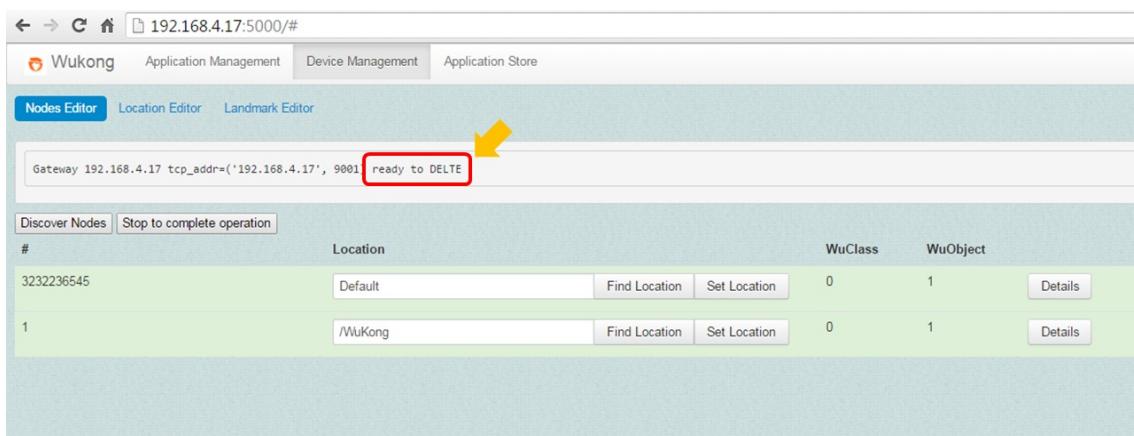
#到Python悟空類別的資料夾下啓動裝置程式
```

- 點擊停止並完成操作(Stop to complete operation)讓主控台進入 停止(STOP)模式



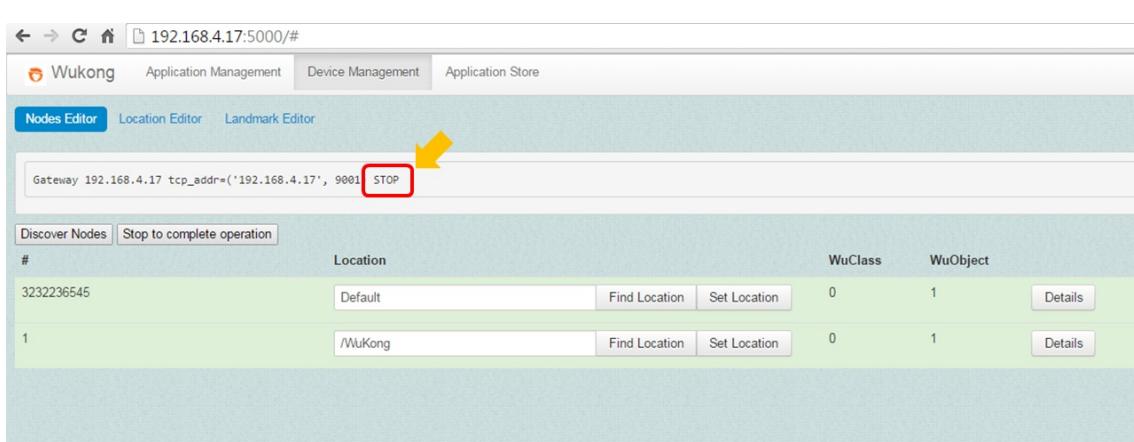
- 從悟空系統中移除裝置

- 點選移除裝置(**Remove Node**)按鈕，主控台會進入移除(**DELETE**)模式，並提示"準備好移除裝置(ready to DELETE)"



- 啓動裝置程式，並令其進入**learing mode**

