EPIC-HUB Gateway – Administration & User Guide

Installation

Prerequisites:

-Windows or Linux based OS
-Java version 1.7 or greater installed. The environment; JAVA_HOME and PATH variables should be configured in order to make the **java** command available from the command line.
-Internet connection
-Access to ports: 1527 (derby), 8888 (rest)

Installation (Windows):

Unzip the archive in the desired folder. Rename **run-openmuc.bat.winfile** by removing it's last suffix to **run-openmuc.bat.** Execute this file. When running the application for the first time, the firewall may ask for admin privileges in order to enable the use of different communication ports.

Installation (Linux):

Unzip the archive in the desired folder. The **run-openmuc.sh** file will be used in order to launch the gateway, but this file needs to be defined as executable first. This can be done via the command line: **chmod 755 run-openmuc.sh.**

Configuration

Once installed the two needs need to be configured: the properties files and the database.

Properties files

The '**conf**' folder includes 3 different configuration files:

config.properties include the properties related to the apache Felix framework, and should NOT be modified.

logback.xml is the file related to the logging configuration. It can change looging level, appenders, file location, etc…

system.properties file includes properties used by the bundles in the gateway. Most of the file properties are related to the original openMUC bundles, but the next ones are used to configure properties of the EPIC-HUB bundles:

```
. . .

//Different timeout and retry intervals
epichub.driver_load_timeout=10000            //Timeout for drivers to load
epichub.connect_timeout=60000                //Timeout for a connection
epichub.connect_retry_interval=60000         //Retry interval after a bad connection
epichub.status_check_retry_interval=2000     //Retry interval for checking a var
```

```
//Defines if the gateway should store the monitored data.
epichub.storeToDB=true

//Properties used in the plug-ins. As the default properties files, any property defined
here will be accessible for all the gateway components
org.openmuc.framework.driver.gdss.scenario=scn/scenario.scn
epichub.generalwsntimeout=60000

//Time intervals to check the correct arrival of the data to middleware
//Period interval to check the data
epichub.check_variables_value_interval=3000000
//Threshold admitted between last data sent and checked to avoid temporal delays
epichub.check_variables_value_time_threshold=2000000
```

Database Configuration

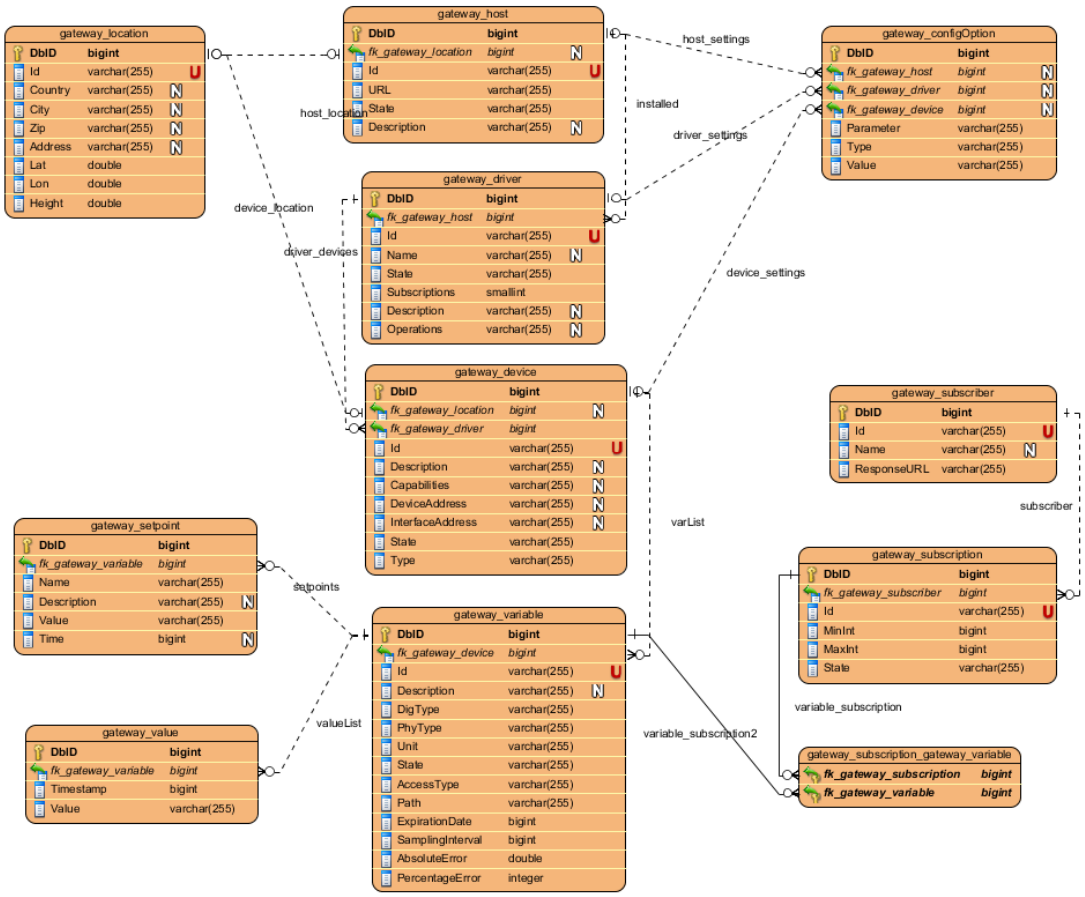
EpicHub Energy gateway will use a Derby database instead of the **openmuc-config.xml** file to store and configure drivers, devices, variables, subscriptions… When running the gateway, one of the bundles will create an endpoint to connect to the Derby database. The database will be accessible through **jdbc:derby://localhost:1527/epichubdb**. The user and password for accessing the database will be both **epichub**.

