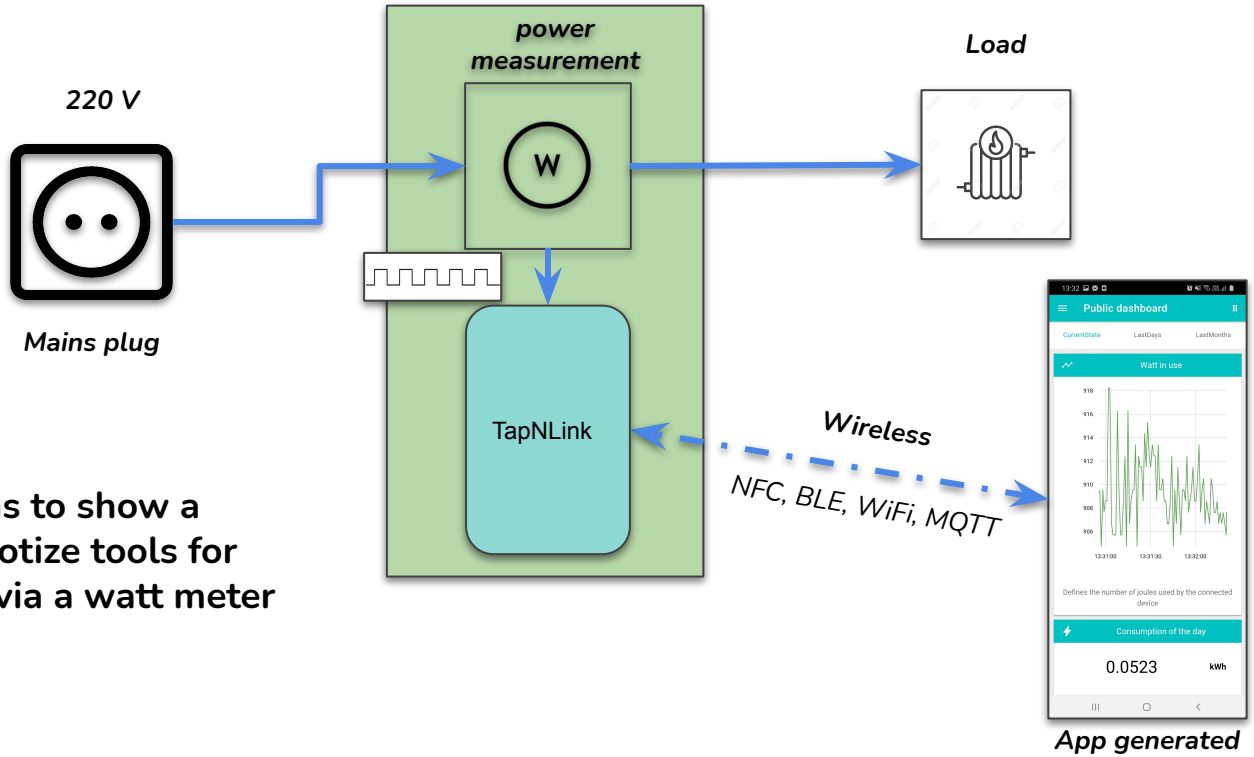
Creating the app generated WattTap

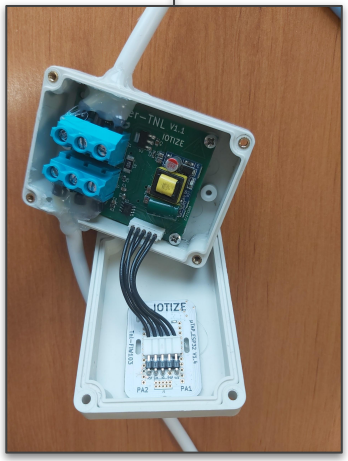
The purpose of this documentation is to show how to create an simple Watt meter based on TapNLink

WattMeter Project Diagram



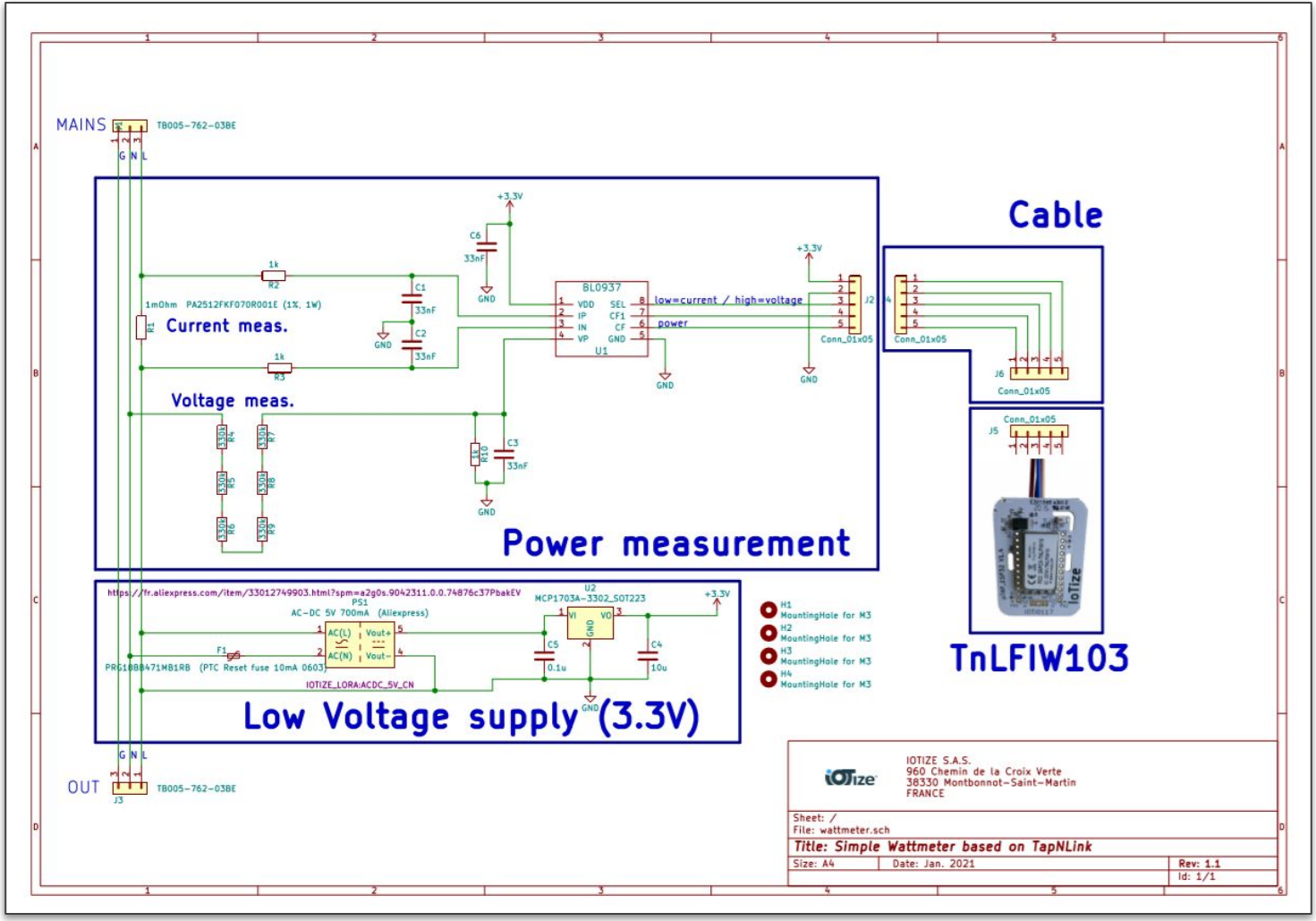
- This demo aims to show a simple use of iotize tools for reading pulse via a watt meter

