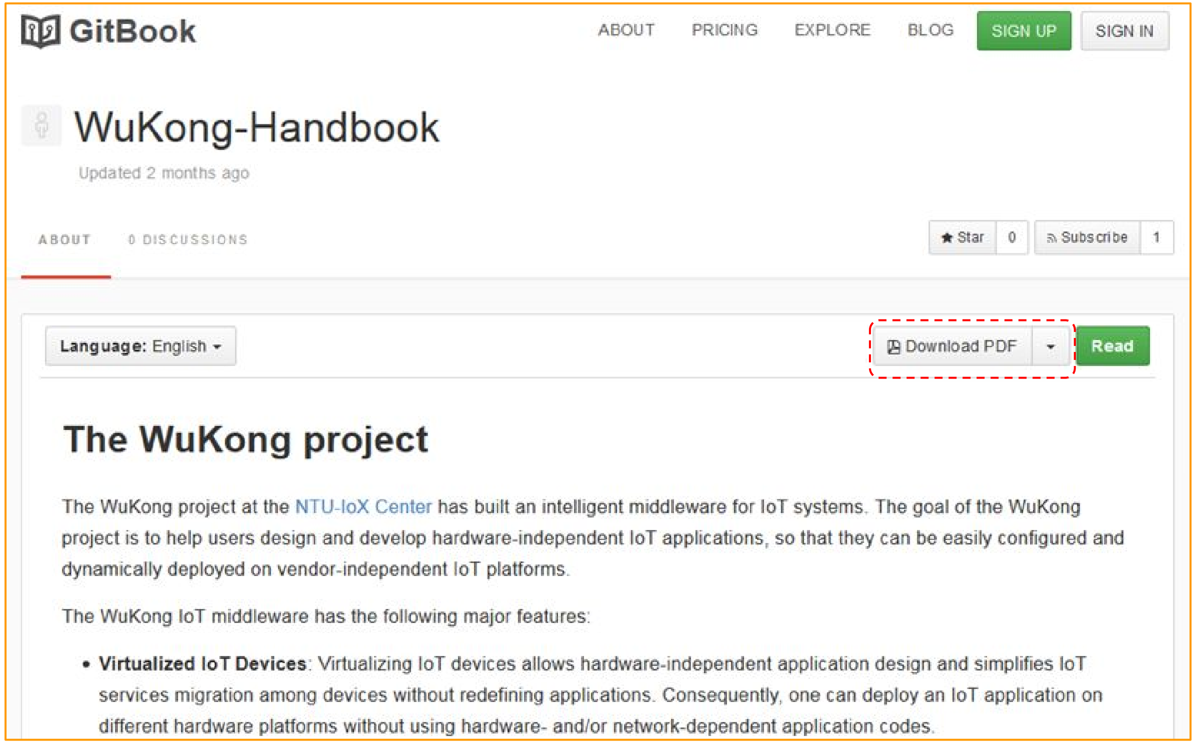
Outline

- **Flow Based Programming UI**
- WuKong Profile Framework
- The First Example

url of this slide:
goo.gl/AHfDPx

WuKong Handbook Website

https://www.gitbook.com/book/wukongsun/wukong-handbook/details



WuKong-Handbook

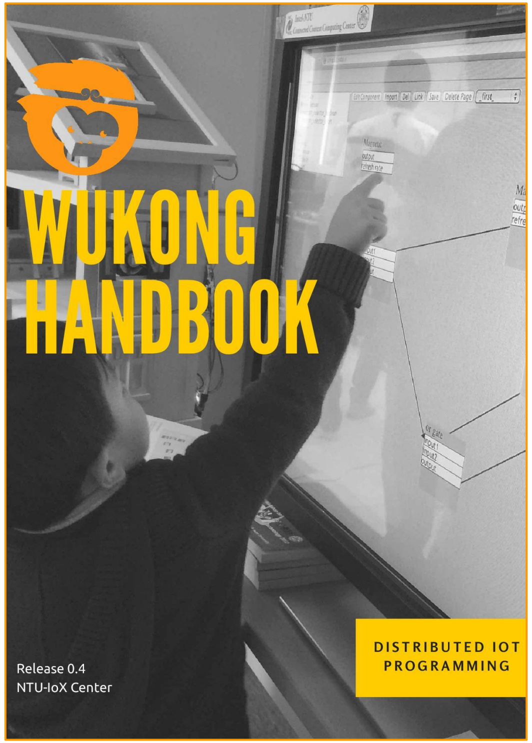
Updated 2 months ago

The WuKong project

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

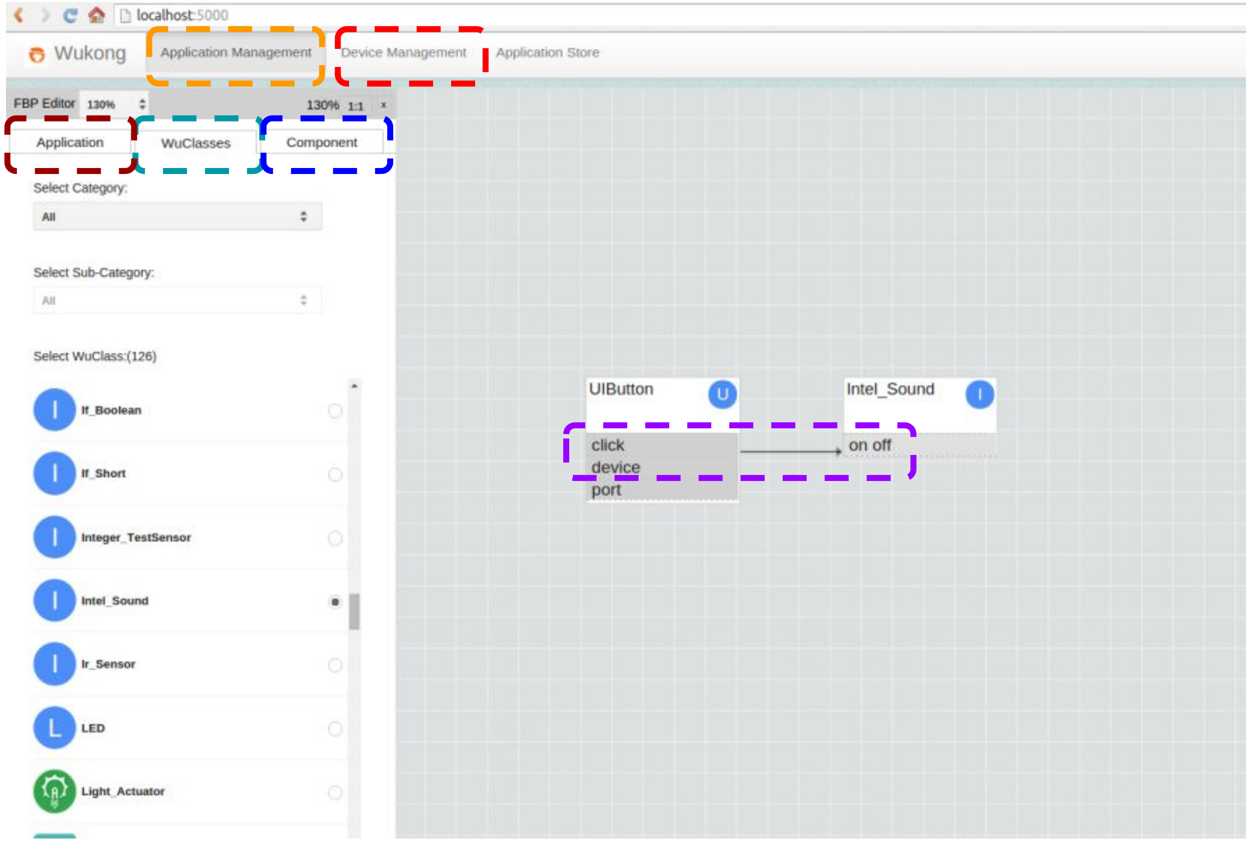
The WuKong IoT middleware has the following major features:

- **Virtualized IoT Devices**: Virtualizing IoT devices allows hardware-independent application design and simplifies IoT services migration among devices without redefining applications. Consequently, one can deploy an IoT application on different hardware platforms without using hardware- and/or network-dependent application codes.



Flow Based Programming UI (FBP)

- This example can be found in the chapter 4 of Gitbook.