Different SQL clients can be used to access the derby database, as long as they have jdbc support. In our case **ij** has been used, the client provided along with the derby installation (http://db.apache.org/derby/papers/DerbyTut/ij_intro.html). Another alternative with a graphical user interface is **SQuirreL SQL** http://squirrel-sql.sourceforge.net/.

In order to populate the databases the use of SQL scripts is recommended, so it can be used to load specific configurations or if the database needs to be resettled. Scripts should connect to the database, clear old data and write the new gateway configuration.

Important! **The database is dynamically started from one of the bundles, so the gateway NEEDS to be running in order to access it. If the gateway is stopped, the database won't be accessible.**

Database Entities



- **Host**: Refers to the physical device where the gateway has been installed and where to access it. **ID** and **URL** will be used by the middleware to detect which gateway is sending information and where it's accessible. **STATE** will provide the state of the gateway [AVAILABLE, DISABLE, ERROR] and **DESCRIPTION** will be used just for informative means

- **Drivers**: The communication protocols that the gateway will use, in this case OpcUa and a dummy driver provided by openMUC. Drivers will be associated to a host's DBID through **FK_GATEWAY_HOST. ID** will be used by the gateway to load the plug-in, **STATE** will to show if the driver is available, disabled or it has some errors [AVAILABLE, DISABLE, ERROR] **SUSBCRIPTIONS** will sow if the driver supports the *Listening* capabilities provided by openMUC. **NAME** and **DESCRIPTION** are used just for additional information

- **Device:** Physical device to which a driver will try to connect. The device will use a driver to be accessed (**FK_GATEWAY_DRIVER**). **ID** will be used to identify the device at both middleware and gateway levels It should follow the next pattern: `urn:rixf:eu.epichub/devices/pilot/xxx` where *pilot* is the pilot test site and *xxx* is the id unique to the device. **STATE** will inform of the status of the device [AVAILABLE, ERROR]. **DEVICEADDRESS** and **INTERFACEADDRESS** will be used by the driver to access

the device and they will be the direct equivalent of openmuc-confi.xml deviceAddress and interfaceAddress. Depending the implementation of the driver, deviceAddress will be optional. **TYPE** and **DESCRIPTION** provide extra information.

- **Variable:** The specific parameters that the device is monitoring. Variables will be associated to a single device (**FK_GATEWAY_DEVICE**). **ID** will be used to identify this variable in both middleware and gateway layers and should follow the next pattern `urn:rixf:eu.epichub/variables/xxxxx`. **PATH** will be openMUC's channel address configuration parameter (channelAddress) , and will be used by the driver to connect to the variable inside the device. **DESCRIPTION** will be a small description of the variable, **DIGTYPE** will define the digital type in which the values are stored [BOOLEAN, INTEGER, DOUBLE, STRING, FLOAT, BYTE, STRING, LONG, SHORT], and will be used for data conversion, **PHYTYPE** provides information about the magnitude of the variable, what is measuring. **STATE** will inform about the status of the variable, if it is available or there has been an error accessing it [AVAILABLE, ERROR]. **ACCESSTYPE** will define if the variable can accessed in read, written or read/write mode[R,W,RW]. **EXPIRATIONDATE, SAMPLINGINTERVAL, ABSOLUTEERROR, PERCENTAGEERROR** will be parameters used by the middleware.