We connect the wattmeter system to a 1000**W** load.



Here we can see that the tap is connected via five pates to the wattmeter circuit.

TapNLink wattmeter schematic

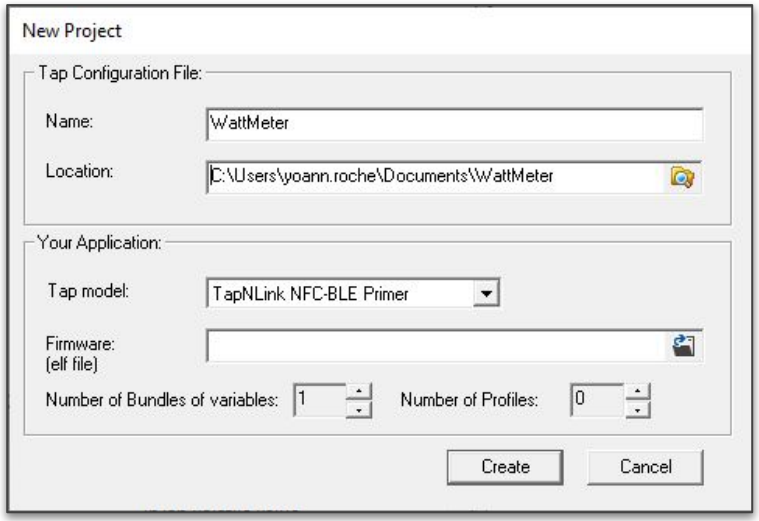


Project management from IoTize Studio

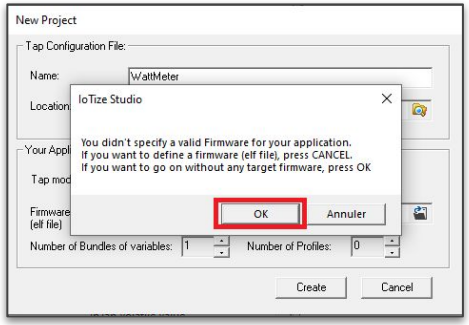
Find the IoTize Studio file on GitHub here : **WattMeterDemo**

1. Implementation of the project

- **First we will create a new project on IoTize Studio : File/New (Ctrl + N).**

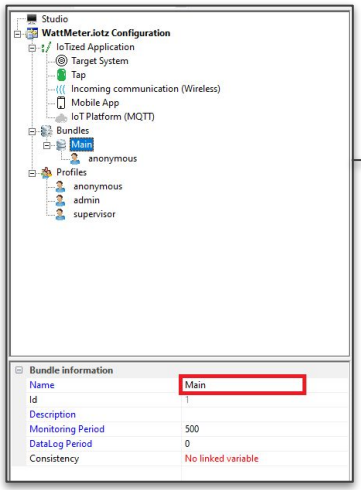


a. Define the name of the project.
b. Choose the location of the project on your computer.
c. Select your tap model.
d. Click on **Create** button.
e. Press ok when the firmware warning appears ->

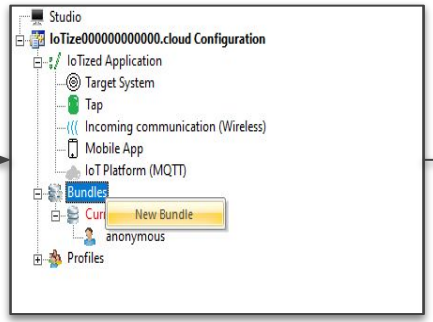


2. Implementation of Bundle

- Here we will create several Bundles to contain the variables

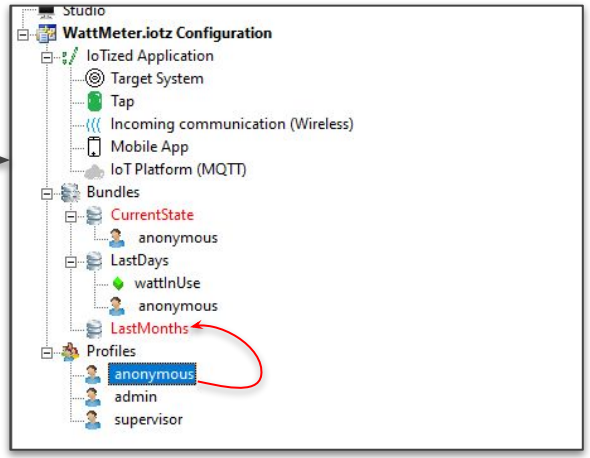


Rename the already existing bundle.



