Local Deployment of the OpenIoT Platform:

Notice: MAC OS X was used as the working environment for this deployment.

Notice: Please follow the sequence in this manual because any change in the sequence may cause errors.

Notice: In this manual, JAVA is in the location:/Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home, MAVEN is in the location:/apache-maven-3.2.5, JBOSS is in the location:/jboss-as-7.1.1.Final and THE CODE in the location: /openiot-0.6.1
Please use appropriate path based on your local installation location

1) Install JDK 1.7 (only for the first time to deploy the platform):

- Download JDK 1.7 from the following link (please use this version): http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

- Open the terminal and open the .bash_profile file to store JAVA configurations so that you can use them every time you use the terminal without the need of writing them down every time:

```
vi ~/.bash_profile
```

- Write in the following configurations then save:

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/
jdk1.7.0_71.jdk/Contents/Home      //(the location of your JDK)

export  JAVA=$JAVA_HOME/bin

export PATH=$JAVA:$PATH
```

- To make sure that every thing is correct, run:

```
java –version  or
env
```

Notice: you can follow any other steps according to your own experience.

Notice: vim is a text editor. For more information about using it, follow this link: http://www.linux.com/learn/tutorials/228600-vim-101-a-beginners-guide-to-vim

2) Install Maven (only for the first time to deploy the platform):

- Download Maven from the following link:
  http://maven.apache.org/download.cgi

- Open the terminal and open the .bash_profile file to store maven configurations so that you can use them every time you use the terminal without the need of writing them down every time:

  ```
  vi ~/.bash_profile
  ```

- Write in the following configurations then save:

  ```
  export M2_HOME=/apache-maven-3.2.5  //(location of your MAVEN folder)
  export M2=$M2_HOME/bin
  export PATH=$M2:$PATH
  ```

- To make sure that every thing is correct run:

  ```
  mvn –version
  ```

Notice: you can follow any other steps according to your own experience.

3) Installing Local Virtuoso (only for the first time to deploy the platform):

- Please follow the link below for detailed instructions for installing Virtuoso.
- Note: for Linux installations, virtuoso requires a 64 bit machine.
  https://github.com/OpenIotOrg/openiot/wiki/InstallingVirtuosoOpensource7Ubuntu

Notice: you can follow any other steps according to your own experience

4) Configuring JBoss:

- Please download JBoss (please use this version: JBoss AS 7.1.1.Final):
  http://jbossas.jboss.org/downloads

- To run JBoss in standalone mode:

  For Linux: JBOSS_HOME/bin/standalone.sh
  For Windows: JBOSS_HOME\bin\standalone.bat

- If JBoss is running on a server and you want to open it from your localhost or any other PC:

  For Linux: JBOSS_HOME/bin/standalone.sh –b 0.0.0.0
  For Windows: JBOSS_HOME\bin\standalone.bat –b 0.0.0.0

- To run JBoss in the background (be able to write commands afterwards):

  For Linux: JBOSS_HOME/bin/standalone.sh &
  For Windows: JBOSS_HOME\bin\standalone.bat &

- To make sure that JBoss is running correctly, go to the following URL:
  http://localhost:8080 or

  http://SERVER_HOSTNAME:8080 (if you are using JBoss from a server not the localhost)

- Enable SSL in JBoss in order to run the security modules illustrated later in this document correctly (this is done only once when you are configuring JBoss for the first time):

  1. Run: mkdir -p /jboss-as-7.1.1.Final/standalone/configuration/ssl

  2. Run: cd /jboss-as-7.1.1.Final/standalone/configuration/ssl

  3. Run: keytool -genkey -alias jbosskey -keypass <password> -keyalg RSA -keystore server.keystore , Use "localhost" as Common Name [as answer to "What is your first and last name?"]. If you are deploying on a server, use the DNS name of the server instead of "localhost". If you are deploying on a server with public IP (no DNS), use ip address as Common Name [as answer to "What is your first and last name?"]. Also add the following command to the keytool -genkey -ext san=ip: 10.0.0.1

4. Run: keytool -export -alias jbosskey -keypass <password> -file server.crt -keystore server.keystore

5. Run: keytool -import -alias jbosscert -keypass <password> -file server.crt -keystore server.keystore Ignore the warninig!

6. In /jboss-as-7.1.1.Final/standalone/configuration/standalone.xml add the following connector in <subsystem xmlns="urn:jboss:domain:web:1.1" ..