- **Subscriber:** Entities that should be informed of the different gateway notifications: alarms, configuration changes, sensor-data updates... In the projects actual approach it will only be one, the THALES middleware. **ID** will identify the different subscribers to notify and **RESPONSEURL** will define the URL where to send the notifications, which in our case is the provided in the CDM WSDL. **NAME** is additional information

- **Subscriptions**: will be used to define the monitoring conditions of the variables. **FK_GATEWAY_SUBSCRIBER** will define the subscriber to notify**, ID** will be used as identifier at gateway level and should follow the next pattern `urn:rixf:eu.epichub/subscriptions/xxxxxx`, **STATE** will define whether the subscription is active or disabled [STARTED, STOPPED]. **MININT** will define the sampling interval of a variable and will limit the response if the driver has Listening capabilities. **MAXINT** can be used to periodically resend the values when no changes have occurred, mainly to inform the middleware that there are no connection problems.

- **Subscription-Variables:** A subscription can be associated to more than one variable in order to send them as a batch. In the same way, a variable could be in various subscriptions in order to notify different subscribers.

The gateway database includes tables for other entities, but in order to test the plug-ins only the previous ones need to be configured. Depending the plug-in implementation, drivers and devices may need additional settings. These setting can be added by defining **gateway_configOption** entities.

In order to avoid deleting other drivers structural data DBID ranges have been assigned for each plug-in. This ID range should be taken into account in **ALL the tables.**

- G-OPC: 1-1000
- G-WSN: 1001-2000
- G-V6000: 2001-3000
- G-DSS: 3001-4000

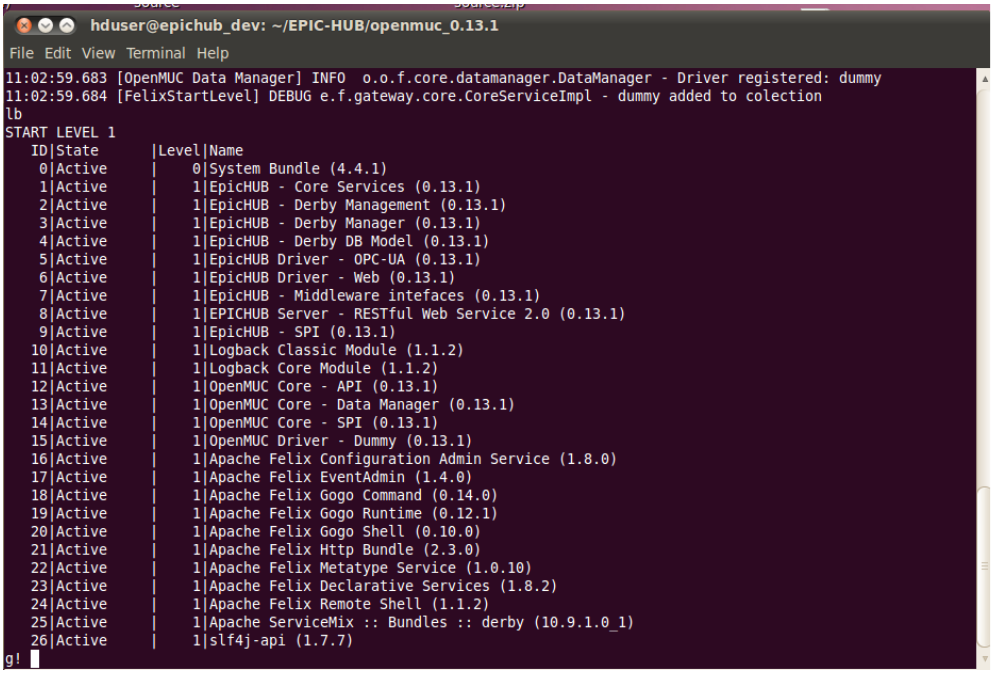
These will affect to the tables;

- Gateway_driver
- Gateway_device
- Gateway_variable
- Gateway_subscription

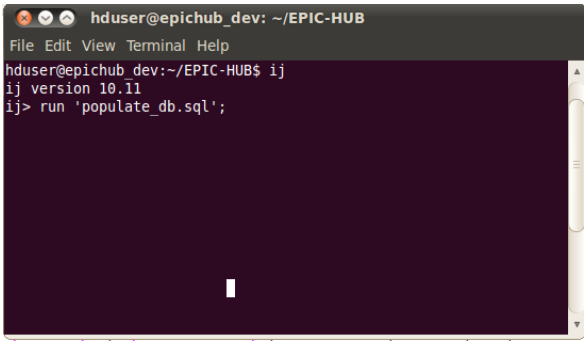
Run

The gateway should be activated and working in order to access the database. To do so, execute the **run-openmuc** launcher located in the ***demo/framework folder*** (.sh for linux based systems and .bat or windows systems). It will activate the Apache Felix server, which will load the bundles located in the ***demo/framework/bundle*** folder. After I has finished, the status of the bundles can be checked with the **lb** command. All the bundles should be in the 'Active' state. In order to list the available commands, use **help.** The usual bundles will include:

- System bundle: This is the main bundle, stopping it will exit the gateway.
  EpicHUB - Core: Provides extra functionalities and manges the other EpicHUB bundles.
  EpicHUB - Derby Management: Sets up Derby database in port 1527
- EpicHUB - Derby Manager: manages the operations between the gateway and database
- EpicHUB - Derby Model: Provides ORM classes for Derby database
  EpicHUB Driver - XXX: Communication protocol plug-ins.
- EpicHUB Middleware Interfaces: Manages the interactions with the middleware.
- EpicHUB Server: REST server launched on port 8888.
- EpicHUB SPI: Packages the common interfaces of every epichub component
- Logback & slf4j: Logging functionalities
- OpenMUC bundles: Original openMUC bundles, used by EpicHUB bundles
- Apache Felix bundles: Manage the apache felix OSGi container and provide various functionalities: shell, runtime, commands, event mangement…
- Apache ServiceMix – derby: Connector bundle for derby support



With the gateway running, use a SQL client to connect to the **`epichubdb`** database open in 1527 port. In this example 'populate_db.sql' file, the connection is already included with the `connect` command `'jdbc:derby://localhost:1527/epichubdb',` so the only instruction that is needed to be executed in **ij** is **`run 'populate_db.sql'`**;



Once the database is configured, stop the gateway (From the felix shell: **Ctrl+C** or **stop 0** will work). Start it again to load the new configuration and check that the bundles are correctly loaded again.

Logging

The gateway will use diverse logs to output messages. The logs can be found in the main folder ***demo/framework.*** This and many other logging parameters can be changed in the logback.xml config file inside de ***demo/framework/conf*** folder. The current configuration publishes all in the log messages both in terminal and in a file (openmuc.log).

The openmuc.log file is rolling every day or when it reachs 20Mb size. Logs files older than 30 days are deleted. This configuration can be changed.

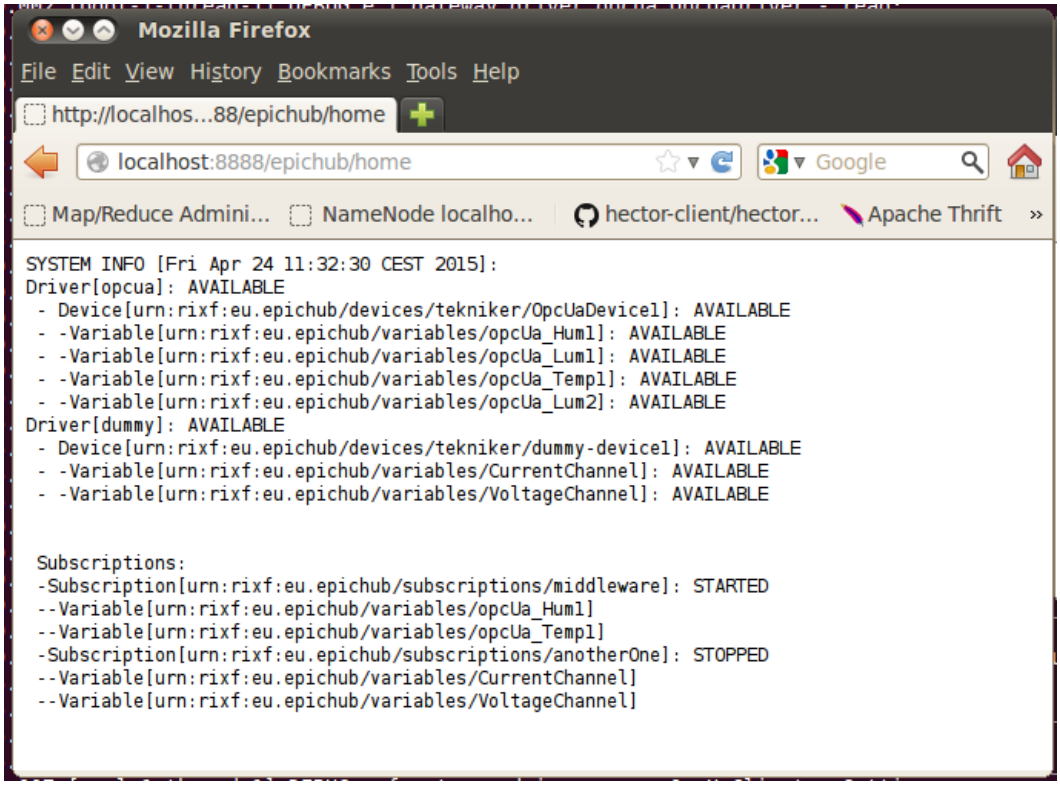
REST Services

The gateway will include a REST server that can be used for accessing different functionalities

EpicHUB will use a modified version of openMUC's REST server.