Create a bundle with a right click on the part of the **Bundle** branch.

Create a bundle for each variable group you want.

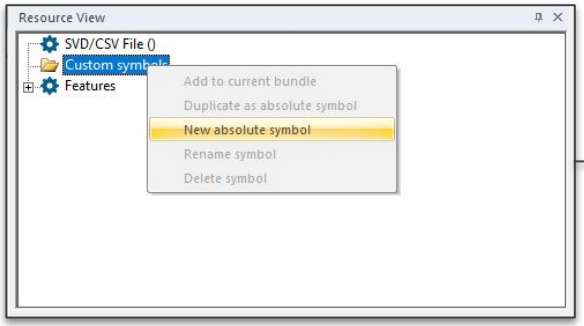


For each bundle created you must select them a profile. Drag'n drop a profile on a bundle.

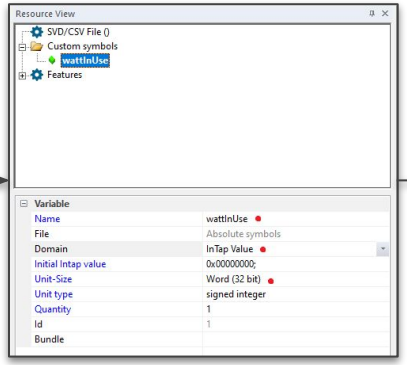
! On this demo we use only VarInTap value (volatile or not) but you can access to GPIO value/config or Target Variable.

3. Implementation of VarInTap

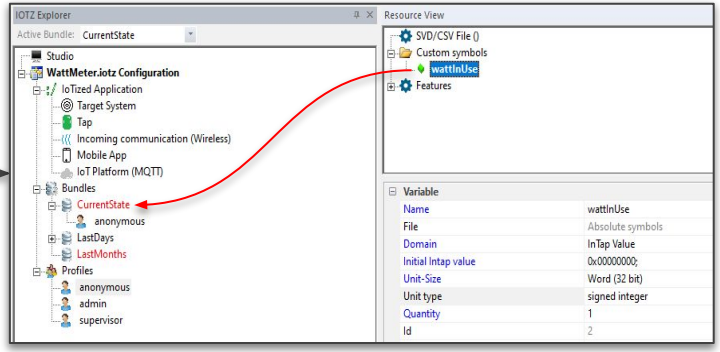
- We will set up a variable for each information that must be retrieved in the generated application or in the java code



In the resource view frame, right-click on **Custom symbol** then **New absolute symbol**.



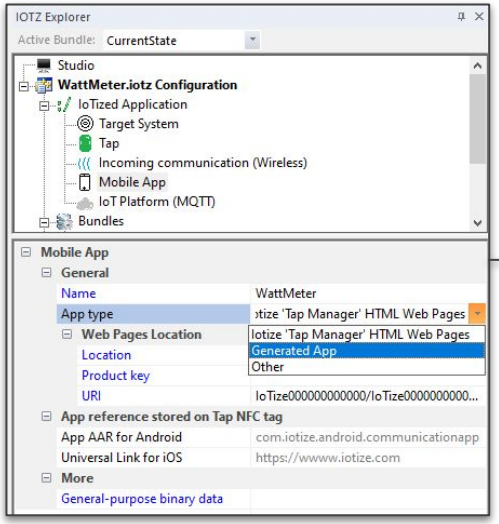
Once the item is created, it is necessary to modify this information as desired.



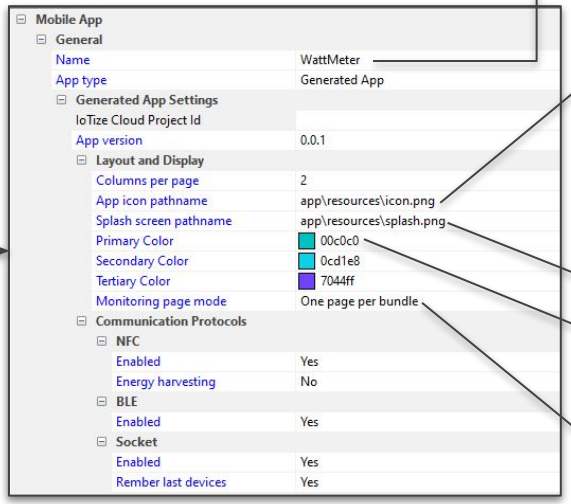
You can then assign the variable to a **Bundle** using drag n drop on the bundle.

Warning! The id may be different in the assigned bundle than resource view.

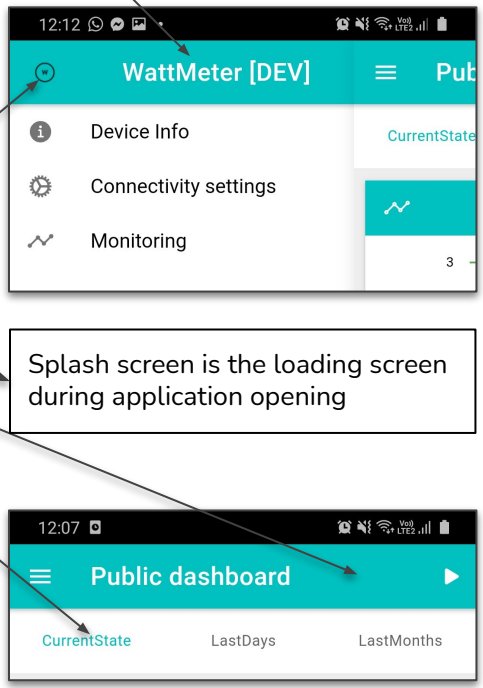
4. Implementation of the generated app



To generate an application, go to the **Mobile App** directory. In **General/App** Type select **Generated App**.

