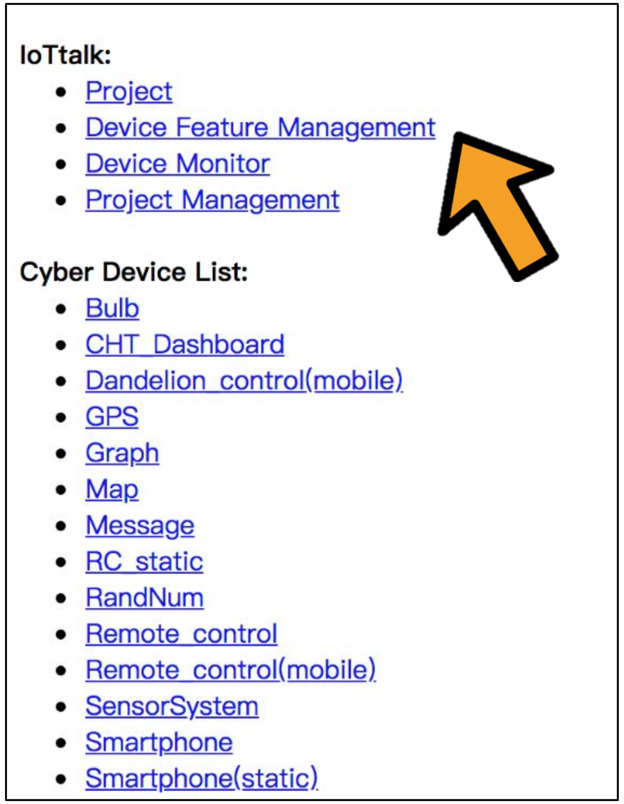
Tutorial - IoTtalk

IoTtalk Application

- An IoT device can be characterized by its functionalities or “device features”.
    - input device feature (IDF), output device feature (ODF)
- An IoT device may be connected to the network (i.e., Internet) using wireless communications directly or indirectly.
- The corresponding software is network application
    - executed by IoTtalk server in the network side, which receives or sends the messages from/to the IoT device

Device Feature Management – Device Feature

- Enter '**Device Feature Management**' from IoTtalk Homepage
- We can define the new **DF** in Device Feature Management page



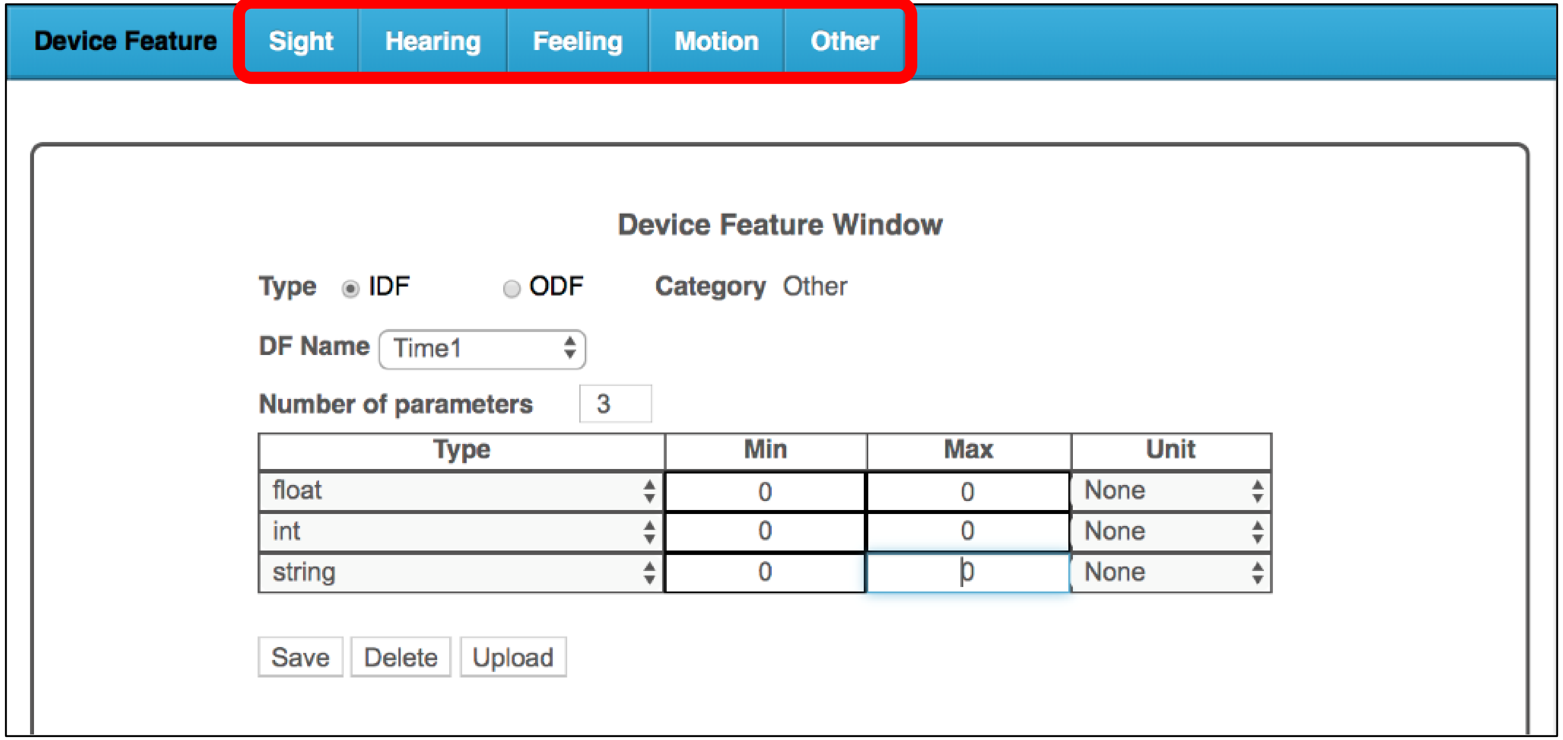
IoTtalk:

- Project
- Device Feature Management
- Device Monitor
- Project Management

Cyber Device List:

- Bulb
- CHT_Dashboard
- Dandelion_control(mobile)
- GPS
- Graph
- Map
- Message
- RC_static
- RandNum
- Remote_control
- Remote_control(mobile)
- SensorSystem
- Smartphone
- Smartphone(static)

IoTtalk Homepage



DF Classification

Device Feature Window

Type ● IDF ○ ODF Category Other

DF Name Time1

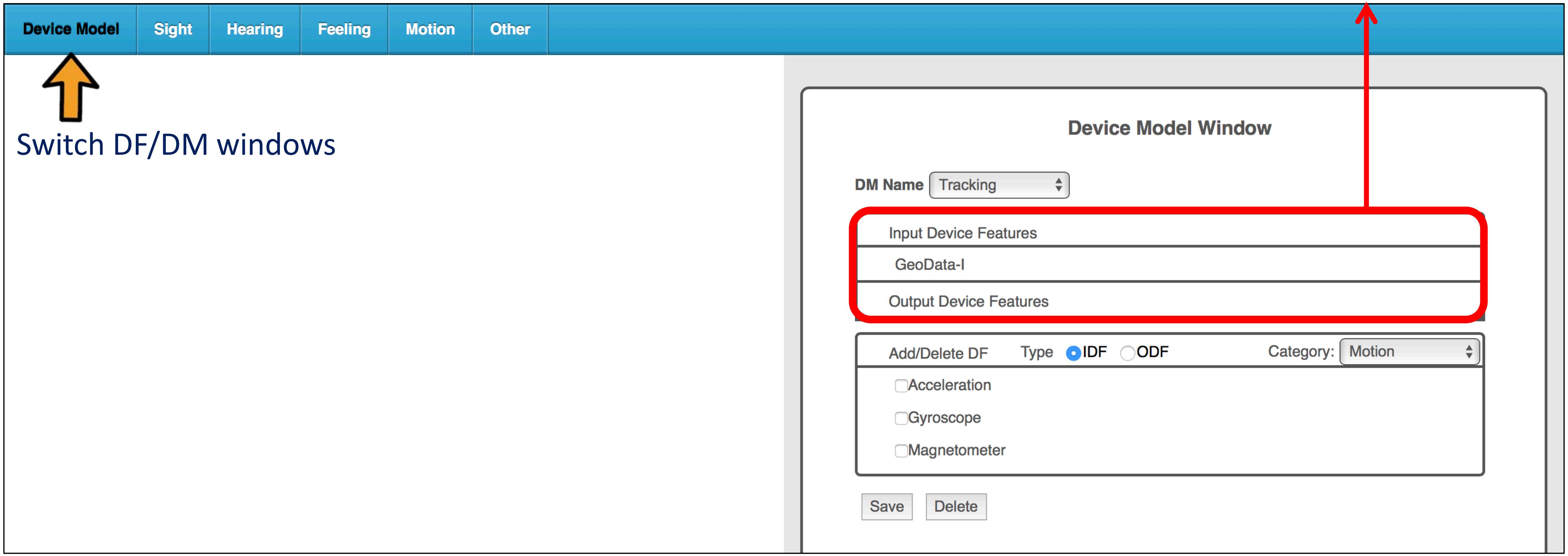
Number of parameters 3

Type	Min	Max	Unit
float	0	0	None
int	0	0	None
string	0	0	None

Device Feature Management page - DF

Device Feature Management – Device Model

- We can also define the new **DM** by adding existing DFs in Device Feature Management page
- After saved, the DM can be used in the IoTtalk project



DFs list in DM"Tracking"