```
python udpdevice_blink_led.py <IP address of gateway program> \
<此網域的通訊閘的IP><此裝置使用的網路介面的 IP>:<任意的通訊埠>
#進入想移除的裝置上，重啓Python程式
#等到Web介面顯示 "STOP" 後停止程式
```



- 點擊停止並完成操作(**Stop to complete operation**)讓主控台進入停止(**STOP**)模式

#	Location	WuClass	WuObject	
3232236545	Default	0	1	<button>Details</button>
1	/MuKong	0	1	<button>Details</button>

- 為每個裝置設定位置(location)

修改裝置位置並點選設定位置(Set Location)儲存

#	Location	WuClass	WuObject	
3232236545	/edison	0	2	<button>Details</button>
1	/MuKong	0	1	<button>Details</button>

- 尋找裝置 (Discover Nodes) 可以顯示有多少裝置是主控台已知的

#	Location	WuClass	WuObject	
3232236545	Default	0	1	<button>Details</button>
3232236546	/FrontDoor	0	1	<button>Details</button>
3232236547	/Logic	0	1	<button>Details</button>
3232236548	/Wall	0	1	<button>Details</button>
1	/MuKong	0	1	<button>Details</button>

- 細節(Details) 可以確認各個裝置上的傳感器資料

The screenshot shows the 'Node Info' dialog box from the WuKong application. The dialog box contains the following information:

- WuClasses:** A list containing one item: "Class Name: Light_Actuator".
- WuObjects:** A list containing two items:
 - "Class Name: Light_Actuator" (Port Number: 1)
 - "Class Name: Button" (Port Number: 2)

On the left side of the dialog, there is a table with columns for '#', Location, and another column. The table rows are:

#	Location	
3232236545	/edison	
1	/WuKong	

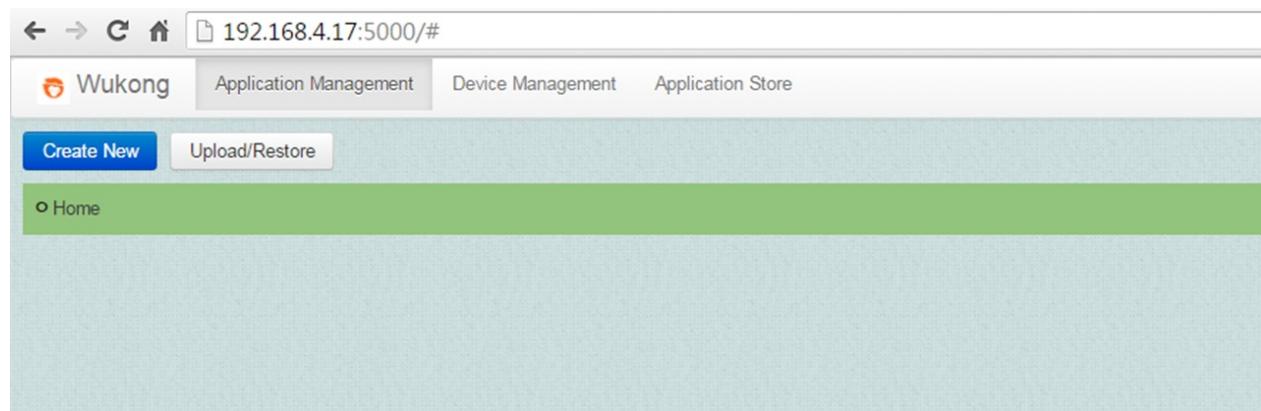
At the bottom right of the dialog box is a 'Close' button. To the right of the dialog, there is a table titled 'WuObject' with two rows. The first row has a yellow arrow pointing to the 'Details' button for the entry '2'. The second row has a 'Details' button for the entry '1'.

5.3 應用程式管理(application management)

創建資料流編程(flow-based programming)

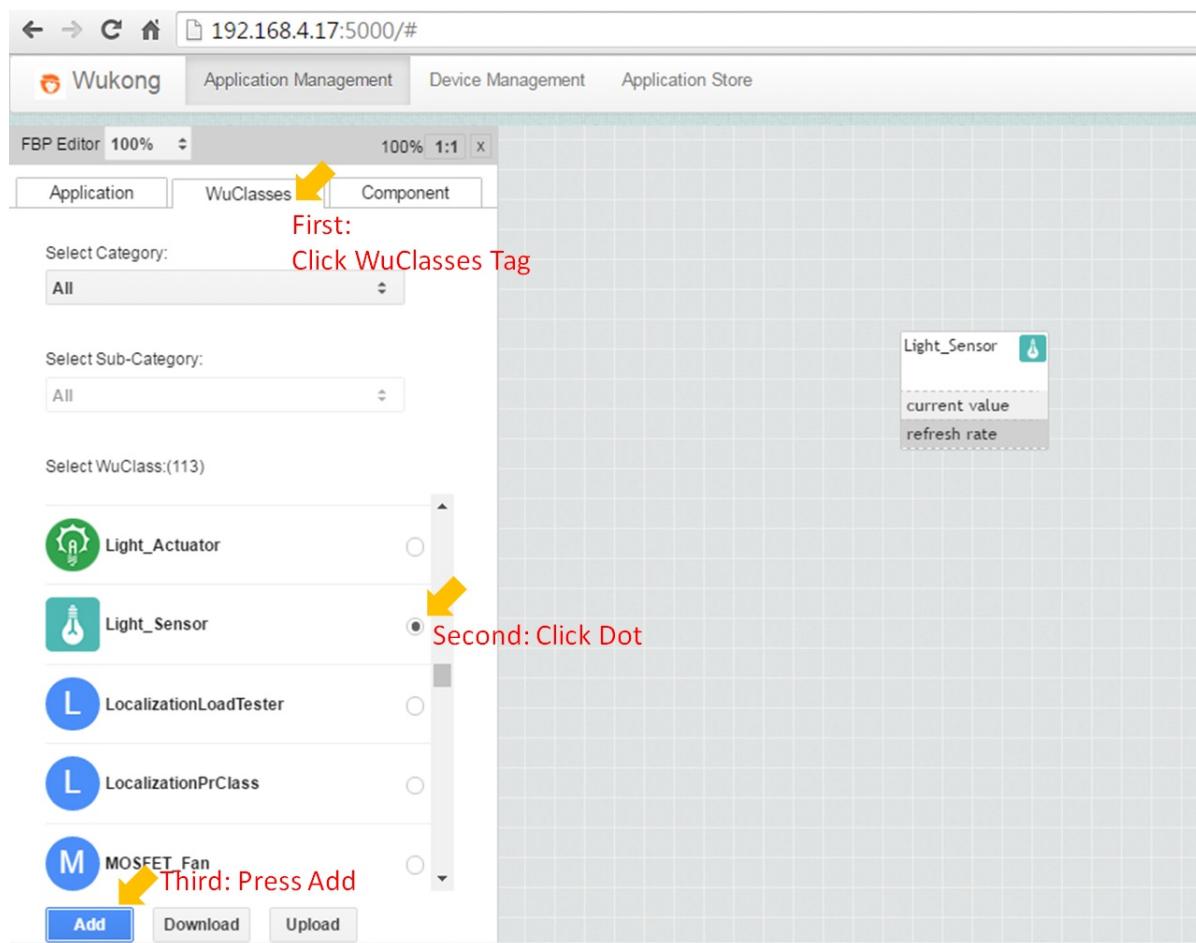
點選 應用程式管理 (Application Management)，會列出目前已有的應用程式(application)。

我們可以經由點選建立新的應用程式(Create New)加入一個新的應用程式，或點選  刪除一個應用程式。點選應用程式的名稱可以進入資料流編程(flow-based programming)的修改模式。



- 加入組件

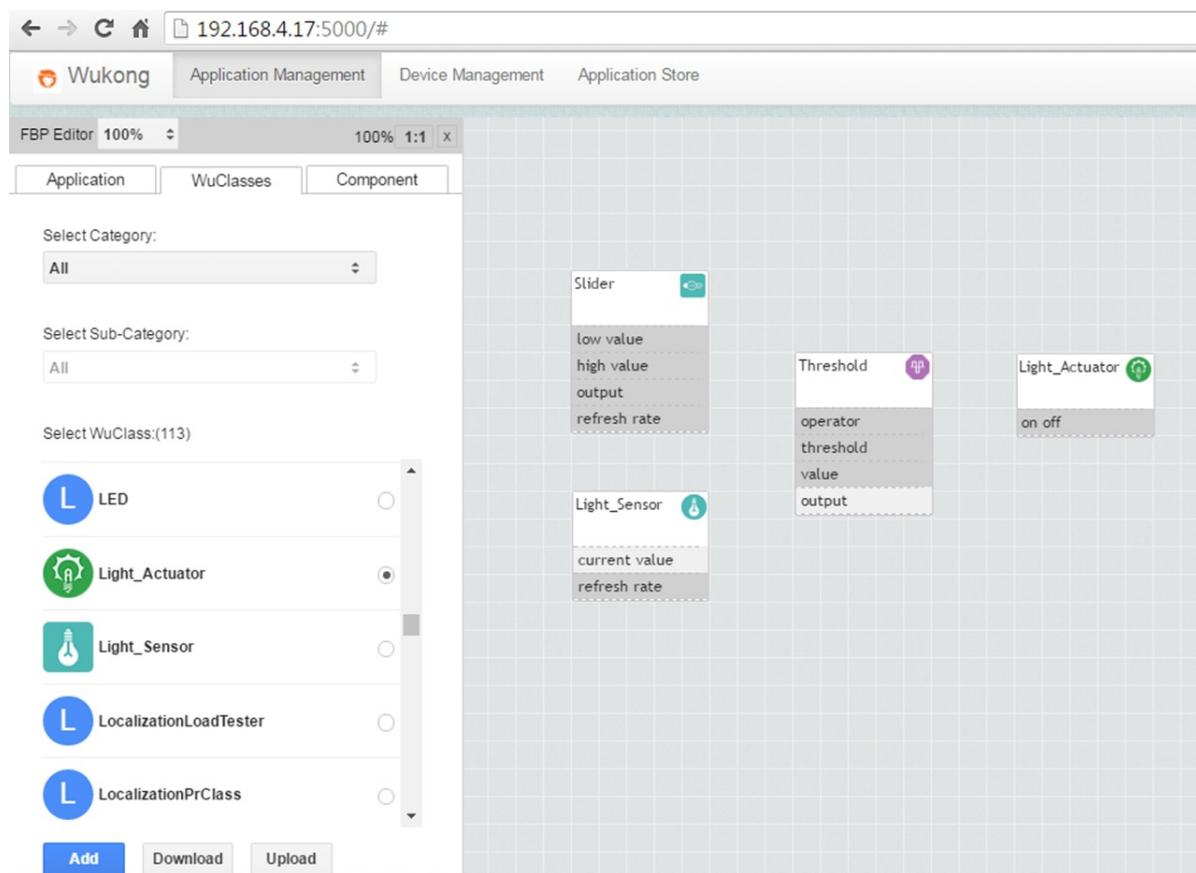
點選左側菜單悟空類別(WuClass)可以看到可用的物件(Object)，物件儲存於 WuKongStandardLibrary.xml。接著，選擇想要的物件後點選加入(ADD)或雙擊在畫布上產生一個新的物件。下圖表示選取了感光器(Light_Sensor)物件。



感光器(Light_Sensor) 物件有兩個參數。輸入屬性會以灰色顯示，輸出屬性會以白色顯示。在這個例子中，"更新速度(refresh rate)" 是表示此傳感器的多久測量一次(毫秒(ms))，"目前值(current value)"表示感光器(Light_Sensor)的收集到的資料。

在接下來的範例中，我們會建立一個應用程式：當感光器(Light_Sensor)的值低於某一個特定數值時，打開燈。這個範例會包含四個物件：

1. 感光器(Light_Sensor) 物件：產生一個0到255的值
2. 閥(threshold) 物件：比較light sensor的值是否低於設定的值
3. 滑標(slider) 物件：設定threshold的值
4. 發光器(light actuator) 物件：代表一個燈泡

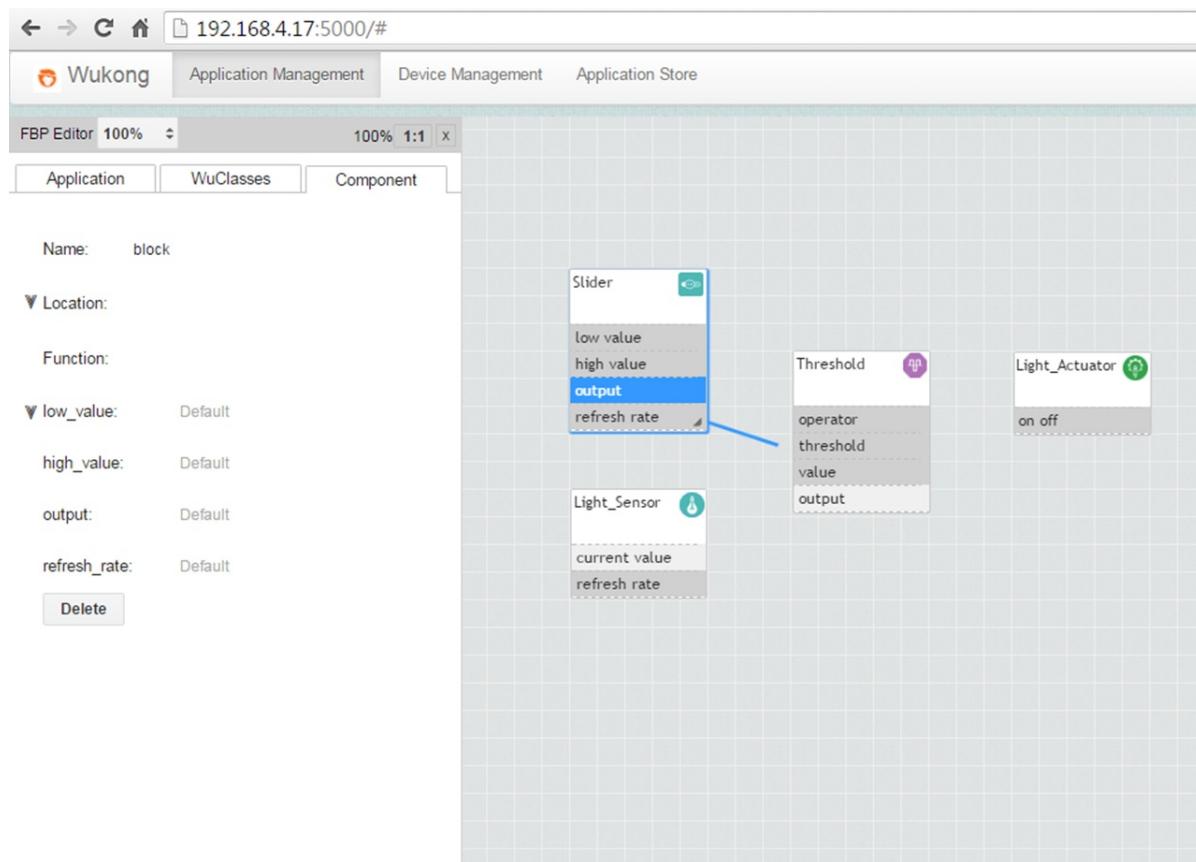


- 建立連線

在將所需的物件放入畫布後，將各物件的屬性鏈接(link)。

1. 點選想連接的來源物件(source object)
2. 點擊來源物件的屬性，一條鏈接(link)將會產生
3. 點選接收端(receiver)的物件的屬性
4. 會跳出一個視窗確認兩側想鏈接的屬性

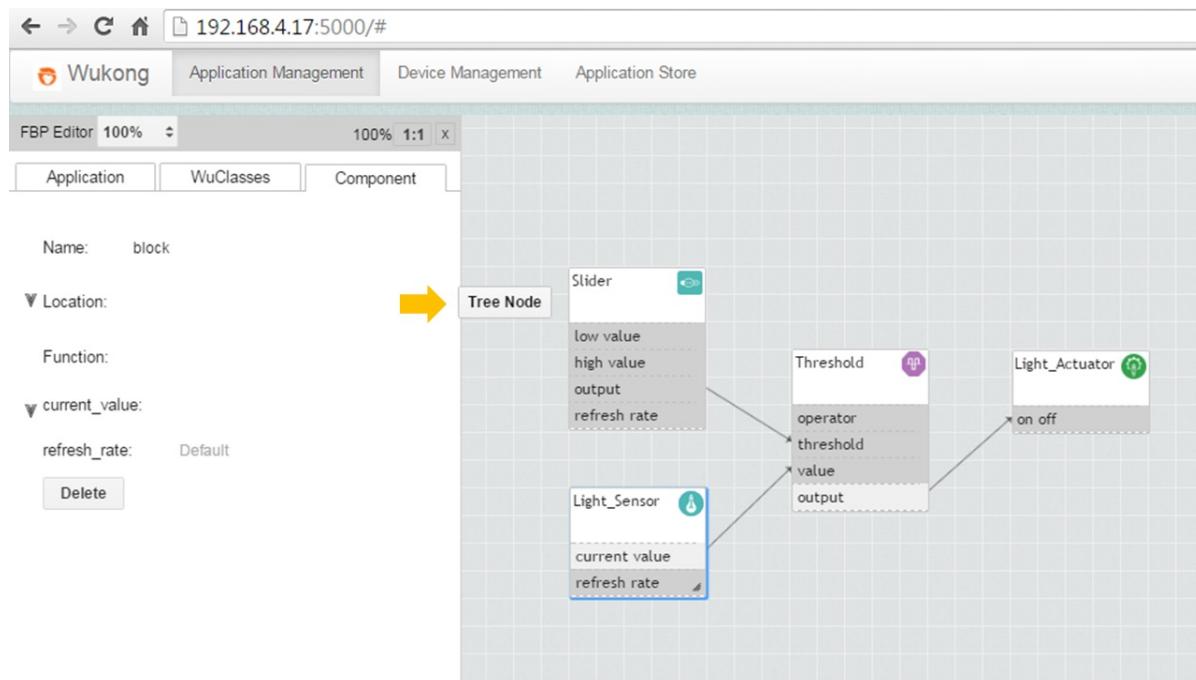
下圖為加入了所需的物件和鏈接。滑標(slider)輸出至閾(threshold)。



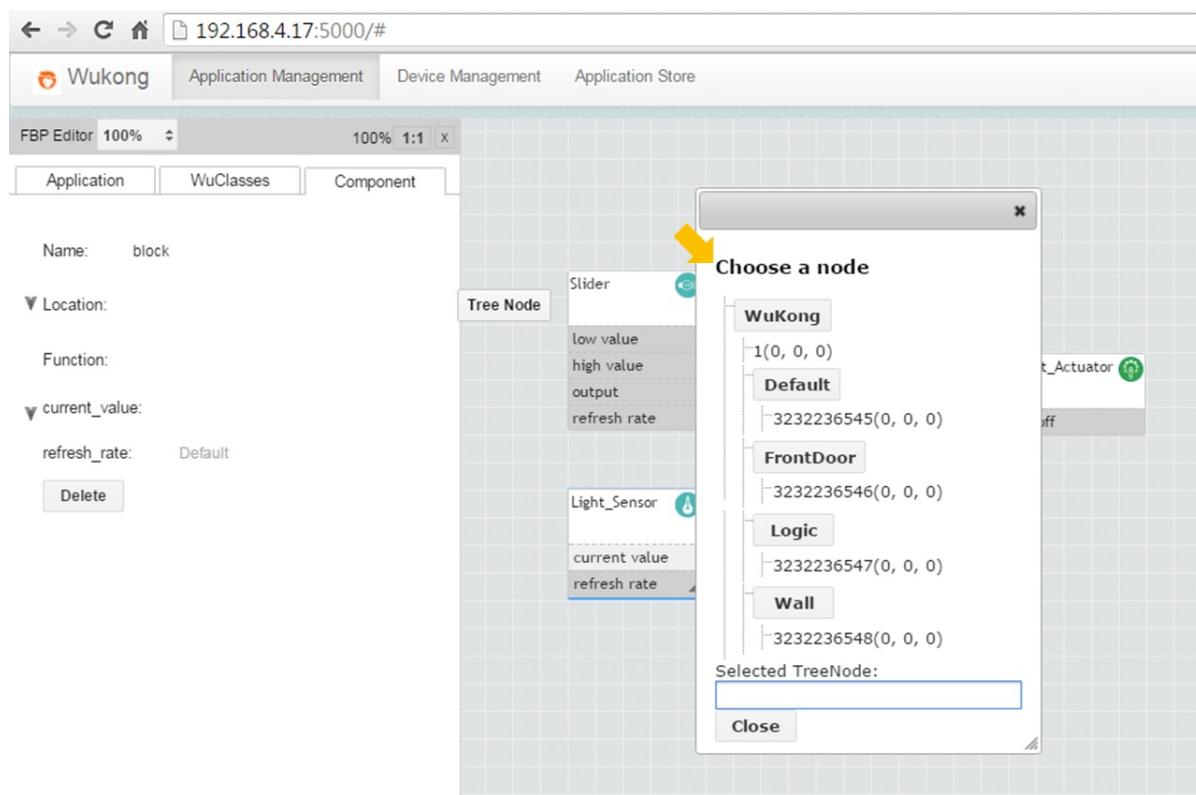
- 設定位置(**location**)

當部屬程式(deploy)時，主控台會嘗試尋找適合的裝置執行物件。由於已知的裝置中可能有複數的物件符合條件，我們使用位置來標示我們想使用的裝置。

為了設定物件的位置，將鼠標移到 空白的位置(**Location**)表格上，直到裝置樹(**Tree Node**)出現，如下圖所示。



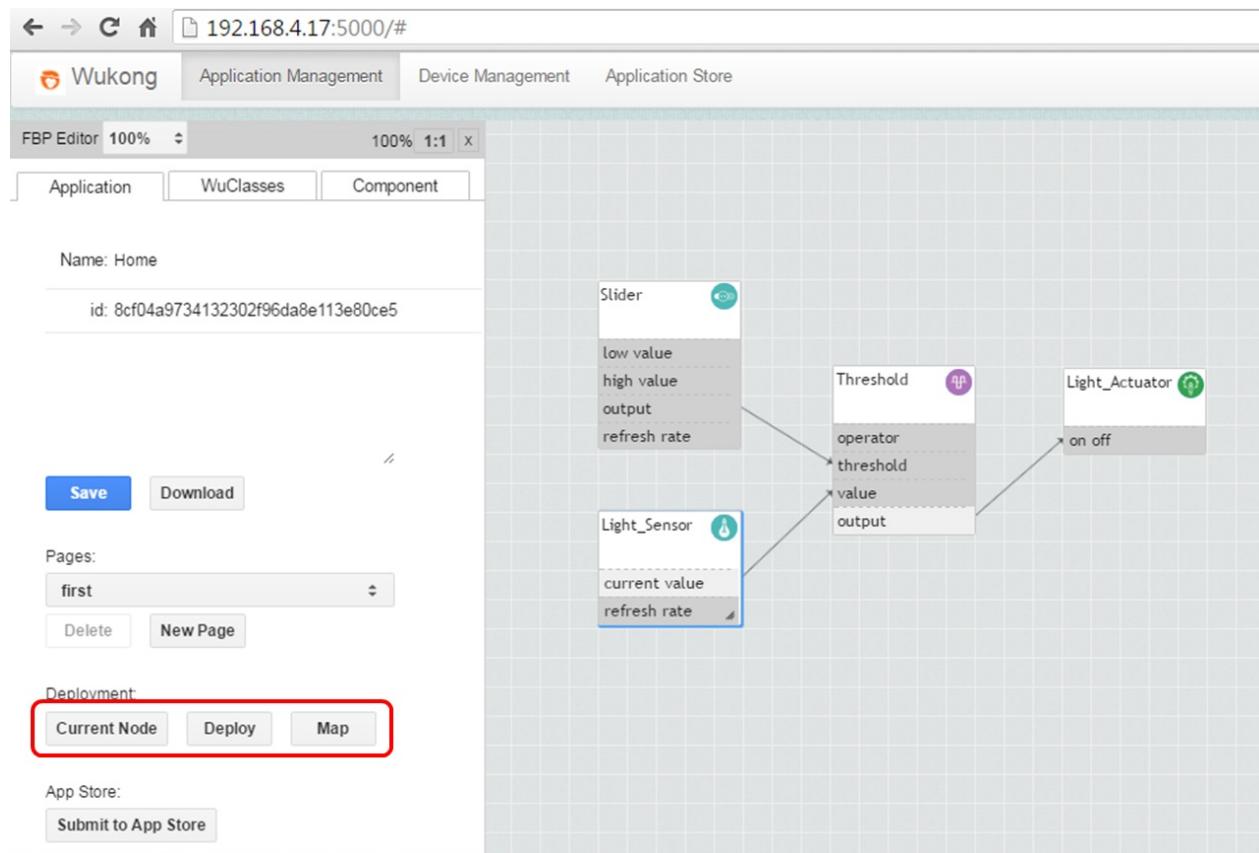
點擊 裝置樹(**Tree Node**)，會跳出一個畫面讓使用者選著目前已知的物件的位置。



部屬應用程式

當使用者完成一個應用程式，使用者可能想執行它。點擊左側菜單的 **應用程式 (Application)**。

為了更好的控制和獲取中間過程的結果，部屬應用程式目前分為三個步驟。在未來，主控台將會自動執行三個步驟。三個步驟為"目前裝置(current)"，"配對(map)"，"部署程式(deploy)"，如圖所示。

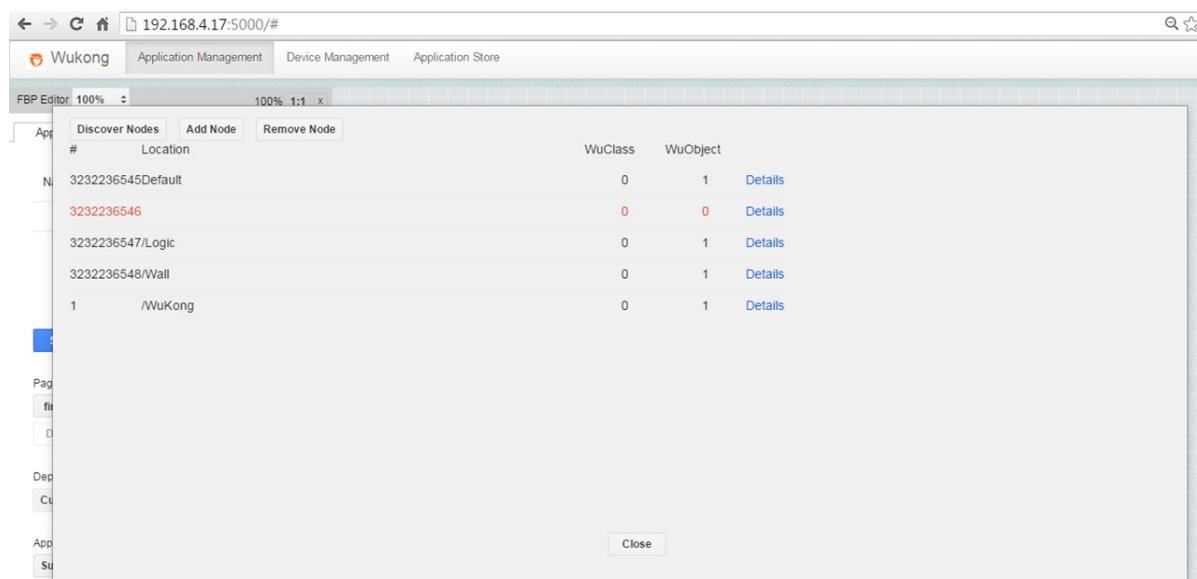


- 第一步 -- 目前裝置

點擊 "目前裝置(current)" 會列出當前尋找裝置的結果。當首次點擊時，主控台會探詢已知的裝置和其能力(WuObject)。會花上一些時間，之後會使用其結果。

每一個裝置的內容會橫列表示，當裝置是紅色時，表示裝置目前無法連接，黑色表示裝置存活。

如果預期與結果不符，回到裝置管理(Device Management)重新作尋找裝置。



- 配對(map)

當主控台知道他有哪個資源可用後，主控台依照資料流編程(FBP)配對物件(Object)和裝置(devices)。

點擊“配對(Map)”將會使主控台嘗試依照資料流編程配對物件和裝置 結果會顯示在一個表格如下：

1. 悟空物件(WuObject)的名稱
2. 配對的裝置ID
3. 配對的裝置的通訊埠(port)

The screenshot shows the 'Mapping Results' section of the FBP Editor. It lists several objects and their corresponding device mappings:

Object ID	Object Name	Device ID	Count
113	(Virtual) Light_Sensor	<192.168.4.17>.3232236546	1
115	(Virtual) Slider	<192.168.4.17>.3232236548	1
116	(Virtual) Threshold	<192.168.4.17>.3232236547	1
114	(Virtual) Light_Actuator	<192.168.4.17>.3232236545	1
1	(Virtual) Server	<192.168.4.17>.1	1

● 部署程式

在配對結束後，應用程式需經由部署程式(Deploy)部屬。這將會持續一段時間，大約30秒。部屬時，log會顯示主控台的過程，包含編譯和上傳到各個裝置。

The screenshot shows the deployment log terminal. It displays the progress of deploying the application to multiple nodes:

```

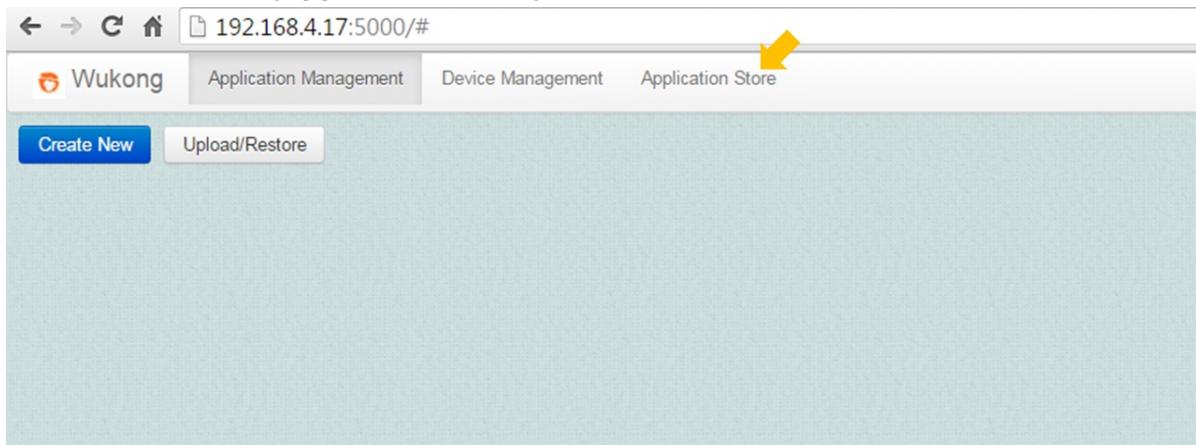
069:[log] Application has been deployed!
068:[log] ...has completed
067:[log] Deploying to node 3232236545, remaining set({})
066:[log] ...has completed
065:[log] Deploying to node 3232236548, remaining set({3232236545})
064:[log] ...has completed
063:[log] Deploying to node 3232236547, remaining set({3232236548, 3232236545})
062:[log] ...has completed
061:[log] Deploying to node 3232236546, remaining set({3232236547, 3232236548, 3232236545})
060:[log] Preparing to deploy to nodes set([1, 3232236546, 3232236547, 3232236548, 3232236545])
059:[log] Compression finished
058:[log] Compressing application code to bytecode format...
057:[log] Generating java application...
056:[log] Preparing java library code...
055:[log] Deploying to node 3232236545, remaining set({})
054:[log] ...has completed
053:[log] Deploying to node 3232236548, remaining set({3232236545})
052:[log] ...has completed

```

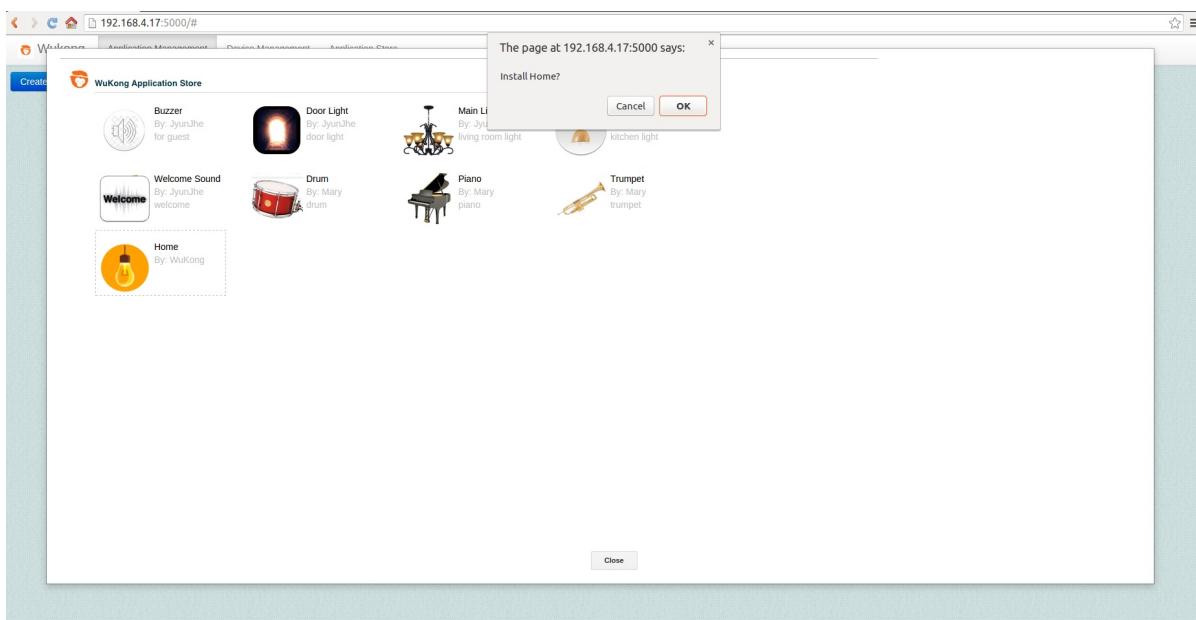
5.4 應用程式商店(Application Store)

除了自己畫資料流編程(flow-based programming)，使用者也可以從 應用程式商店 (Application Store) 選擇應用程式(application)。 應用程式商店 (application store) 中提供了一些基本使用範例供使用者參考。

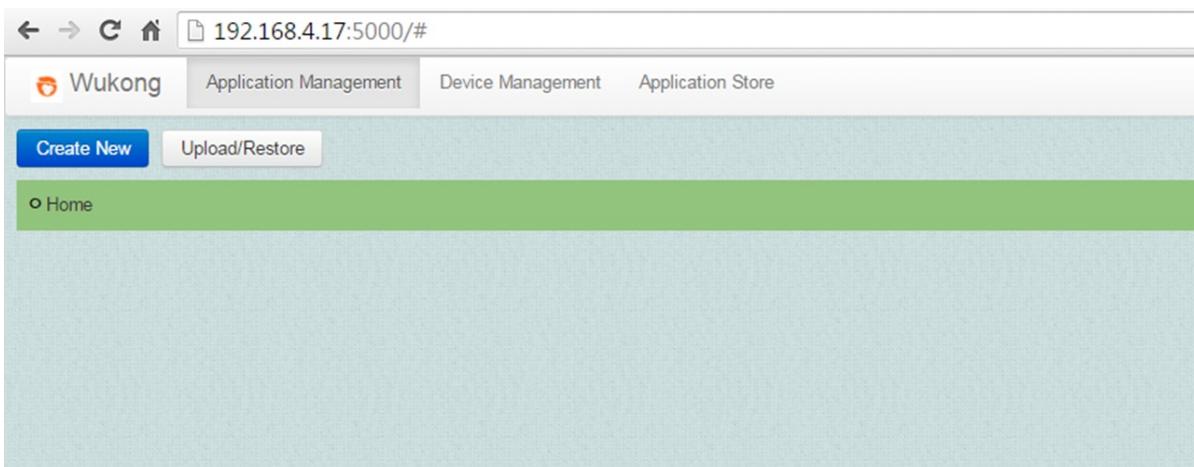
1. 點擊 應用程式商店 (Application Store)



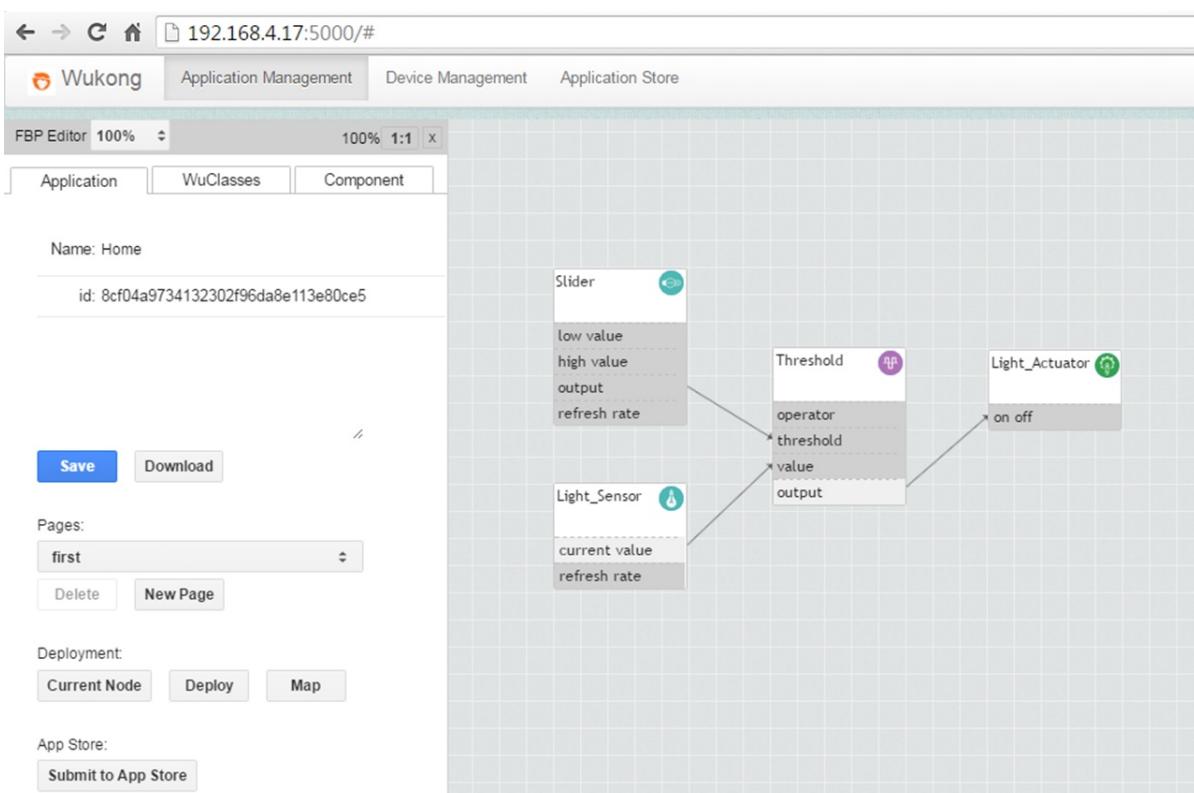
2. 選擇範例應用程式。在此，我們使用"Home"作為範例。



3. 在選好範例後，剛才選擇的應用程式會載入到應用程式管理頁面 (application management)。



- 點選 "Home"，會出現資料流編程編輯畫面。使用者也可以從 "應用程式商店 (application store)" 選擇其他的應用程式。



目前，應用程式商店尚處於起始開發階段，我們需要更進一步討論應用程式該如何存放與分享，使其成為使用者們分享他們物聯網應用程式的渠道。

第六章：動手寫悟空類別元件

這章主要介紹如何在悟空系統中設計新的資料流編程(FBP)元件，讓開發者可以整合新的軟件服務與新的物聯網商品至悟空系統。以下將分成四個小節來說明：

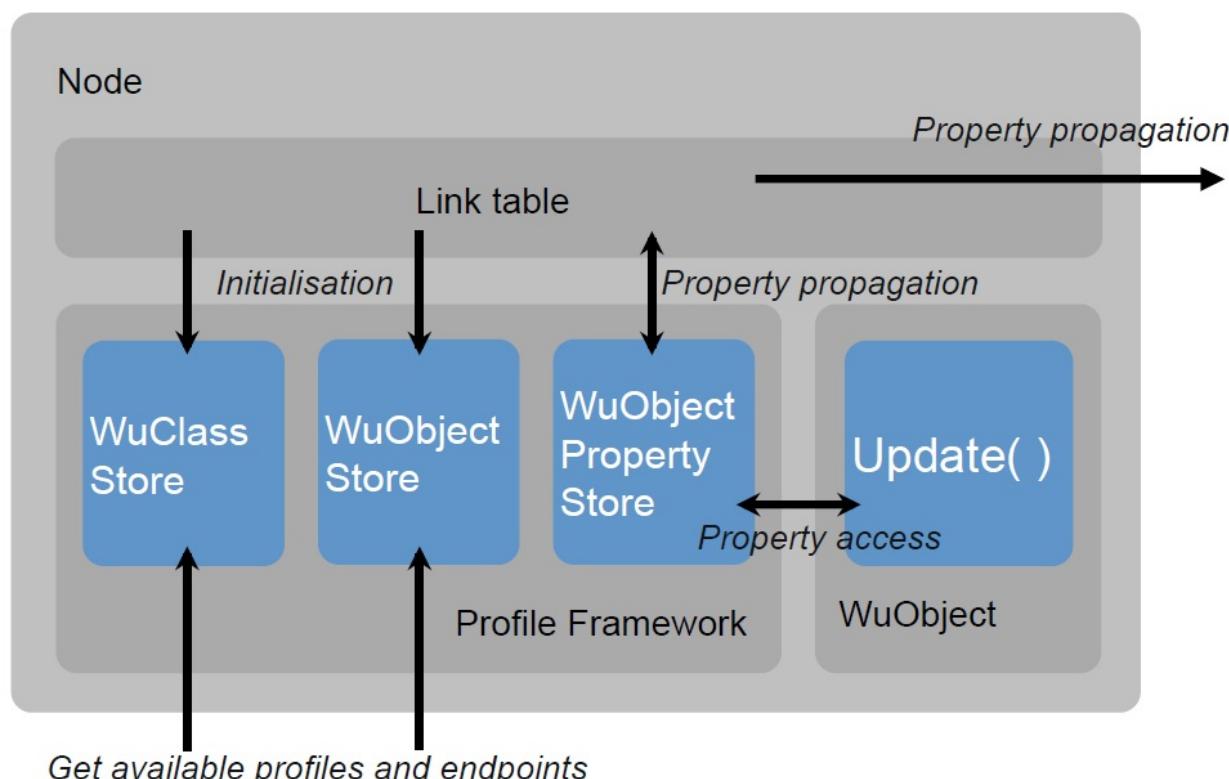
1. 第一節為悟空屬性架構(WuKong Profile Framework, WKPF)，我們將解釋悟空系統的架構，以及這個架構如何執行在單一裝置上。
2. 第二節將說明如何新增一個悟空類別(WuClass)的定義在函式庫中，因為在開始實作元件之前，我們需要先定義元件的屬性和參數，並讓此元件出現在資料流編程介面的清單上。
3. 接者，第三節將介紹實作類別元件的樣版，我們將按照所定義的屬性和參數來實作此類別元件，讓開發者可以在樣版中新增所需要的功能。
4. 最後，我們將介紹現有的範例，這些範例整合了悟空系統與Grove感測器入門套件，這個套件在物聯網的開發板應用上十分普遍。

6.1 悟空屬性架構

在介紹如何實作悟空類別(WuClass)之前，將先解釋悟空裝置(Node)是如何透過悟空類別元件(Component)傳遞資料，這項說明有助於掌握接下來的實作章節。

悟空裝置中的軟體架構可以用以下這張圖清楚地描述：

- 圖中最核心的區塊為屬性架構(Profile Framework, WKPF)，這個架構是用來管理悟空類別與悟空物件(WuObject)如何在裝置上運作。
- 接著，圖中最上方的區塊為資料流表單(Link table)，這張表單是由主控臺(Master)在部署程式時所產生，表單內容記錄了每一筆資料流的來源屬性與目的屬性，因此當資料在傳遞(Property propagation)前，都會先確認這張表單。最後，圖中右下角的區塊為悟空物件，每個裝置可以包含多個悟空物件，而每個物件中都有一個更新函式(Update())，更新函式的執行將視何時被屬性架構呼叫，若該物件的類別有定義刷新速度(refresh rate)這項屬性的話(於下一節詳述)，這個函式會被定期呼叫；
- 另外，當屬性架構被任一個更新函式通知這個物件的屬性有新資料時，這個函式也會被呼叫，這個函式為悟空類別實作的一部分，將於6.3節說明。



屬性架構(Profile Framework, WKPF)

悟空屬性架構運行於悟空裝置上，使得裝置具有服務架構。當主控臺部署一個資料流應用時，屬性架構會監控網路中的每一個悟空裝置，並管理彼此的通訊。

這個架構負責四項主要的功能：

1. 載入悟空類別並產生悟空物件到悟空裝置。
2. 使得主控臺能尋找每個裝置中的類別與物件清單(Get available profiles and endpoints)。
3. 呼叫物件內的更新函式（定期或是被觸發）。
4. 依照資料流表單上的來源與目的，在各個物件屬性間傳遞新的資料。

屬性(Property)

屬性是屬性架構裏最小的資料單位，屬性也是最主要控制悟空物件的方式。正如我們將於下一小節提到的，每一個屬性有四個元素的定義，包含名稱(Name),存取(Access),資料形態(Datatype)以及初始值(Default)。

悟空類別與悟空物件(WuClass and WuObject)

如同名稱所示，悟空類別與物件導向程式中的類別具有相似的定義，各個變數與函式均定義於悟空類別內，並用以產生悟空物件；悟空物件是用以處理應用服務的資料的最小單位，悟空物件均被執行在每個裝置中。

悟空物件屬性儲存區(WuObject Property Store)

每個悟空物件上屬性的資料，均由屬性架構用一個共有的儲存區來管理，屬性架構並提供訪問儲存區的函式(Property access)，讓每個悟空物件在更新函式被呼叫的時候，可以用儲存區的函式存取其資料。這個設計讓屬性架構能監控每個物件屬性的改變，並適時將改變的值傳遞到該資料流的目的地。

更新函式(Update Method)

更新函式實作了悟空類別的行為，這個函式會在該物件屬性的值改變時被執行，或是依照所定義的刷新速度被規律地執行。一旦更新函式被呼叫執行後，悟空物件會透過儲存區的函式讀出被改變的值，並在更新函式中使用。因此，撰寫悟空類別相當於撰寫更新函式。

另外，悟空物件之所以要透過儲存區函式來讀寫值的原因是，悟空物件並不儲存本身屬性的值，這個值會被儲存於屬性架構中，以便於管理資料傳遞與監控，因此，悟空物件需要在更新函式中訪問屬性架構，以讀取更新其屬性的值。

更新函式的執行過程可分為以下三個階段：

1. 向屬性架構的屬性儲存區讀取某一屬性的值
2. 根據更新函式內的程式做資料運算

3. 寫入數值到屬性架構的屬性儲存區以更改某一屬性的值

以一個簡易的按鈕感測器(Button)為例，它的更新函式包含了第2和3階段，但跳過了第1階段，原因是它身為感測器，無需向屬性架構讀取值，而是定期將感測到的值寫入屬性架構。另一個例子是燈光致動器(Light actuator or LED)，由於它沒有輸出的屬性，所以它的更新函式包含了第1和2階段，但跳過了第3階段，一旦當它的更新函式被呼叫時，它會向屬性架構讀取值，來決定是否開燈或關燈。

資料流表單(Link Table)

這個表單記錄了每一筆資料流的來源屬性與目的屬性，並由主控臺在部署應用服務時傳給悟空裝置。

訪問儲存區的函式(Property Access)

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

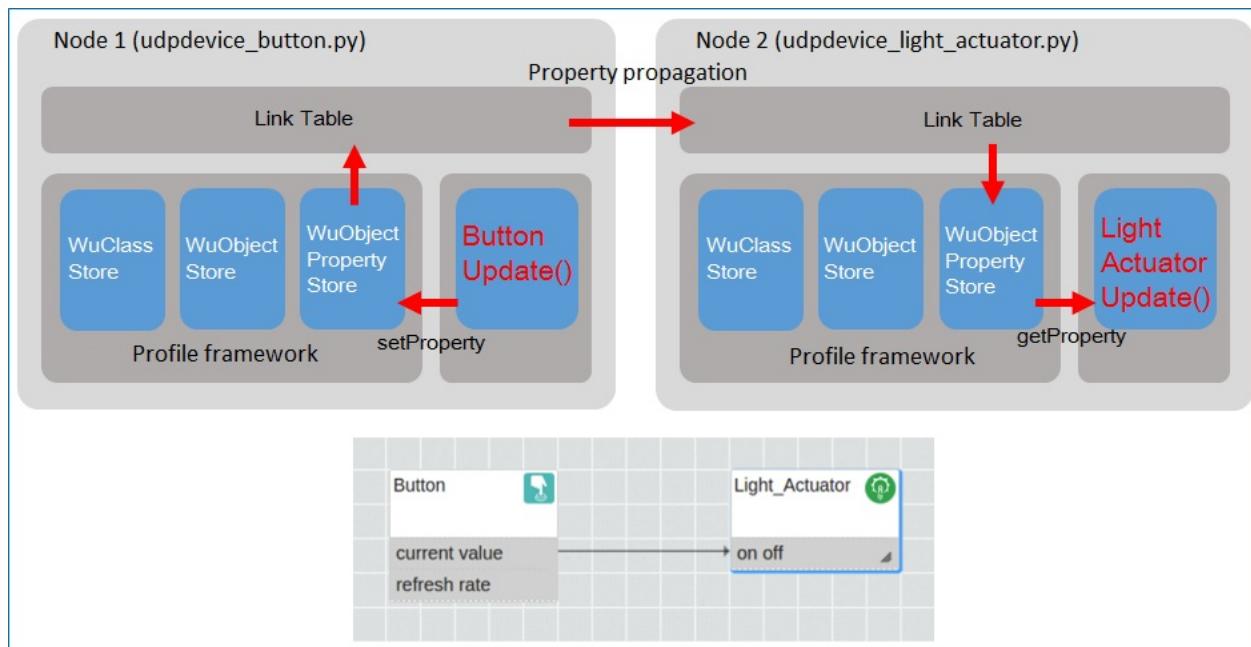
`setProperty` 可以讓悟空物件將某屬性的值寫入到儲存區

`getProperty` 可以讓悟空物件從儲存區讀取某屬性的值

此函式實作在 <https://github.com/wukong-m2m/wukong-darjeeling/blob/release0.4/wukong/gateway/udpkpf/udpkpf.py>。

在接下來的章節中，我們會時常使用這兩個函式訪問儲存區。

範例：用按鈕控制燈光開關



經過以上的說明，我們將比較4.2節的執行步驟與這張圖的關聯，透過這個比較，能進一步了解悟空如何讓按鈕控制燈光開關。

1. 新增裝置(Add Node)

當裝置被新增之後，裝置會先進行初始化，在初始化的過程中，會依序載入悟空類別與產生悟空物件。

2. 尋找裝置(Discovery)

當我們按下"尋找裝置"的按鈕時，主控臺會收集每個裝置中的類別與物件清單(Get available profiles and endpoints)。

3. 配對(Map)

主控臺會依照所取得的類別與物件清單，從中配對資料流編程(FBP)所需要的悟空物件。以此圖為例，資料流編程所需要的悟空物件為1個按鈕(Button)和1個燈光(Light Actuator)，由於裝置1(Node 1)有1個按鈕物件，裝置2(Node 2)有1個燈光物件，因此，資料流編程的按鈕將會被配對至裝置1，資料流編程的燈光將會被配對至裝置2。

4. 部署程式(Deploy)

當我們按下"部署程式"按鈕時，每個裝置會得到從主控臺來的資料流表單，用以傳遞需要被更新的屬性的值，同時也會得到資料流編程所需要的悟空物件清單。

5. 測試(Testing)

當我們按下按鈕後，由於按鈕的悟空類別有定義刷新速度屬性，所以裝置1上的按鈕悟空物件的更新函式會被定期執行，並且定期將感測到的資料寫入屬性儲存區，此時，當悟空屬性架構在儲存區發現按鈕的第一個屬性(current value)改變時，就會按照資料流清單上的來源與目的，將這個改變的值傳遞給裝置2。

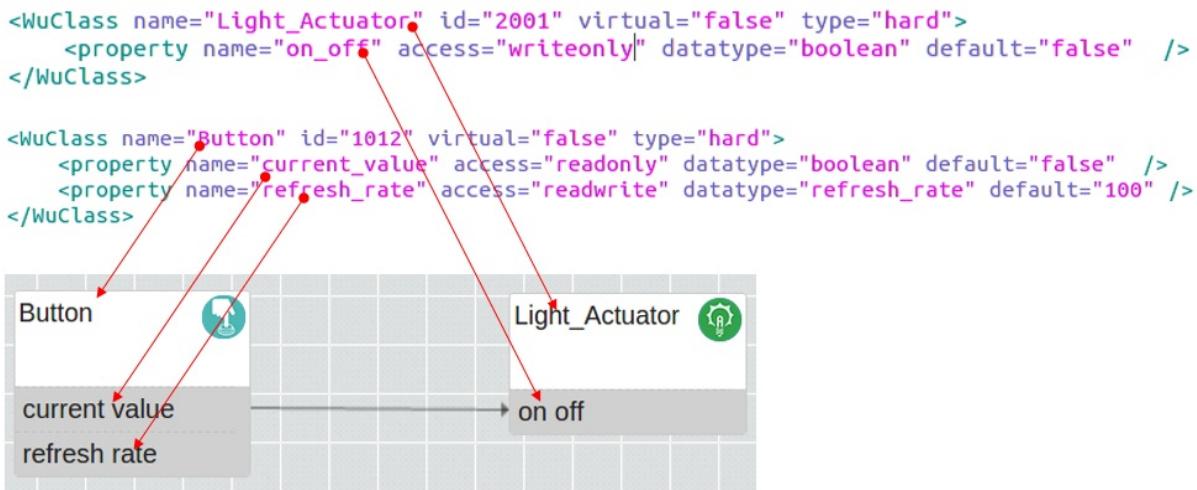
接著，裝置2的屬性架構會發現其儲存區有屬性改變，便呼叫具有該屬性的燈光悟空物件的更新函式，在函式執行的過程中，燈光悟空物件會拿到被更新的屬性的值，也就是燈光悟空物件的第一個屬性(on off)，最後，按照這個屬性的值來開關燈光。

6.2 新增一個悟空類別的定義至主控台

在開始實作悟空類別之前，我們需要先決定悟空類別的介面，包括具有哪些屬性，以及各屬性的參數，並將這個介面定義寫入函式庫 <https://github.com/wukong-m2m/wukong-darjeeling/blob/release0.4/wukong/ComponentDefinitions/WuKongStandardLibrary.xml>，如同以下的步驟：

- 定義悟空類別名稱(**WuClass name**)與各屬性名稱(**property name**)

下圖為比較4.2節的資料流編程範例與其悟空物件的定義，從中可觀察出兩者的名稱定義是一致的，此外，元件的屬性數量也與所定義的屬性數量相同。以按鈕(Button)來說，我們定義了兩個屬性名稱(current_value, refresh_rate)，因此，按鈕的資料流編程元件也就有兩個屬性；以燈光(Light Actuator)來說，我們只定義了一個屬性名稱(on_off)，因此，燈光的資料流編程元件只有一個屬性。



- 定義悟空類別的識別號(**id**)

每一個悟空類別都需要給予一個獨特的識別號，一般我們習慣將運算用的悟空類別從號碼1開始給予；將感測用的悟空類別從號碼1001開始給予；將控制用的悟空類別從號碼2001開始給予。

- 定義每個屬性的存取(**access**)

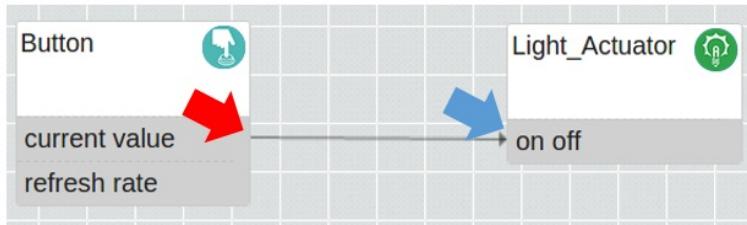
透過定義每個屬性的存取，我們可以決定該屬性為資料流上的來源或是目的，以按鈕的第一個屬性(current_value)來說，這個屬性的存取定義為唯讀(readonly)，代表我們預設這個屬性是作為資料流的來源；以燈光的第一個屬性(on_off)來說，這個屬性的存取定義為唯寫(writeonly)，代表我們預設這個屬性是作為資料流的目的；若屬性的存取定義為可讀可寫(readwrite)，則這個屬性可以同時作為某一個資料流的來源以及另一個資料流的目的。