Procedures:

Include new devices

Create a new FBP

Add components

Set properties and locations

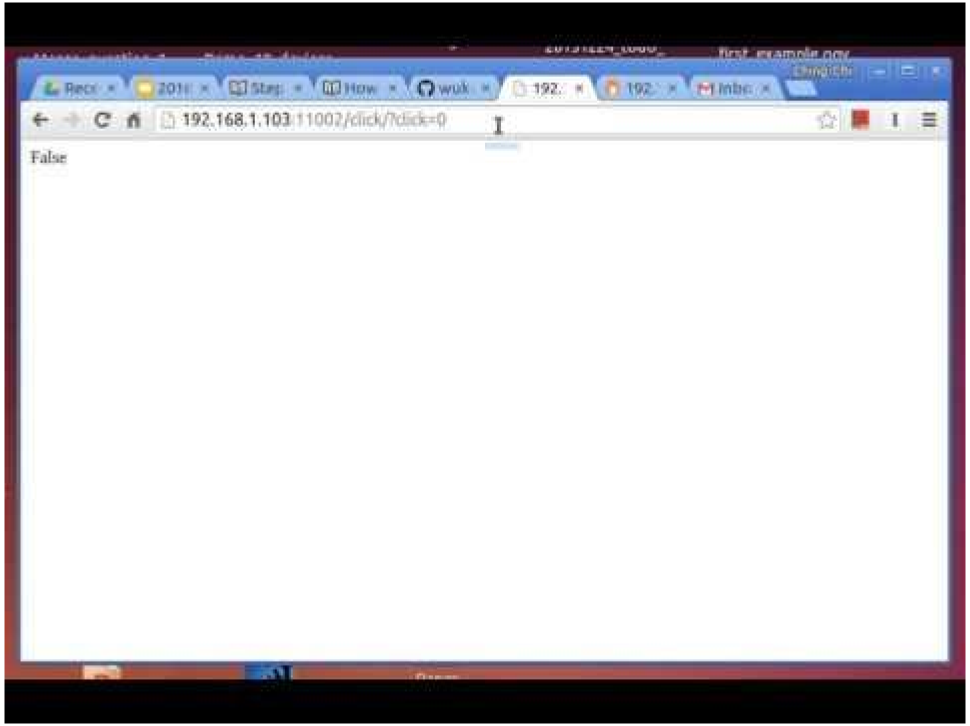
Add links between components

Save FBP

Mapping components to physical devices

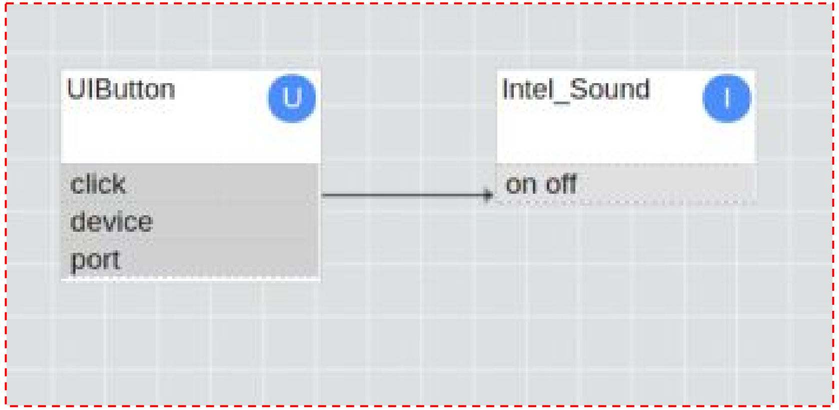
Deploy FBP to physical devices

Flow Based Programming UI (FBP)



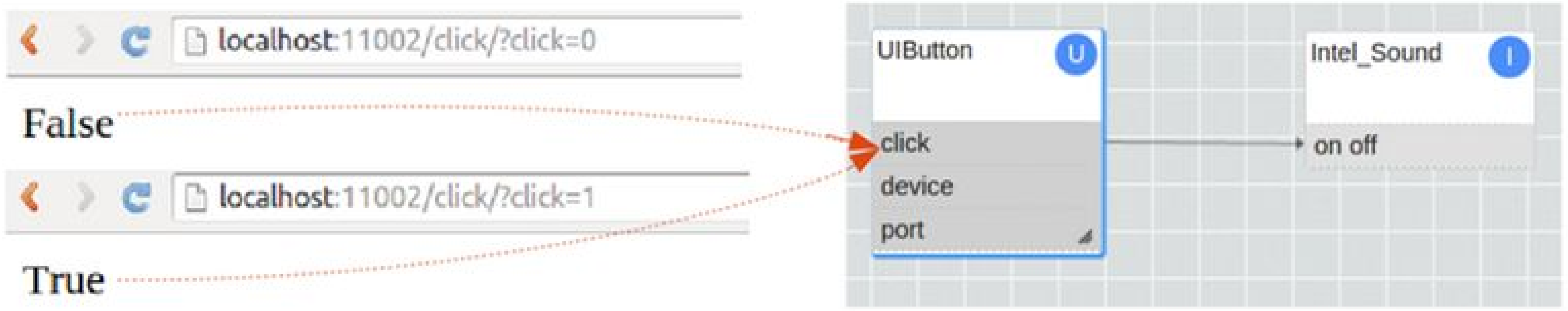
192.168.1.103:11002/click/?click=0

False



Flow Based Programming UI (FBP)

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

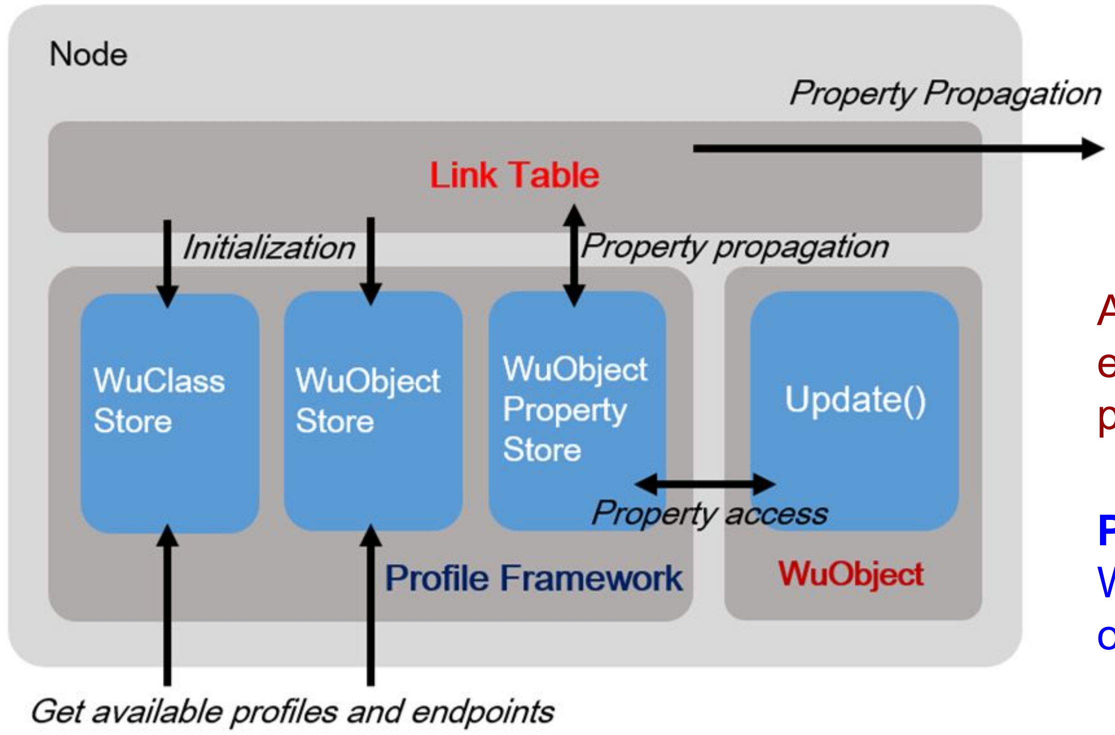

localhost:11002/click/?click=0
False
localhost:11002/click/?click=1
True
UIButton
U
click
device
port
Intel_Sound
I
on off

Outline

- Flow Based Programming UI
- **WuKong Profile Framework**
- The First Example

WuKong Profile Framework (WKPF)

- The following figure shows a view on **the software structure of a WuKong node.**
- This section can be found in the chapter 6 of Gitbook.