Switch DF/DM windows

Device Model Window

DM Name: Tracking

Input Device Features

GeoData-I

Output Device Features

Add/Delete DF Type ● IDF ○ ODF Category: Motion

□ Acceleration

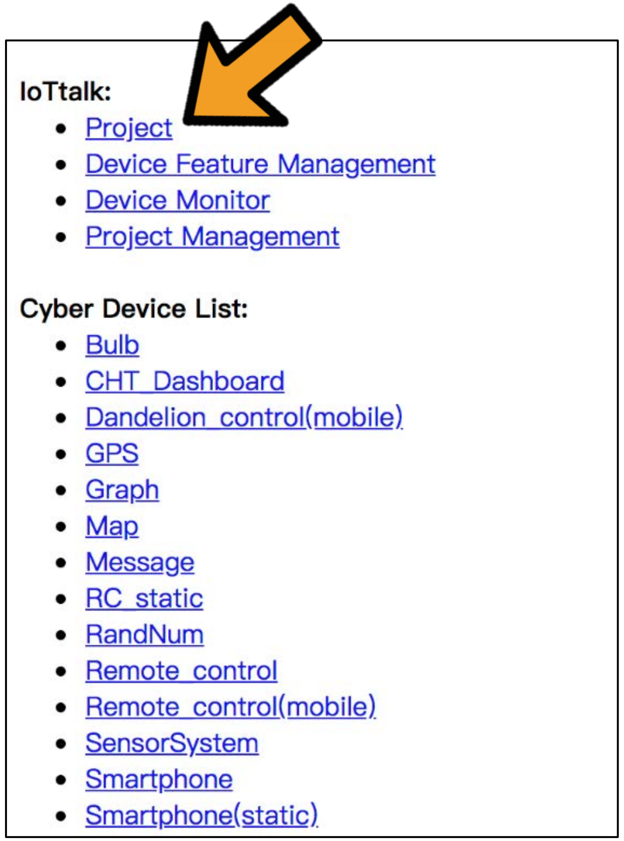
□ Gyroscope

□ Magnetometer

Device Feature Management page - DM

IoTtalk Project Creation

- Enter '**Project**' from IoTtalk Homepage
- Create your own IoTtalk project



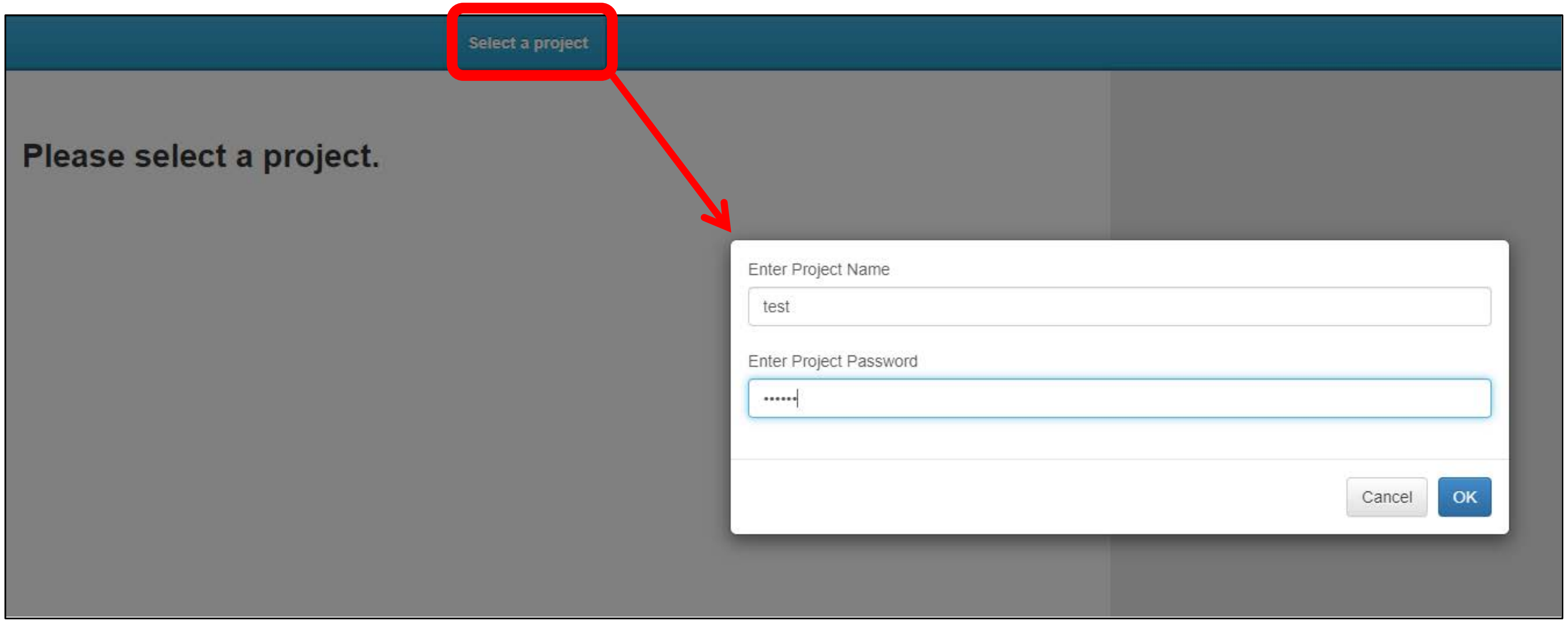
IoTtalk:

- Project
- Device Feature Management
- Device Monitor
- Project Management

Cyber Device List:

- Bulb
- CHT_Dashboard
- Dandelion_control(mobile)
- GPS
- Graph
- Map
- Message
- RC_static
- RandNum
- Remote_control
- Remote_control(mobile)
- SensorSystem
- Smartphone
- Smartphone(static)

IoTtalk Homepage



Please select a project.