```

<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>

```



- 定義每個屬性的資料型態(**datatype**)

每個屬性都需要定義資料型態，而目前可支援的資料型態為短整數型態(short integer, 16 bit), 布林型態(boolean), 列舉型態(enum)和刷新速度型態(refresh rate)。

短整數與布林型態的理解較為直接，於是我們從列舉型態開始說明。使用列舉型態的目的是，讓資料流編程以及悟空類別的程式碼更容易被讀懂，此型態常被用在邏輯運算用的悟空類別，如臨界值比較(Threshold)和四則運算(Math)。

接者是刷新速度型態，此型態是為了感測器的更新函式而設計的，用來定義更新函式的執行頻率，另外，按照一般對感測器的量測要求，刷新速度值的單位設為毫秒(ms)，但由於屬性架構的資料型態有16位元的限制，若設定超過上限會被截除。

以按鈕感測器為例，若按鈕的屬性定義中沒有刷新速度的資料型態的話，按鈕的更新函式將不會被屬性架構呼叫，按鈕的值也不會定期更新到屬性儲存區內。

- 定義每個屬性的初始值(**default**) 每個屬性都需設定初始值，若沒有定義初始值，則預設值為零。以按鈕為例，若刷新速度的初始值設為100，則按鈕的更新函式會每100毫秒執行一次，若未給予按鈕的刷新速度初始值，按鈕的刷新速度就會為零，代表按鈕的更新函式將不會被執行。

- 定義悟空類別的進階參數(**virtual and type**)

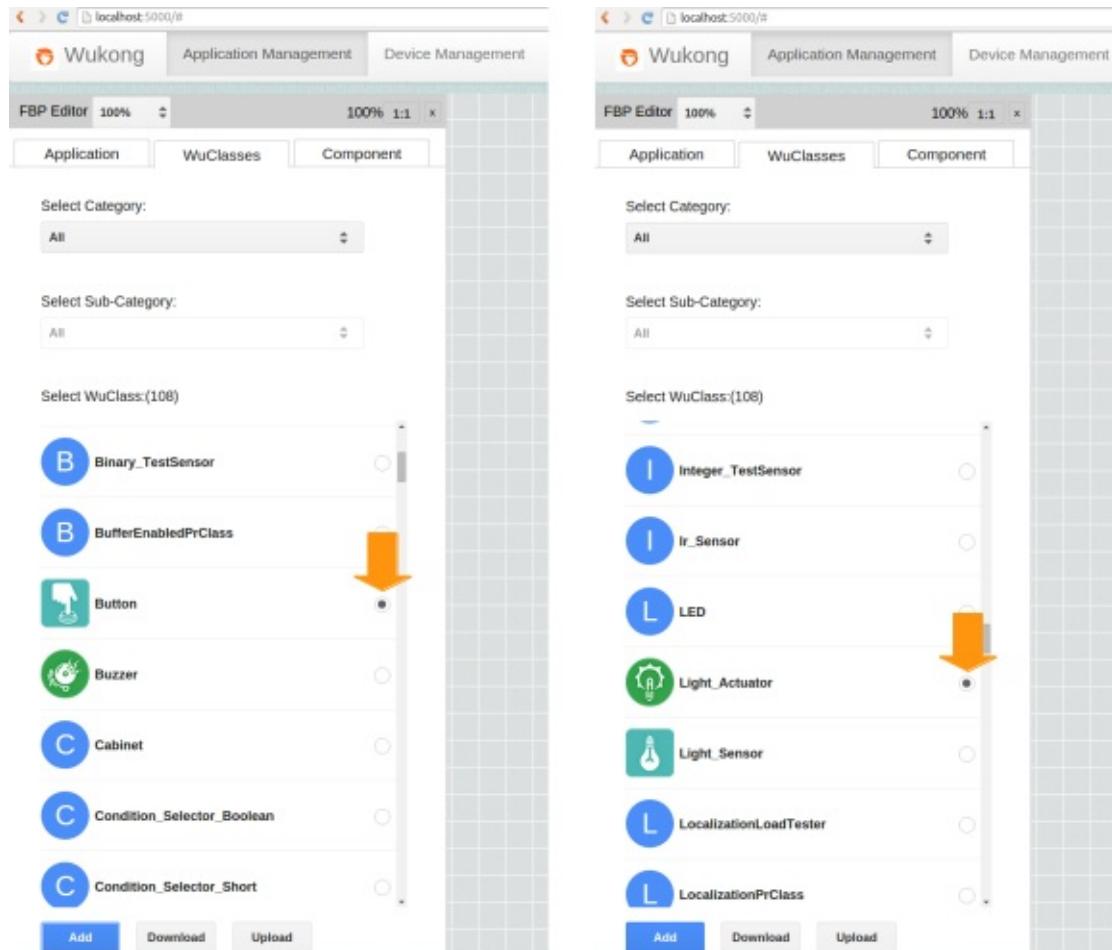
這份文件將先跳過進階參數的描述，直到下一個版本再做詳細說明。目前請先參考最相近的悟空類別來給予定義，舉例來說，若要新增的悟空類別屬於感測器，則可以參考按鈕(Button)的進階參數定義；若要新增的悟空類別屬於控制物件，則可以參考燈光(Light Actuator)的進階參數定義。

- 刪除暫存檔並重啓主控臺

當新增悟空類別至函式庫(WuKongStandardLibrary.xml)後，我們需要先把之前的暫存檔清除，確保主控臺會產生新的悟空類別清單，接者才重啓主控臺，以下為執行步驟：

```
cd <path_of_source_code>/wukong-darjeeling/wukong/master/static/js/
rm __comp_*
# 接者，清除Chrome瀏覽器關於http://localhost:5000的歷史資料
# 重啓主控臺
```

完成以上步驟並重啓主控臺後，在資料流編輯介面的左側悟空類別清單上，就可以看到新增的悟空類別，如同下圖的按鈕與燈光。



6.3 實作悟空類別

在這一章節中，我們會搭配4.2節的資料流編程範例來介紹如何實作悟空類別，實作出來的悟空類別可以同時用於Edison, Galileo和Raspberry Pi，使用的方式如同5.2節所述。

悟空裝置的Python樣版

以下為悟空裝置的樣版，從樣版中可看到，悟空裝置會加入悟空類別(addClass)，並產生悟空物件(addObject)。目前所有的悟空裝置程式都存放在 <https://github.com/wukong-m2m/wukong-darjeeling/tree/release0.4/wukong/gateway/udpkpf>