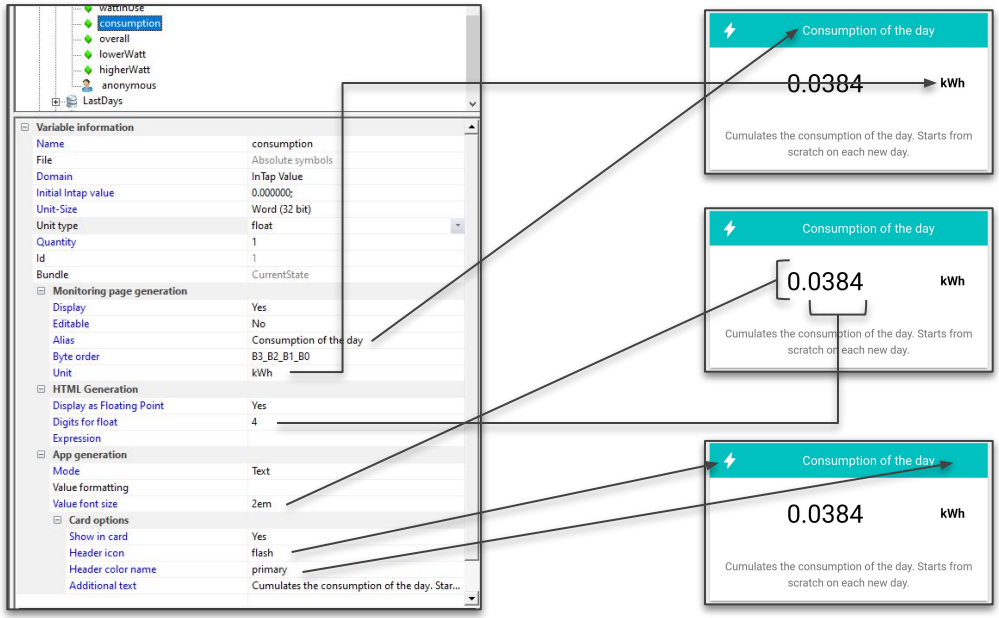


You can change the general appearance of the application via the **Generated App Settings** tab.



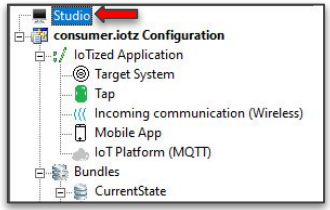
5. Improve the visual of the application

- For each created variable it is possible to define its visual aspect on the generated application



6. Tap Connection

- To connect to the tap, go to the **Studio** branch and select the **Protocol**.



Protocol

WIFI / SOCKET

Connection to Tap	
Tap Id (serial number)	IoTize00610000101F
Encryption	Mid
Protocol	Socket
Socket host name/IP	192.168.20.999
Service name/Port	2000

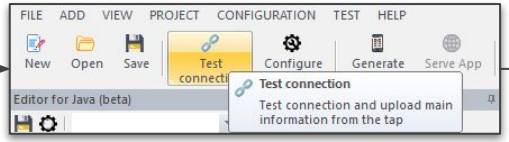
The tap must be connected to the same wifi as the computer. Once connected to the wifi the tap, retrieve the ip of the tap connected to the wifi

BLE

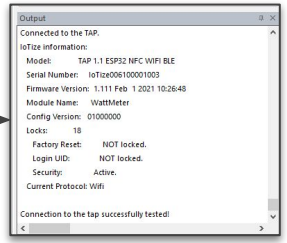
Connection to Tap	
Tap Id (serial number)	IoTize00610000101F
Encryption	Mid
Protocol	BLE
BLE device name/address	"consumer_01003 (70:b3:d5:9c:f0:04)"

Click on **BLE device name/address** to launch the available tap search.
Warning! Only one connection is possible at the same time.

To test the connection, click on the **Test connection** button



If the connection is good the **Output** tab should return this



Implementation of the Java code

Find the Java code on GitHub here : **WattMeterDemo**

7. Prepare a Java class for the JVM

- Below we find the body of a Java class with the strict minimum to be build

```java
import com.iotize.jvm.*;
import com.iotize.jvm.hal.*;



public class WattMeter {


    WattMeter() {

    }

    public void check(final int id) {

    }

    public void onException(int errorcode, int par1, int par2) {

    }
}
```

The class constructor. It is run only once at tap start. It can be used to instantiate variables.

The check method is called for each TapNLinkVar at the frequency defined in the variable constructor. The passed id is that of the TapNLinkVar which makes its check call.

The onException method is called if there is a serious error during the JVM process. If this method is called the call of the check methods is interrupted.

Warning ! If no TapNLinkVar is instantiated in the Java class, then the check method will never run.

8. Setting up the MQTT in the Java code

TapNLinkSys

We prepare an array of bytes to recover the topic.

```
6   public class WattMeter {
7
8       private TapNLinkSys system;
9       private String topic;
10
11      WattMeter() {
12          byte[] topicPrefixPayload = new byte[256];
13          system.engineCommand(TapNLinkSys.GET, 1024, 0, 58, topicPrefixPayload );
14          this.topic = new String(topicPrefixPayload) + "/result";
15          system.sendMQTTMessage(topic, "initClass()", 0);
16      }
17
18      public void check(final int id) {
19
20      }
21
22      public void onException(int errorcode, int par1, int par2) {
23          system.sendMQTTMessage(topic, String.format("Error code : " + errorcode, errorcode, par1, par2), 0);
24      }
25  }
```

The system's engineCommand is used to retrieve the mqtt topic defined in the tap configuration. Pass the byte array that will be transformed and contain the topic.

One transforms the array of byte in String to recover the topic.

We use the sendMQTTMessage method of the system interface to signal that the class is well instantiated.

Here we send a mqtt message if the onException method is called, the message contains information about the error.