System Information

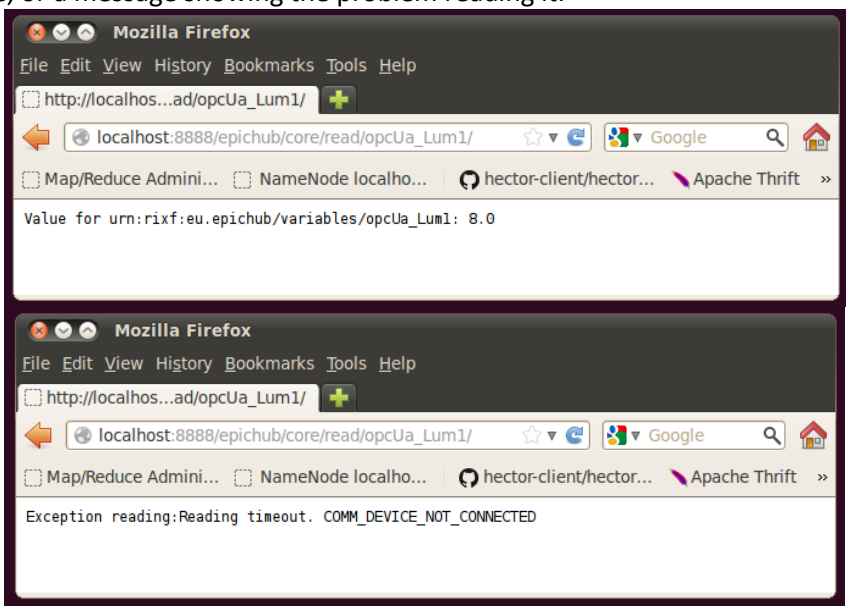
The plug-ins can be tested with the REST server:
http://x.x.x.x:8888/epichub/home will show the stored information in the database. It will list in a tree-like list the loaded drivers, devices and variables, as well as the subscriptions and their state.



Driver functionalities

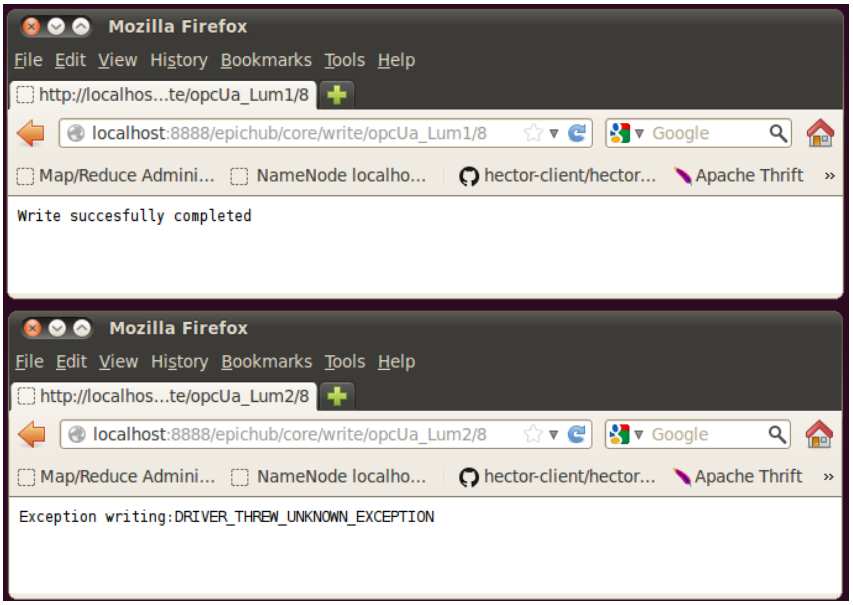
Read

http://x.x.x.x:8888/epichub/core/read/{site_id}/{variable_id}:
The plug-in will try to connect to the variable `urn:rixf:eu.epichub/variables/{site_id}/{variable_id}` and read it. It will show the actual value of that variable, or a message showing the problem reading it.