```
from twisted.internet import reactor
from udpkpf 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()
```

悟空裝置範例程式：按鈕和燈光

此範例程式是根據以上的樣版所寫成，在此裝置樣版中實作了兩個悟空類別，一個是第10行的按鈕(Button)，另一個是第25行的燈光(Light_Actuator)，有了這兩個悟空類別的實作，我們就可以照者4.2節的步驟來部署應用程式，因此，在往下看程式之前，建議可以先照者4.2節所述的步驟操作一遍，將有助於對此程式的理解。接者，每個悟空類別都有初始函式(init)與更

新函式(update)，初始函式是用來設定各變數，狀態與硬體參數的起始值，而更新函式則是用來被屬性架構重覆地呼叫執行。最後，當悟空裝置範例程式被執行後，這兩個悟空類別將會被加入(addClass)悟空裝置中，並用來產生(addObject)悟空物件。

```

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)
21                 print "Button pin: ", Button_Pin, ", value: ", current_value
22             except IOError:
23                 print "Error"
24
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):          ❺
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) ❸
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 <ip> <dip>:<port>' % sys.argv[0]
59             print '      <ip>: IP address 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() ❹
67         device_cleanup()

```

❶ 為了透過GPIO控制按鈕和燈光，我們需要在悟空裝置最前面匯入(import)物聯網開發板的GPIO函式庫，而且，為了盡可能讓同一個悟空裝置程式能執行在不同的物聯網開發板，我們新增了這個介面(udpdevice_io_interface.py)來決定如何使用三個不同的GPIO函式庫，下一個章節會繼續說明這個GPIO介面。

❷ 這個函式的輸入參數必須是悟空類別的名稱，並且，此名稱要與WuKongStandardLibrary.xml所定義的名稱相同，當這個函式取得悟空類別名稱後，就會分析WuKongStandardLibrary.xml所定義的內容，截取這個悟空類別的每個參數。

❸ 使用第1點所提的GPIO介面來設定哪個GPIO腳位會被用到，並設定該腳位是數位(digital)還是類比(analog)，以及設定是輸入(input)或輸出(output)。

❹ 由於在6.2節中，按鈕感測器有定義刷新速度(refresh rate)屬性，所以更新函式(update)會定期被執行並透過GPIO讀取其感測到的值。

❺ setProperty函式是用來將某屬性的值寫入屬性架構的屬性儲存區，當屬性架構監測到此屬性的值改變後，就會按照資料流表單來傳遞值。setProperty函式的第一個參數是屬性辨別號(property ID, pID)，在第20行中，pID等於零表示按鈕的第1個屬性的值將被寫入屬性儲存區，若我們對照WuKongStandardLibrary.xml，可知道按鈕的第1個屬性是current_value，也就是說，按鈕的感測數值會被寫入屬性儲存區。

❻ 由於燈光悟空類別沒有定義刷新速度(refresh rate)，所以燈光的更新函式只會在其屬性的值改變時被呼叫。以4.2節的範例來說，當按鈕被按下去時，屬性架構會知道按鈕的第1個屬性的值已改變，於是就照着資料流清單把值傳遞給燈光屬性儲存區，並且呼叫燈光的更新函式，當函式被執行時，就會照者讀取到的值來開關燈光。在屬性架構呼叫更新函式時，同時也會傳遞改變的obj(悟空物件),pID(屬性識別號)和value(值)給更新函式做運用。

⑦ 這個函式爲用來加入悟空類別到悟空裝置，在這個範例程式中，按鈕和燈光均加入到同一個悟空裝置程式，所以會共同使用一個屬性架構。關於此函式的第2個輸入參數，是用來表示這個悟空類別可否自行產生悟空物件，零代表此悟空類別不能自行產生悟空物件，因此需要用下一行的函式(`addObject`)來產生悟空物件。在7.3節時，我們將會使用到能自行產生悟空物件的悟空類別，屆時就不需要再用`addObject`來產生悟空物件。

⑧ 由於屬性架構的實作是用Python的Twisted函式庫，所以最後記得要加這行程式，否則按鈕的值不會被定期更新。

要點

這一章節的最後，我們歸納出實作悟空類別的幾個要點：

- 首先是新增並仿照悟空裝置樣版到
`<path_of_source_code>/wukong/gateway/udpwkpf/udpdevice_XXX.py`
- 若是要新增感測器(sensor)，請參考按鈕的悟空類別寫法；若是要新增控制器(acutator)，請參考燈光的悟空類別寫法。
- 最後加入悟空類別到裝置上(`addClass`)，並在裝置上產生悟空物件(`addObject`)。

6.4 悟空類別範例：Grove模組

在悟空類別的實作介紹之後，這章節會繼續了解一些相仿的範例，讓開發者能夠針對個別應用的需要，來實作新的悟空類別。

我們的範例是使用Grove模組(http://www.seeedstudio.com/wiki/Grove_System)，選擇Grove模組的原因是，這個模組可以透過各家的擴充板，輕易地連接到物聯網開發板(Edison, Galileo and Raspberry Pi)，並且，Grove系列提供了非常豐富的元件與函式庫給物聯網的開發者。

關於更多Grove開發模組(Grove Starter Kit)的資訊，請參考以下的網頁：

使用Intel Edison or Galileo: <https://software.intel.com/iot/hardware/devkit>

使用Raspberry Pi: <http://www.dexterindustries.com/grovepi/>

輸入/輸出介面(IO Interface)

如同上節所述，為了讓同一個Grove模組的範例能執行在Edison, Galileo和Raspberry Pi，我們新增了這個介面(udpdevice_io_interface.py)來決定如何使用三個不同的GPIO函式庫，對Edison, Galileo來說，所需要匯入的GPIO函式庫為mraa；對Raspberry Pi來說，所需要匯入的GPIO函式庫為rpi，若使用Grove Pi擴充板的話，所需要匯入的GPIO函式庫則為grovepi，因此，在這個介面的一開始(如下圖)，我們定義三種裝置的選項：

DEVICE_TYPE_MRAA, DEVICE_TYPE_GPIO以及DEVICE_TYPE_RPI，並於第12行決定要使用哪一個選項來執行Grove模組的範例。

```

1  DEVICE_TYPE_MRAA = 0 # Edison, Galileo
2  DEVICE_TYPE_RPI  = 1 # Raspberry Pi
3  DEVICE_TYPE_GPIO = 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.BCM)
20 elif device_type == DEVICE_TYPE_GPIO:
21     import grovepi
22 else:
23     raise NotImplementedError

```

GPIO使用方式

除了用這個介面決定使用哪一種裝置外，這個介面也定義了如何在裝置上使用GPIO，目前為止，我們在這個介面中定義了6個使用GPIO的共同方式，如下所示：

```
pin_mode(pin, pin_type, pin_mode, **kwargs)
# 這個函式是用來初始化GPIO腳位(pin)的設定，包括型態(pin_type)是數位或類比(digital/analog)，
# 或者模式(pin_mode)是輸入或輸出。

digital_read(pin_obj)
# 若腳位的型態設定為數位，模式設定為輸入，我們可以用這個函式來讀取GPIO的值，
# pin_obj是pin_mode的回傳物件。

digital_write(pin_obj, val)
# 若腳位的型態設定為數位，模式設定為輸出，我們可以用這個函式來寫入GPIO，
# pin_obj是pin_mode的回傳物件，val是要寫入GPIO的值。

analog_read(pin_obj)
# 若腳位的型態設定為類比，模式設定為輸入，我們可以用這個函式來讀取GPIO，
# pin_obj是pin_mode的回傳物件。
# 但由於Raspberry Pi本身不支援類比輸入，
# 所以此函式只適合用在Edison, Galileo以及使用grovepi擴充板的Raspberry Pi。

analog_write(pin_obj, val)
# 若腳位的型態設定為類比，模式設定為輸出，我們可以用這個函式來寫入GPIO，
# pin_obj是pin_mode的回傳物件，val是要寫入GPIO的值。
# 但目前只有使用grovepi擴充板的Raspberry Pi支援類比輸出。

temp_read(pin_obj)
# 這是針對Grove模組中的溫度感測器所定的介面函式，
# pin_obj是pin_mode的回傳物件。
# 但由於Raspberry Pi本身不支援類比輸入，
# 所以此函式只適合用在Edison, Galileo以及使用grovepi擴充板的Raspberry Pi。
```

目前關於Grove開發模組的悟空裝置程式，大都是使用以上的介面函式來讀取或寫入開發板的GPIO，然而，這個介面目前只限於使用數位和類比GPIO，尚未支援共同的PWM,I2C,SPI，以及UART介面，主要的原因是，各個GPIO函式庫在處理數位和類比GPIO的方式較為接近，所以容易定義一個共同的介面，但其餘的傳輸方式就有許多差異，因此，目前若需使用PWM,I2C,SPI，以及UART傳輸的Grove模組時，就不會使用這個介面，而是針對各個開發板獨自開發，比方說蜂鳴器(Buzzer)，mraa函式庫是用PWM來控制蜂鳴器，但grovepi函式庫只支援用類比輸出(analog write)來控制蜂鳴器，於是蜂鳴器的範例就沒有使用這個介面。

Grove開發模組範例(Grove Starter Kit)

以下列出使用各個GPIO介面函式的範例：

- 使用數位輸入GPIO的按鈕(Button)

```

4  from udpwkpf_io_interface import *
5
6  Button_Pin = 5
7
8  class Button(WuClass):
9      def __init__(self):
10         WuClass.__init__(self)
11         self.loadClass('Button')
12         self.IO = pin_mode(Button_Pin, PIN_TYPE_DIGITAL, PIN_MODE_INPUT)
13
14     def update(self,obj,pID=None,val=None):
15         try:
16             current_value = digital_read(self.IO)
17             print "Button value: %d" % current_value
18             obj.setProperty(0, current_value)
19         except IOError:
20             print "Error"

```

- 使用數位輸出GPIO的繼電器(Relay)

```

9  class Relay(WuClass):
10     def __init__(self):
11         WuClass.__init__(self)
12         self.loadClass('Relay')
13         self.relay_gpio = pin_mode(Relay_Pin, PIN_TYPE_DIGITAL, PIN_MODE_OUTPUT)
14         print "Relay init success"
15
16     def update(self,obj,pID=None,val=None):
17         try:
18             if pID == 0:
19                 if val == True:
20                     digital_write(self.relay_gpio, 1)
21                     print "Relay On"
22                 else:
23                     digital_write(self.relay_gpio, 0)
24                     print "Relay Off"
25             else:
26                 print "Relay garbage"
27         except IOError:
28             print ("Error")

```

- 使用類比輸入的旋轉電位器(Slider)

```

9  class Slider(WuClass):
10     def __init__(self):
11         WuClass.__init__(self)
12         self.loadClass('Slider')
13         self.slider_aio = pin_mode(Slider_Pin, PIN_TYPE_ANALOG)
14         print "Slider init success"
15
16     def update(self,obj,pID=None,val=None):
17         try:
18             current_value = analog_read(self.slider_aio)
19             obj.setProperty(2, current_value)
20             print "Slider analog pin: ", Slider_Pin, ", value: ", current_value
21         except IOError:
22             print ("Error")

```

- 使用特定函式的溫度感測器(**Temperature Sensor**)

```

6  from udpwkpf_io_interface import *
7
8  PIN = 3 #Analog pin 0
9
10 class Temperature_sensor(WuClass):
11     def __init__(self):
12         WuClass.__init__(self)
13         self.loadClass('Temperature_Sensor')
14         print "temperature sensor init!"
15
16     def update(self,obj,pID=None,val=None):
17         try:
18             current_value = temp_read(PIN)
19             obj.setProperty(0, current_value)
20             print "WKPFUPDATE(Temperature): %d degrees Celsius" % current_value
21         except IOError:
22             print ("Error")