Enter Project Name

test

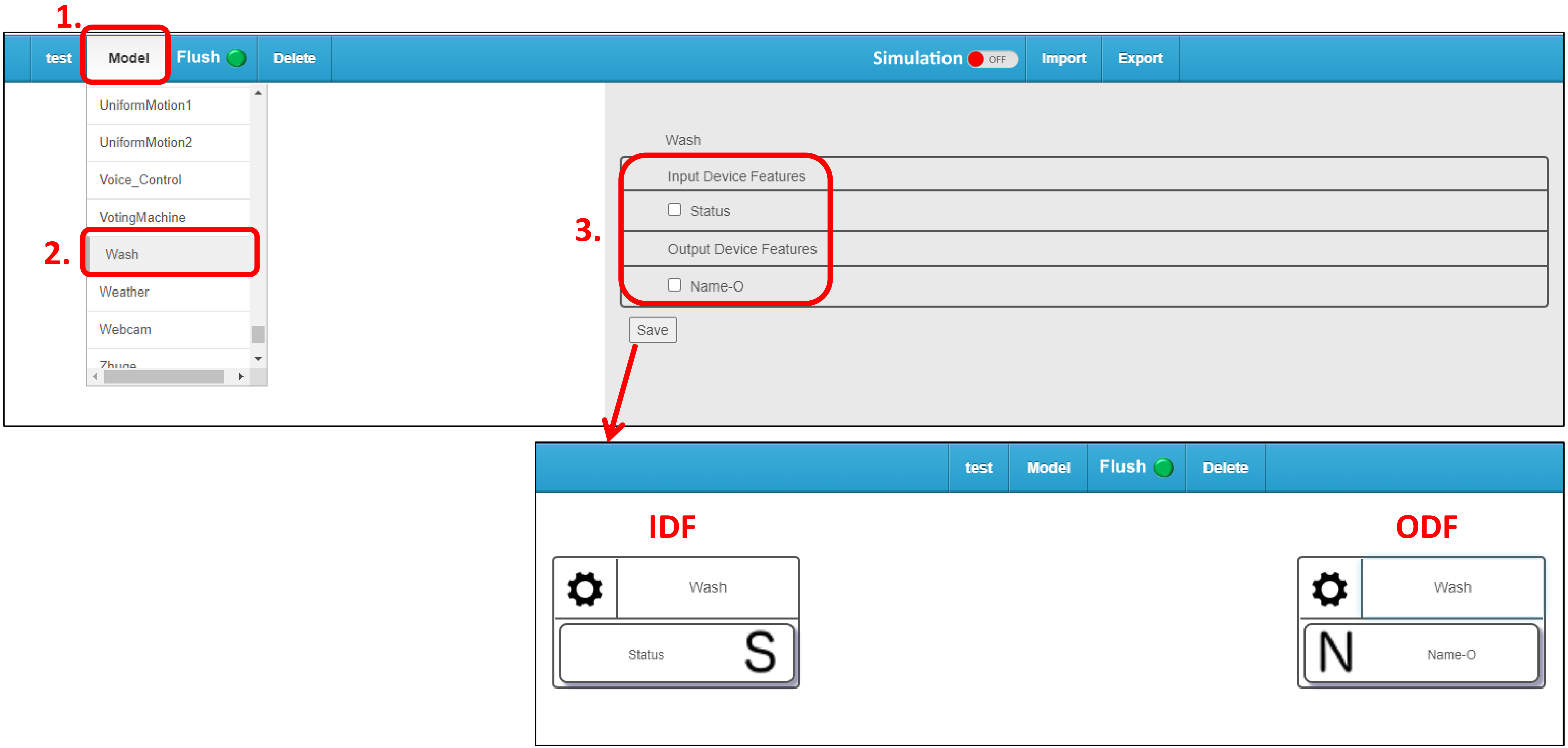
Enter Project Password

••••••

Project page

Project Design

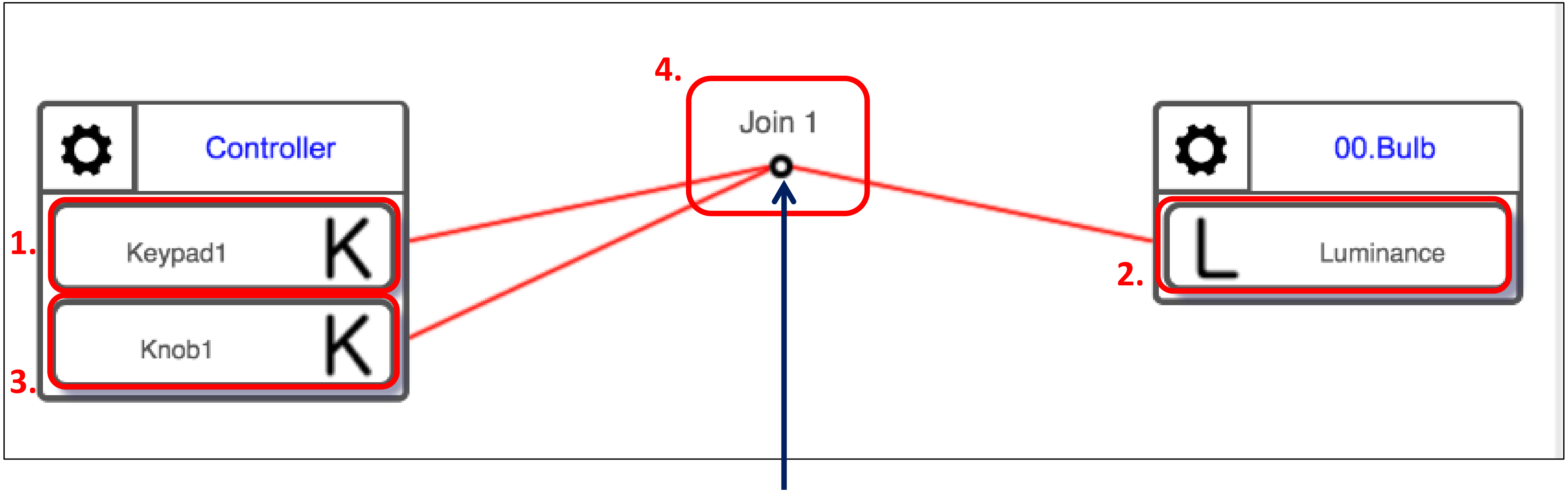
- Add those DMs you need in the project



I/O Connection

- Add those DMs you need in the project

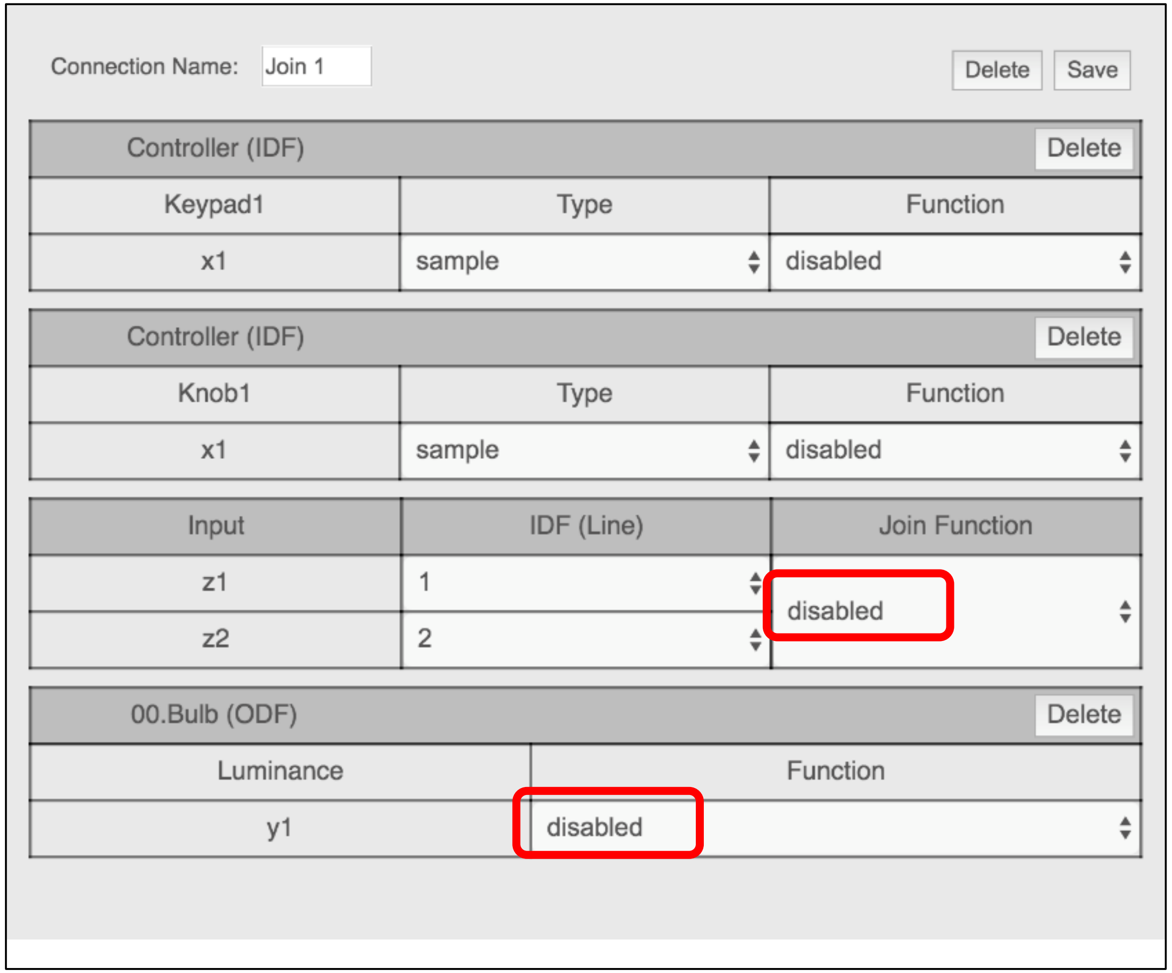
- Click the DF you want to link, and a red line will appear between the two sides (Join 1)



Right click to show function setting window (next slide)

Function Setting(1/3)

- Join Function
  - Set up the connection between IDF and ODF
- IDFs/ODFs Function
  - Design the required service logic for each IDF/ODF
- Design the functions (red box)
  - Drop down the combobox
  - Click “add new function”



Connection Name: Join 1

Controller (IDF)

Keypad1	Type	Function
x1	sample	disabled

Controller (IDF)

Knob1	Type	Function
x1	sample	disabled

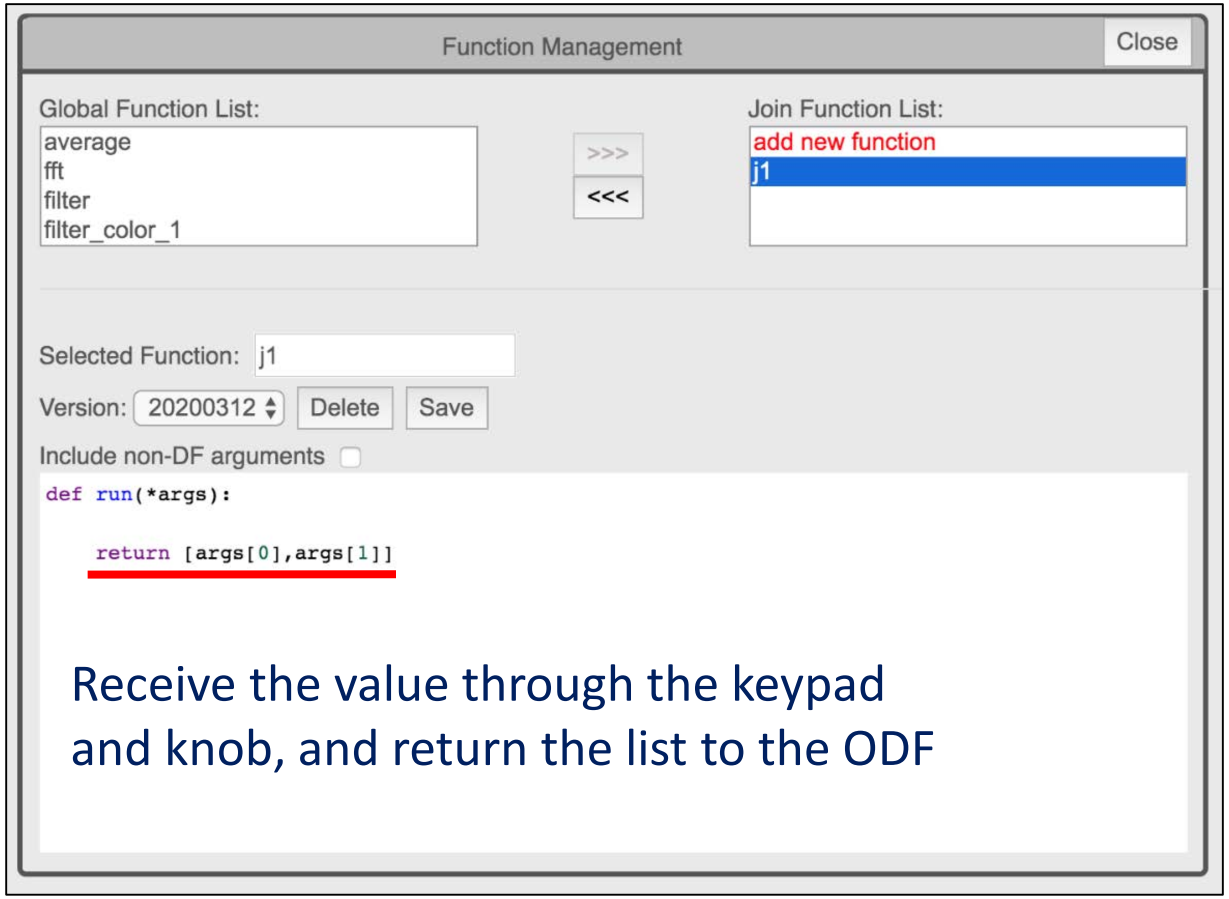
Input	IDF (Line)	Join Function
z1	1	disabled
z2	2	

00.Bulb (ODF)

Luminance	Function
y1	disabled

Function Setting(2/3)

- You can use the built-in basic functions directly, or you can define your own
- Take Controller <--> Bulb on page 7 for example



Selected Function: j1

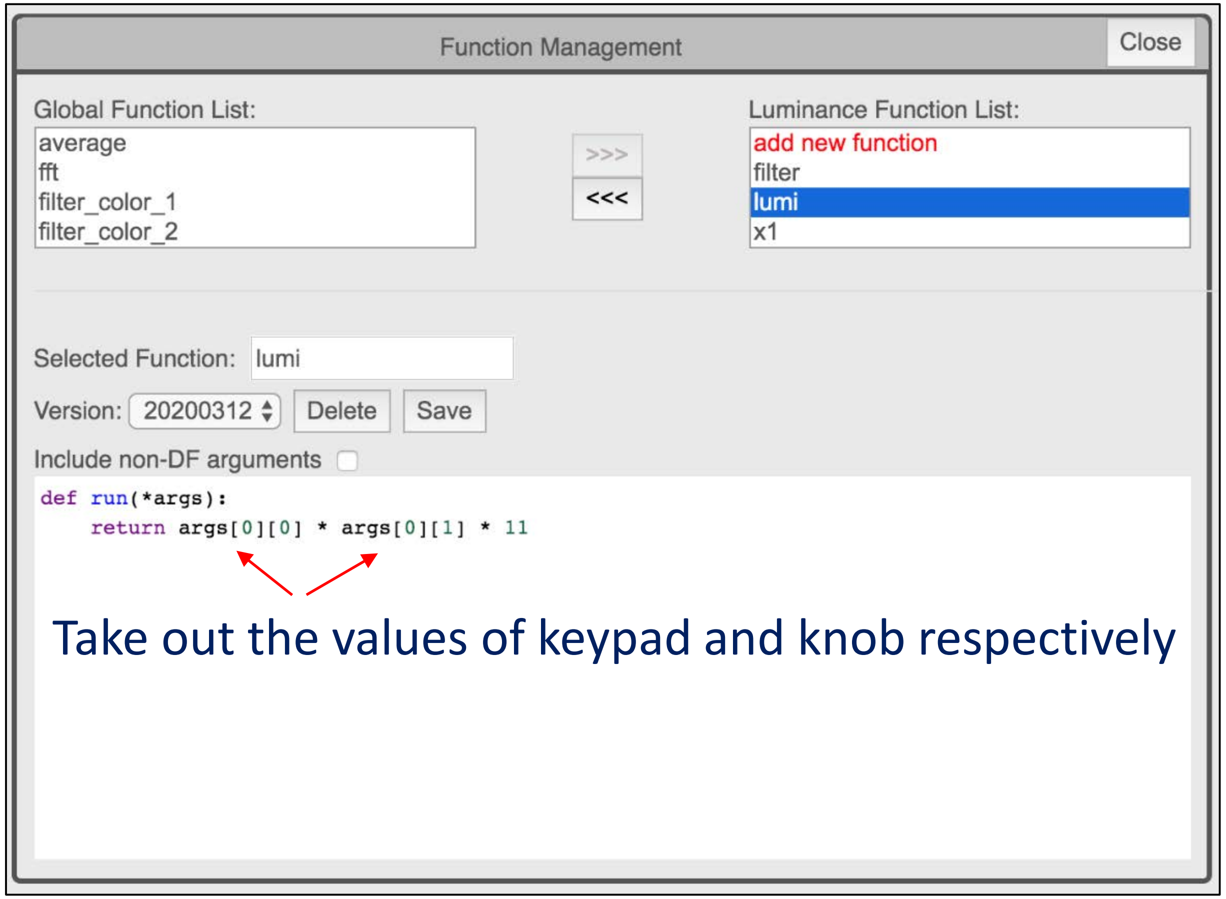
Version: 20200312

```
def run(*args):

    return [args[0],args[1]]
```