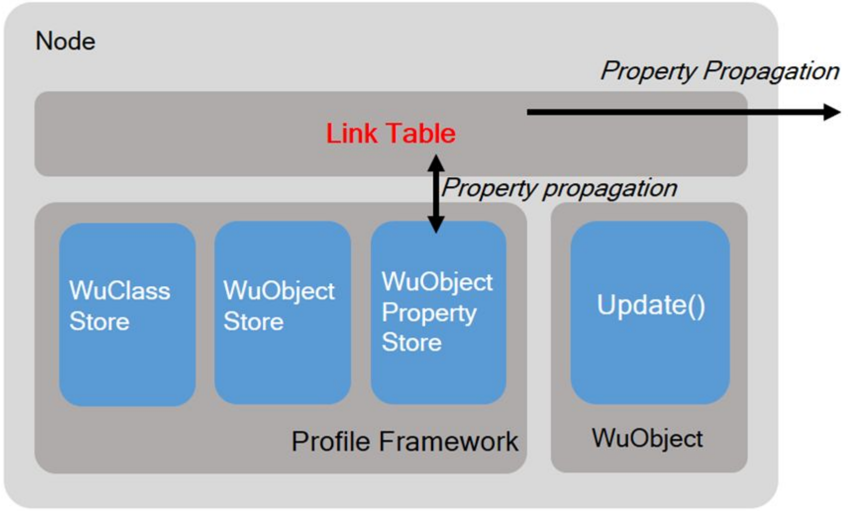
Link Table specifies the destination where a property value should be propagated.

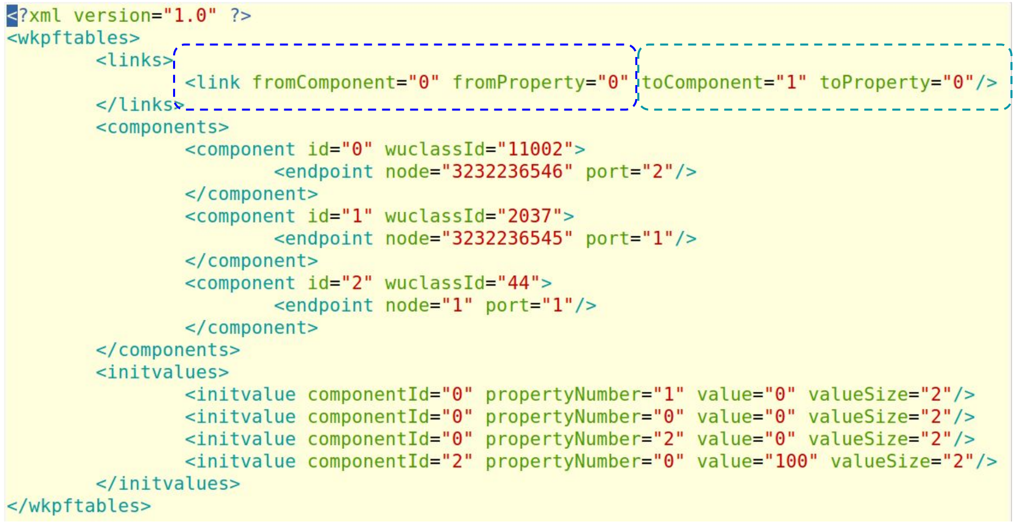
A node may have many **WuObjects**, each with an update function running periodically.

Profile Framework manages the WuKong classes and objects running on the node.

WuKong Profile Framework --- Link Table

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



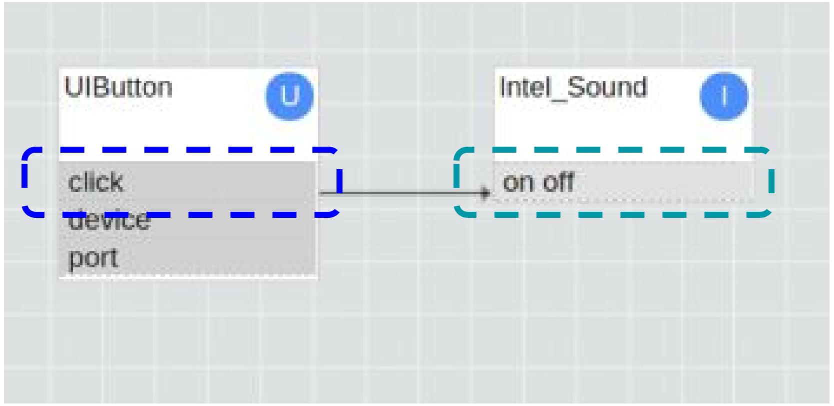


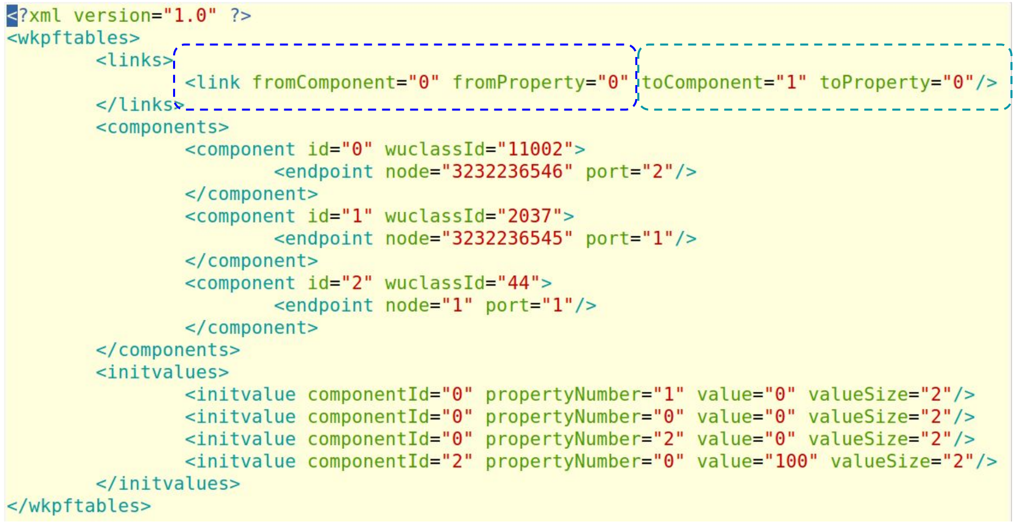
```
<?xml version="1.0" ?>
<wkpftables>
        <links>
                <link fromComponent="0" fromProperty="0" toComponent="1" toProperty="0"/>
        </links>
        <components>
                <component id="0" wuclassId="11002">
                        <endpoint node="3232236546" port="2"/>
                </component>
                <component id="1" wuclassId="2037">
                        <endpoint node="3232236545" port="1"/>
                </component>
                <component id="2" wuclassId="44">
                        <endpoint node="1" port="1"/>
                </component>
        </components>
        <initvalues>
                <initvalue componentId="0" propertyNumber="1" value="0" valueSize="2"/>
                <initvalue componentId="0" propertyNumber="0" value="0" valueSize="2"/>
                <initvalue componentId="0" propertyNumber="2" value="0" valueSize="2"/>
                <initvalue componentId="2" propertyNumber="0" value="100" valueSize="2"/>
        </initvalues>
</wkpftables>
```