```
<connector name="https" protocol="HTTP/1.1" scheme="https"
        socket-binding="https" secure="true">
   <ssl name="https" key-alias="jbosskey" password=<password>
        certificate-key-file="YOUR_SSL_DIR_PATH/server.keystore" />
</connector>
```

7. Restart JBoss and go to https://localhost:8443 to see if SSL is enabled and works correctly.
8. Next, you will have to import this certificate into the java trust-store with the command keytool -import -keystore /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/security/cacerts -file server.crt -alias incommon. In Linux you will have to do this as root. (the location of your cacerts file)

9. For troubleshooting please follow this link:

   https://github.com/OpenIotOrg/openiot/wiki/Security-Use-Server#jboss-ssl-troubleshooting (Only troubleshooting)

- If you want to stop JBoss, run the following command:

  cd /jboss-as-7.1.1.Final/bin

  For Linux: ./jboss-cli.sh --connect command=:shutdown

Notice: you can follow any other steps according to your own experience

5)Download the openIoT code:

https://github.com/OpenIotOrg/openiot

6) Running X-GSN:

- ✶ Navigate to /openiot-0.6.1/modules/x-gsn/and run the command: mvn clean package
- ✶ A folder named target will be created under this directory. This folder contains a file called: xgsn-1.1.4.jar
- ✶ Run the script:

  For Linux: ./gsn-start.sh

- ✶ If you want to stop X-GSN for any reason, run the following script:

  For Linux: ./gsn-stop.sh

Notice: the first two steps are done only for the first time to deploy the platform.

7) Deployment of the Modules (only for the first time to deploy the platform):

Notice: OpenIoT modules depend on each other, so it is recommended that you follow the below order while deploying the full platform.

- **utils.commons:**
  - ✶ Navigate to /openiot-0.6.1/utils/utils.commons and run the command: mvn install
  - ✶ A folder named target will be created under this directory. This folder contains a file called: utils.commons.jar
  - ✶ Navigate to /openiot-0.6.1/utils/utils.commons/src/main/resources
  - ✶ Copy the file security-config.ini to the location: /jboss-as-7.1.1.Final/standalone/configuration

    cp security-config.ini /jboss-as-7.1.1.Final/standalone/configuration

  - ✶ Navigate to /openiot-0.6.1/utils/utils.commons/src/main/resources/properties

- Copy the file openiot.properties to the location: /jboss-as-7.1.1.Final/standalone/configuration

  cp openiot.properties /jboss-as-7.1.1.Final/standalone/configuration

- **Security Client:**

  - Navigate to /openiot-0.6.1/modules/security/security-client and run the command: mvn install
  - A folder named target will be created under this directory. This folder contains a file called: security.client-0.0.1-SNAPSHOT.jar

- **Lsm-light Client:**

  - Navigate to /openiot-0.6.1/modules/lsm-light/lsm-light.client and run the command: mvn install