Receive the value through the keypad and knob, and return the list to the ODF

Join Function



Selected Function: lumi

Version: 20200312

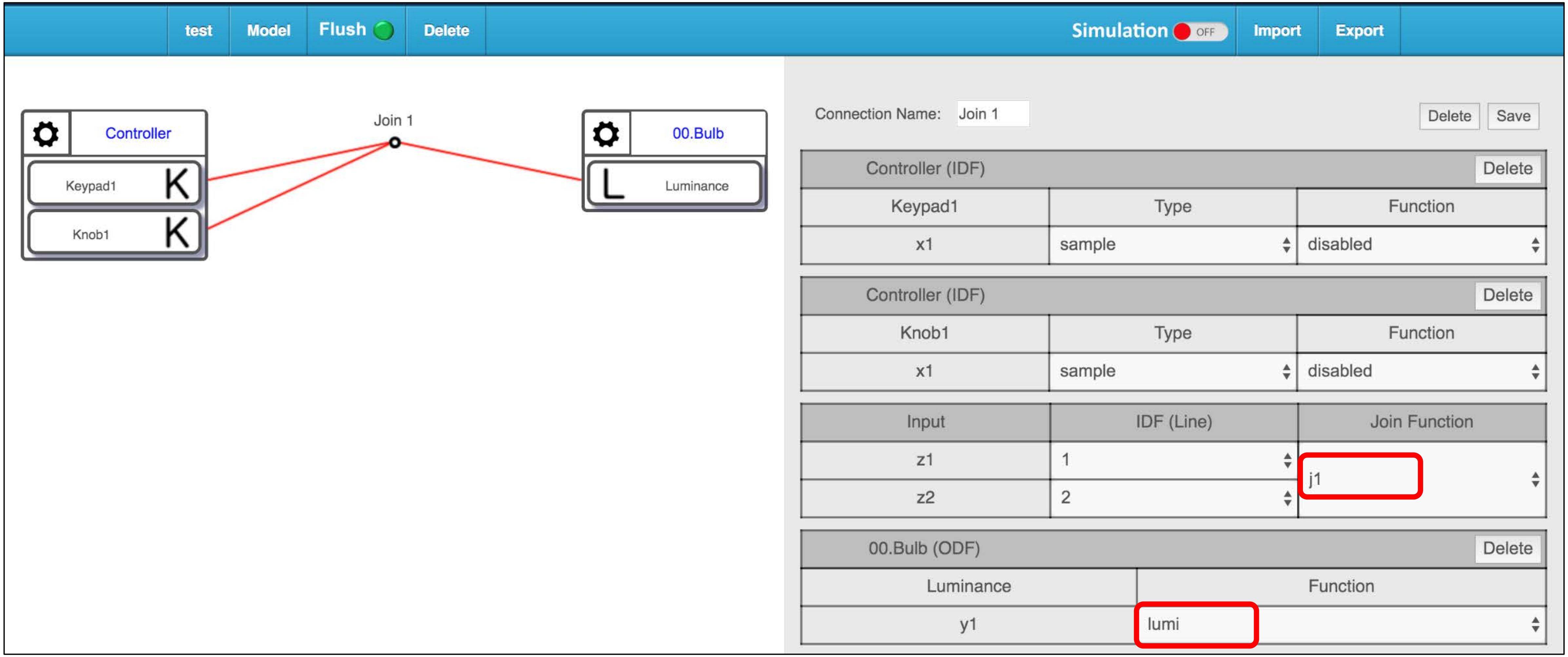
```
def run(*args):
    return args[0][0] * args[0][1] * 11
```

Take out the values of keypad and knob respectively

IDF/ODF Function

Function Setting(3/3)

- After the new function is saved, it can be selected from the combobox



Connection Name: Join 1

Controller (IDF)

Keypad1	Type	Function
x1	sample	disabled

Controller (IDF)

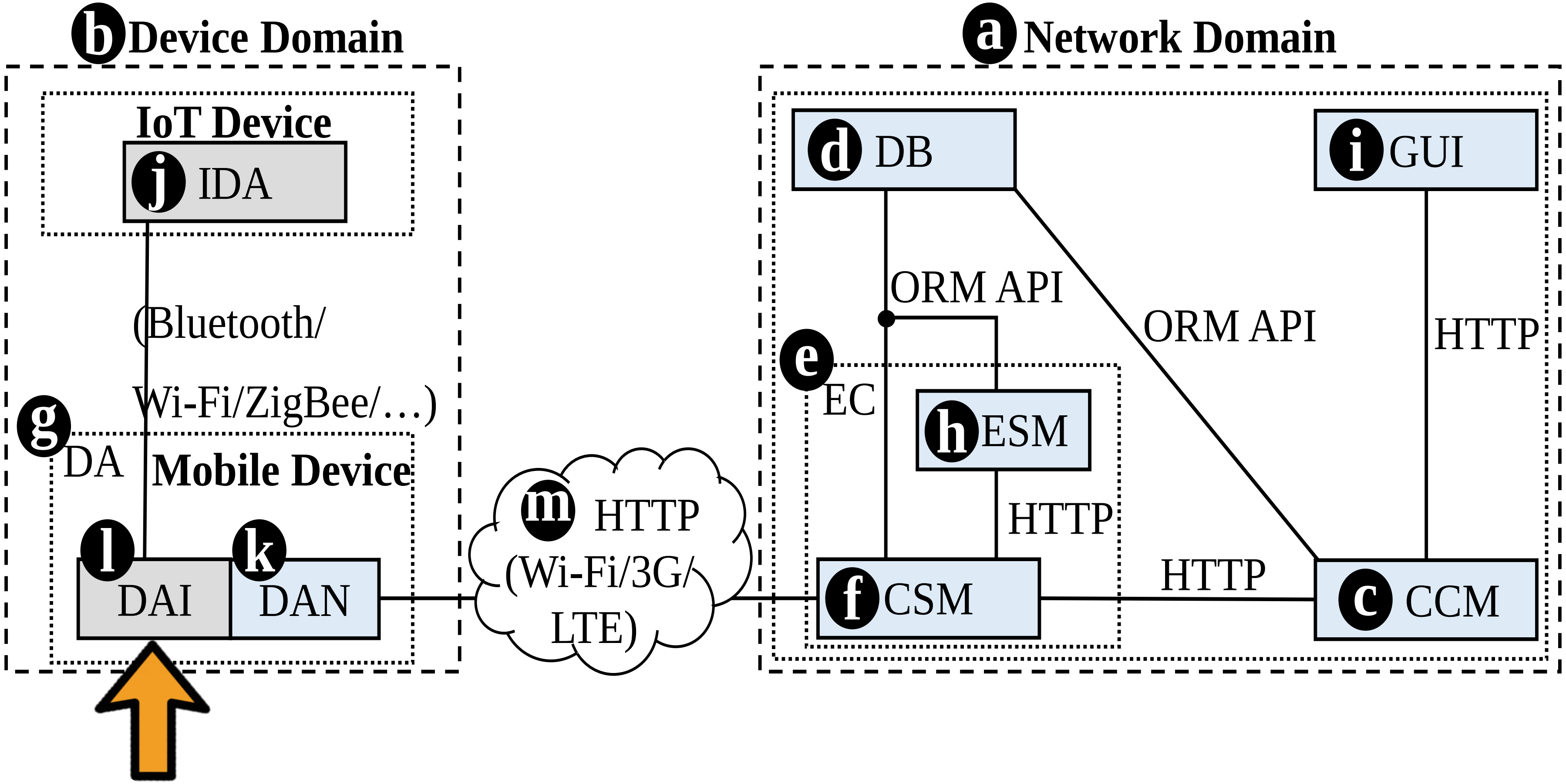
Knob1	Type	Function
x1	sample	disabled

Input	IDF (Line)	Join Function
z1	1	j1
z2	2	

00.Bulb (ODF)

Luminance	Function
y1	lumi

DAI Setting



DAI = Device Application to IoT Device

Sample DAI.py(1/2)

```
import time, random, requests
import DAN
```

Connect to IoTtalk server

```
ServerURL = 'http://IoTtalk Server IP:9999' #with non-secure connection
Reg_addr = None #if None, Reg_addr = MAC address
```

Define device profile

```
DAN.profile['dm_name']='Wash'
DAN.profile['df_list']=['Status','Name-O']
#DAN.profile['d_name']= 'Assign a Device Name'
```

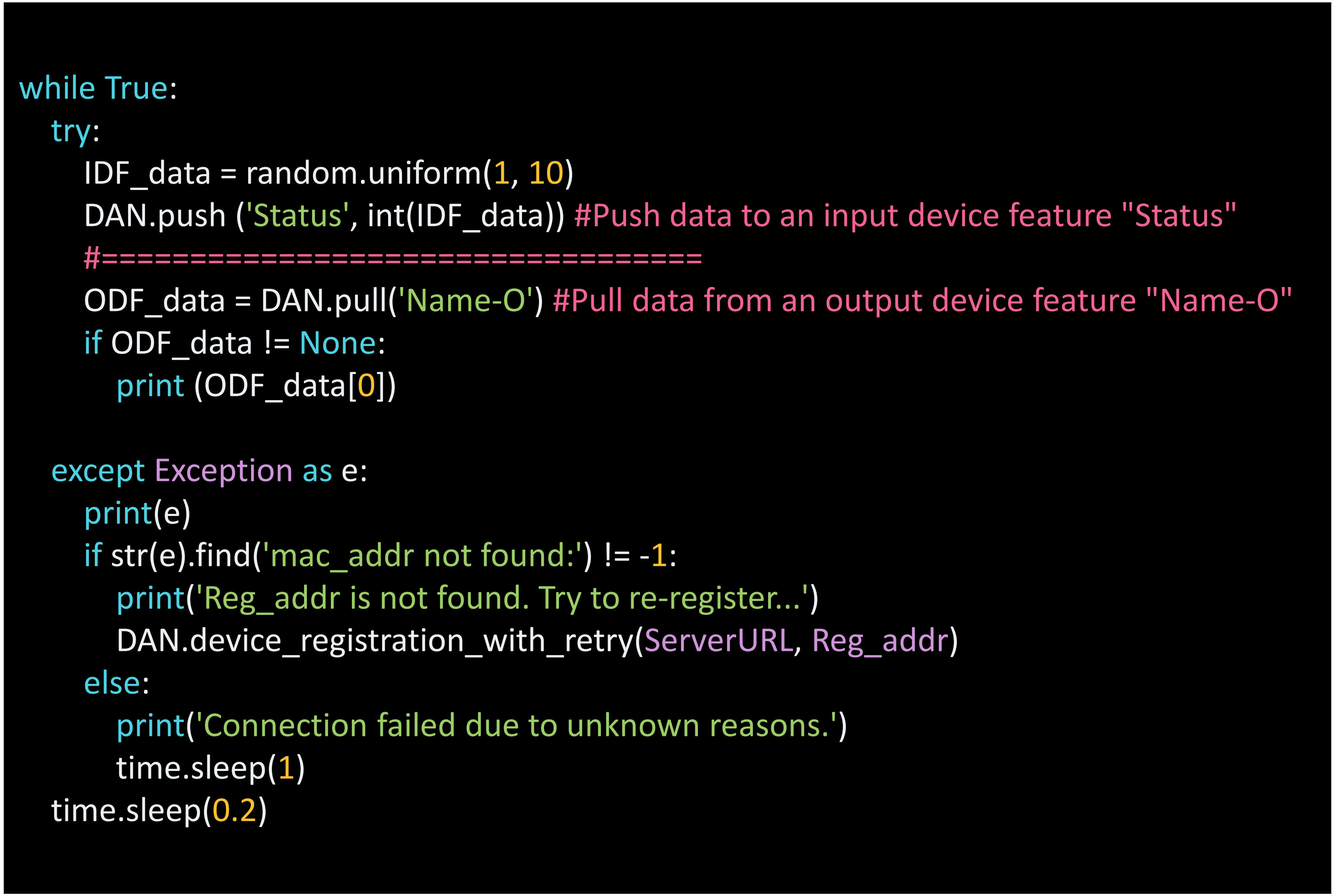
Register/Deregister device to IoTtalk

```
DAN.device_registration_with_retry(ServerURL, Reg_addr)
#DAN.deregister() #if you want to deregister this device, uncomment this line
#exit() #if you want to deregister this device, uncomment this line
```