Note: <path>/src/app/wkdeploy/java/WKDeploy.xml

WuKong Profile Framework --- Link Table

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



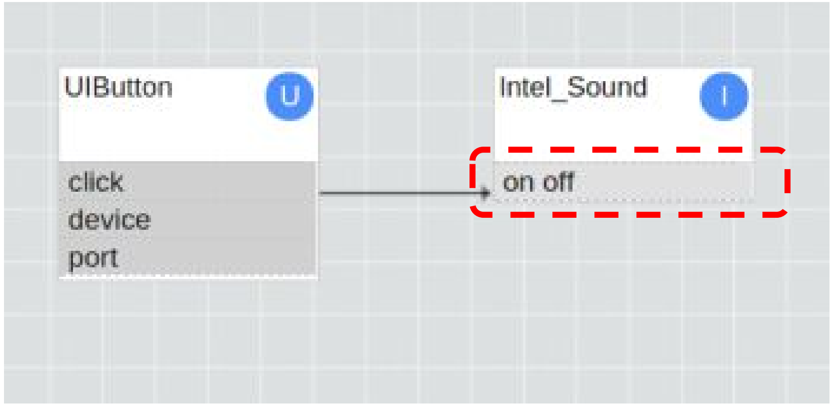


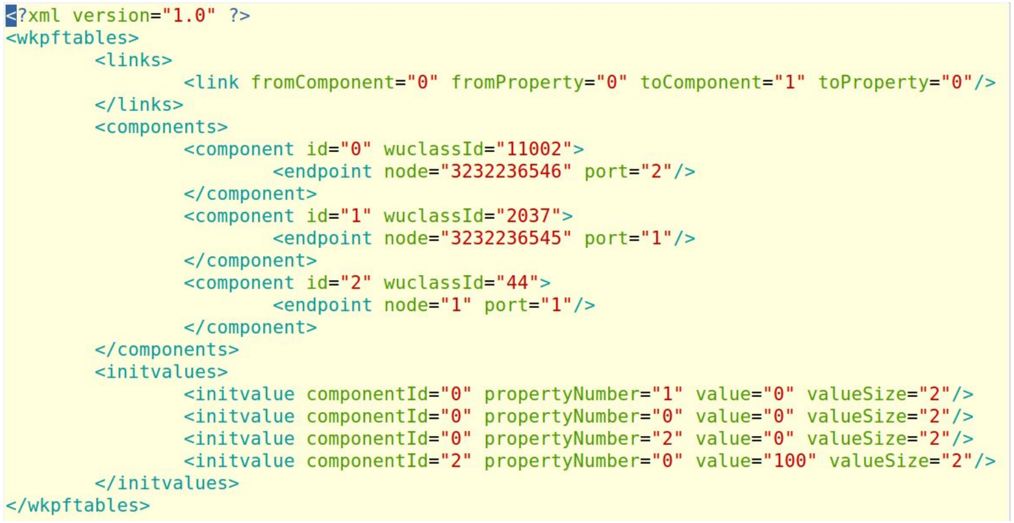
```
<?xml version="1.0" ?>
<wkpftables>
        <links>
                <link fromComponent="0" fromProperty="0" toComponent="1" toProperty="0"/>
        </links>
        <components>
                <component id="0" wuclassId="11002">
                        <endpoint node="3232236546" port="2"/>
                </component>
                <component id="1" wuclassId="2037">
                        <endpoint node="3232236545" port="1"/>
                </component>
                <component id="2" wuclassId="44">
                        <endpoint node="1" port="1"/>
                </component>
        </components>
        <initvalues>
                <initvalue componentId="0" propertyNumber="1" value="0" valueSize="2"/>
                <initvalue componentId="0" propertyNumber="0" value="0" valueSize="2"/>
                <initvalue componentId="0" propertyNumber="2" value="0" valueSize="2"/>
                <initvalue componentId="2" propertyNumber="0" value="100" valueSize="2"/>
        </initvalues>
</wkpftables>
```

Note: <path>/src/app/wkdeploy/java/WKDeploy.xml

WuKong Profile Framework --- Property

- Property is the basic data unit of WKPF. Each property has four attributes including **name**, **access**, **datatype** and **default** value. For Intel_Sound, four attributes of its first property are **on_off**, **input**, **boolean** and **none**.



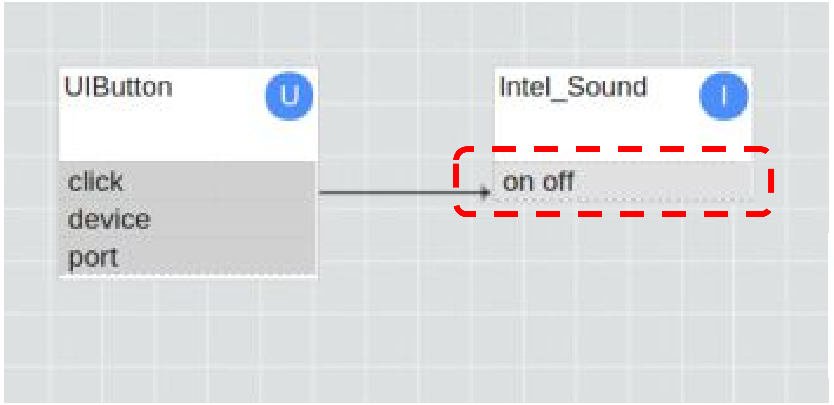


```
<?xml version="1.0" ?>
<wkpftables>
        <links>
                <link fromComponent="0" fromProperty="0" toComponent="1" toProperty="0"/>
        </links>
        <components>
                <component id="0" wuclassId="11002">
                        <endpoint node="3232236546" port="2"/>
                </component>
                <component id="1" wuclassId="2037">
                        <endpoint node="3232236545" port="1"/>
                </component>
                <component id="2" wuclassId="44">
                        <endpoint node="1" port="1"/>
                </component>
        </components>
        <initvalues>
                <initvalue componentId="0" propertyNumber="1" value="0" valueSize="2"/>
                <initvalue componentId="0" propertyNumber="0" value="0" valueSize="2"/>
                <initvalue componentId="0" propertyNumber="2" value="0" valueSize="2"/>
                <initvalue componentId="2" propertyNumber="0" value="100" valueSize="2"/>
        </initvalues>
</wkpftables>
```

Note: <path>/src/app/wkdeploy/java/WKDeploy.xml

WuClass Definition

- This definition can be found in the chapter 6.2 of Gitbook.
- These components will be generated to the WuClass list of FBP editor once defined in the WuKongStandardLibrary.xml

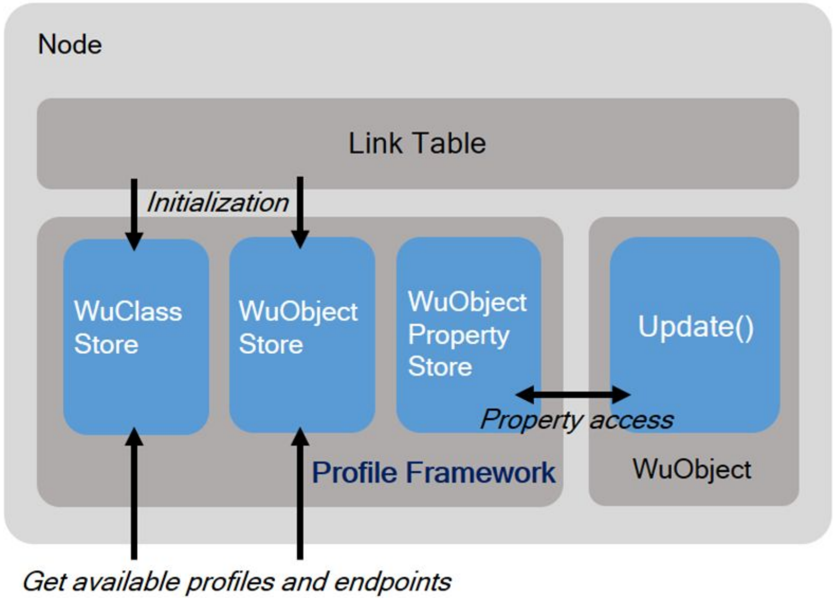


```
<WuClass name="UIButton" id="11002" virtual="false" type="soft">
  <property name="click" access="readwrite" datatype="short" default="0" />
  <property name="device" access="readwrite" datatype="short" default="0" />
  <property name="port" access="readwrite" datatype="short" default="0" />
</WuClass>

<WuClass name="Intel_Sound" id="2037" virtual="false" type="hard">
    <property name="on_off" access="writeonly" datatype="boolean"  />
</WuClass>
```

Note:
<path>wukong/ComponentDefinitions/WuKongStandardLibrary.xml

WuKong Profile Framework --- Update Function

- An WuObject does not store its properties by itself.
- The properties are stored by WKPF which is responsible for monitoring and propagating changes.
- An WuObject has to communicate with WKPF to read and update its properties.

