

# M3 developer documentation

To set up, run, understand the M3 project and exploit M3 web services

Creator	Amelie Gyrard (Eurecom)
Goal	<ul style="list-style-type: none"><li>• This documentation guides developers to set up and run the M3 project.</li><li>• Understanding the M3 project code (description of packages)</li><li>• Understanding M3 web services</li><li>• It also help understand the web services (to use it or add new ones if you have the code)</li><li>• Add a new Semantic Web of Things (SWoT) template</li><li>• Reference a new ontology-based project in LOV4IoT (HTML web page &amp; RDF dataset)</li></ul> <p>The code is not shared online (need refactoring and ask for permissions).</p>
Requirements	Java 1.7, Eclipse Kepler, App Engine SDK 1.9, Jena framework 2.11.
System	Windows 7, 64 bits
Last updated	April 22th 2015
Status	Work in progress
URL	<a href="http://www.sensormeasurement.appspot.com/documentation/M3DeveloperDocumentation.pdf">http://www.sensormeasurement.appspot.com/documentation/M3DeveloperDocumentation.pdf</a>
Time estimated	1 day to set up the project



**After three years and half of research and development, this project really needs a refactoring (even if I have done it frequently), and even maybe to be re-think from scratch. Anyway, this is the documentation to reuse, set up and understand the M3 project.**

## Table of Contents

Part I :	Setting up the M3 project.....	3
I.	Requirements.....	3
II.	Install Java and JRE 1.7.....	5
III.	Install Eclipse Kepler .....	5
1.	Use the default Eclipse workspace .....	7
2.	Optional: Reference your own workspace.....	8
3.	Result expected: Check that you can see the M3 project .....	9
IV.	(Optional) Install the Google Plugin for Eclipse .....	9
Part II :	Running the M3 project on localhost .....	12

I.	Discover the M3 project and run it .....	12
1.	Fixing Google App Engine Problem .....	13
2.	Java Build Path Problems .....	14
II.	(Optional) Install Jena inside Eclipse .....	15
III.	Run M3 on localhost .....	15
Part III :	Deploying M3 on the Web .....	16
I.	Creating an application on app engine .....	22
II.	Error App engine when deploying .....	23
Part IV :	Understanding the M3 project .....	24
I.	M3 framework overview .....	24
II.	Libraries (Jars) required .....	26
III.	Descriptions of packages .....	27
1.	eurecom.common.util .....	28
2.	eurecom.constrained.devices .....	29
3.	eurecom.data.converter .....	29
4.	eurecom.generic.m2mapplication .....	29
5.	eurecom.sparql.result .....	30
6.	eurecom.store.jdo .....	30
7.	eurecom.search.knowledge .....	30
8.	eurecom.web.service .....	30
IV.	The WAR directory .....	31
1.	WAR/CSS .....	32
2.	WAR/dataset .....	32
3.	WAR/documentation .....	32
4.	WAR/html .....	32
5.	WAR/images .....	32
6.	WAR/javascript .....	32
7.	WAR/ont .....	33
8.	WAR/publication .....	33
9.	WAR/RULES .....	33
10.	WAR/SPARQL .....	33
11.	WAR/WEB-INF/ontologies .....	34

Part V :	Understanding M3 web services.....	35
1.	APIJsonWS Java class (deprecated) .....	35
2.	LOV4IoTWS Java class .....	35
3.	M3JsonWS (deprecated).....	36
4.	M3WS.....	36
5.	NaturopathyWS .....	37
6.	RestaurantWS .....	38
7.	STACWS.....	38
8.	SWOTWS .....	39
9.	TourismWS.....	40
10.	TransportWS .....	41
11.	Design a new web service .....	42
Part VI :	Adding a new SWoT template.....	42
Part VII :	Adding a new ontology in LOV4IoT .....	43
1.	HTML web page .....	43
2.	LOV4IoT RDF dataset .....	43
Part VIII :	Additionnal.....	44
I.	Jena TDB and Jena Fuseki (Internship Amira) .....	44
II.	Security issues to execute JavaScript with the project.....	45

# Part I : Setting up the M3 project

## I. Requirements

- Java 1.7
- Eclipse Kepler
- Plugin Google App Engine SDK App engine 1.9
- Jena
- Jars required

To avoid any incompatibility issues, we provide all tools with the good version:

On BSCW (For Eurecom users)