9. Definition of TapNLinkVar in Java code

TapNLinkVar

This value defines the call frequency of the check method with the id of this variable. The value is in milliseconds.

```
public  TapNLinkVarInt              higherWatt          = new TapNLinkVarInt((short) 0x07, VariableType.INT16, "higherWatt", 1000);
```

We choose the class according to the variable created in the **Bundle**

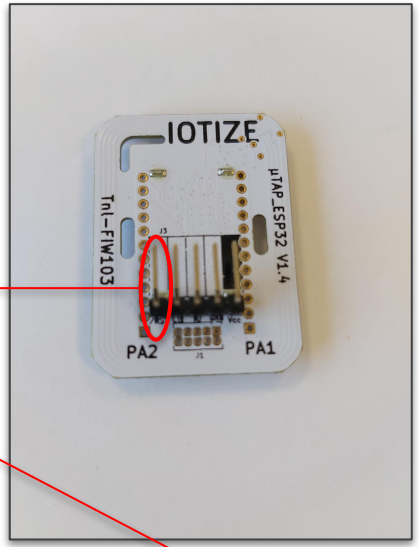
Variable information	
Name	higherWatt
File	Absolute symbols
Domain	InTap Value
Initial Intap value	0x0000;
Unit-Size	Half-Word (16 bit)
Unit type	signed integer
Quantity	1
Id	7
Bundle	CurrentState

10. Reading pulses on Pin in Java code

Pin

The first element of the constructor contains the paw of the tap which recovers the pulses of the device.

```
public  Pin                    p1                    = new Pin(Pin.TGT_NRST, 4, Pin.HIZ);
```



p1.getValue();

The second element of the constructor is pin mode.

The last element of the constructor is pin pull.

1sec

0 → **158** → **314**

1sec

+158 pulses

+156 pulses

Watt/s = calibration Watt * pulses = **Watt/s = 0.9625*158** = **Active power consumption = 152.075 Watt/s**

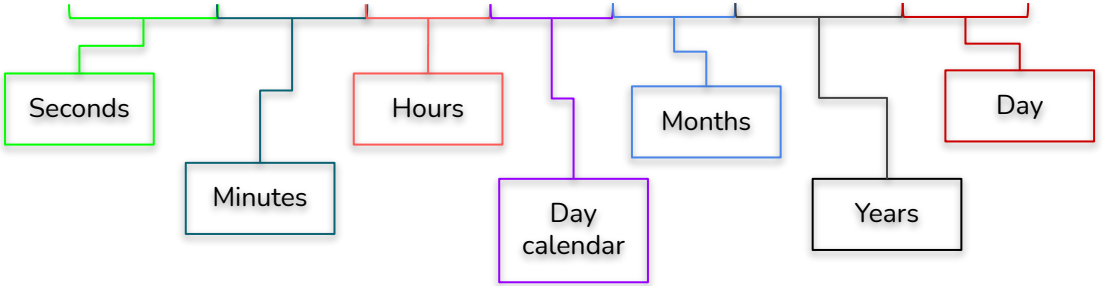
11. Absolute time recovery

```
public void currentTime() {
    try {
        system.engineCommand(TapNLinkSys.GET, RESSOURCE_ID, 0x00, 79, date);
        String month = "" + date[16] + date[17] + date[18] + date[19];
        String day = "" + date[24] + date[25] + date[26] + date[27];
        currentDay = (int) Integer.valueOf(day);
        currentMonth = (int) Integer.valueOf(month);
    } catch (TapNLinkException e) {
        system.sendMQTTMessage(topic,"Error on method currentTime(), error code : " + e.getCode(), 0);
    }
}
```

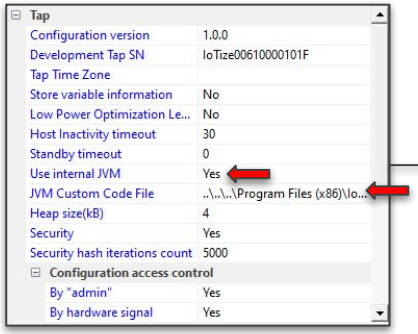
```
private byte[]          date                        = new byte[36];
private int             currentDay, currentMonth;
```

The system get command on 79 will recover the absolute time and transform the date object. In our example of the WattMeter only the day and the month are necessary.

date[] = 0.0.0.39.0.0.0.33.0.0.0.8.0.0.0.3.0.0.0.1.0.0.0.121.0.0.0.3.0.0.0.33.0.0.0.0

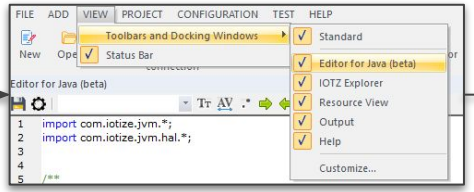


12. Set up the Java code in IoTizeStudio

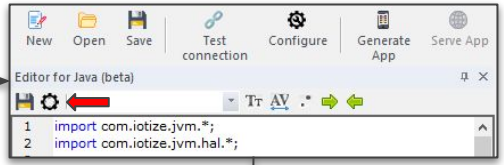


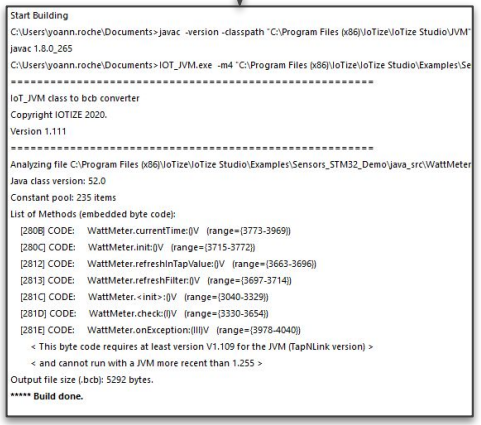
Pass the **Use internal JVM** parameter to Yes.
In **JVM Custom Code File** set the path of your java file.