Sample DAI.py(2/2)

Random values and push them into IoTtalk

Pull the value from IoTtalk

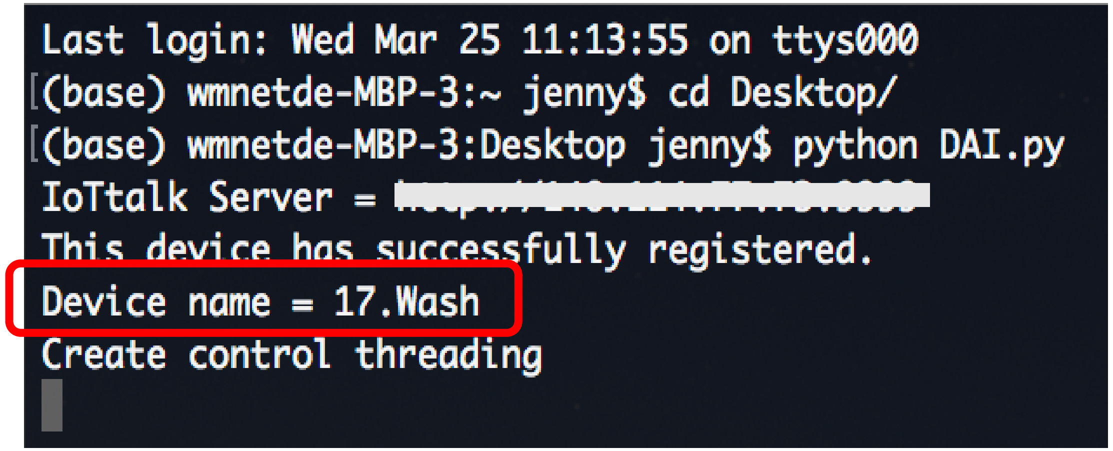


```
while True:
    try:
        IDF_data = random.uniform(1, 10)
        DAN.push ('Status', int(IDF_data)) #Push data to an input device feature "Status"
        #==================================
        ODF_data = DAN.pull('Name-O') #Pull data from an output device feature "Name-O"
        if ODF_data != None:
            print (ODF_data[0])

    except Exception as e:
        print(e)
        if str(e).find('mac_addr not found:') != -1:
            print('Reg_addr is not found. Try to re-register...')
            DAN.device_registration_with_retry(ServerURL, Reg_addr)
        else:
            print('Connection failed due to unknown reasons.')
            time.sleep(1)
    time.sleep(0.2)
```

DAI Execution

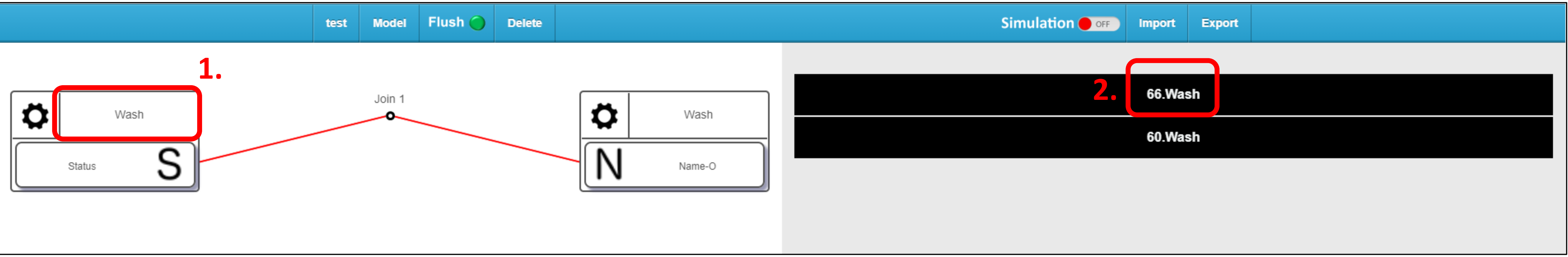
- Execute DAI.py
    - *python DAI.py*
- Information from terminal
    - **Device name = 17.Wash**
    - The terminal will display the number of registered Device



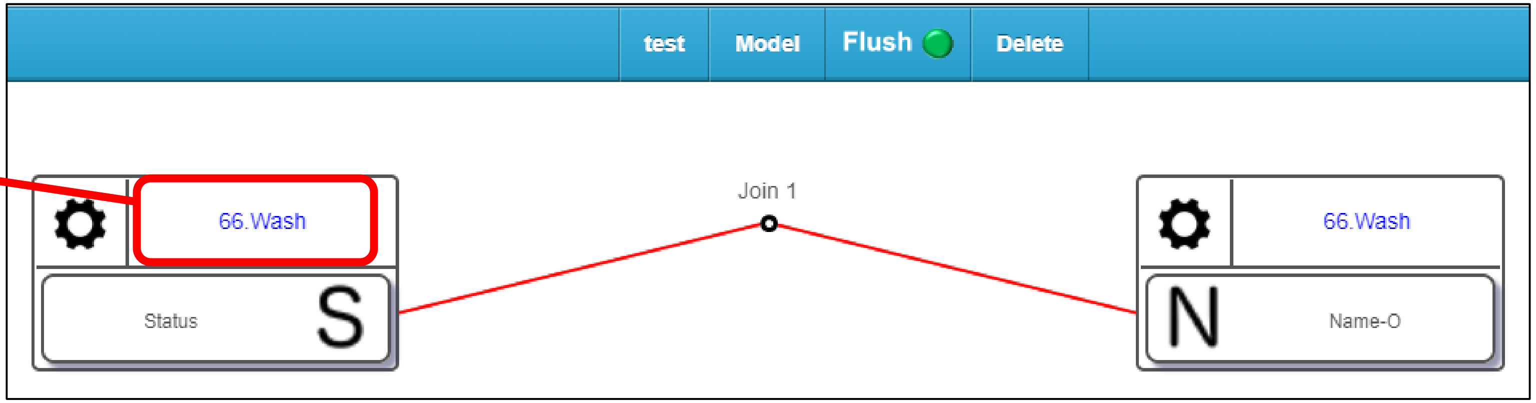
```
Last login: Wed Mar 25 11:13:55 on ttys000
[(base) wmnetde-MBP-3:~ jenny$ cd Desktop/
[(base) wmnetde-MBP-3:Desktop jenny$ python DAI.py
IoTtalk Server = [redacted]
This device has successfully registered.
Device name = 17.Wash
Create control threading
```

Device Binding

- After DAI.py is executed, the registered device will appear on the right window, and then bind the devices
- **Remember to correspond the device number**

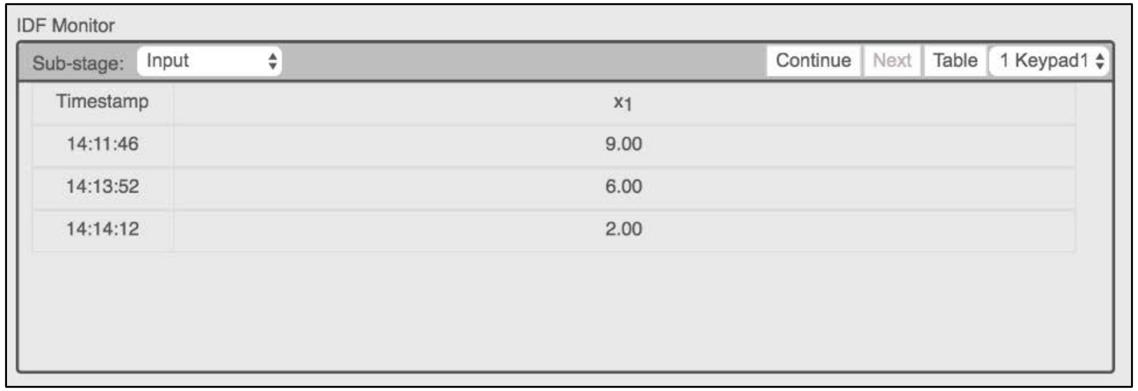


After binding,
the text will be displayed in blue



Other - Monitor

- Click the left button at the Join point
  - can observe the input and output data

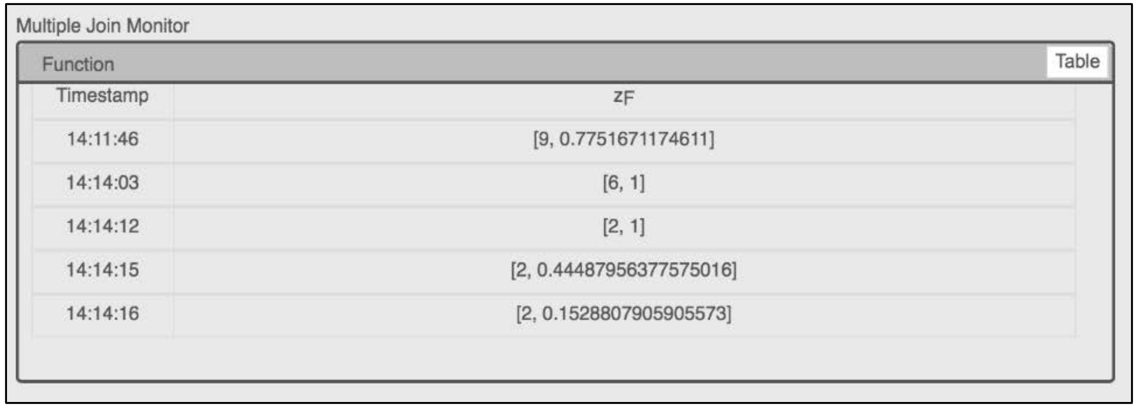


IDF Monitor

Sub-stage: Input

1 Keypad1

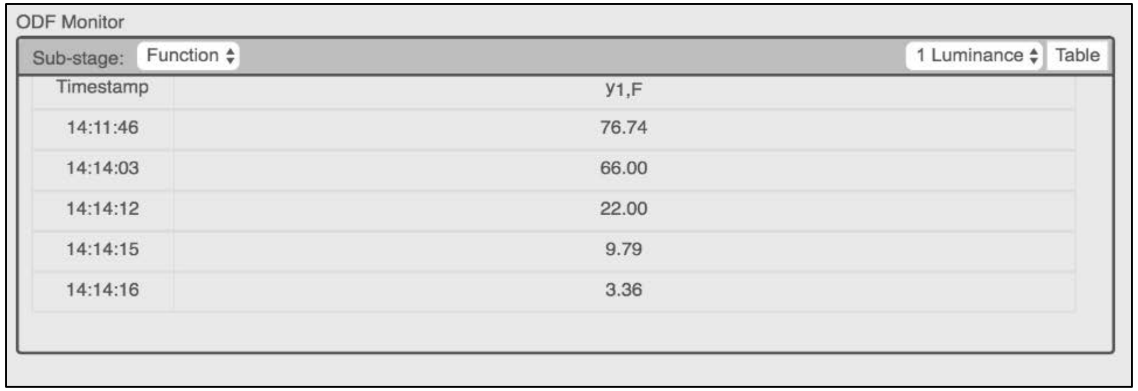
Timestamp	x_1
14:11:46	9.00
14:13:52	6.00
14:14:12	2.00