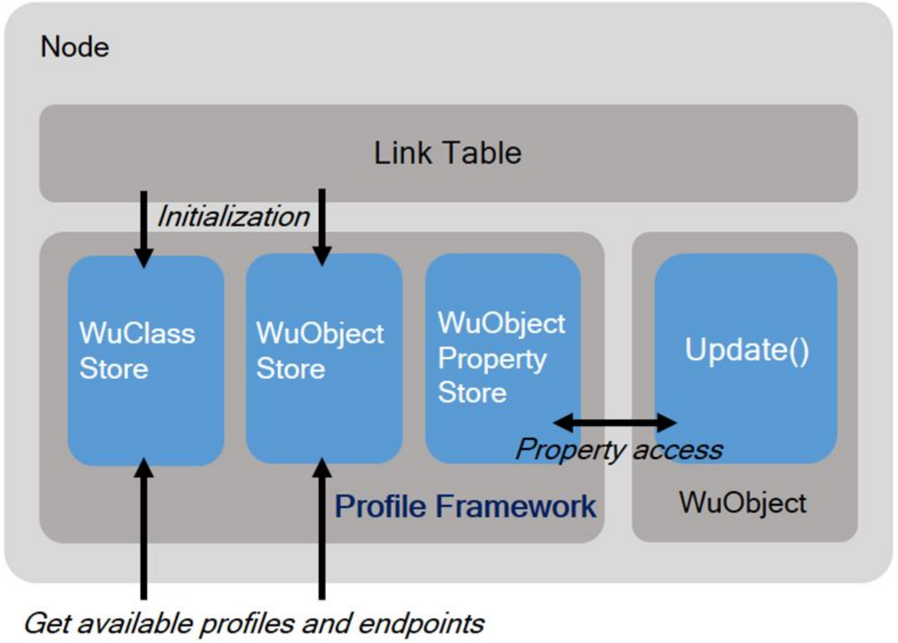


Basically each update function has three phases:

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

WuKong Profile Framework --- Update function

- The APIs to communicate with WKPF are **setProperty** and **getProperty**.
- setProperty will write output properties to the profile framework.
- getPropert will read input properties from the profile framework.



#pID means property ID, which starts from 0.

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

Note:
https://github.com/wukong-m2m/wukong-darjeeling/blob/develop/wukong/gateway/udpwkpf/udpwkpf.py#L727-L730

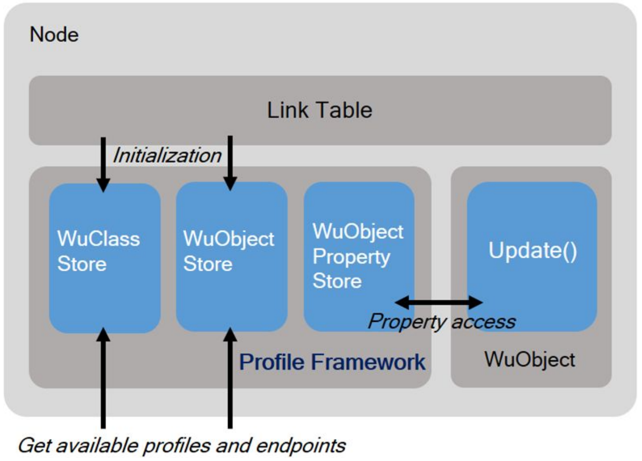
WuKong Profile Framework --- Update function

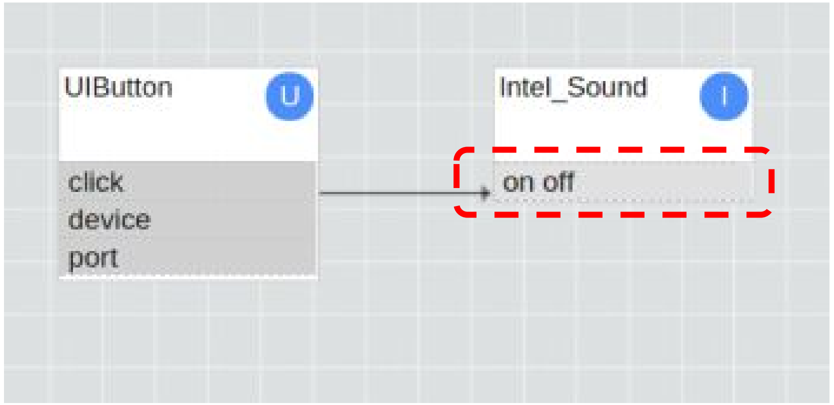
[Question] For on_off property of Intel_Sound, which API will be used?
Does update function run all three phases? Or some of them?

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

Basically each update function has three phases:

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





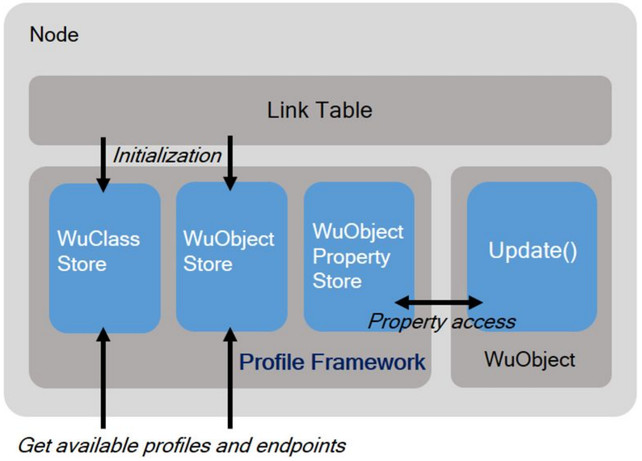
WuKong Profile Framework --- Update function

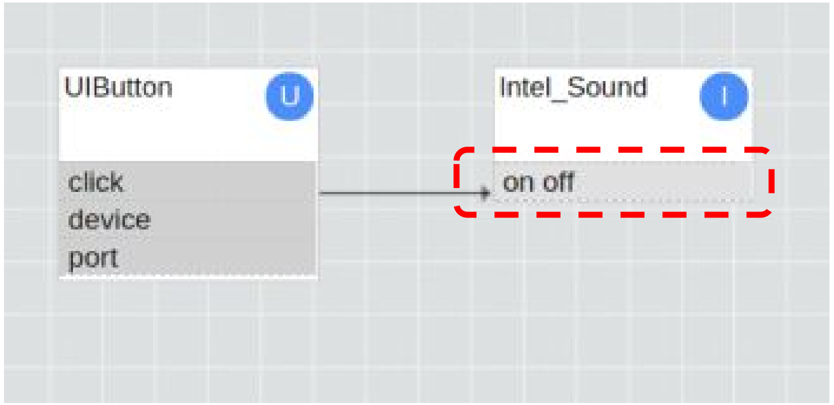
[Answer] Update function of Intel_Sound will use getProperty(0) to read data from WKPF. After reading data, the update function will turn on the music.

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

Basically each update function has three phases:

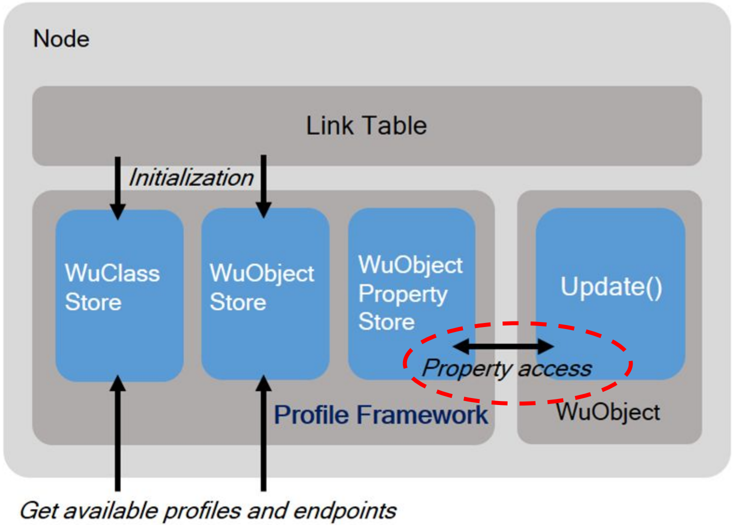
1. read input properties from the profile framework
2. do some processing
3. write output properties to the profile framework