If the page that displays the java code is not present, you must go to the directory **View/Toolbars** and **Docking Windows/Editor for java**



Run the build via the engravings button





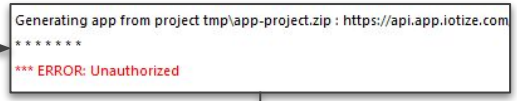
Generation of the application

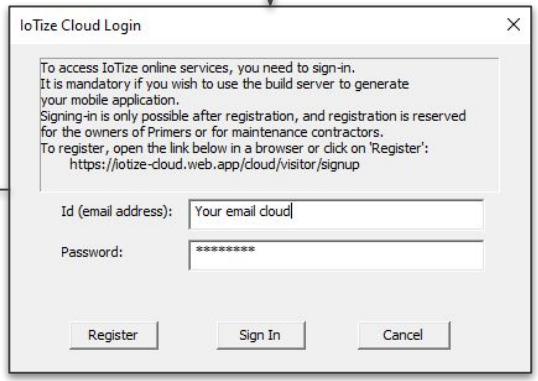
13. Generate App with IoTizeStudio

Click on the **Generate App** button to start the generation of your application

If you are not logged on IoTize cloud

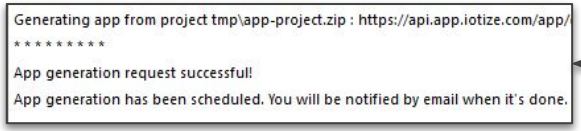
Information retrieved from the **Output** frame





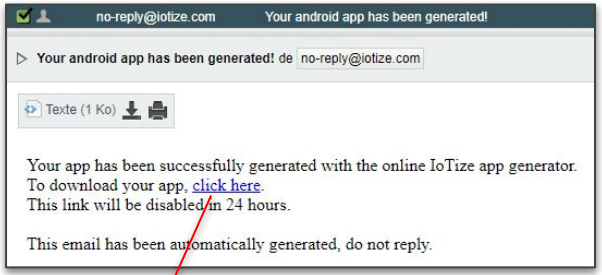
After logging

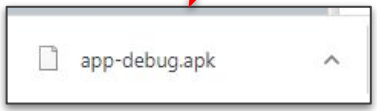
Information retrieved from the **Output** frame



14. Retrieve the generated app on your mobile

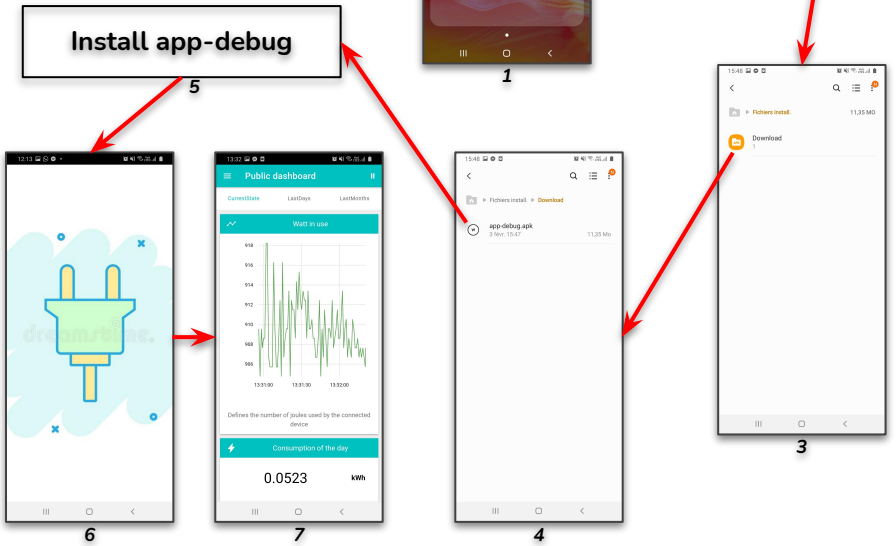
After running the app generation you must have received an email containing a link to click to download the generated app.





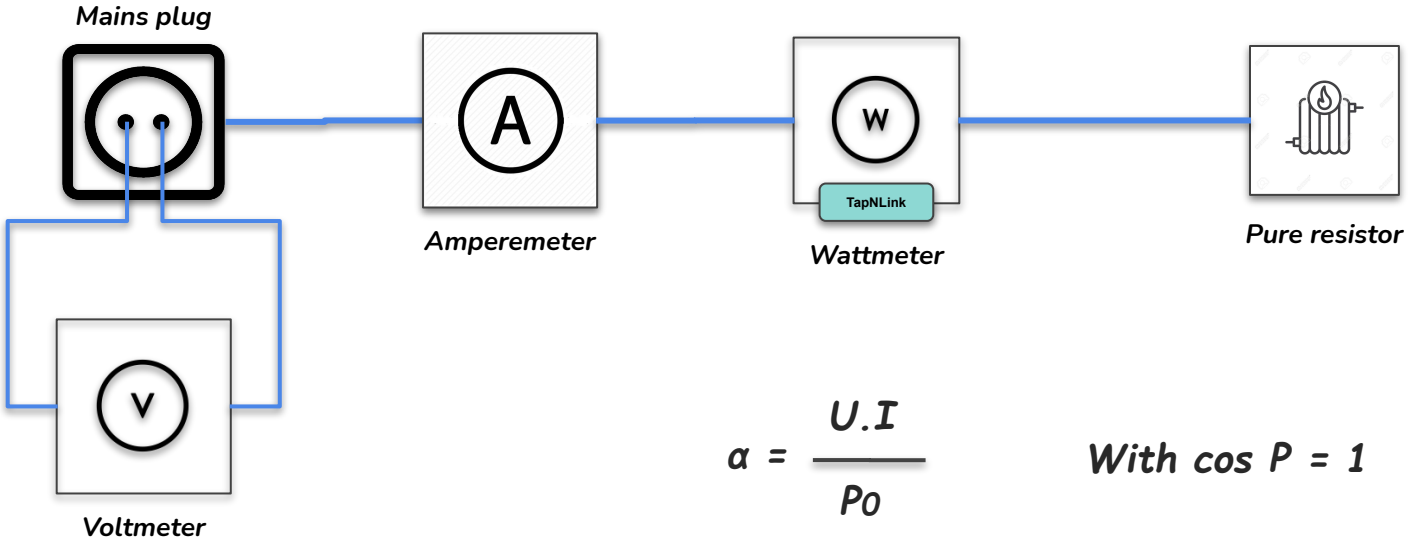
Drop the app file on your mobile **Download** folder.