Multiple Join Monitor

Function

Timestamp	z_F
14:11:46	[9, 0.7751671174611]
14:14:03	[6, 1]
14:14:12	[2, 1]
14:14:15	[2, 0.44487956377575016]
14:14:16	[2, 0.1528807905905573]



ODF Monitor

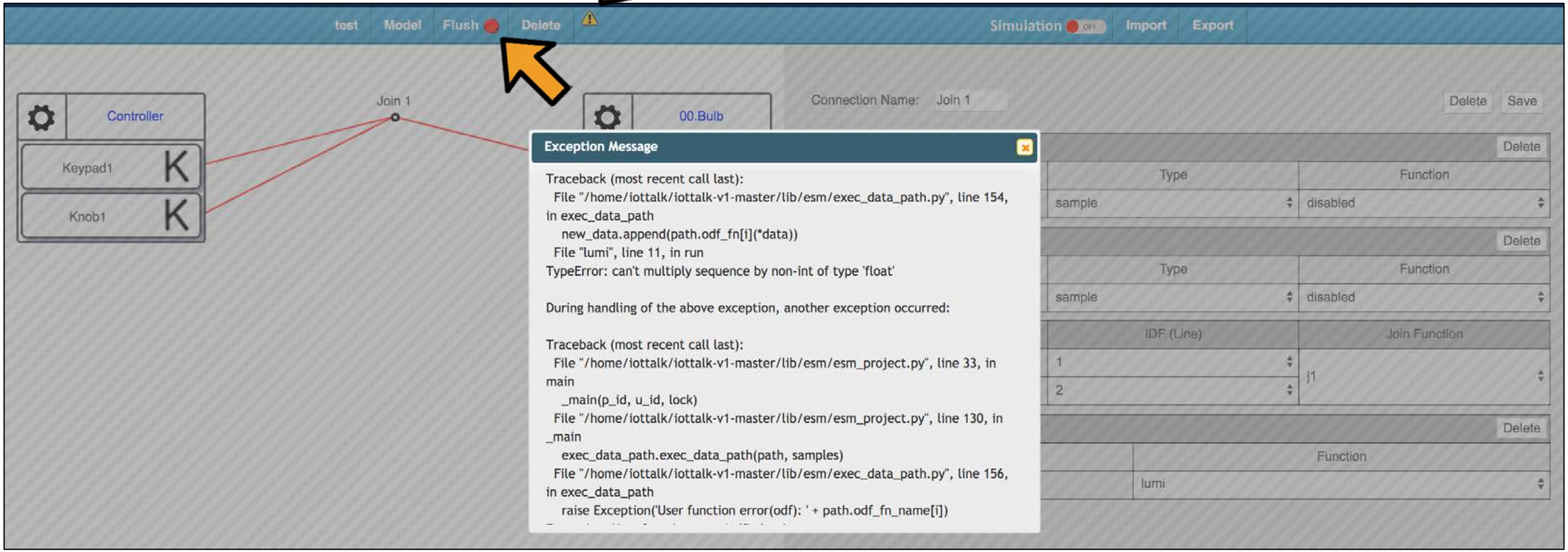
Sub-stage: Function

1 Luminance

Timestamp	$y_{1,F}$
14:11:46	76.74
14:14:03	66.00
14:14:12	22.00
14:14:15	9.79
14:14:16	3.36

Other - Exception message & Flush

- **Exception message**
  - If a warning appears during execution, it should be a mistake in your code.
- **Flush**
  - Click here to flush and restart



Connection Name: Join 1

Type	Function
sample	disabled

Type	Function
sample	disabled

Function
lumi

Exception Message

```
Traceback (most recent call last):
  File "/home/iottalk/iottalk-v1-master/lib/esm/exec_data_path.py", line 154,
in exec_data_path
    new_data.append(path.odf_fn[i](*data))
  File "lumi", line 11, in run
TypeError: can't multiply sequence by non-int of type 'float'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/iottalk/iottalk-v1-master/lib/esm/esm_project.py", line 33, in
main
    _main(p_id, u_id, lock)
  File "/home/iottalk/iottalk-v1-master/lib/esm/esm_project.py", line 130, in
_main
    exec_data_path.exec_data_path(path, samples)
  File "/home/iottalk/iottalk-v1-master/lib/esm/exec_data_path.py", line 156,
in exec_data_path
    raise Exception('User function error(odf): ' + path.odf_fn_name[i])
```

Tutorial - LineBot & Heroku

LineBot Application

- **Building a LineBot with Heroku**
- **Heroku**
  - Heroku is a platform as a service that enables developers to build, run, and operate applications entirely in the cloud
- **Deploy with Git or Docker**
  - Git

    As long as the developer pushes the code to the Heroku repository, Heroku can automatically determine the language and deploy it.
  - Docker

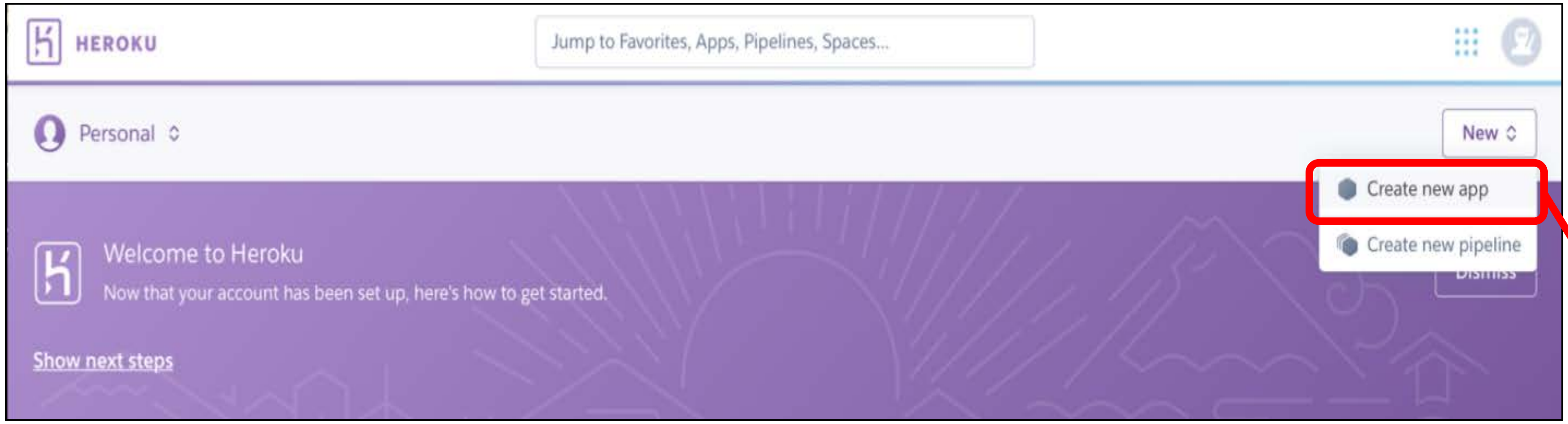
    As long as the developer put a Dockerfile in the repository and upload the Docker container to the Heroku Docker Registry through Heroku CLI, Heroku can automatically deploy the website.

Preliminary

- You need to have :
  - A Line Account (https://developers.line.biz/en/)
  - A Heroku Account (https://www.heroku.com/)
- You need to download :
  - Heroku CLI (https://devcenter.heroku.com/articles/heroku-cli)
  - Git (https://git-scm.com)
- Check if you install successfully
  - **git --version**
  - **heroku --version**

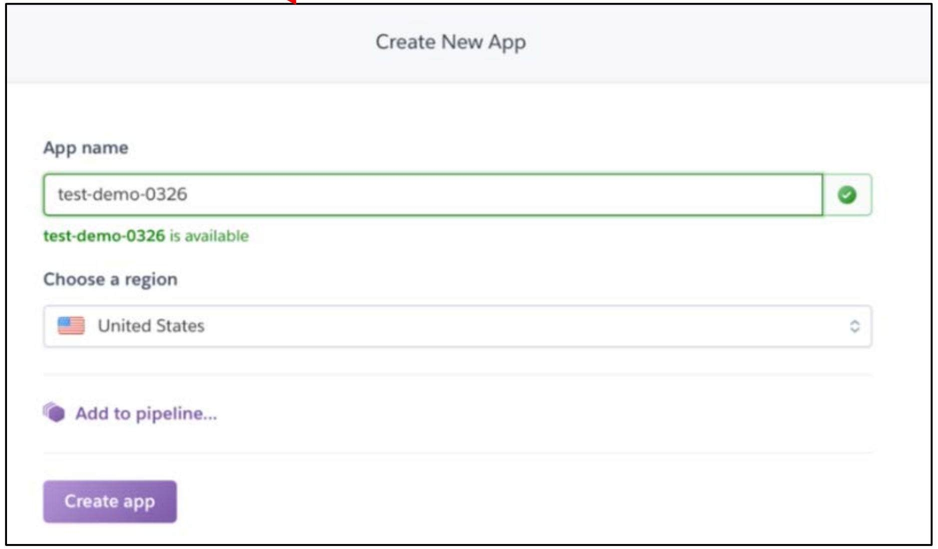
Create a Heroku Project

- Login Heroku >> Create new app >> Type your <Heroku App name>



Welcome to Heroku

Now that your account has been set up, here's how to get started.