Write

http://x.x.x.x:8888/epichub/core/write/{site_id}/{variable_id}/{value}:
The plug-in will try to connect to the variable `urn:rixf:eu.epichub/variables/{site_id}/{variable_id}` and write the provided value. It will show a message with "Write successfully completed" or an exception if cannot be completed.

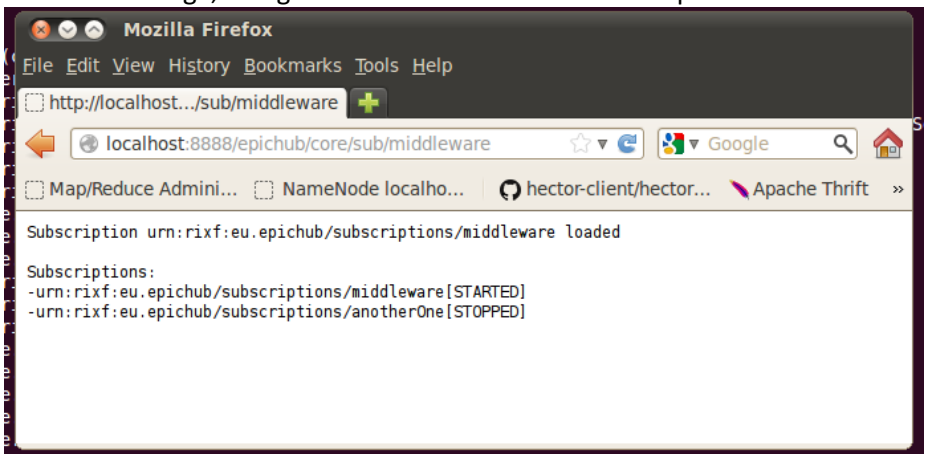


Subscribe

In the current version of the gateway the subscriptions will 'publish' in the log the information of the subscribed variables, as well as sending a publication messages to the middleware. If the option is enabled in the configuration file, it will also store the readings in the DDBB

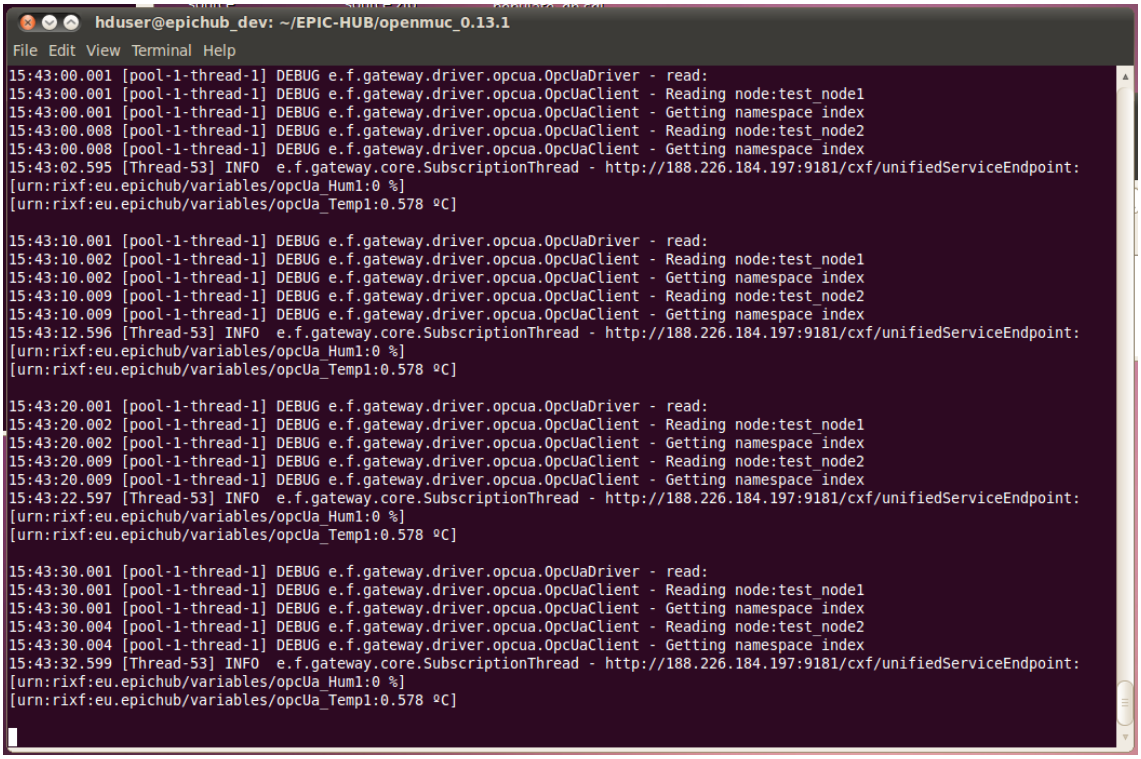
http://x.x.x.x:8888/epichub/core/sub/{subscription_id}/:

The gateway will try to load the subscription `urn:rixf:eu.epichub/subscriptions/{subscription_id}` from the database. It will first check if the variables associated are available. It will return a success or failure message, along with the status of all the subscriptions.



http://x.x.x.x:8888/epichub/core/unsub/{subscription_id}/:

The gateway will stop the subscription `urn:rixf:eu.epichub/subscriptions/{subscription_id}`. It will return a success or failure message, along with the status of all the subscriptions.



Middleware communications

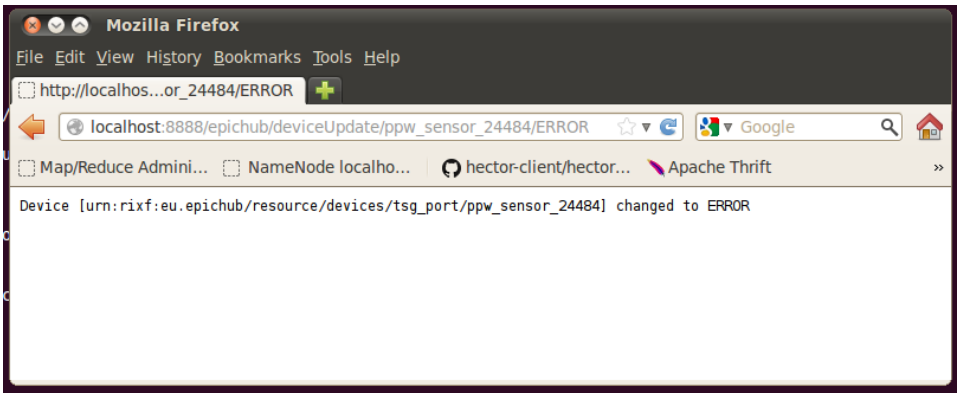
The REST service includes functionalities to simulation the device status change for the panoramic power devices installed in the TSG pilot. The status can be changed manually or through a simulation cycle. This was developed mainl to test the communication capabilities with the THALES middleware.

deviceUpdate

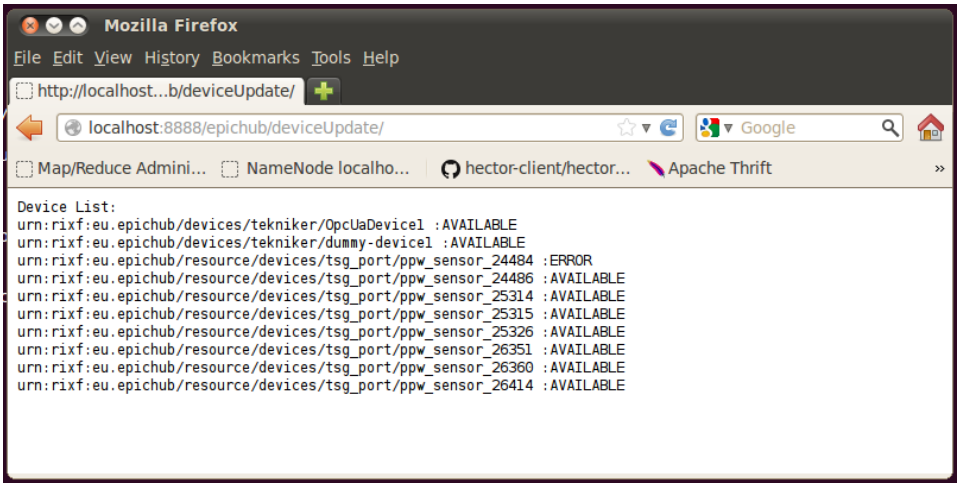
This function is used to manually simulate device failing/recovering, publishing different messages to the CDM middleware.
When accessing the URL into a web browser:
http://x.x.x.x:8888/epichub/deviceUpdate/{id}/{state} the gateway will first check if the provided state is a valid one (ERROR or AVAILABLE). Then it will check if the device with the id **urn:rixf:eu.epichub/resource/devices/tsg_port/{id}** exists in the database and if its state is different to the provided one. If everything is correct, the application will produce an event and the state of the device will change, sending two messages to the CDM middleware:

1. A resource update
2. An alarm definition/update.

The first time a device receives the event, its state will change to “ERROR” and a new alarm will be created.
The next time a device receives the event, its state will change to “AVAILABLE” and the previous alarm created will be change its state to “CLOSE”
If everything went correctly, a response message will notify the device state change.