Then



Optimize your results

15. Define the calibration



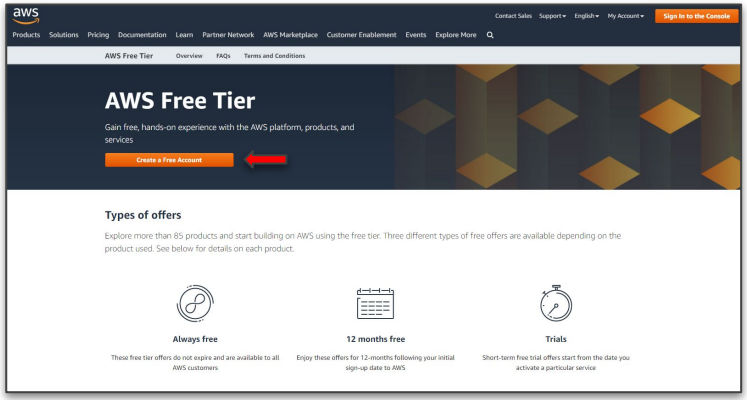
$$\alpha = \frac{U.I}{P_0}$$

With cos P = 1

Implementation of AWS

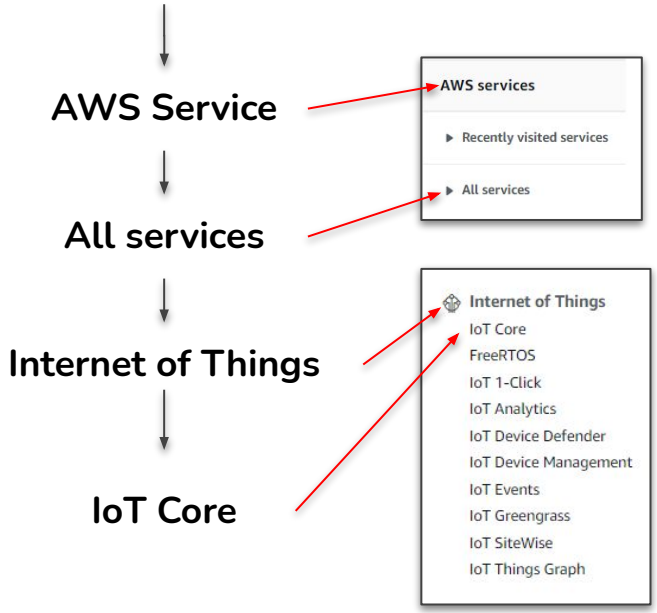
Find more information on doc IoTize here : **AWS IoT**

16. Create your AWS account

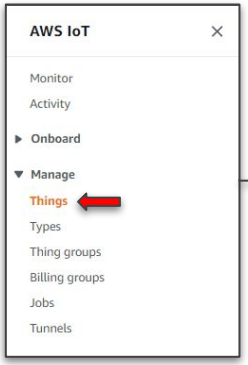


Create AWS account

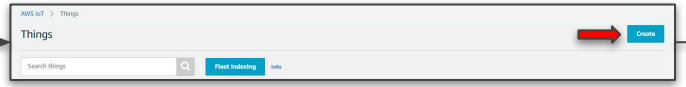
Once your account is created, go to the **AWS Management Console** page.
Then connect to **AWS** and go to main menu



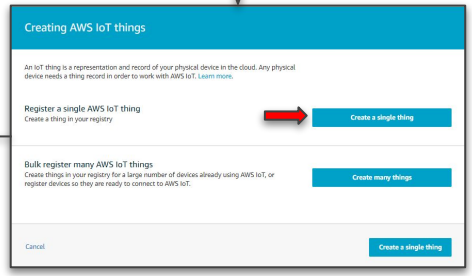
17. Create things on AWS



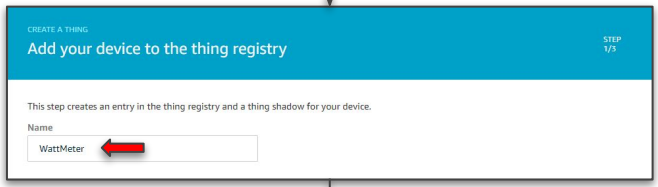
Click on **Things** in the left bar on tab **Manage**



Click on **Create** to access on **Creating AWS IoT things**



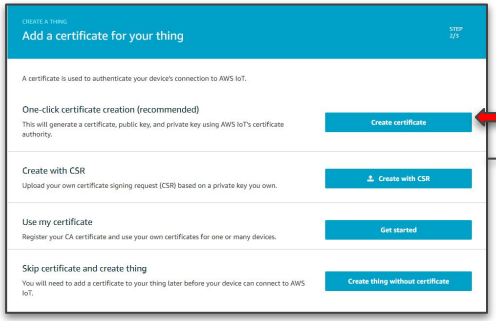
Click on **Create a single thing**



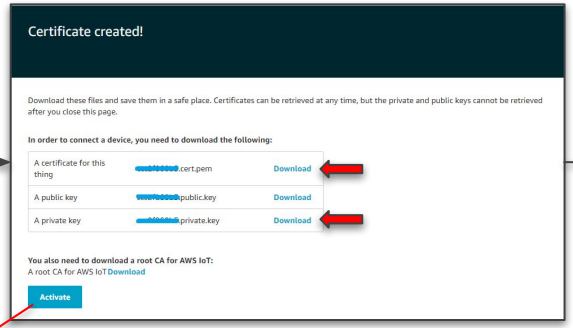
Set the name of the thing to **WattMeter**

In the WattMeter example the other information in the form is not necessary.
Press **Next** at the bottom of the form. sd

18. Create certificate for the thing



Select **Create certificate** to generate AWS IoT's certificate authority.

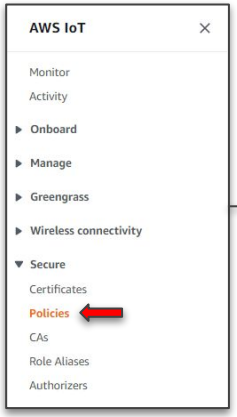


Download **A certificate for this thing** and **Private key**. Place them in a file reserved for your WattMeter project.