  - A folder named target will be created under this directory. This folder contains a file called: lsm-light.client-0.0.1.jar

- **Ui.requestCommons:**

  - Navigate to /openiot-0.6.1/ui/ui.requestCommons and type the command: mvn install
  - A folder named target will be created under this directory. This folder contains a file called: ui.requestCommons.jar

- **Lsm-light Server:**

  - Navigate to /openiot-0.6.1/modules/lsm-light/lsm-light.server and run the command: mvn clean package jboss:deploy
  - A folder named target will be created under this directory. This folder contains a file called: lsm-light.server.war

- Copy this file to the location: /jboss-as 7.1.1.Final/standalone/deployments

  cp lsm-light.server.war /jboss-as 7.1.1.Final/standalone/deployments

- To make sure that everything is working, go to link: http://localhost:8080/lsm-light.server/ (or replace localhost with the name of your server if you are running JBoss from a remote server)

Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

•Security Management and Server:

- Navigate to /openiot-0.6.1/modules/security/security-management and run the command: mvn clean package jboss:deploy

- A folder named target will be created under this directory. This folder contains a file called: security.management.war

- Copy this file to the location: /jboss-as-7.1.1.Final/standalone/deployments :

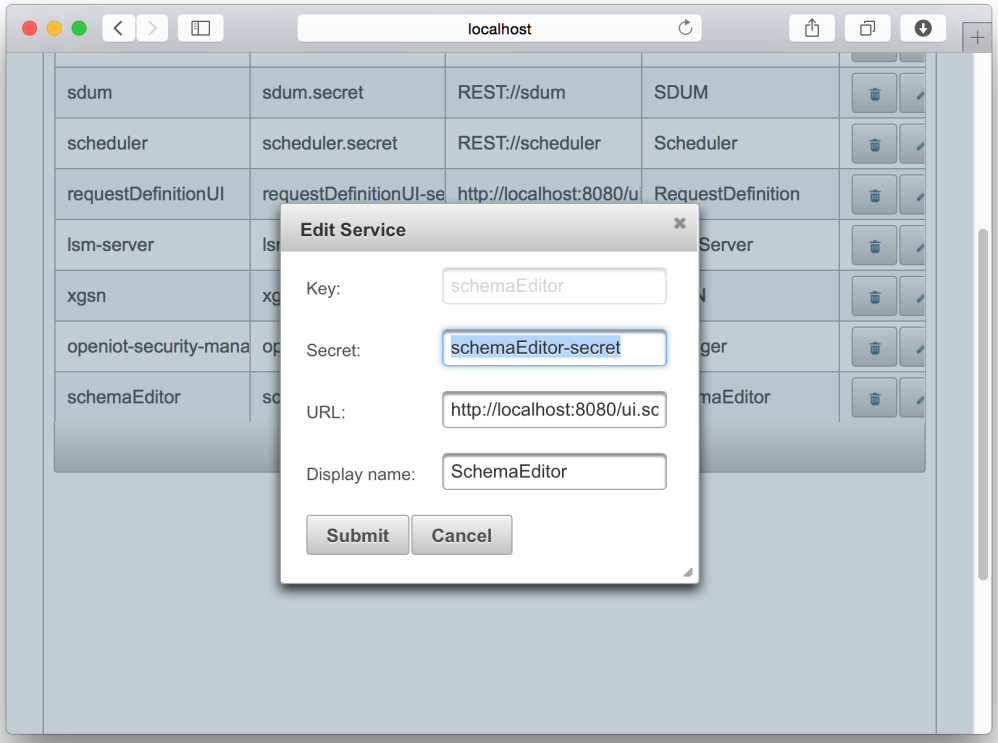
  cp security.management.war /jboss-as 7.1.1.Final/standalone/deployments

- To make sure that everything is working, go to link: http://localhost:8080/security.management (or replace localhost with the name of your server if you are running JBoss from a remote server)

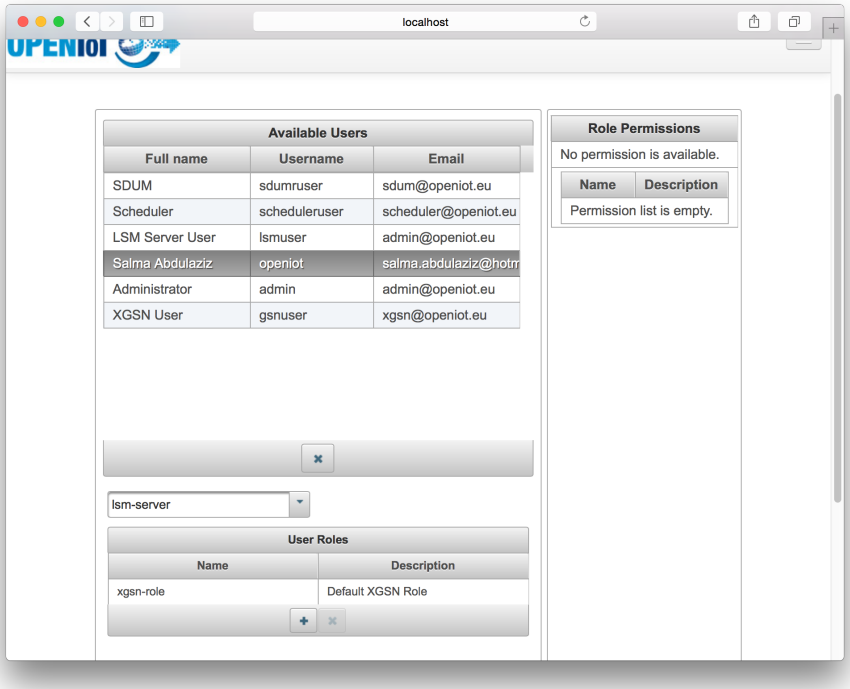
Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

- **Adding a New User and Some Modifications:**

  - Sign up for a new user: eg: openiot/openiot
  - Log-out (make sure you are logged out from the application and from CAS log-out at the end of the page) and close the browser
  - Log-in again with: admin/secret
  - Go to Manage Services and edit the shemaEditor



- Change the URL to be: http://localhost:8080/ui.schemaeditor/callback?client_name=CasOAuthWrapperClient
- Change the Secret to be: schemaEditor-secret and press Submit
- Go to Manage Users and click on the user you have created



- Click on Select Service and add roles to each of the following:
    a. LSM usage role
    b. Schema editor usage role
    c. Scheduler usage role
    d. Sdum usage role
    e. X-GSN usage role
- The following link will help you understand what is a role and how to use the security management interface: https://github.com/OpenIotOrg/openiot/wiki/Security-Use-Console
- Log-out again (from both) and close the browser.

✶ Navigate to /openiot-0.6.1/modules/security/security-server and run the command: mvn clean package jboss:deploy

✶ A folder named target will be created under this directory. This folder contains a file called: openiot-cas.war