WuKong Profile Framework --- Update function

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

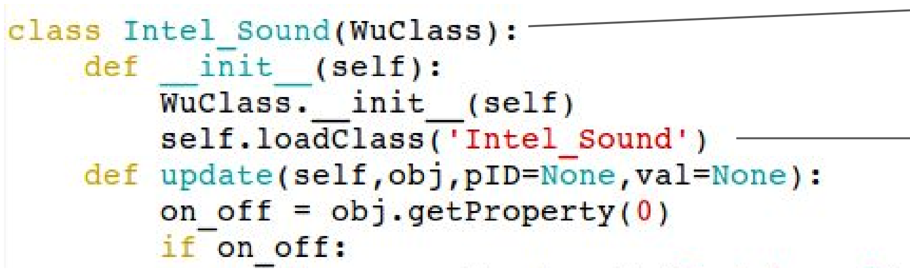


```
class Intel_Sound(WuClass):
    def __init__(self):
        WuClass.__init__(self)
        self.loadClass('Intel_Sound')
    def update(self,obj,pID=None,val=None):
        on_off = obj.getProperty(0)
        if on_off:
            path = os.path.abspath("intel.wav")
            chunk = 1024
            wf = wave.open(path, 'rb')
            p = pyaudio.PyAudio()

            stream = p.open(
                format = p.get_format_from_width(wf.getsampwidth()),
                channels = wf.getnchannels(),
                rate = wf.getframerate(),
                output = True)
            data = wf.readframes(chunk)

            while data != '':
                stream.write(data)
                data = wf.readframes(chunk)

            stream.close()
            p.terminate()
        else:
            pass
```

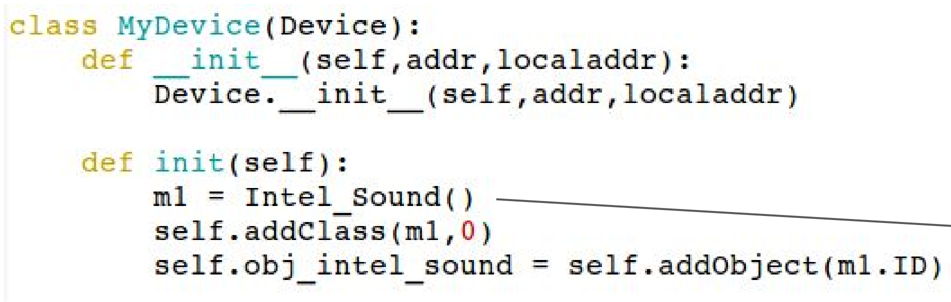
Read Input

Do some processing

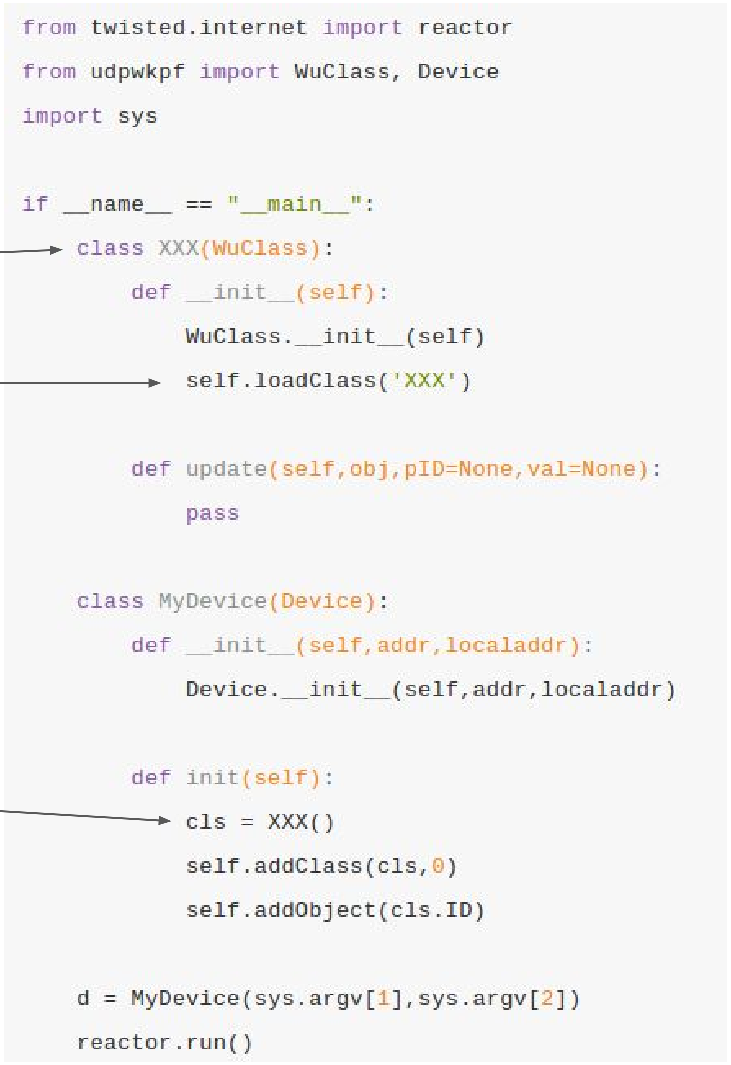
WuKong Device Python Template



```
class Intel_Sound(WuClass):
    def __init__(self):
        WuClass.__init__(self)
        self.loadClass('Intel_Sound')
    def update(self,obj,pID=None,val=None):
        on_off = obj.getProperty(0)
        if on_off:
```