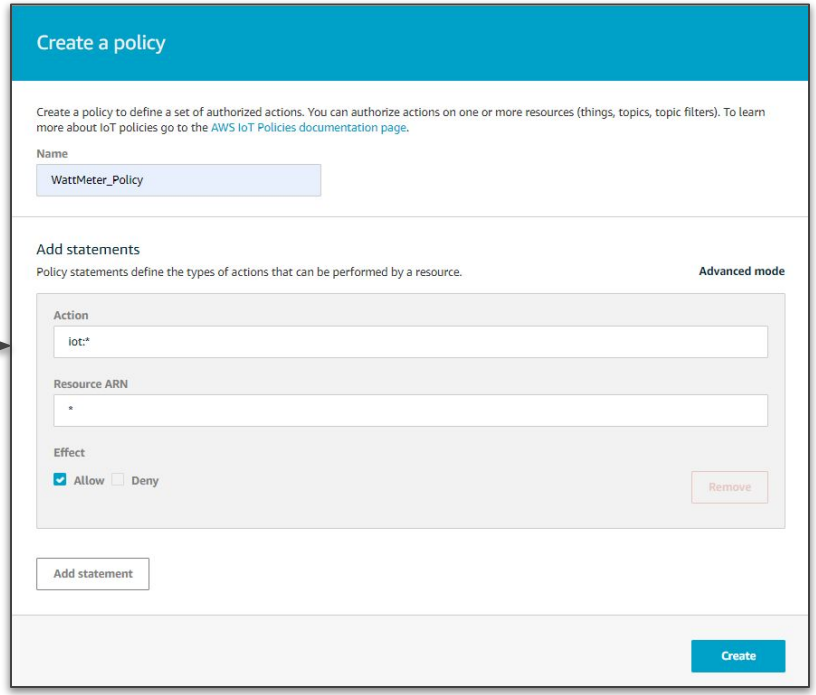
It is important to click **Activate** for the root CA for AWS IoT. Once activated click on **download** above to be taken to the download page of Amazons root CA.

To complete the creation, click **Done** to return to the list of things page with your new thing

19. Create a policy for the thing



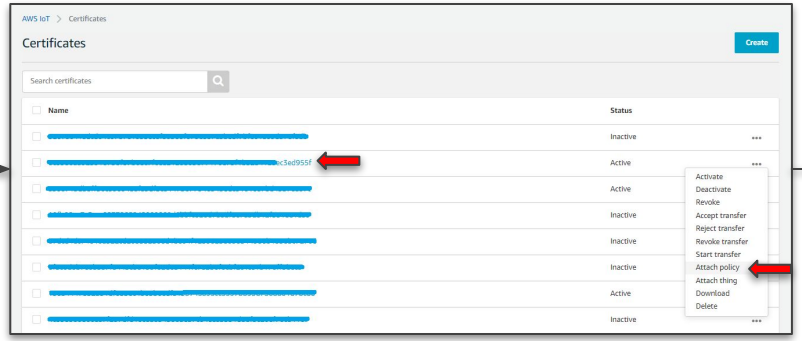
Click on **Policies** in the left bar on tab **Secure**



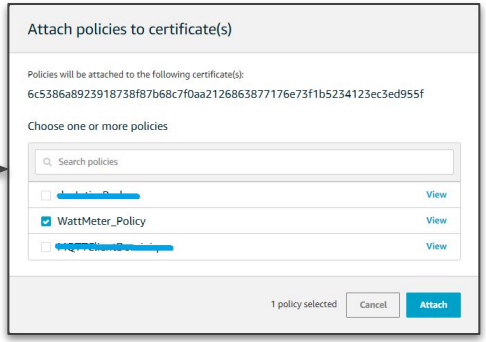
Complete the policy creation form as shown in the picture. Once completed click on **Create**.

20. Define a policy for certificate of the thing

Click on **Certificates** in the left bar on tab **Secure**

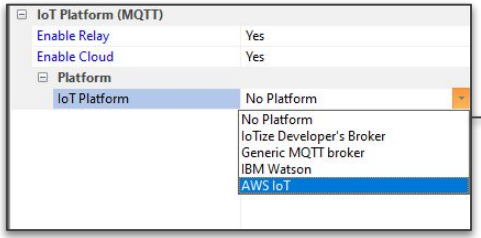


Select the certificate that was created with your thing, then select **attach policy**.

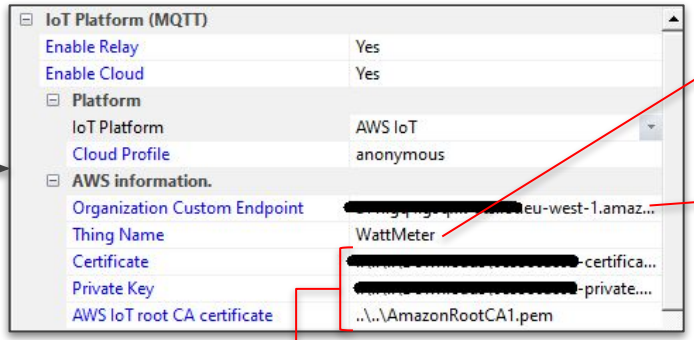


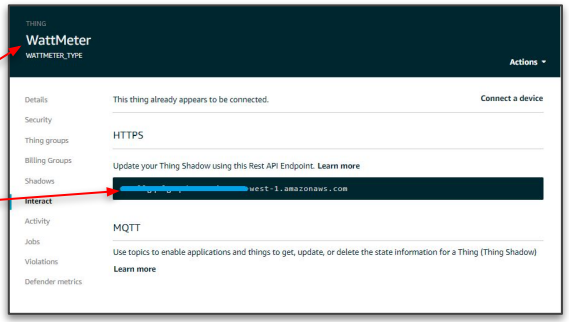
Select the previously created policy and click on **Attach**.

21. Define IoT Platform (MQTT) on IoTize Studio



Select the **IoT Platform** with **AWS IoT**





Go back to your created thing and go to the **Interact** tab.

Recover previously uploaded files in your WattMeter project folder

22. Implementation of the MQTT message to AWS on Java code

```
system.sendMQTTMessage("$aws/things/WattMeter/shadow/name/inUse/update",
"{\"state\": {\"reported\": {\"ID\": \"WattMeter\", \"CurrentDayConsumption\": " + consumptionDay + " ,\"wattInUse\": "+ wattInUse.getValue() +"}}}", 0);
```

This command will send a mqtt message on the aws topic that will define a new shadow if it is not existing. The message contains two values that will be implemented in the **inUse** shadow