✶ Copy this file to the location: /jboss-as-7.1.1.Final/standalone/deployments

cp openiot-cas.war /jboss-as 7.1.1.Final/standalone/deployments

✶ To make sure that everything is working, go to link: https://localhost:8443/openiot-cas (or replace localhost with the name of your server if you are running JBoss from a remote server)

✶ Log-in with the new user you have created before in security management.

Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

•Scheduler Core:

✶ Navigate to /openiot-0.6.1/modules/scheduler/scheduler.core and run the command: mvn clean package jboss:deploy

✶ A folder named target will be created under this directory. This folder contains a file called: scheduler.core.war

✶ Copy this file to the location: /jboss-as-7.1.1.Final/standalone/deployments :

cp scheduler.core.war /jboss-as 7.1.1.Final/standalone/deployments

- To make sure that everything is working, go to link: http://localhost:8080/scheduler.core/rest/services (or replace localhost with the name of your server if you are running JBoss from a remote server)

Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

•SDUM Core:

- Navigate to /openiot-0.6.1/modules/sdum/sdum.core and run the command: mvn clean package jboss:deploy

- A folder named target will be created under this directory. This folder contains a file called: sdum.core.war

- Copy this file to the location: /jboss-as-7.1.1.Final/standalone/deployments :
  cp sdum.core.war /jboss-as 7.1.1.Final/standalone/deployments

- To make sure that everything is working, go to link: http://localhost:8080/sdum.core/rest/services (or replace localhost with the name of your server if you are running JBoss from a remote server)

Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

Notice: Both SDUM and Scheduler provide clients to test the server functions. These are SWING java programs. Please execute them as any other java swing program if needed.

•ui.requestDefinition:

- Navigate to /openiot-0.6.1/ui/ui.requestDefinition and run the command: mvn clean package jboss:deploy

- A folder named target will be created under this directory. This folder contains a file called: ui.requestDefinition.war

- Copy this file to the location: /jboss-as-7.1.1.Final/standalone/deployments

cp ui.requestDefinition.war /jboss-as 7.1.1.Final/standalone/deployments

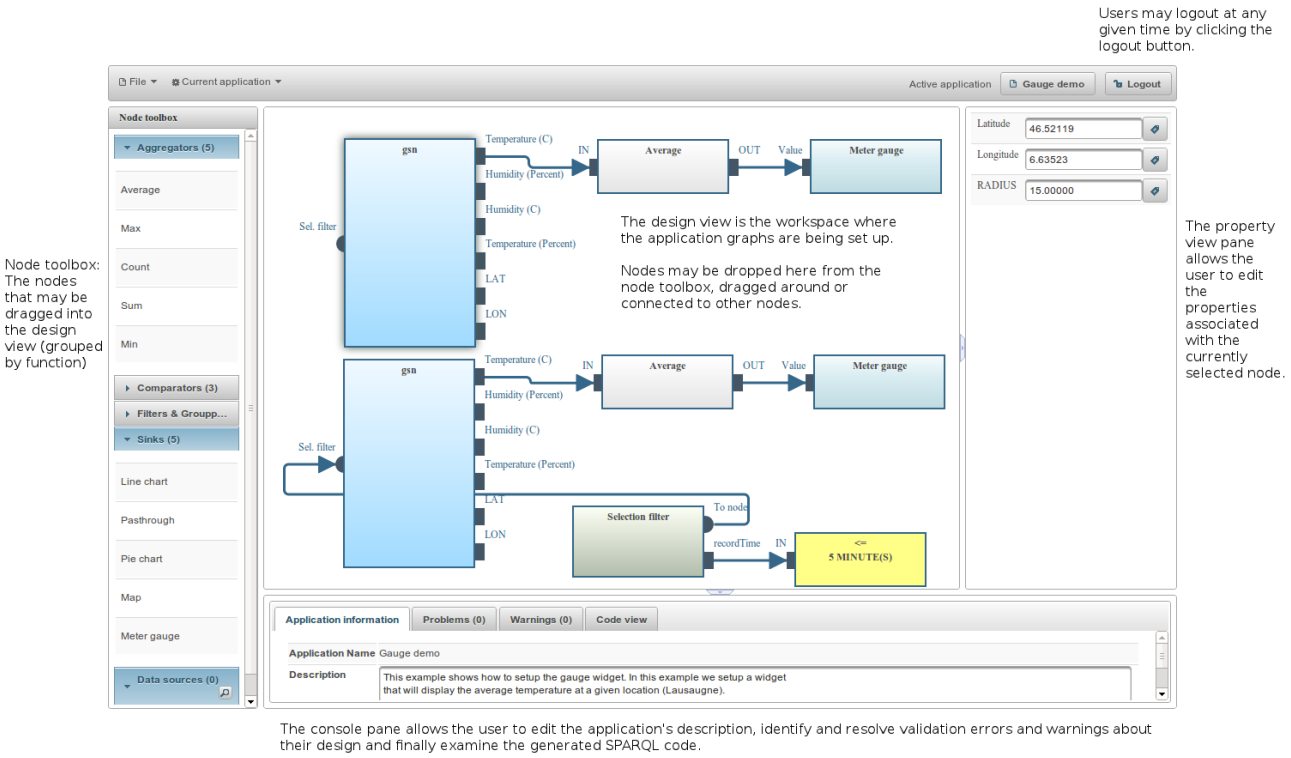
✶ To make sure that everything is working, go to link: http://localhost:8080/ui.requestDefinition/ (or replace localhost with the name of your server if you are running JBoss from a remote server)

✶ Log-in with the new user you have created before in security management

✶ To know how to use this interface after pushing data into the platform (will be shown later), please follow this link:

https://github.com/OpenIotOrg/openiot/wiki/Request-definition-use

- **UI Overview**



Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

•ui.requestPresentation:

✶ Navigate to /openiot-0.6.1/ui/ui.requestPresentation and run the command: mvn clean package jboss:deploy

✶ A folder named target will be created under this directory.
This folder contains a file called: ui.requestPresentation.war

✶ Copy this file to the location: /jboss-as-7.1.1.Final/standalone/deployments

cp ui.requestPresentation.war /jboss-as 7.1.1.Final/standalone/deployments

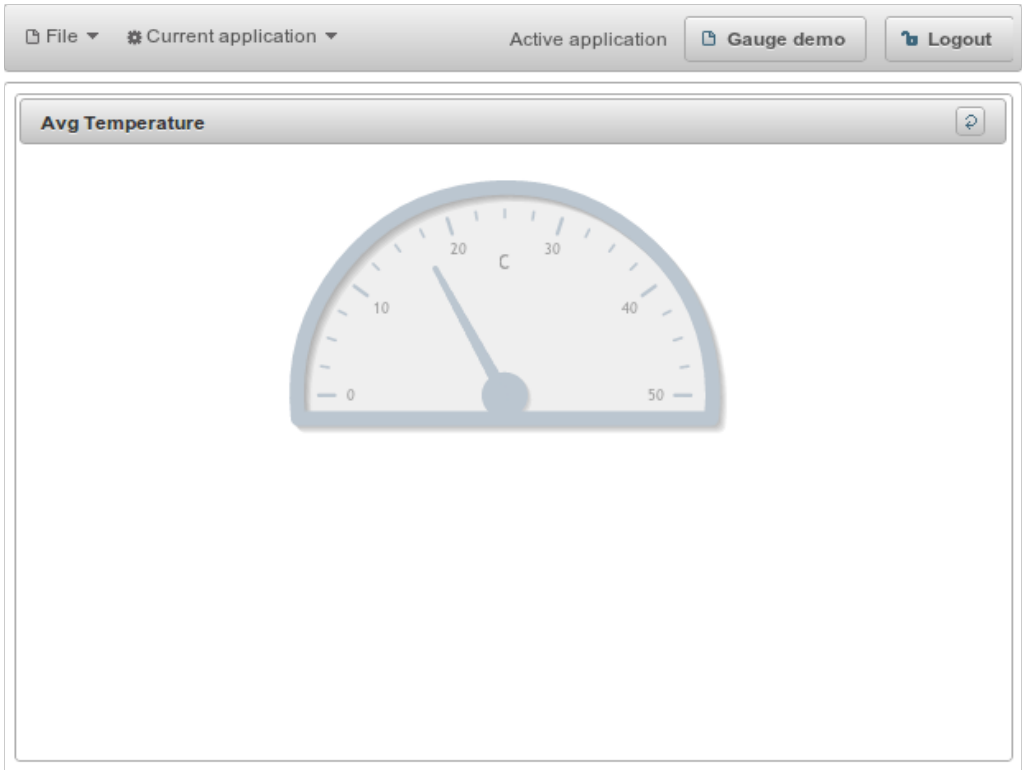
✶ To make sure that everything is working, go to link: http://localhost:8080/ui.requestPresentation/ (or replace localhost with the name of your server if you are running JBoss from a remote server)

✶ Log-in with the new user you have created before in security management

✶To know how to use this interface after pushing data into the platform (will be shown later), please follow this link:

https://github.com/OpenIotOrg/openiot/wiki/Request-presentation-use

- **UI Overview**



Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

- **ui.schemaeditor:**

  - Navigate to /openiot-0.6.1/ui/ui.schemaeditor and run the command: mvn clean package jboss:deploy

  - A folder named target will be created under this directory. This folder contains a file called: ui.schemaeditor.war

  - Copy this file to the location: /jboss-as-7.1.1.Final/standalone/deployments

    cp ui.schemaeditor.war /jboss-as 7.1.1.Final/standalone/deployments

  - To make sure that everything is working, go to link: http://localhost:8080/ui.schemaeditor (or replace localhost with the name of your server if you are running JBoss from a remote server)

  - Log-in with the new user you have created before in security management

  - Details on how to use this interface are given later in this document.

Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

- **Ide Core:**

  - Navigate to /openiot-0.6.1/ui/ide/ide.core and run the command: mvn clean package jboss:deploy

  - A folder named target will be created under this directory. This folder contains a file called: ide.core.war

  - Copy this file to the location: /jboss-as-7.1.1.Final/standalone/deployments

    cp ide.core.war /jboss-as 7.1.1.Final/standalone/deployments

  - To make sure that everything is working, go to link: http://localhost:8080/ide.core/ (or replace localhost with the name of your server if you are running JBoss from a remote server)

Notice: if jboss:deploy doesn't work, replace it with jboss-as:deploy

For better understanding of the platform, please read the documentation found here:

https://github.com/OpenIotOrg/openiot/wiki/Documentation

https://github.com/OpenIotOrg/openiot/wiki/Developer-Setup-Instructions-Eclipse---OpenIoT-v0.6.1

Using Arduino Mote with OpenIoT - Manual

1) Creating Virtual Sensor XML File

- You have to write an XML file describing your sensor
- It has to be put in this directory:

/openiot-0.6.1/modules/x-gsn/virtual-sensors

An example of an XML file describing a light sensor:

```
<?xml version="1.0" encoding="UTF-8"?>
<virtual-sensor name="light_sensor" priority="10" >
  <processing-class>
    <class-name>org.openiot.gsn.vsensor.LSMExporter</class-name>
    <init-params>
      <param name="allow-nulls">false</param>
      <param name="publish-to-lsm">true</param>
    </init-params>
    <output-structure>
      <field name="light" type="double" />
    </output-structure>
  </processing-class>
  <description>Arduino station</description>
  <life-cycle pool-size="10"/>
  <addressing>
  </addressing>
  <streams>
    <stream name="lightInput" sampling-rate="1" storage-size="1">
      <source alias="lightSource">
        <address wrapper= "arduino">
          <predicate key="serialport">/dev/tty.usbmodem1411</predicate>
          <predicate key="baudrate">9600</predicate>
        </address>
        <query>select * from wrapper</query>
      </source>
      <query>select * from lightSource</query>
    </stream>
  </streams>
```

```
</virtual-sensor>
```

- The Name: each virtual sensor has a unique name

- The Priority: controls the processing priority of a virtual sensor (1 is the lowest priority while 20 is the highest, 10 is the default)

- The Processing Class: is a piece of code which acts in the final stage of data processing as sits between the wrapper (discussed later in this document) and the data publishing engine
  - ✶ In order to push data to LSM, the LSMExporter processing class has to be used
  - ✶ It is possible to use other processing classes for other purposes. These processing classes can be found in :

/openiot-0.6.1/modules/x-gsn/src/main/java/org/openiot/gsn/vsensor

  - ✶ Examples of virtual sensors using other processing classes can be found in :

/openiot-0.6.1/modules/x-gsn/virtual-sensors/samples

  - ✶ You can write your own processing class using the help of this link: https://github.com/OpenIotOrg/openiot/wiki/X-GSN-Develop#writing-new-processing-classes

    - ✶ init-params: include some of the initialization parameters of the processing class (inputs of the processing class depending on which class is being used)

    - ✶ The Output Structure: includes the name of the output and its type

- The Description: includes the description of the sensor as a string

- The Life Cycle: enables the control and management of resources provided to a virtual sensor such as the maximum number of threads/queues available for processing.

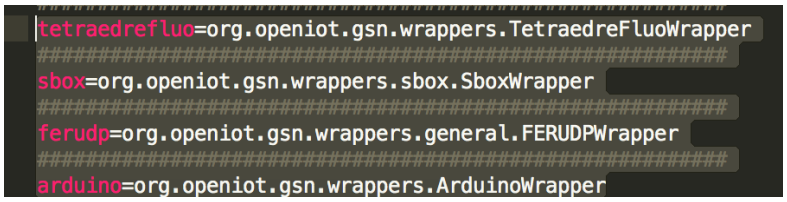
  - ✶ In the example above, the Pool Size specifies a maximum number of 10 threads, which means that if the pool size is reached, data will be dropped (if no pool size is specified, it will be controlled by GSN depending on the current load)

- Addressing: can be left empty because it will be specified in the metadata file afterwards.

- The Stream Name: it is the name of the stream (a stream may contain more than one source)
- The Sampling Rate: it will tell how long the stream should sleep
- The Storage Size: it will tell how much data should be stored locally
- The Source Alias: is the name of the source within the stream
- The Wrapper: Each sensor should have a supported wrapper to be attached to the GSN server, in order to communicate with the sensor hardware.
  - ✶ In the above example, the arduino wrapper is used since the data in collected from an arduino sensor through the serial port
  - ✶ There are many wrappers available in the platform in:

/openiot-0.6.1/modules/x-gsn/src/main/java/org/openiot/gsn/wrappers

  - ✶ You can write your own wrapper following this link: https://github.com/OpenIotOrg/openiot/wiki/X-GSN-Develop#writing-new-wrappers
  - ✶ If you wrote a new wrapper, please don't forget to add it in wrappers.properties file found in /openiot-0.6.1/modules/x-gsn/conf/wrappers.properties



In the example above, ArduinoWrapper in the name of the created wrapper and arduino in the name to be used in the XML file to refer to this wrapper.

  - ✶ Each wrapper has certain predicates to be known after reading the wrapper code. In case of the arduino wrapper, the serial port and the baudrate have to be defined in order for the wrapper to work properly.
- The first query is to specify what to choose from the wrapper in case the wrapper has more than one output

- The second query is to select what to choose from the outputs of the sensor in case the sensor has more than one output (for example measures temp and humidity)

- More virtual sensor XML files can be found here:

/openiot-0.6.1/modules/x-gsn/virtual-sensors/LSM

- More information about GSN, wrappers and processing classes can be found here:

https://github.com/LSIR/gsn/wiki/Documentation

Notice: all the above fields should be present in the XML file in order to work properly. Only values can be changed. Any missing field will cause errors in the platform.

2) Creating Virtual Sensors Metadata File:

- This file describes the data in the virtual sensor XML file

- It has to be located in the same location of the virtual sensor XML file in:

/openiot-0.6.1/modules/x-gsn/virtual-sensors

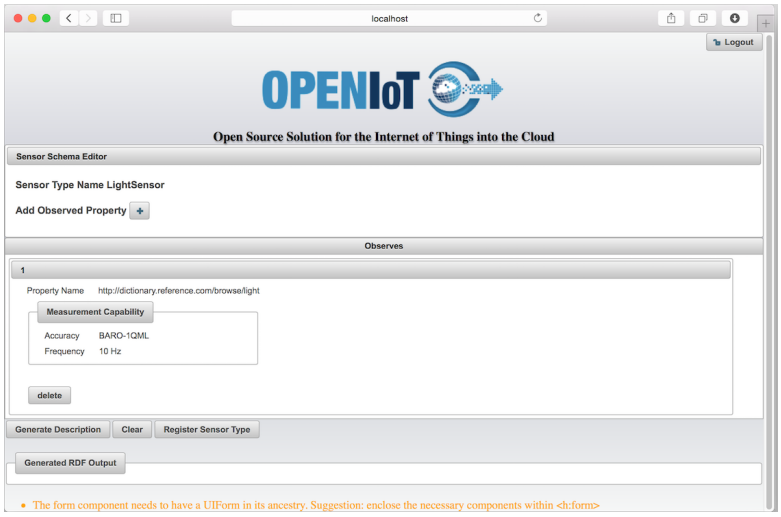
- It must have the same name as the the virtual sensor XML file but with the extension (.metadata). For example, a virtual sensor named *sensor1.xml* will have an associated metadata file named *sensor1.metadata*.

- All the sensors have to be registered to LSM
- To create the metadata file, name it with the right name, place it in the right folder and register it to LSM, all you have to do is to
  - ✶ Run JBOSS
  - ✶ Navigate to http://localhost:8080/ui.schemaeditor if the JBOSS is deployed to localhost. Please replace localhost with the public IP or host name depending on your installation.

- The landing page is a navigator page for
  - ✶ Sensor Type Editor
  - ✶ Sensor Instance Editor

- **Sensor Type Editor:**

- The sensor type editor is used to create new sensor classes which will be automatically stored as an extension to the OpenIoT ontology in LSM server.
- Here is a screenshot of the sensor type editor with a sample sensor description



- **The Fields:**

  - **Sensor Type Name**: the name of the new sensor class to be created

  - **Add Observed Property**: the property is the entity to be measured like the light. If you want to measure only one entity so press on it once to add one property. If you want to measure more than one entity, then press on it more than once to add more than one property

  - **Property Name**: this is a URL containing a description of the entity to be measured (Not necessarily a real one for this stage)
    http://dictionary.reference.com/browse/light

  - **Accuracy**: The accuracy is a string value. There are some well-known accuracy definitions for some sensors or you can write any string describing the accuracy. The application is responsible to convert the accuracy value to corresponding domain specific knowledge

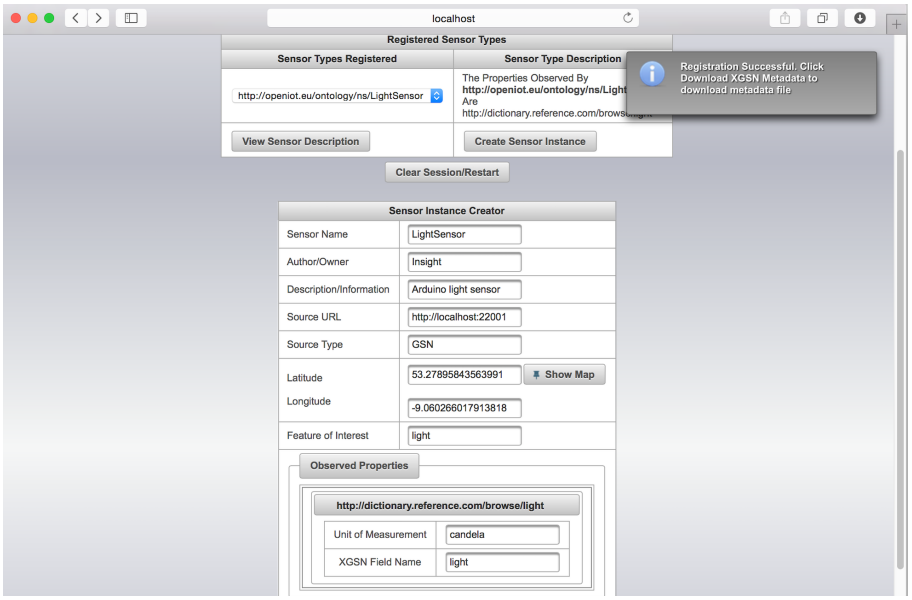
  - **Frequency**: The frequency is a string value. This is the frequency by which the data is being pushed into the platform. The application is responsible to convert the frequency value to corresponding domain specific knowledge

✶ Press Generate Description then Register Sensor Type. If everything is written correctly, you will have a message stating that the sensor is registered correctly.

- **Sensor Instance Editor:**

✶ The sensor instance editor is used to define new instances of a specific sensor type (either defined by the user or by other users).

✶ The screenshot below presents a sample of sensor instance editor.



✶ **Properties:**

- **Sensor Types Registered**: This drop down will display the list of sensor types already registered with the OpenIoT system.

- **Sensor Name**: Name of the sensor instance (e.g. LightSensor).
  Note: Use _ for text separation

- **Author/Owner**: The owner of the sensor.

Note: The owner has to be a single string without any spaces or special character. This is currently a limitation which will be fixed in the next release

- **Description**: Description of the sensor. This can be a freeform text.

- **Source URL:** This is the URL of the data source. E.g. if it is GSN, it will be http://localhost:22001

- **Source Type:** The type of the source which produces data. For e.g. when using GSN to push data into OpenIoT, the source type will be GSN

- **Latitude and Longitude:** used to specify the position of the sensor on the map.

- **Feature of Interest:** A relation between an observation and the entity whose quality was observed. For example, in an observation of the weight of a person, the feature of interest is the person and the quality is weight. (http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#featureOfInterest)

- **Observed Properties:** These fields will be auto generated and is dependent on the sensor type which was defined earlier.

  - **Unit of Measurement:** Defines the measurement unit for the property. E.g.Candela for light

  - **XGSN Field Name:** This is a very important field as it maps the sensor observation to the XGSN fields defined in the XGSN *.xml file and the wrapper. Use the same field name when defining the XGSN XML file. A sample of the XML file is shown below:

```
<output-structure> <field name="light" type="double" /> </output-structure>
```

✶ **Usage: Create and Register a new Sensor Instance:**

1. Select the sensor type for which an instance needs to be created
2. Click the View Sensor Description Button
3. Verify if the sensor instance you wish to create matches the sensor type definition
4. If so click the Create Sensor Instance Button
5. Fill all the necessary field
6. Ensure the page does not display any errors e.g. invalid data format, empty field etc
7. Click the Register Sensor Instance
8. If the registration is successful, you will see a success message.
9. A new "Download XGSN Metadata" Button will appear
10. Click on this to download the sensorname.metatdata file.
11. Save this file under the XGSN virtual director folder (/openiot-0.6.1/modules/x-gsn/virtual-sensors)

✶ **On clicking the register button, the following operation occur:**

1. The sensor instance input is verified for consistency
2. A check if performed to see if a sensor with same name exists
3. The sensor instance is registered with LSM
4. A sensor ID from LSM is then stored locally in the metadata file along with the other user defined sensor instance descriptions

A sample .metadata file is shown below:

```
sensorName=LightSensor
source="http://localhost:22001"
sourceType=GSN
sensorType="http://openiot.eu/ontology/ns/LightSensor"
information=Arduino light sensor
author=Insight
feature="light"
fields="light"
field.light.propertyName="http://dictionary.reference.com/browse/light"
field.light.unit="candela"
latitude="53.278"
longitude="-9.061"
sensorID="http://lsm.deri.ie/resource/14229763638602390000"
```