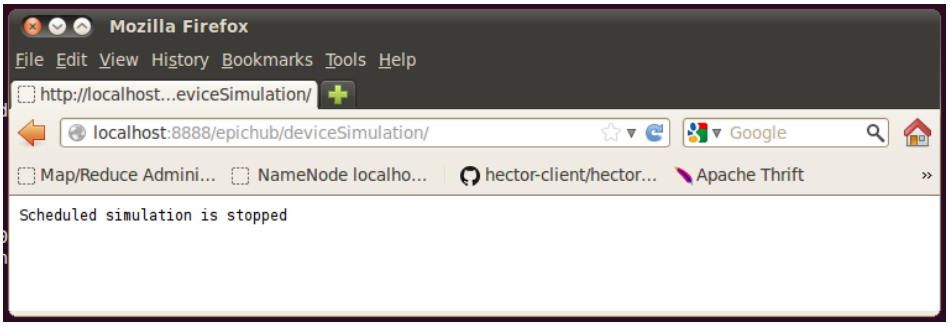
The status of all the devices can be checked in **http://x.x.x.x:8888/epichub/deviceUpdate/**



deviceSimulation

Through this service the gateway will simulate devices going on and off in a cyclic manner. The event will modify the state of the devices, publishing ResourceStateUpdate and Alarma Definition/Update messages to the CDM.

http://x.x.x.x:8888/epichub/deviceSimulation/start : Start the simulation
http://x.x.x.x :8888/epichub/deviceSimulation/stop : Stop the simulation
http://x.x.x.x:8888/epichub/deviceSimulation/: Check simulation status



Once started the simulation will run continuously cycling each 720 seconds, or until it is manually stopped.

Time	Device Ids	Status Change
0 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_24484	ERROR
60 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_24484	AVAILABLE
120 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_24486 urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_25314	ERROR
180 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_24486 urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_25314	AVAILABLE
240 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_25315 urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_25326 urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_26351	ERROR
300 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_25315 urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_25326	AVAILABLE
360 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_26351	AVAILABLE
420 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_26360	ERROR
480 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_26414	ERROR
540 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_26414	AVAILABLE
600 s	urn:rixf:eu.epichub/resource/devices/tsg_port/ppw_sensor_26360	AVAILABLE
720 s	Cycle restart	

Update

For updating an existing installation of the gateway it is very important to know how the update affects to the database because all information about devices and variables that are being monitoring is configured in the database.

Before an update is recommended backup directories “conf” and “epichubdb” that stores the configuration of the gateway.

If there is no change in the “system.properties” file you can substitute directly the new version with your old version. If there is any change you must move the values from the old file to the new file.

The same procedure is valid for the database. If there is no change in the database you can substitute the new “epichubdb” directory for your existing database “epichubdb” directory.

If there is any change in the database there are two options;

- Use the new database and populate the complete configuration using SQL scripts. This approach can be easy if you have a complete DB creation script available and changes does not affect them. Using this approach you lose all data values recorded by the gateway.
- Use the old database and change the structure executing the corresponding SQL scripts. Using this approach you don’t lose any data values but can be more complicated depending of the database structure changes.

FAQ:

- **Derby Management/Manager doesn't load correctly:**
  This could be due to the Derby database not being correctly initialized, IP naming problems or bad hibernate mappings.
  - Check if **localhost** is enables as the local IP name by using the command **'ping localhost'.**
  - Check if the **epichub-derby-model-0.13.1** is mapping correctly the database entities. To do this open the .**jar** bundle in the 'bundle' folder, and search for the **EHReq.cfg.xml** file inside the **ormmapping** folder.

    ...
    <property name="dialect">org.hibernate.dialect.DerbyDialect</property>
    <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
    <property name="connection.url">jdbc:derby://localhost:1527/epichubdb</property>
    <property name="connection.username">epichub</property>
    <property name="connection.password">epichub</property>
    <property name="hibernate.connection.provider_class">
    - org.hibernate.service.jdbc.connections.internal.C3P0ConnectionProvider
      </property>

    ...

  The gateway is able to use a remote database to map the entities, modifying the attributes on this file.

- **Error creating GatewayClient: 2 counts of InaccessibleWSDLException:**
  The bundle cannot reach the designed CDM middleware endpoint. Check inside the '**conf**' folder the **system.properties** file. The attribute **wsdl.client.url** should point to the desired wsdl as follows:

  ...
  wsdl.client.url = http://ip:port/cxf/unifiedServiceEndpoint