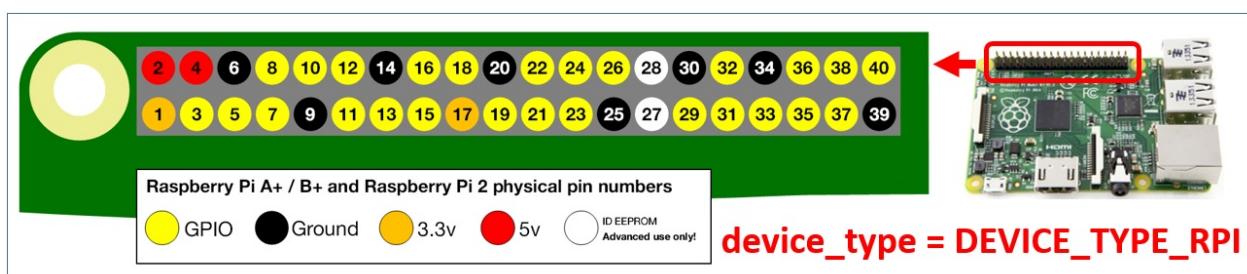
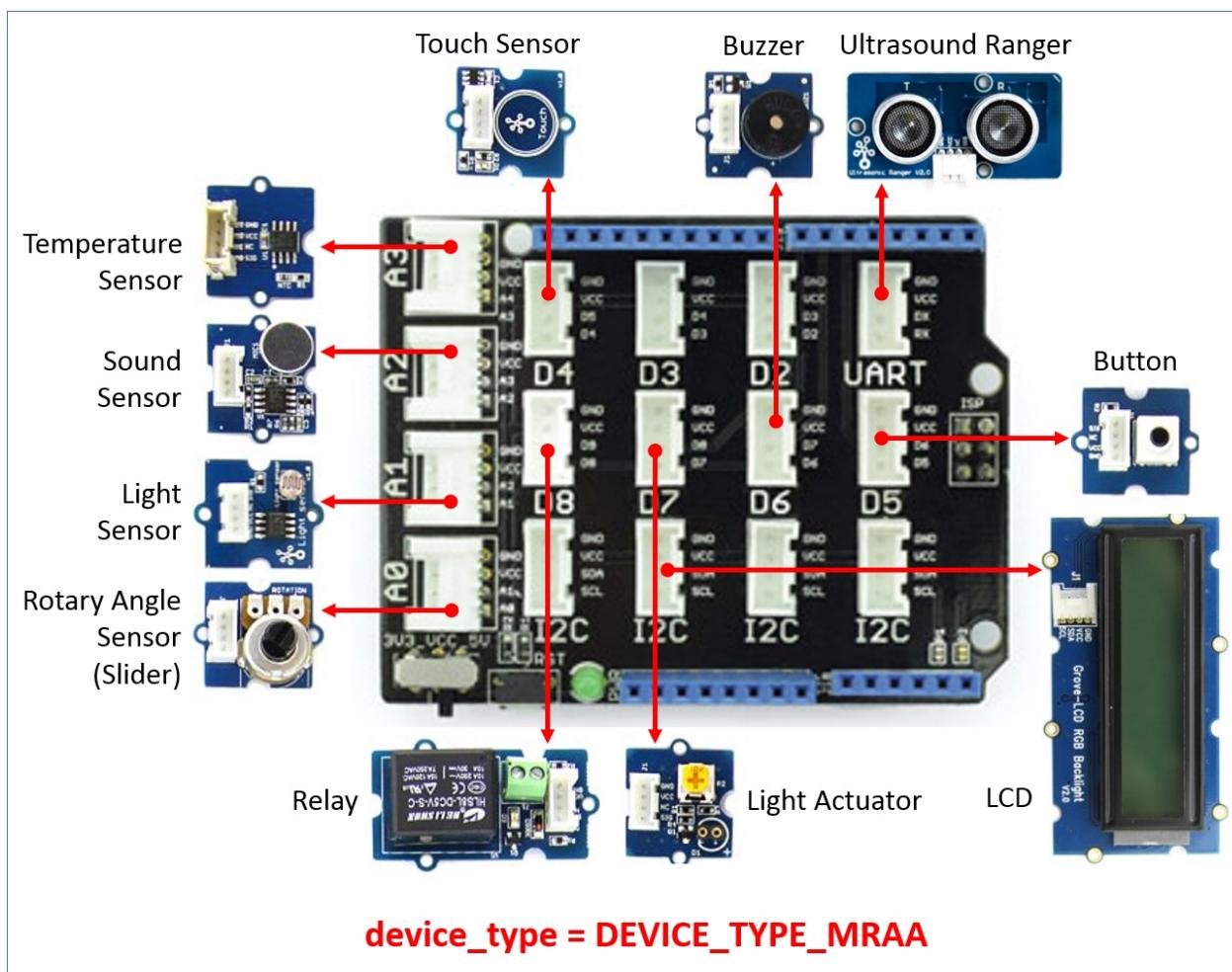
```

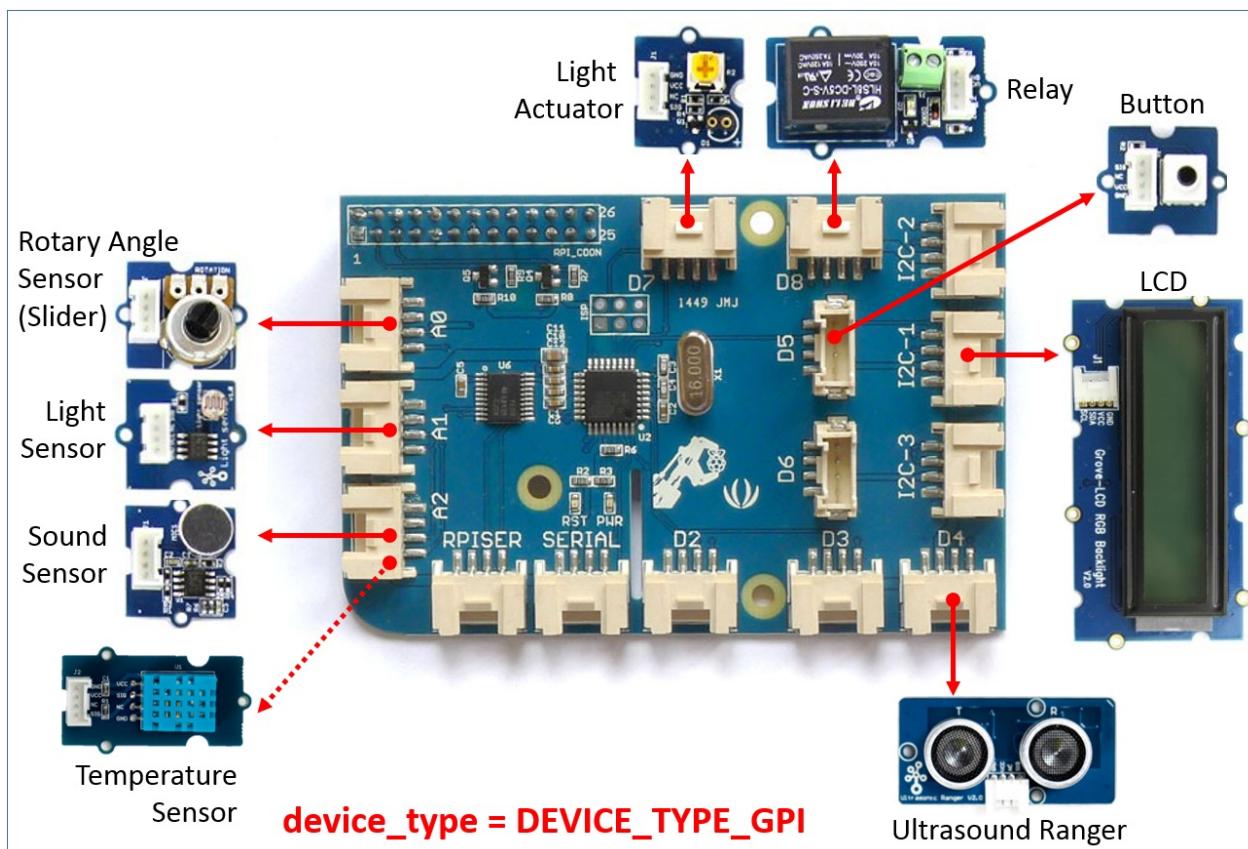
這些範例程式均存放於：

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

開發板的腳位編號

在選擇使用哪一個裝置執行Grove開發模組的範例後，我們需要確認範例程式中所定義的腳位編號，並將Grove開發模組連接到合適的位置，以下分別是Edison的Grove擴充板,Raspberry Pi與其GrovePi擴充板的腳位定義，這些腳位編號通常定義在每個悟空裝置程式的最前面。





第七章：建立與連接非悟空系統(Non-WuKong)的服務

在這章中，我們將透過一些範例來介紹如何建立與非悟空系統所提供的服務通訊，並在悟空類別中使用TCP封包(TCP socket)與網頁封包(Web socket)的通訊協定，有了這些通訊方式，我們就能將現有的資料流連接至外部的網頁服務以及市售的物聯網商品，以實現更有彈性與強大的分散式物聯網應用。

這章的悟空類別範例將會比上一章的複雜，有的悟空類別是使用該產品專有的API來取得服務，有的悟空類別則是用網頁存取的協定(web-access protocols)來取得服務，因此，為了讓這些範例能被清楚地呈現，部分非必要的實作細節將被省略，介紹將着重於如何與各式的物聯網商品建立資料流。以下是各章節範例的簡述：

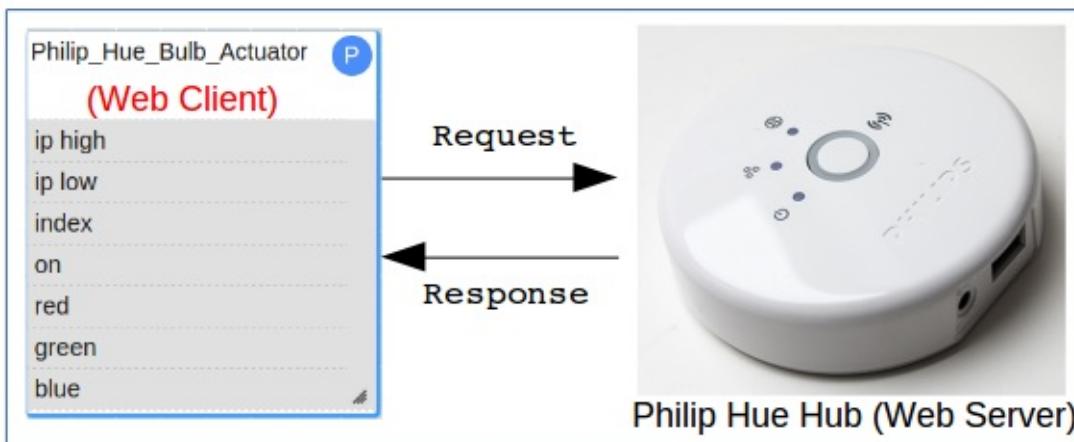
- 使用TCP伺服器(TCP Server)連線的悟空類別

在這個範例中，我們使用Python的Twisted函式庫來建立TCP連線，將悟空類別與Intel RealSense攝影機連接起來，如此一來，每當攝影機接受到新的輸入訊號後，都會將讀取到的值傳送給悟空類別，這樣的通訊方式也適合用於與Intel RealSense攝影機相仿的產品，如Kinect for Xbox One。



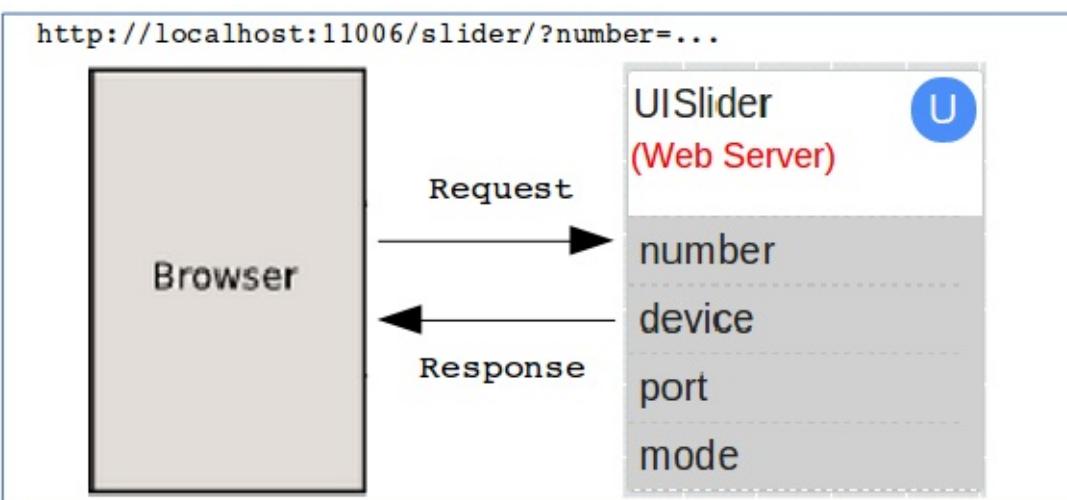
- 使用網頁客戶端(Web Client)連線的悟空類別

在這個範例中，我們使用Python的Twisted函式庫來傳送HTTP請求訊息(HTTP requests)給Philip Hue燈泡系列產品，當Hue的伺服器接受到訊息後，該伺服器就會依照訊息內容來調控Hue的燈光。



- 使用網頁伺服器(Web Server)連線的悟空類別

在這個範例中，我們使用Python的Twisted函式庫來實作一個網頁伺服器，用來接收瀏覽器發出的網址請求(URL requests)，只要我們在網頁伺服器中設計好HTTP APIs，並實作使用這個API的網頁客戶端(Web Client)，就能透過這個網頁伺服器，使外部的瀏覽器或應用程式控制悟空類別的輸出。

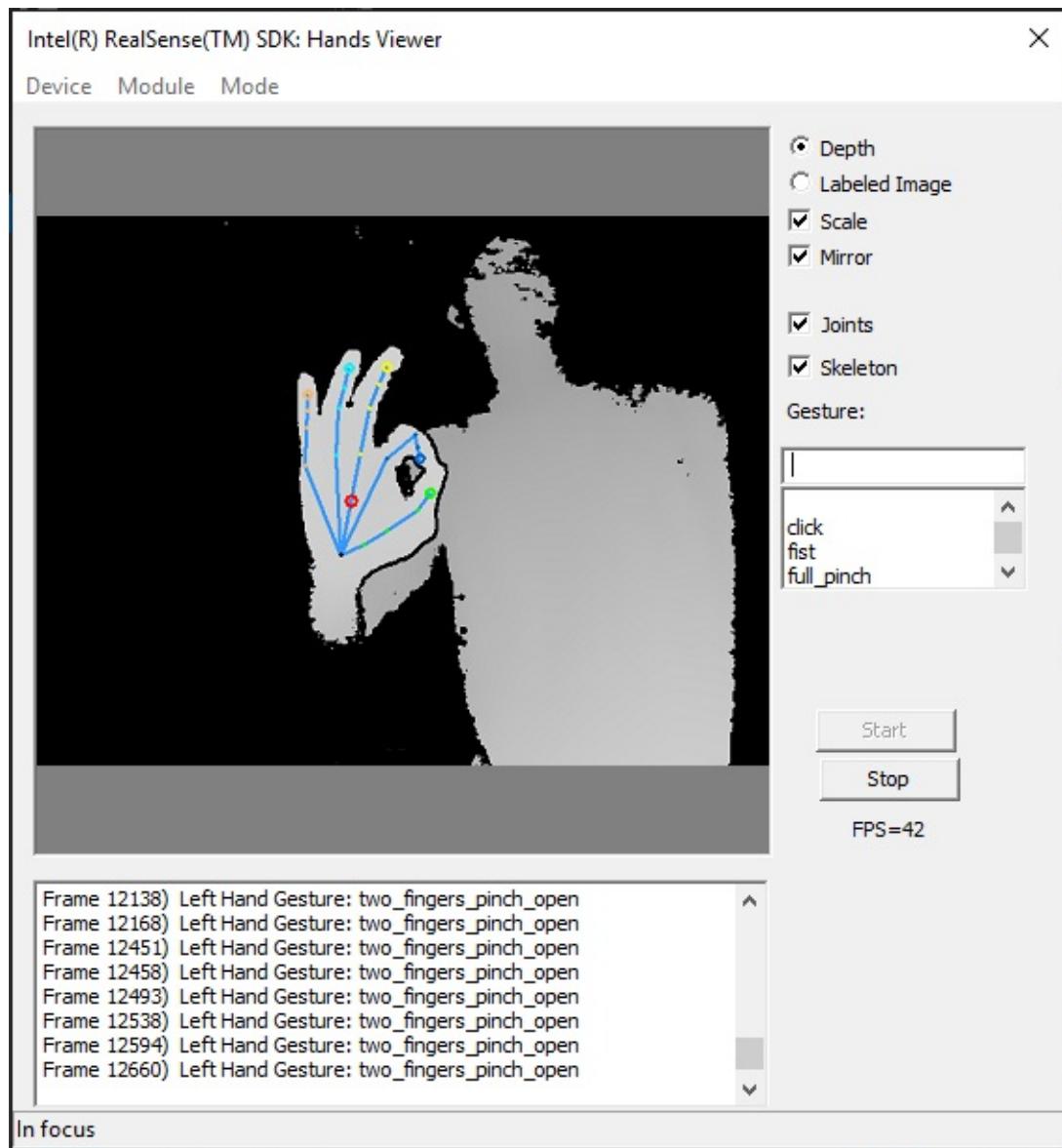


7.1 TCP伺服器連線

在這個章節中，我們介紹如何將Intel RealSense SDK與悟空資料流編程的物件透過TCP互相連線，同時也介紹如何在悟空資料流編程的介面上，部署使用RealSense攝影機的物聯網應用程式。

• 將Intel RealSense攝影機與悟空類別互相連線

1. Intel RealSense攝影機提供了許多專屬的功能，如臉部辨識，手勢辨識，聲音分析與擴增實境，在這個章節中，我們只使用手勢辨識的功能，目的是展示如何在悟空類別內新增TCP客戶端(TCP Client)，而Intel RealSense攝影機的手勢辨識範例可以直接從它的SDK(<https://software.intel.com/en-us/realsense/home>)取得，範例的檔案資料夾位置在Intel/RSSDK/sample/FF_HandsViewer/.
2. 為了要將RealSense攝影機捕捉的手勢訊號傳給悟空物件，我們在原有的手勢辨識範例中加了一些檔案與函式，用來定義如何傳送訊號，這份改版的程式可以從以下連結(<https://drive.google.com/open?id=0B31WAUDq3a3uZWZzb21iR0NwMDQ>)取得，若要使用這份下載的程式，需要取代掉這個資料夾下(Intel/RSSDK/sample/FF_HandsViewer/src)的原有檔案，接者用Microsoft Visual Studio來打開名稱為FF_HandsViewer_vs201*的SLN檔。當RealSense攝影機連接至USB3.0的插槽後，便可執行SLN檔，執行後跳出的畫面如下，代表攝影機已在接收手勢訊號並做辨識分析。



在 HandsViewer.cpp 的檔案中，新增TCP客戶端的連線是發生在第535行，接者將這個 TCP 檔案描述子(socket file descriptor)傳送給 DisplayGesture 函式，這個函式會持續地辨認手勢，並將事前定義好的手勢的值用 SendMessage 傳送給悟空類別上的 TCP 伺服器，如此一來，收到訊號的悟空類別就可進一步做反應。

3. 而這個悟空類別的實作是在 `udpdevice_mux_gesture_control.py`，如同下圖中的 `RealSenseProtocol` 類別所示，這個類別主要的功能在於分析與反應 TCP 客戶端傳回來的資料。

```

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 這個類別繼承了basic.LineReceiver，這是Twisted函式庫內用來通訊的一種協定，它有許多種接收TCP連線訊號的模式，其中一種是lineReceived模式，用lineReceived函式來接收一整行(lines)的數據。詳細的資訊可以參考以下網頁：

<https://twistedmatrix.com/documents/9.0.0/api/twisted.protocols.basic.LineReceiver.html>

❷ 在lineReceived模式中，每接收到一筆數據，lineReceived函式就會被呼叫(callback)，換句話說，lineReceived會接收到其它TCP客戶端的封包，也就是手勢辨識範例(HandsViewer.cpp)透過SendMessag函式所傳遞的。

❸ 一旦接收到數據，便會按照程式的設計被分析與處理。在這個悟空類別中，我們定義了模式(mode)與手勢(gesture)兩種屬性，手勢的值是在RealSense範例中所事先定義好的數字，而模式屬性的設計目的是，為了可以讓同一組手勢運用在不同的場景，所以模式屬性的值就是用來做場景的切換，由於在RealSense範例中，不能捕捉比數字的手勢，於是我們選擇用它內建的輕拍(Tap)手勢來表示要更換模式，並且，輕拍手勢很像在按虛擬的按鈕，手勢屬性的值也與按鈕一樣是布林值(True or False)。