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

    def init(self):
        m1 = Intel_Sound()
        self.addClass(m1,0)
        self.obj_intel_sound = self.addObject(m1.ID)
```



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

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

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

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

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

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

Outline

- Flow Based Programming UI
- WuKong Profile Framework
- **The First Example**

Check Material

For Edison,

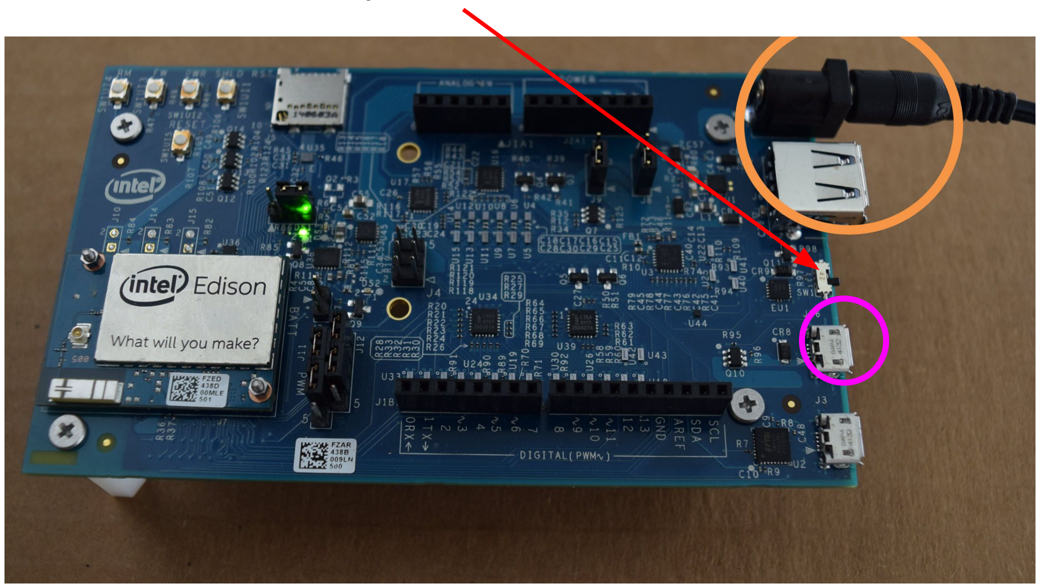
Charger:	a micro-usb to usb or a power adapter with **DC 12Volt**
Sensor:	a smoke sensor
	a touch pad
Actuator:	a LED strip (with 12 lights)
Wire:	female-to-male x 9
	female-to-female x 1

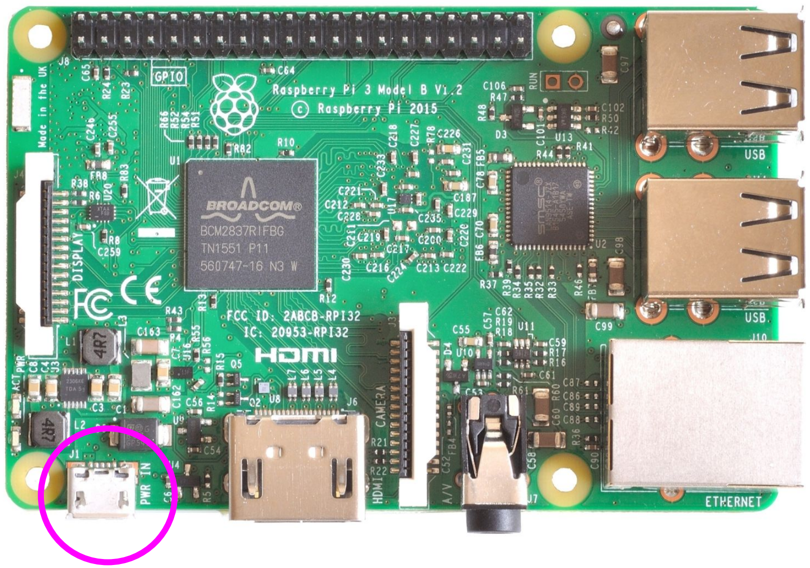
For Raspberry Pi,

Charger:	a micro-usb to usb
Sensor:	a smoke sensor
	a touch pad
Actuator:	a LED strip (with 12 lights)
Wire:	female-to-female x 10

Plug in the Power Supply

Switch to power cable or micro usb cable





7-15V DC/ At least 1500mA

Micro USB cable

Access to Edison or Raspberry Pi

- connect to AP according to the number labeled on each device.
  For number **10-29,** please connect to ssid **wukong_workshop_1**
  For number **41-55,** please connect to ssid **wukong_workshop_2**
  For number **30-40,** please connect to ssid **wukong_workshop_3**

- use ssh software to login in edison or pi
  For Edison,
  ssh root@192.168.0.xxx (xxx is the number labeled on each device.)
  username is root.
  password is wukong2016

  For Raspberry Pi,
  ssh pi@192.168.0.xxx (xxx is the number labeled on each device.)
  username is pi
  password is raspberry

Three Possible Configurations for WuKong Systems

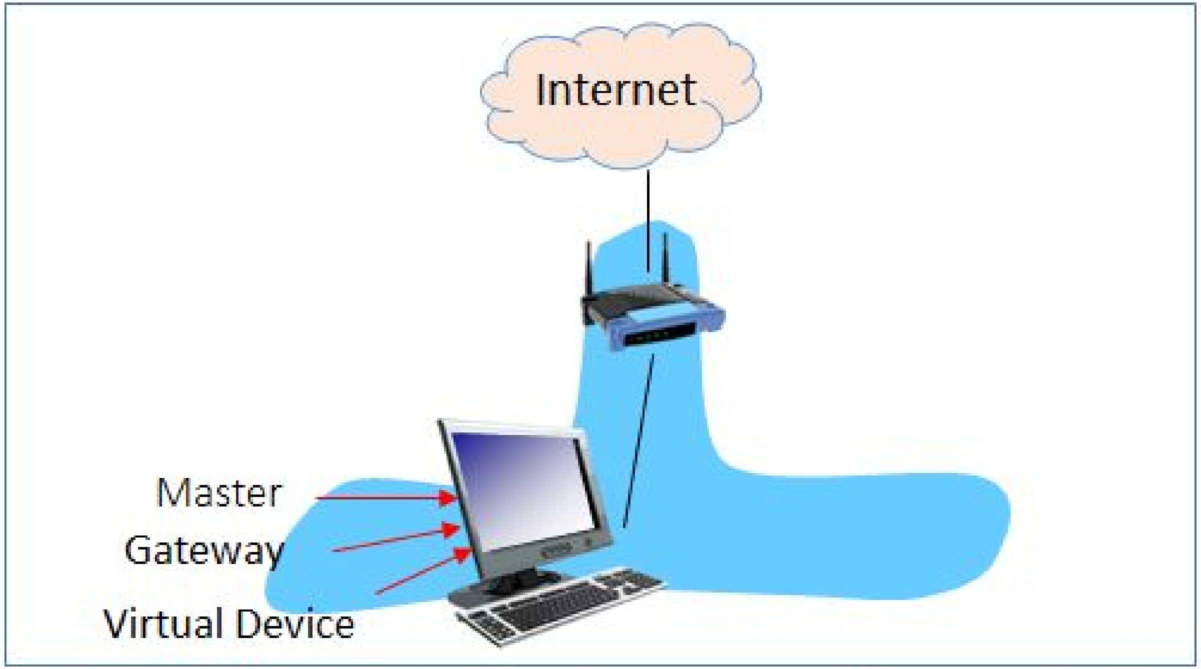


Fig 1a. A WuKong-based IoT application can be run using only a Linux-based computer, with Master, gateway, and virtual devices all on the same machine.

Three Possible Configurations for WuKong Systems