Create New App

App name

test-demo-0326

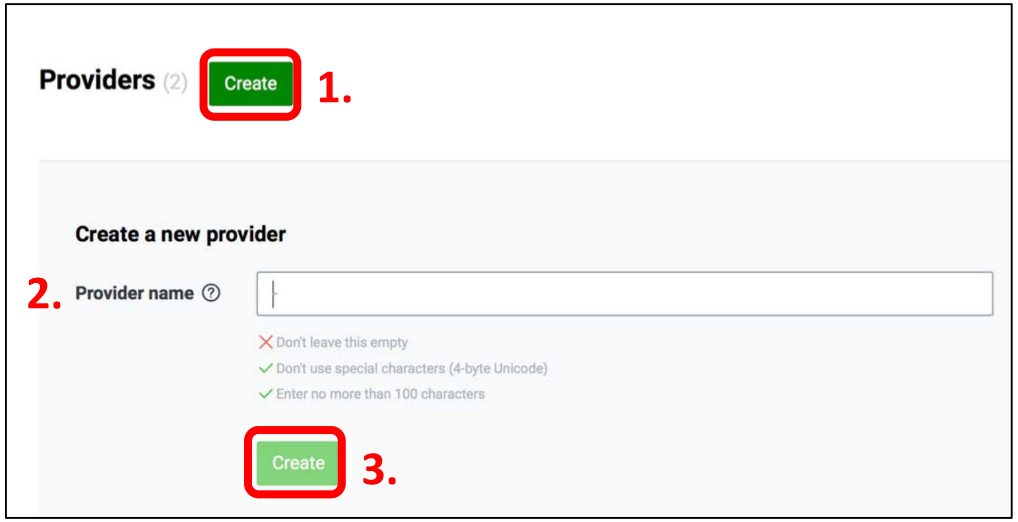
test-demo-0326 is available

Choose a region

United States

Create a LineBot Channel

- Enter the Line Control Console (with your Line Account)
- Create a provider
- Choose “Create a Message API channel”
  - Setting some information: Channel type, Provider, Channel name, Channel description, Category, Subcategory, Email address



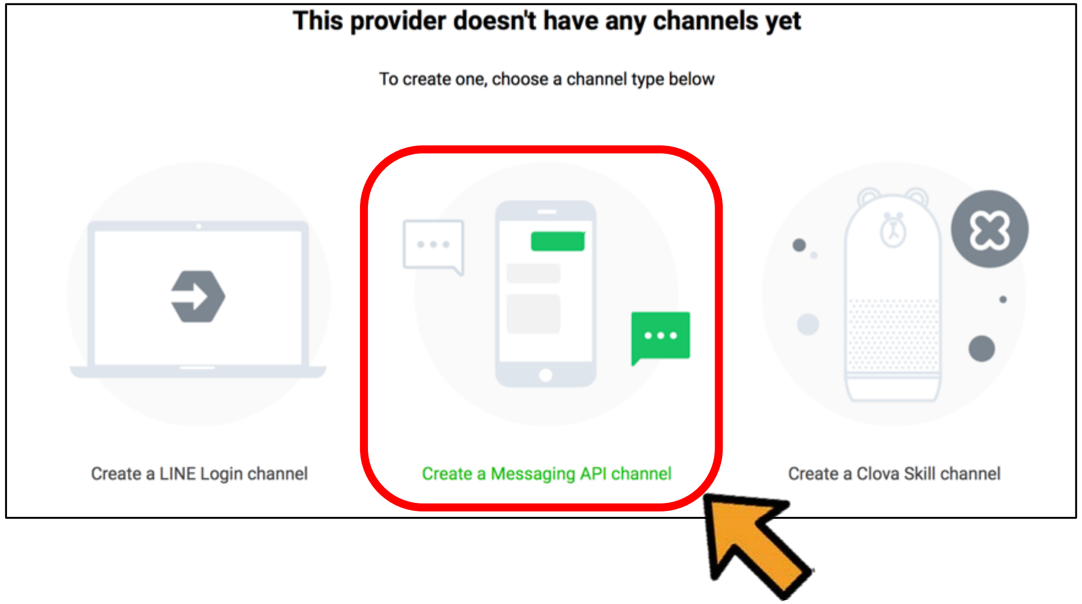
Providers (2) Create 1.

Create a new provider

2. Provider name ⓘ

✗ Don't leave this empty
✓ Don't use special characters (4-byte Unicode)
✓ Enter no more than 100 characters

Create 3.



This provider doesn't have any channels yet

To create one, choose a channel type below

Create a LINE Login channel

Create a Messaging API channel

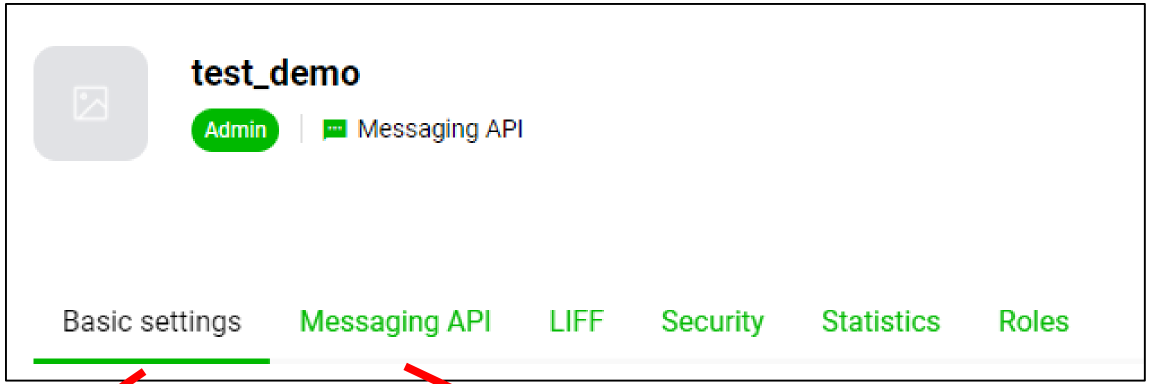
Create a Clova Skill channel

Get Channel Information

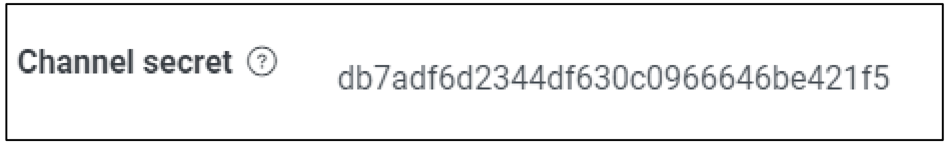
- Record **Channel Access Token** and **Channel Secret**, which will be used in linebot implementation

- Get Channel Secret on **Basic settings** page

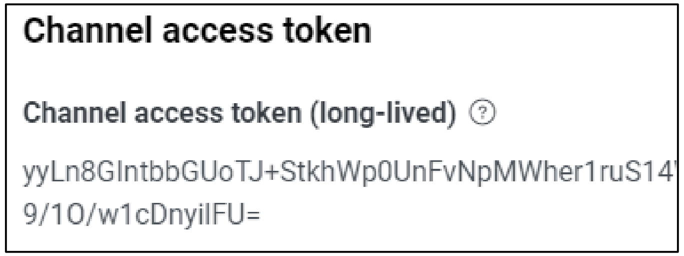
- Get Channel Access Token on **Messaging API** page



test_demo



Channel secret ⓘ db7adf6d2344df630c0966646be421f5



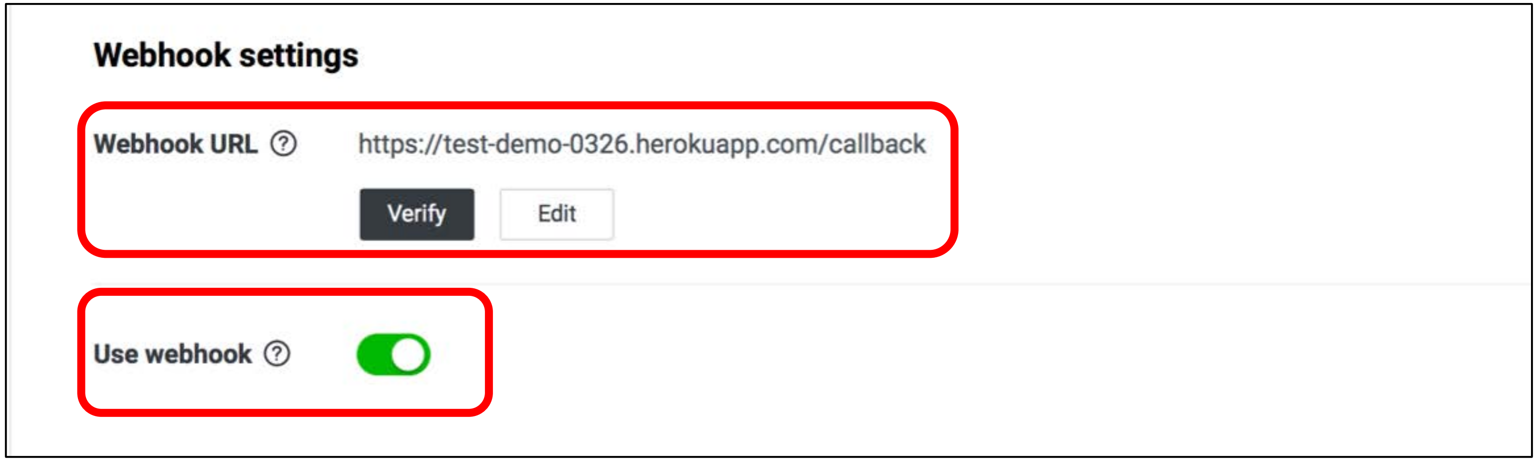
Channel access token

Channel access token (long-lived) ⓘ

yyLn8GIntbbGUoTJ+StkhWp0UnFvNpMWher1ruS14
9/1O/w1cDnyilFU=

Message API Settings

- **Webhook Settings**
  - Webhook URL
    - https://{HEROKU_APP_NAME}.herokuapp.com/callback
    - Click “Update”, do not need to click “Verify”
  - Enable “Use webhook”



Webhook settings

Webhook URL ⓘ https://test-demo-0326.herokuapp.com/callback

Use webhook ⓘ ●

LineBot Sample Package – echo response

- **Procfile**
  - Heroku apps include a **Procfile** that specifies the commands that are executed by the app on startup
  - Procfile Format >> web: [language] [file to be uploaded]
  - E.g. web: python app.py
- **requirements.txt**
  - List all the packages we could use, and Heroku will install these based on the document
- **app.py**
  - The file is a sample code for echo response.
  - The function handle_message() is used to control the message reply

Heroku App Deployment(1/2)

- Enter the folder where your packages are

```
>>   cd <folder>
```

- Login the Heroku

```
>> heroku login
```

- Initial Git (Type this command if you run the code at the first time)

```
>> git config --global user.name ”Your Name"
>> git config --global user.email Your Email
>> git init
```

Heroku App Deployment(2/2)

- Link your folder and Heroku

```
>> heroku git:remote -a <HEROKU_APP_NAME>
```

- Upload your code to Heroku

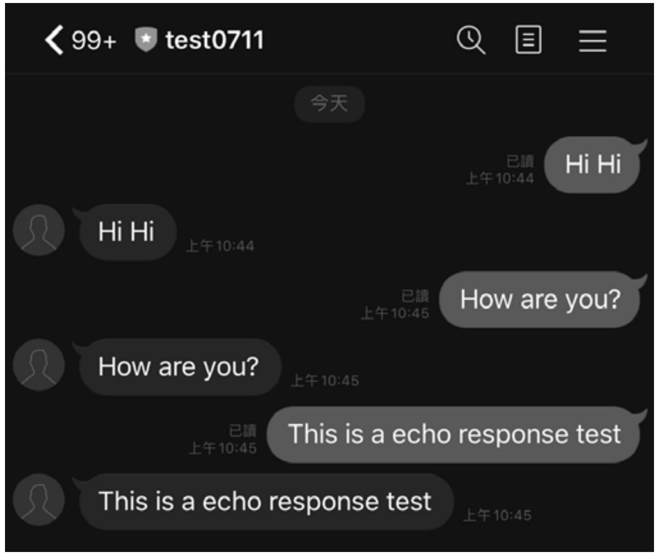
```
>> git add .
>> git commit -m "Add code”
>> git push -f heroku master
```

- **Start** the worker dyno on the Heroku app

```
>> heroku ps:scale web=1
```

- **Stop** the worker dyno on the Heroku app

```
>> heroku ps:scale web=0
```



99+ test0711

今天

已讀 上午10:44 Hi Hi

Hi Hi 上午10:44

已讀 上午10:45 How are you?