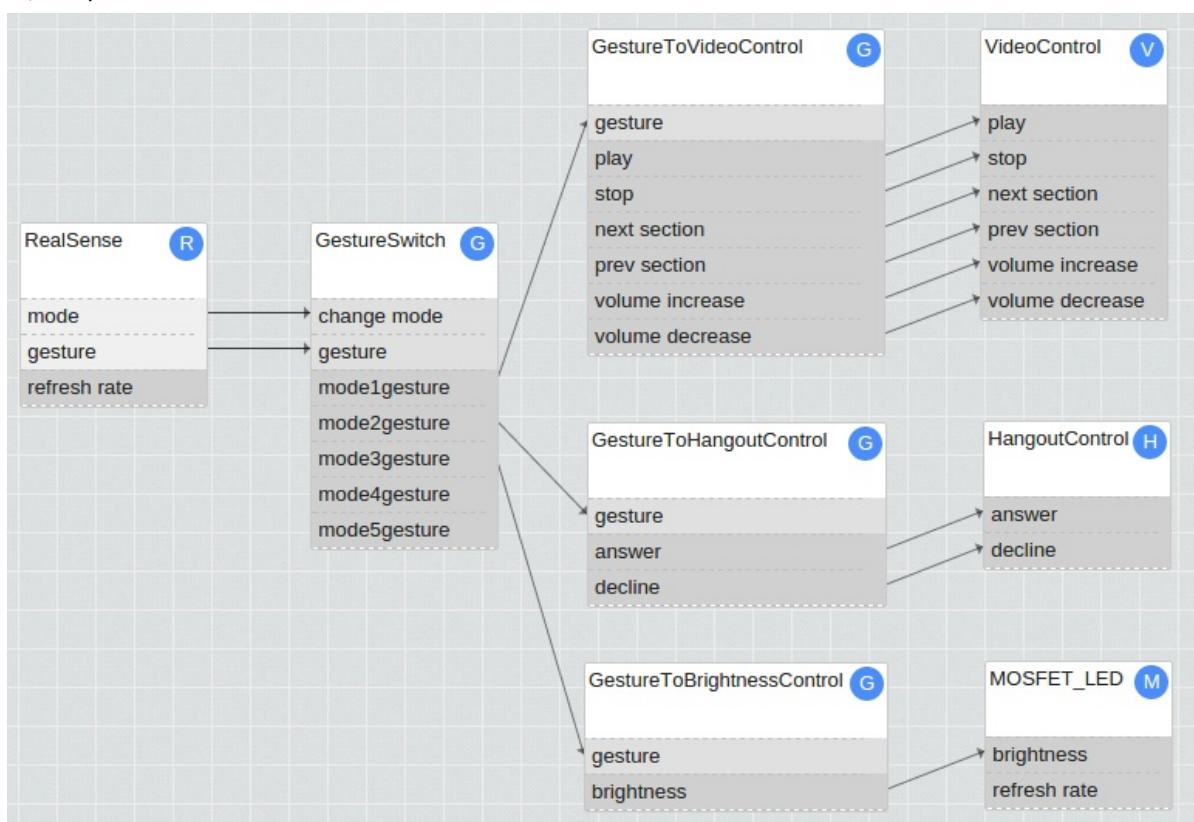
④ 每次當我們接收到一筆新數據後，都要用這兩行程式將模式屬性的值設爲False，將手勢屬性的值設爲零，因爲如同第六章所述的屬性架構，只有當屬性的值被屬性架構發現有改變後，這個值才會按照資料流的來源與目的傳遞，若沒有重新設定這兩個屬性的值，輕拍的手勢將無法更換場景，因爲輕拍手勢的值每次都相同，所以不會被屬性架構傳遞。

⑤ 每次模式屬性的值都會被儲存，並用於跟下一回新的值做比較。

⑥ 最後，手勢屬性的值透過這行程式傳遞至資料流的目的。

• 部署使用 RealSense 攝影機的物聯網應用程式

這張圖可以用來說明，如何透過悟空系統部署使用 RealSense 攝影機的物聯網應用程式。如同上述，圖中的 RealSense 元件有模式與手勢的屬性，這兩個屬性的值會連接到一個交換器 (GestureSwitch)，此交換器會使用模式屬性的值，來決定要更新哪一個場景的資料流，分別有影片播放 (GestureToVideoControl)，視訊通話 (GestureToHangoutControl) 與燈光明暗控制 (GestureToBrightnessControl)，再由該場景元件根據手勢屬性的值來執行應用服務。



7.2 使用網頁客戶端(**Web Client**)的悟空類別

在這一章節中，我們將先說明如何使用現有的悟空類別來控制Philip Hue燈泡，接者說明此悟空類別如何使用網頁客戶端來連線Philip Hue的智慧橋接器(bridge)。

- **用悟空物件來控制Philip Hue燈泡**

1. 為讓使用者能開發應用程式來控制Hue系列的產品，Philip Hue開放了HTTP API供開發者使用，但在使用之前，我們需先取得智慧橋接器的網址，請參照Hue官方網站(<http://www.developers.meethue.com/documentation/getting-started>)的說明，並使用API除錯工具(API Debugger Tool)來取得網址。網址如同以下樣式：

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

中間的長字串代表智慧橋接器的使用者名稱，此名稱會在取得網址的過程中產生出來。

2. 取得使用者名稱後，需要更改Philip Hue的悟空類別中原有的使用者名稱，更改步驟如下：

```
cd <path_of_source_code>/wukong-darjeeling/wukong/gateway/udpwkpf/
vim udpdevice_philip_hue_*.py
# 在大約第34行的地方，將原有的使用者名稱改寫成新取得的名稱。
```



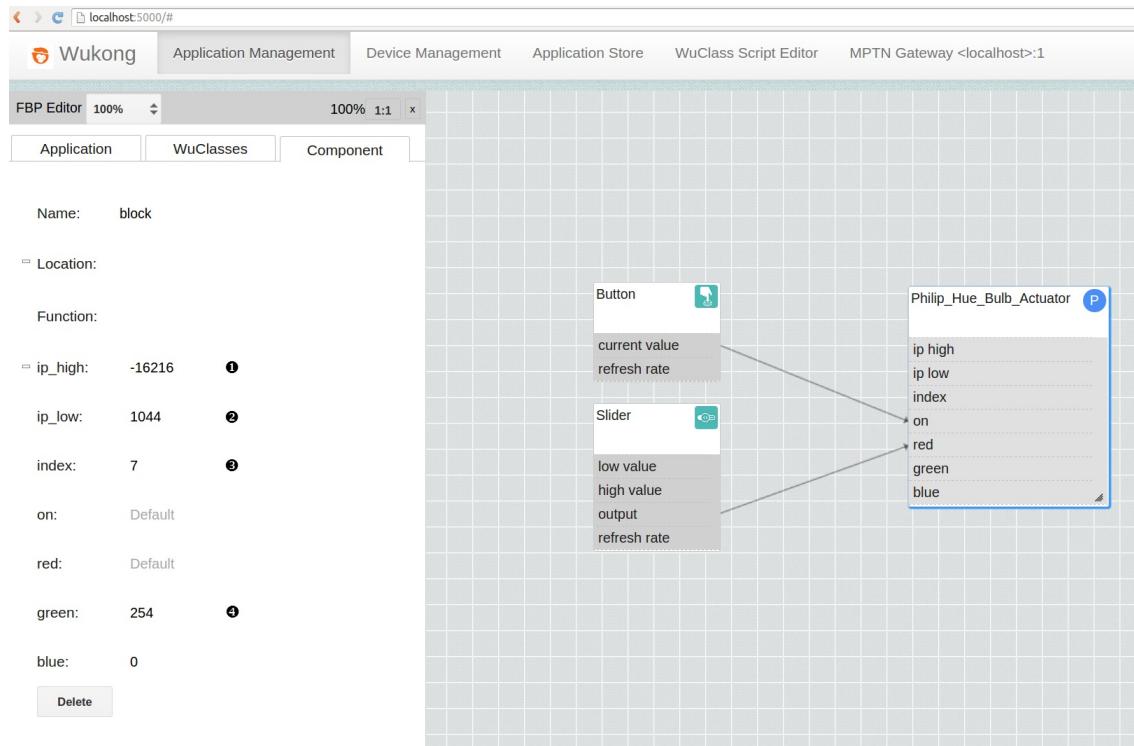
```

32     def update(self,obj,pID,val):
33         debug_name = "Philip_Hue_Bulb_Actuator"
34         user = "newdeveloper"
35         ip = ((int)(obj.getProperty(0)) & 0xffff) << 16
36         ip |= ((int)(obj.getProperty(1)) & 0xffff)
37         ip = socket.inet_ntoa(struct.pack('!L',ip))
38         index = obj.getProperty(2)
39         on = obj.getProperty(3)
40         r = obj.getProperty(4)
41         g = obj.getProperty(5)
42         b = obj.getProperty(6)
43         hwc = HWC(ip, user, index, self.gamma)

```

3. 將Philip Hue的悟空裝置程式，按照第5.2節所述的步驟加入至悟空系統。
4. 按照第5.3節所述的步驟建立一個Hue燈泡的應用程式，並設定初始值。

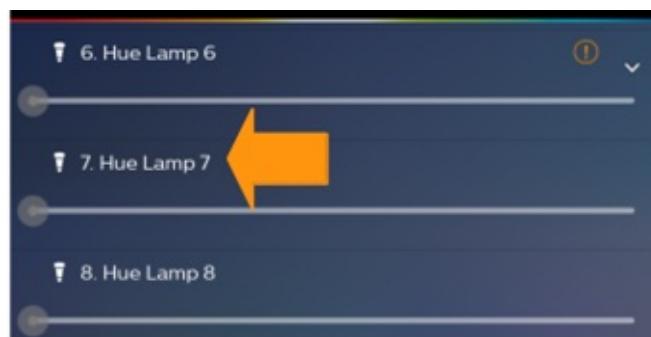
一個基本的Hue燈泡應用程式如下圖所示，按鈕物件(Button)連接到Hue燈泡物件的開關屬性(on)，旋轉電位器(Slider)連接到Hue的紅色屬性(red)以調整紅色佔有的比例，另外，當點選Hue燈泡物件時，左方的欄位就會顯示出來，這個欄位是用來填寫其餘屬性的初始值。



① ②這兩個欄位填寫的是智慧橋接器的IP4位址，由於將應用程式部署到開發板的執行過程不支援字串型態(string datatype)，並且每個屬性的型態最長支援16位元，因此，我們必須將IP4位址轉換成短整數，並且需要用兩個屬性(ip_high, ip_low)來存取IP4位址，以下是計算如何轉換的範例：

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

③指數屬性(index)代表每個Hue燈泡的識別號，這個識別號可以從Hue提供的應用程式(app)來做確認，如下圖所示：



④ 這個欄位代表三原色各自的比例，我們定義的範圍值是從0到254，也就是短整數能表示的範圍。

5. 按照第5.3節所述的步驟來部署Hue燈泡的應用程式

• 實作網頁客戶端的悟空類別

以下將說明悟空類別如何使用網頁客戶端來連線智慧橋接器，藉此控制燈泡的顏色與亮度。一旦了解Hue的悟空類別如何實作後，就能夠用相同的方式，跟其它有提供HTTP API的物聯網商品互通。由於Hue有一系列的產品，為減少重覆的程式在各個Hue的悟空類別中，因此我們把部分共用的程式獨立成一個工具檔案，名叫philip_hue_utils.py，其中包含網頁客戶端，如下圖所示：

```

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):
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)
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

```

```

43         except Exception as e:
44             self.gamma.message = e.message
45             self.gamma.gamma = -101
46
47             self.remaining -= len(display)
48
49     def connectionLost(self, reason):
50         print 'Finished receiving body:', reason.getErrorMessage()
51         self.finished.callback(None)
52
53 class hue_web_client():
54     def __init__(self, ip, user, index, obj):
55         self.http_get_path = 'http://'+ip+'/api/'+user+'/lights/'+str(index)
56         self.http_put_path = 'http://'+ip+'/api/'+user+'/lights/'+str(index)+'/state/'
57         self.gamma = obj
58
59     def get_gamma(self):
60         agent = Agent(reactor)
61         defer = agent.request(
62             'GET', self.http_get_path,
63             Headers({'User-Agent': ['Twisted Web Client Example'],
64                      'Content-Type': ['text/x-greeting']}),
65             None)
66         defer.addCallback(self.cbRequest)
67         # defer.addBoth(self.cbShutdown)
68
69     def put_command(self, body):
70         agent = Agent(reactor)
71         defer = agent.request(
72             'PUT', self.http_put_path,
73             Headers({'User-Agent': ['Twisted Web Client Example'],
74                      'Content-Type': ['text/x-greeting']}),
75             body)
76         # defer.addBoth(self.cbShutdown)
77
78     def cbRequest(self, response):
79         #print 'Response version:', response.version
80         #print 'Response code:', response.code
81         #print 'Response phrase:', response.phrase
82         #print 'Response headers:'
83         #print pformat(list(response.headers.getAllRawHeaders()))
84         finished = Deferred()
85         response.deliverBody(BeginningPrinter(finished, self.gamma)) ④
86         return finished
87
88     def cbShutdown(self, ignored):
89         reactor.stop()

```

附註：關於如何透過Twisted使用網頁客戶端的詳細內容，請參考Twisted官方網頁

(<http://twistedmatrix.com/documents/current/web/howto/client.html>)

① 這個類別初始化的變數包含http_get_path與http_put_path，這兩組字串是根據Hue所提供的HTTP API而訂。

② put_command函式是用來傳送HTTP PUT請求(request)的信息，請求信息中的參數分別有，HTTP請求方法，請求的URI編碼，HTTP的標頭(header)，以及負責產生(produce)HTTP body的物件，接者，HTTP的代理物件(agent)會根據請求信息的參數來建立連線。

③ get_gamma函式是透過傳送HTTP GET請求(request)的信息，來取得Hue燈泡的gamma值，由於Hue系列的產品對色彩範圍的支援並不相同，按照產品的區別可以分成三種範圍，然而，並非每個三原色的組合都會落在這些色彩範圍內，因此需要gamma值來做校正，才能夠將顯示的顏色接近原本的三原色組合。

④ HTTP GET請求信息的回傳物件(response)能透過deliverBody顯示回傳物件的HTTP body。

- 5** BeginningPrinter是專用在deliverBody函式的協議(Protocol)，當有HTTP GET的物件回傳時，dataReceived函式會接收到物件的HTTP body。
- 6** 在分析所收到的HTTP body之後，就能判斷gamma的值是多少。
- 7** 當HTTP body被接收完畢後，BeginningPrinter協議的connectionLost函式會緊接地被執行。

接者，以下的悟空類別使用了工具檔案中的網頁客戶端來控制Hue燈泡。

```

11 import philip_hue_utils; HWC = philip_hue_utils.hue_web_client      ❶
12 import philip_hue_utils; HC = philip_hue_utils.hue_calculation

13
14 class Philip_Hue_Bulb_Actuator(WuClass):
15     def __init__(self):
16         WuClass.__init__(self)
17         self.loadClass('Philip_Hue_Bulb_Actuator')
18         self.lasttime = int(round(time.time() * 1000))
19         self.loop_rate = 500
20         self.gamma = Gamma()
21         print "Hue Bulb Actuator init success"
22
23     def update(self,obj,pID=None,val=None):
24         debug_name = "Philip_Hue_Bulb_Actuator"
25         user = "newdeveloper"
26         ip = ((int)(obj.getProperty(0)) & 0xffff) << 16
27         ip |= ((int)(obj.getProperty(1)) & 0xffff)
28         ip = socket.inet_ntoa(struct.pack('!L',ip))
29         index = obj.getProperty(2)
30         on = obj.getProperty(3)
31         r = obj.getProperty(4)
32         g = obj.getProperty(5)
33         b = obj.getProperty(6)
34         hwc = HWC(ip, user, index, self.gamma)          ❷
35
36         currenttime = int(round(time.time() * 1000))

37         if (currenttime - self.lasttime > self.loop_rate):    ❸
38             if(self.gamma.gamma < 0):                          ❹
39                 hwc.get_gamma()                                ❺
40                 if (self.gamma.gamma < 0):
41                     print "\n____%s____GET gamma error:%d\n" % (debug_name, self.gamma.gamma)
42                     if (self.gamma.gamma < -99):
43                         print "\n____%s____JSON error:%s\n" % (debug_name, self.gamma.message)
44                     else:
45                         print "\n____%s____Error!ip:%s,index:%d\n" % (debug_name, ip, index)
46             self.lasttime = currenttime
47             return

48             hc = HC()
49             hc.RGBtoXY(self.gamma, r, g, b)                  ❻
50
51             if(on):
52                 command = ("{\"on\":true, \"xy\":[%.2f,%.2f], \"bri\":%d}" % (self.gamma.x, self.gamma.y
53                                         , (int)(self.gamma.bri*255)))           ❼
54             else:
55                 command = ("{\"on\":false}")
56
57             time.sleep(0.05)
58             print "\n____%s____PUT command:%s\n" % (debug_name, command)

59             body = FileBodyProducer(StringIO(command))        ❽
60             hwc.put_command(body)
61             self.lasttime = currenttime                      ❾

```

- 1** 匯入Hue工具檔的網頁客戶端類別。
- 2** 從悟空屬性架構的屬性儲存區取得智慧橋接器的網路位址(ip),智慧橋接器的使用者名稱(username)以及Hue燈泡的識別號(index)之後，就能新增一個Hue的網頁客戶端物件。
- 3** 由於智慧橋接器接收到太頻繁的HTTP訪問訊息時，會導致延遲反應或者丟失訊息，