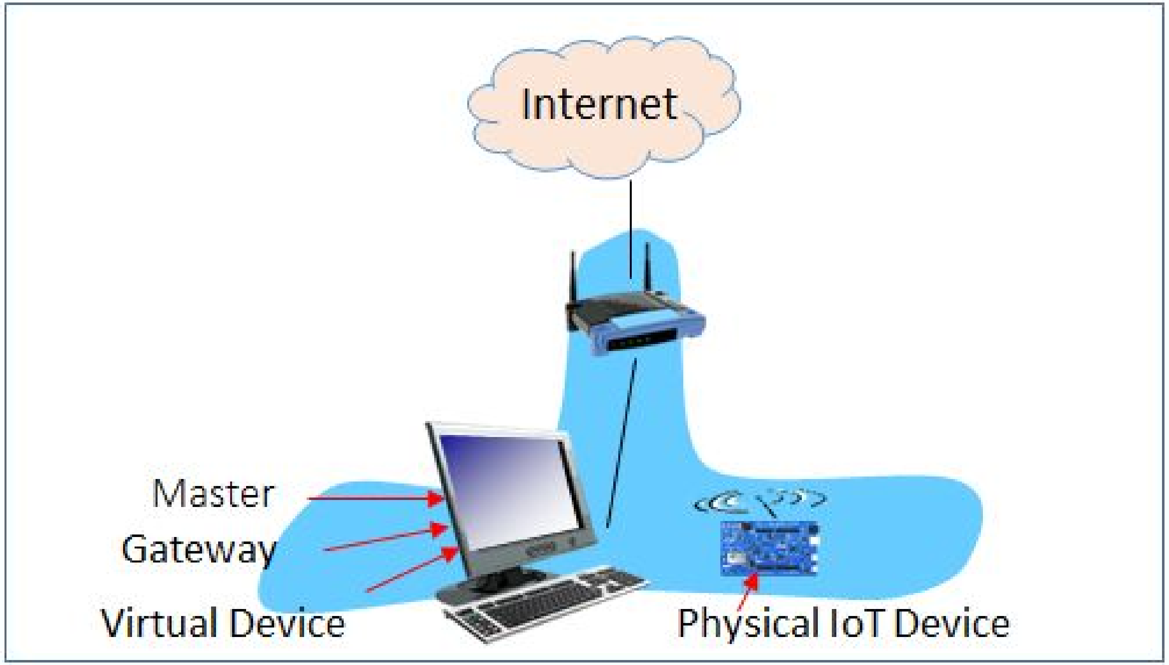


Fig 1b. WuKong-based IoT applications can be run on physical IoT devices connected to a Linux server running Master and gateway.

Three Possible Configurations for WuKong Systems

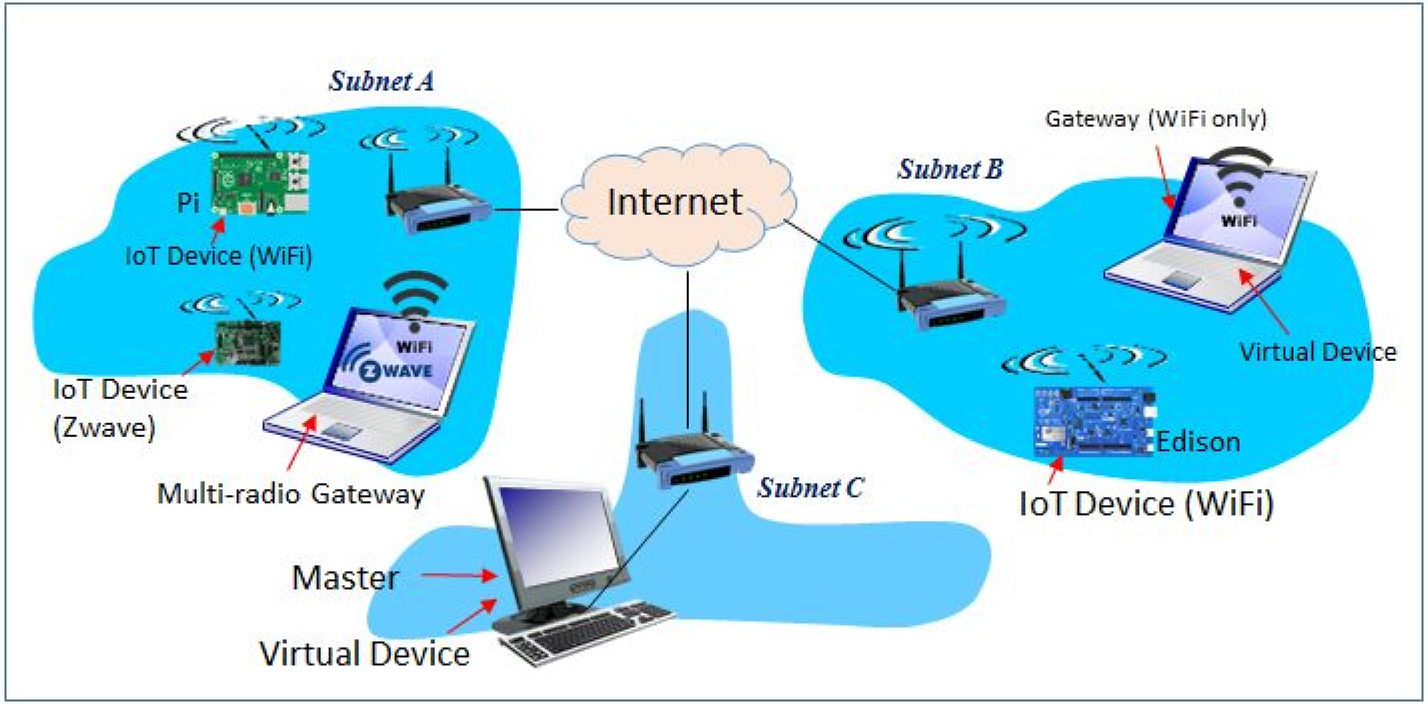


Fig 1c. A full scale WuKong system has a dedicated Master, several gateways each managing devices on different subnets or protocols (e.g. WiFi, ZWave, Zigbee), and IoT devices running on physical and/or virtual machines.

The Configurations for Today

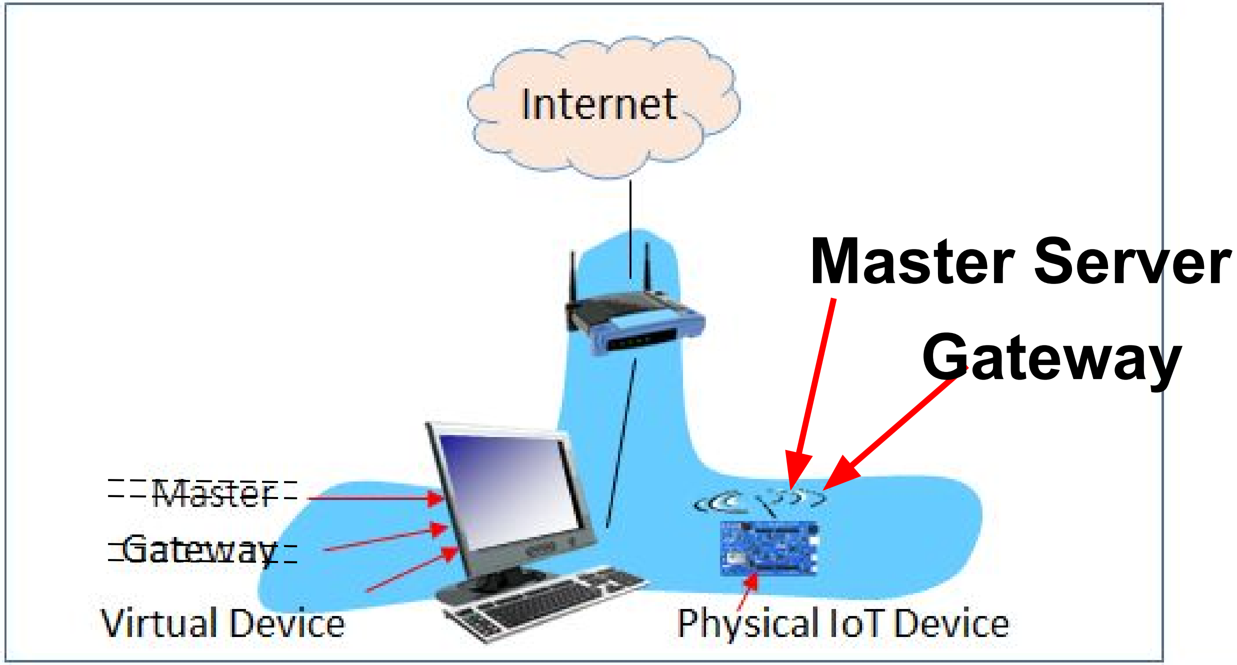


Fig 1b. WuKong-based IoT applications can be run on physical IoT devices connected to a Linux server running Master and gateway.

Start the Master and Gateway

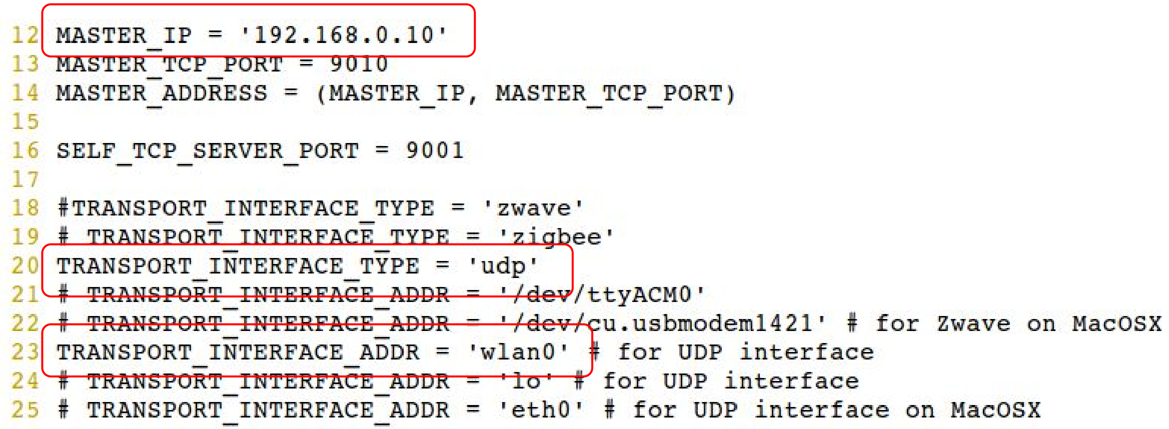
- On Edison or Pi, download the source code from github as below:
```
git clone -b workshop http://github.com/wukong-m2m/wukong-darjeeling
```
- Build infuser:
```
cd ~/wukong-darjeeling/src/infuser/
gradle
```
- Copy the configuration file for Master:
```
cd ~/wukong-darjeeling/wukong/config/
cp master.cfg.dist master.cfg
```
- Run the Master:
```
cd ~/wukong-darjeeling/wukong/master/
python master_server.py
```

Start the Master and Gateway

- On the computer, open a new terminal to log in Edison or Pi again.

- Copy the configuration file for Gateway:
```
cd ~/wukong-darjeeling/wukong/gateway/
cp gtwconfig.py.dist gtwconfig.py
```

- Configure gtwconfig.py:
```
ifconfig                #check ip address and network interface
nano gtwconfig.py       #change MASTER_IP to the IP address of the Master.
                        #change TRANSPORT_INTERFACE_ADDR to your network interface
```



```
MASTER_IP = '192.168.0.10'
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
```

Start the Master and Gateway

- Run the gateway program
  cd ~/wukong-darjeeling/wukong/gateway/
  python start_gateway.py

Include a New Device

- Use the Chrome browser to open the Master interface:
  http://<ip address of edison or pi>:5000

- Then, following the chapter 4 of Gitbook to achieve this example!!