How are you? 上午10:45

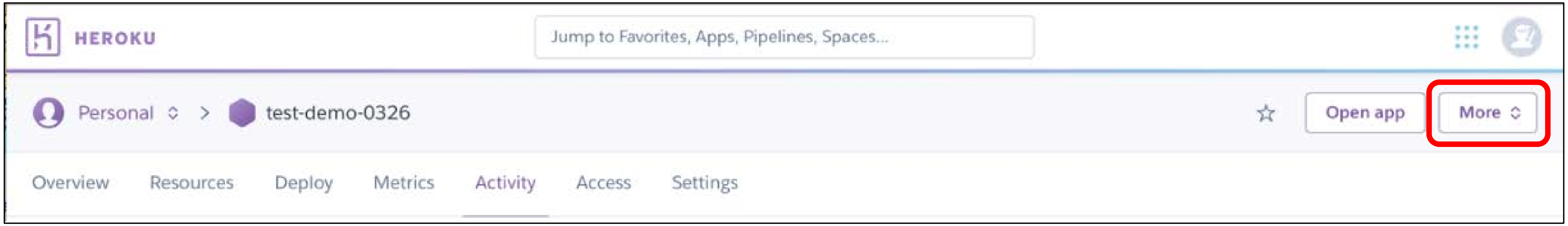
已讀 上午10:45 This is a echo response test

This is a echo response test 上午10:45

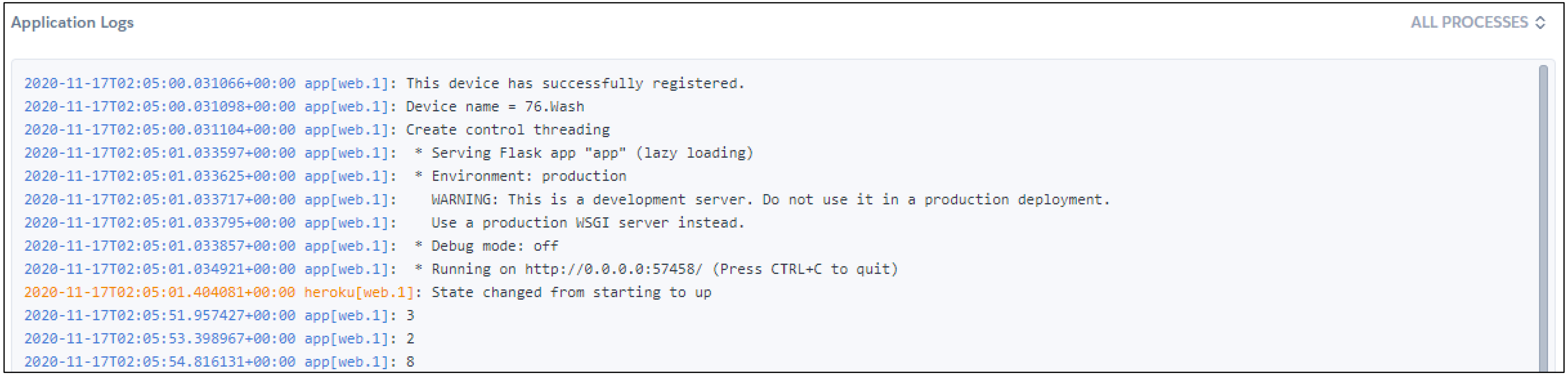
- Then, scan the **QR code**(on Messaging API page) with LINE to add your LINE Official Account as a friend and test

Log Information(1/2)

- You can log in the website of Heroku to check if the deployment is done successfully and view logs



View logs

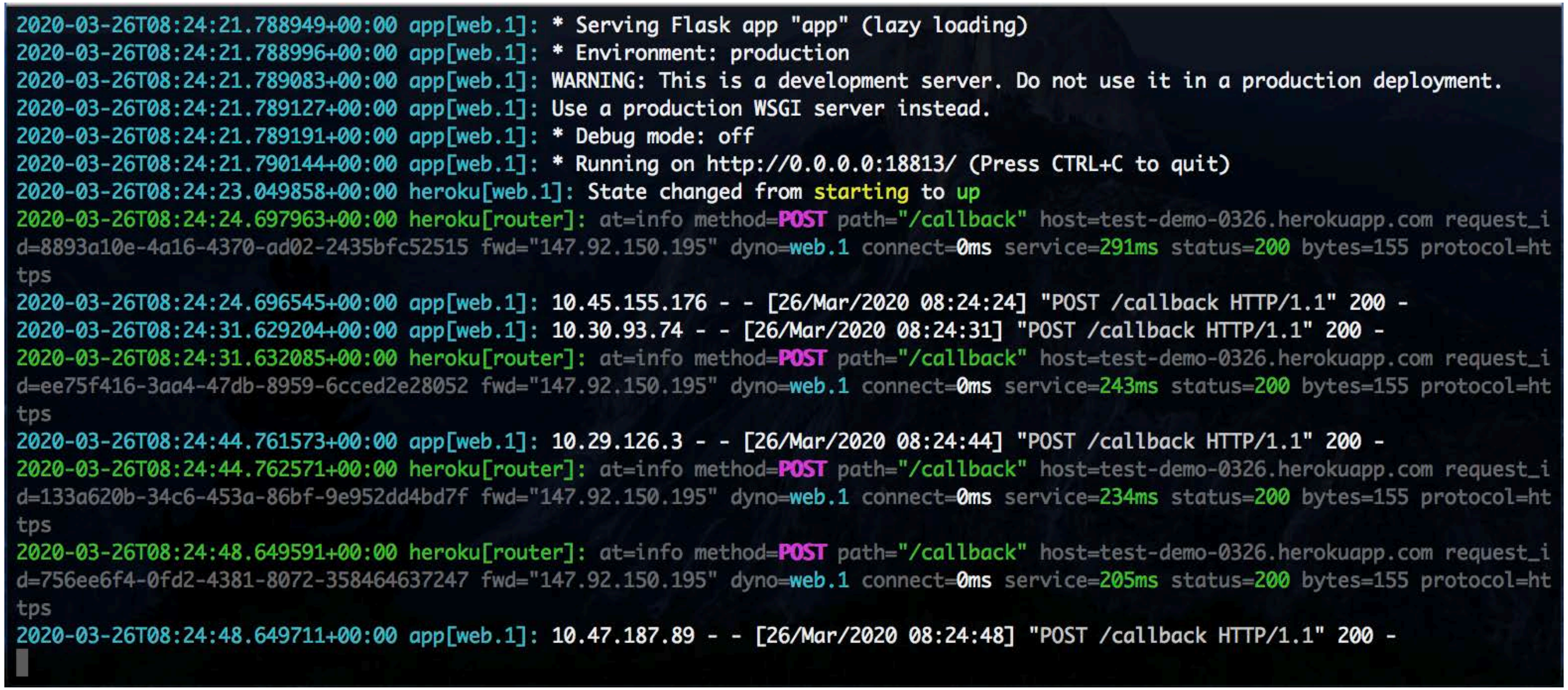


```
2020-11-17T02:05:00.031066+00:00 app[web.1]: This device has successfully registered.
2020-11-17T02:05:00.031098+00:00 app[web.1]: Device name = 76.Wash
2020-11-17T02:05:00.031104+00:00 app[web.1]: Create control threading
2020-11-17T02:05:01.033597+00:00 app[web.1]:  * Serving Flask app "app" (lazy loading)
2020-11-17T02:05:01.033625+00:00 app[web.1]:  * Environment: production
2020-11-17T02:05:01.033717+00:00 app[web.1]:    WARNING: This is a development server. Do not use it in a production deployment.
2020-11-17T02:05:01.033795+00:00 app[web.1]:    Use a production WSGI server instead.
2020-11-17T02:05:01.033857+00:00 app[web.1]:  * Debug mode: off
2020-11-17T02:05:01.034921+00:00 app[web.1]:  * Running on http://0.0.0.0:57458/ (Press CTRL+C to quit)
2020-11-17T02:05:01.404081+00:00 heroku[web.1]: State changed from starting to up
2020-11-17T02:05:51.957427+00:00 app[web.1]: 3
2020-11-17T02:05:53.398967+00:00 app[web.1]: 2
2020-11-17T02:05:54.816131+00:00 app[web.1]: 8
```

Log Information(2/2)

- Or you can type the command on cmd/terminal :

  >> heroku logs --tail --app {HEROKU_APP_NAME}



```
2020-03-26T08:24:21.788949+00:00 app[web.1]: * Serving Flask app "app" (lazy loading)
2020-03-26T08:24:21.788996+00:00 app[web.1]: * Environment: production
2020-03-26T08:24:21.789083+00:00 app[web.1]: WARNING: This is a development server. Do not use it in a production deployment.
2020-03-26T08:24:21.789127+00:00 app[web.1]: Use a production WSGI server instead.
2020-03-26T08:24:21.789191+00:00 app[web.1]: * Debug mode: off
2020-03-26T08:24:21.790144+00:00 app[web.1]: * Running on http://0.0.0.0:18813/ (Press CTRL+C to quit)
2020-03-26T08:24:23.049858+00:00 heroku[web.1]: State changed from starting to up
2020-03-26T08:24:24.697963+00:00 heroku[router]: at=info method=POST path="/callback" host=test-demo-0326.herokuapp.com request_i
d=8893a10e-4a16-4370-ad02-2435bfc52515 fwd="147.92.150.195" dyno=web.1 connect=0ms service=291ms status=200 bytes=155 protocol=ht
tps
2020-03-26T08:24:24.696545+00:00 app[web.1]: 10.45.155.176 - - [26/Mar/2020 08:24:24] "POST /callback HTTP/1.1" 200 -
2020-03-26T08:24:31.629204+00:00 app[web.1]: 10.30.93.74 - - [26/Mar/2020 08:24:31] "POST /callback HTTP/1.1" 200 -
2020-03-26T08:24:31.632085+00:00 heroku[router]: at=info method=POST path="/callback" host=test-demo-0326.herokuapp.com request_i
d=ee75f416-3aa4-47db-8959-6cced2e28052 fwd="147.92.150.195" dyno=web.1 connect=0ms service=243ms status=200 bytes=155 protocol=ht
tps
2020-03-26T08:24:44.761573+00:00 app[web.1]: 10.29.126.3 - - [26/Mar/2020 08:24:44] "POST /callback HTTP/1.1" 200 -
2020-03-26T08:24:44.762571+00:00 heroku[router]: at=info method=POST path="/callback" host=test-demo-0326.herokuapp.com request_i
d=133a620b-34c6-453a-86bf-9e952dd4bd7f fwd="147.92.150.195" dyno=web.1 connect=0ms service=234ms status=200 bytes=155 protocol=ht
tps
2020-03-26T08:24:48.649591+00:00 heroku[router]: at=info method=POST path="/callback" host=test-demo-0326.herokuapp.com request_i
d=756ee6f4-0fd2-4381-8072-358464637247 fwd="147.92.150.195" dyno=web.1 connect=0ms service=205ms status=200 bytes=155 protocol=ht
tps
2020-03-26T08:24:48.649711+00:00 app[web.1]: 10.47.187.89 - - [26/Mar/2020 08:24:48] "POST /callback HTTP/1.1" 200 -
```