於是，我們需要限定最短的訪問時間間隔，來確保智慧橋接器正常工作，以這個範例來說，最短間隔設定為0.55秒。

- ④ 使用Hue的網頁客戶端物件來取得gamma值。
- ⑤ 使用gamma值來校正輸入的三原色值。
- ⑥ 這兩個command將會是HTTP body裏的訊息，於put_command函式傳給智慧橋接器。
- ⑦ FileBodyProducer函式負責產生(produce)HTTP body的物件。
- ⑧ 使用Hue的網頁客戶端物件，透過put_command函式將HTTP請求的訊息傳給智慧橋接器。

7.3 使用網頁伺服器(Web Server)的悟空類別

在這個章節中，我們將先部署一個有使用網頁伺服器的悟空類別的應用程式，這個悟空類別的名稱是UISlider，意思是UI介面的旋轉電位器(以下稱為UI旋轉電位器)，當了接如何使用這個悟空類別後，我們接着說明如何實作這個悟空類別。

- 使用UI旋轉電位器的應用程式

1. 這個應用程式共有三個元件：

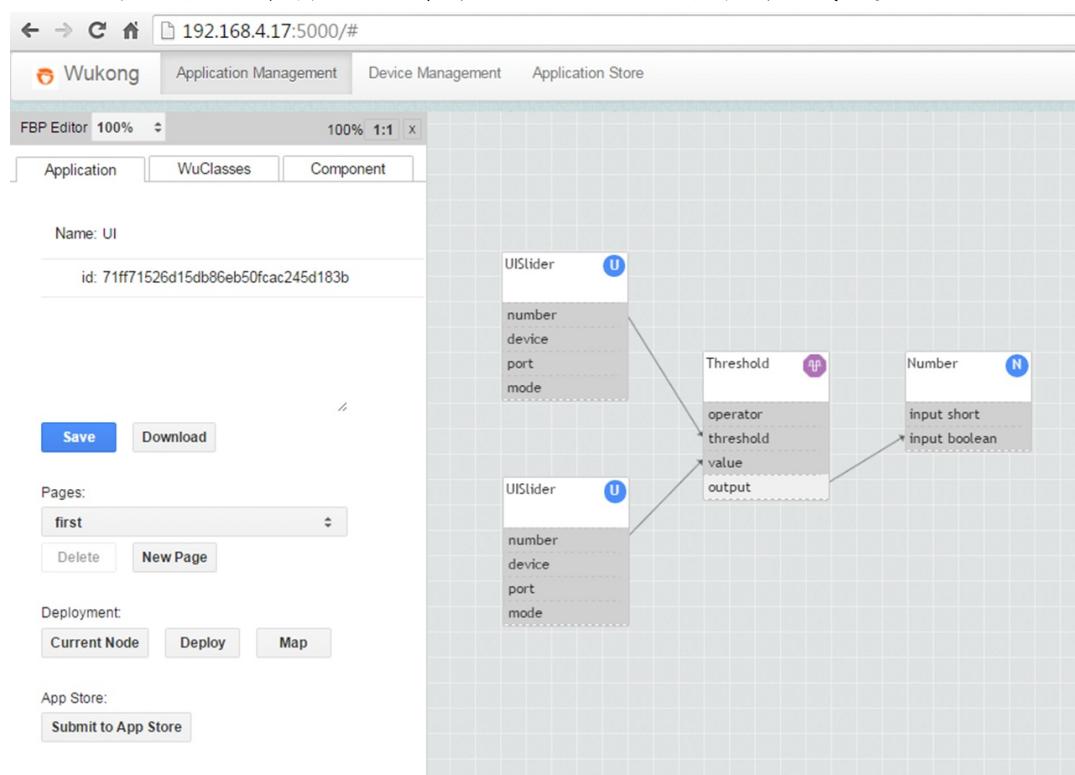
- UI旋轉電位器(UISlider)

如同旋轉電位器(Slider)的輸出從零到電源電壓(vcc)伏特，UI旋轉電位器會接收從瀏覽器傳來的數字(number)，並將這個數字轉換成短整數，短整數就成為UI旋轉電位器的第一個屬性(number)的值，當值改變後，就會按照資料流被傳遞到下一個屬性。

- 閥值比較器(Threshold) 閥值比較器會將其第二個屬性(threshold)的值當做閥值，用來和第三個屬性(value)的值做比較，比較的方式將由第一個屬性(operator)的設定來決定，如果仔細對照元件函式庫的話，會發現這個屬性的資料型態是列舉，其中有LT(小於),LTE(小於和等於),GT(大於)和GTE(大於和等於)這四種選項，預設值是LT，比較後的結果會由第四個屬性(output)傳遞下去。

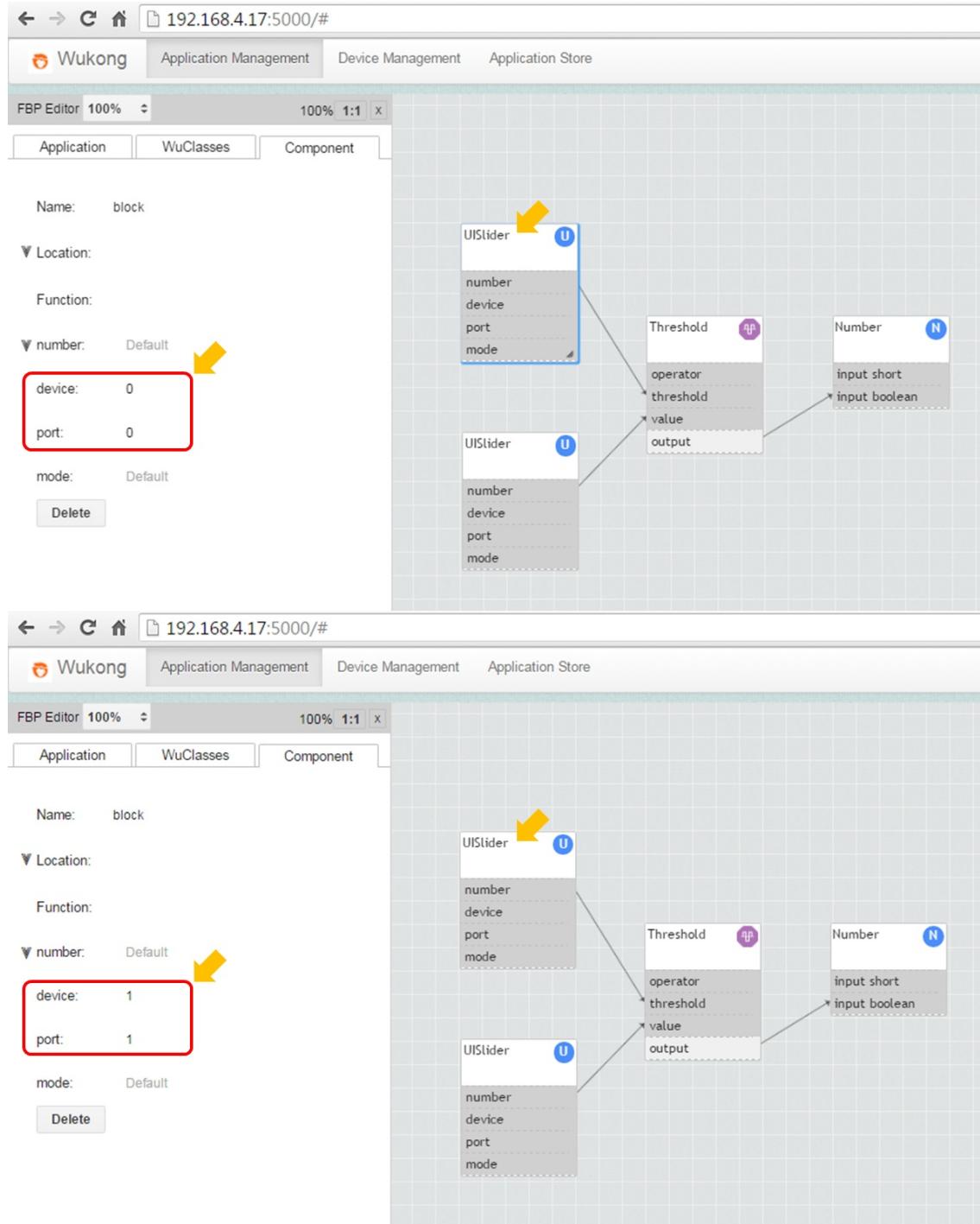
- 數值標示(Number)

這個元件主要是用來偵錯，它會將接收到的值顯示在執行程式的畫面上。

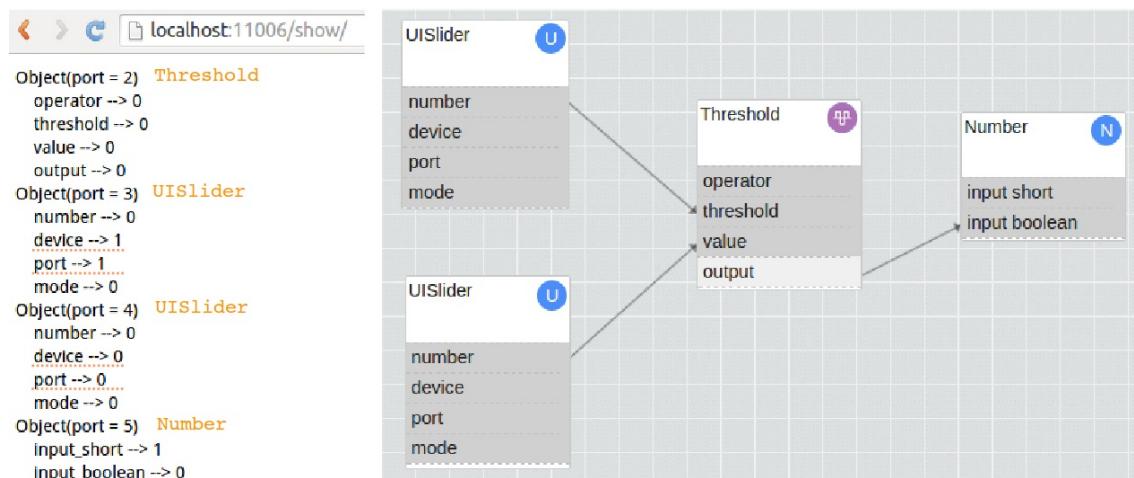


2. 初始值設定

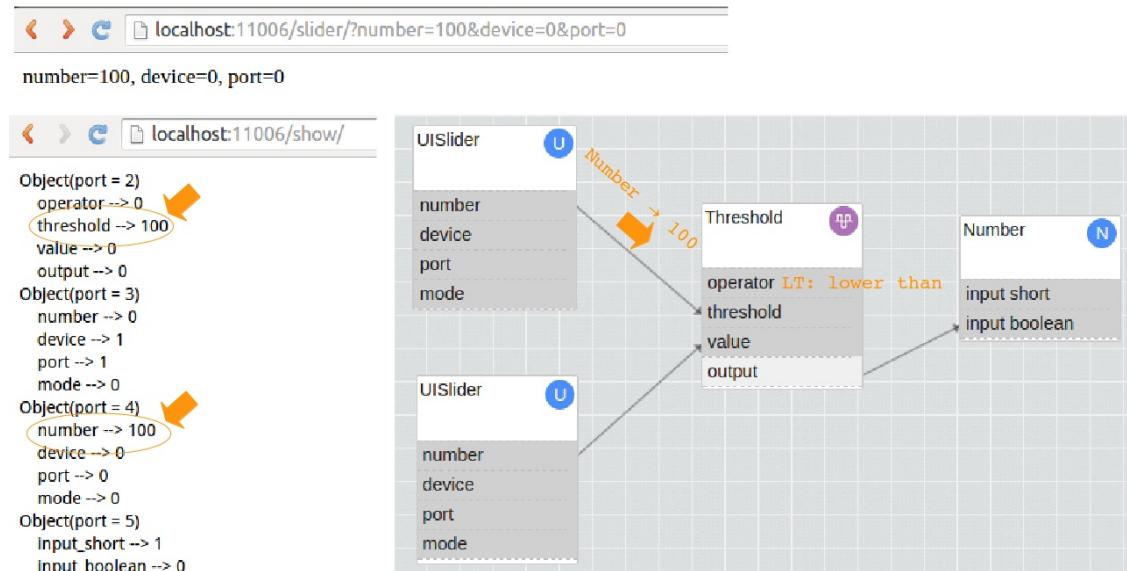
爲了要在應用程式中使用多個UI旋轉電位器物件，我們多定義了兩個屬性，分別是裝置識別號(device)與埠識別號(port)，每個UI旋轉電位器在部署前都需設定這兩個屬性的初始值，如此一來，當瀏覽器傳值給UI旋轉電位器的時候，才能明確地指定是傳給哪一個UI旋轉電位器。



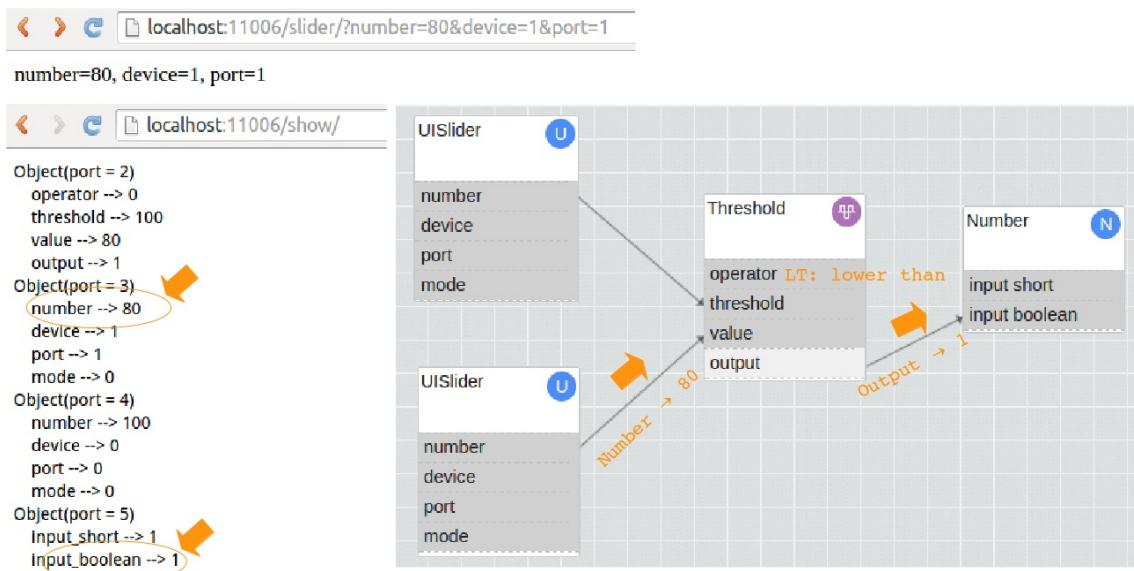
3. 為了解數據在資料流的傳遞過程，我們在UI旋轉電位器內加了一個名稱爲ShowAll的類別，可以透過瀏覽器把每個物件屬性即時的變化印出來，如同下圖中的左半部，在瀏覽器的網址列輸入localhost:11006/show。



4. 在下圖中，當我們透過瀏覽器傳值(number)100給UI旋轉電位器(較上的)時，從呼叫ShowAll的瀏覽器頁面中，可以看出來閥值比較器的閥值屬性(threshold)改變了，同時也可以確認傳遞的值為100。



5. 當我們透過瀏覽器傳值(number)80給UI旋轉電位器(較下的)時，從呼叫ShowAll的瀏覽器頁面中，可以看出來閥值比較器的比較值(value)改變了，接者，比較完這兩個屬性(threshold和value)後，就把比較的結果傳遞給數字標示物件。



- 實作UI旋轉電位器(UISlider)的悟空類別

```

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)) ❶
61 site=server.Site(root)
62 site.device = d
63 reactor.listenTCP(11006,site)
64 reactor.run() ❷

```

❶ render_GET函式的用途在於回應客戶端的HTTP GET請求，回傳的資料型態是字串，字串的內容視開發者實作而定。

❷ 這三行程式是從HTTP的請求中把關於UI旋轉電位器屬性的值取出。

❸ 當我們使用這個標記(FLAGS_APP_CAN_CREATE_INSTANCE, FLAGS_VIRTUAL)來加入悟空類別時，這個悟空類別就可以在部署應用程式的階段，被用來產生多個新的悟空物件，而不需先用addObject函式來產生新的悟空物件。

❹ 由於沒有使用更新函式，悟空屬性架構不會傳要被更新的物件給render_GET，於是這兩行程式比對同一個裝置上所有的悟空物件，直到找到UI旋轉電位器的悟空物件為止。

❺ 由於同一個裝置上有可能有多個UI旋轉電位器，於是這行程式是用來篩選出所指定的UI旋轉電位器，這也是我們先前需要設定裝置識別號(device)與埠識別號(port)的初始值的原因。

❻ putChild函式是用來定義UI旋轉電位器的網址列關鍵字。

❼ 標示哪一個埠會用來接收HTTP GET請求，在UI旋轉電位器的範例中，埠的識別碼是11006，網址列關鍵字是slider，所以<http://localhost:11006/slider>是傳送HTTP GET請求的最前段的網址列。