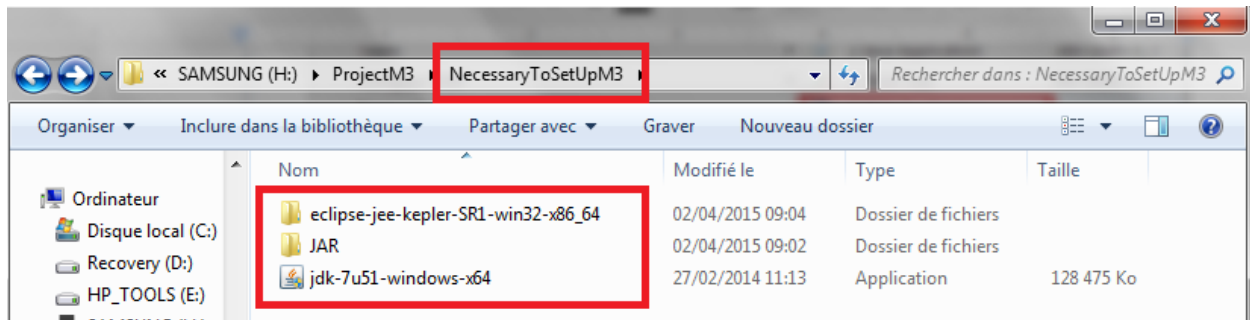
You have access to the directory SetUpM3 which is comprised of:

- ➔ NecessaryToSetUpM3 with eclipse Kepler, all JAR required and the JDK and JRE 1.7 (e.g., NecessaryToSetUpM3.zip)
- ➔ workspace with the M3 project (e.g., m3\_save2Avril2015.zip)

Download and unzip both!

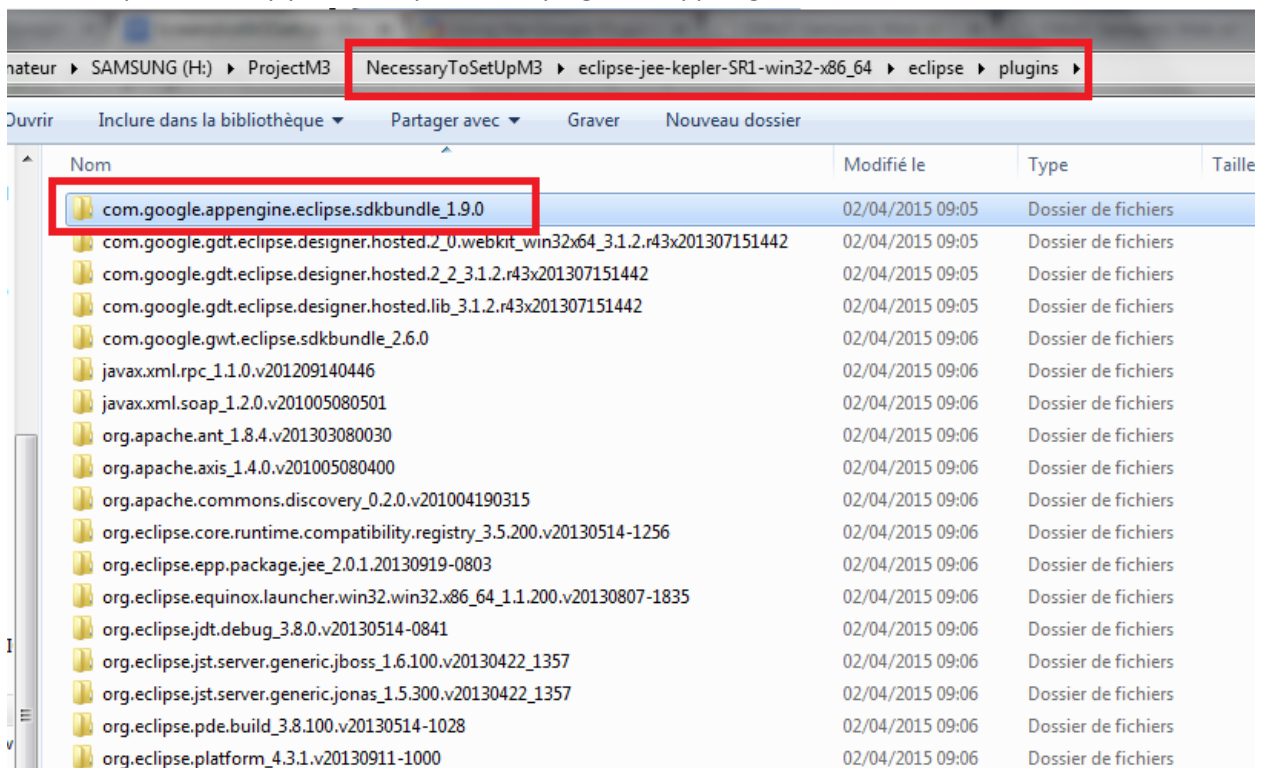
The NecessaryToSetUpM3 directory is comprised of:

- ➔ Eclipse Kepler
- ➔ All jar required to run the M3 project
- ➔ JDK Java 1.7

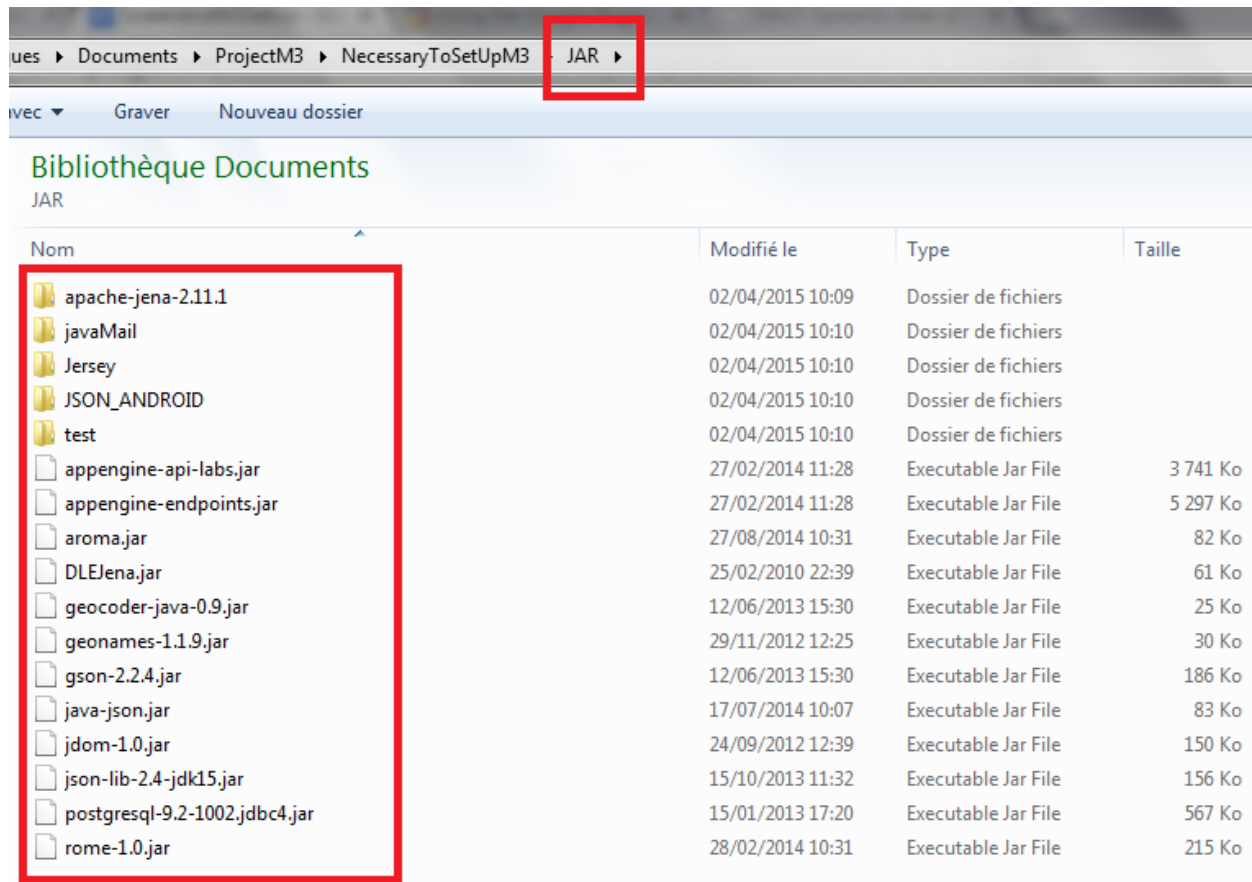


**Figure 1. NecessaryToSetUpM3 directory**

In the Eclipse directory you already have the plugin for App Engine SDK 1.9:



**Figure 2. App Engine SDK 1.9 already in the Eclipse that we provided**



**Figure 3. JARs required in the M3 project are provided in the JAR directory**

## II. Install Java and JRE 1.7

In our case: jdk-7u51-windows-x64

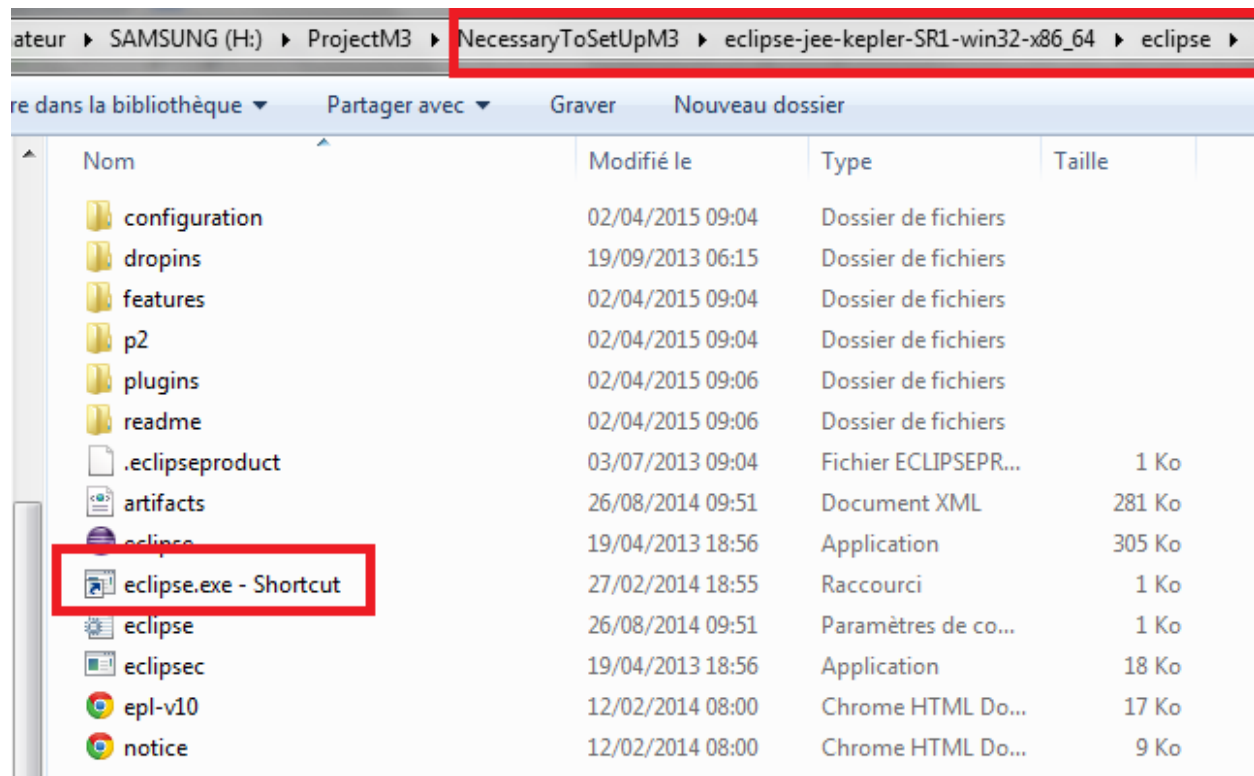
Because of Google App Engine, we cannot use anymore Java 1.6.

If you have Java 1.6, you should uninstall it, restart the machines, to avoid issues.  
(if it does not compromise the other projects running on your machine).

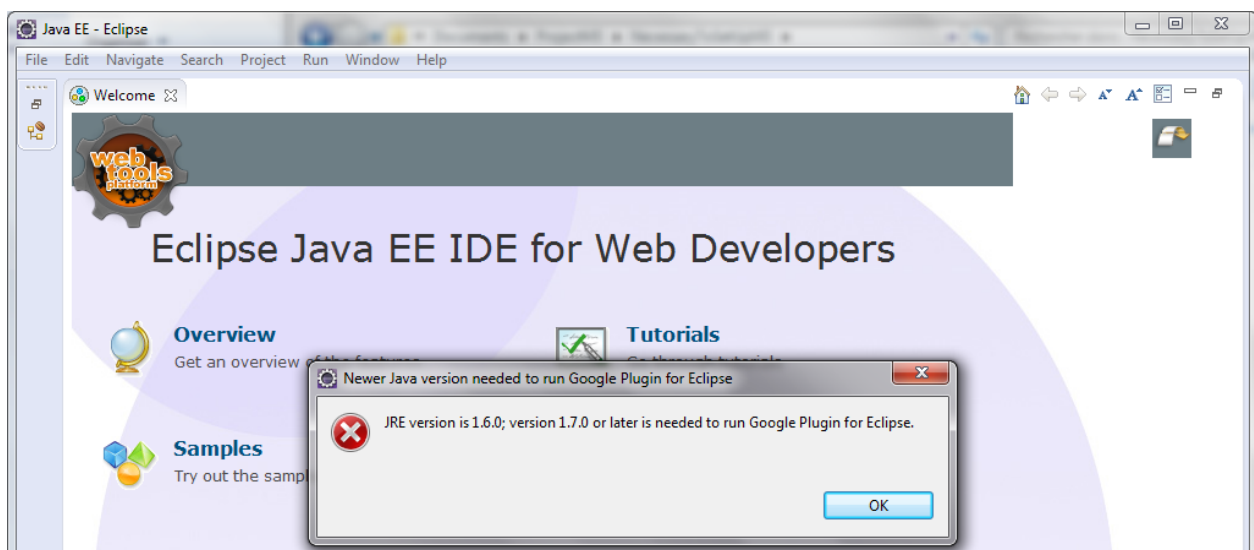
Double click on the JDK and install it.

## III. Install Eclipse Kepler

In our case: eclipse-jee-kepler-SR1-win32-x86\_64

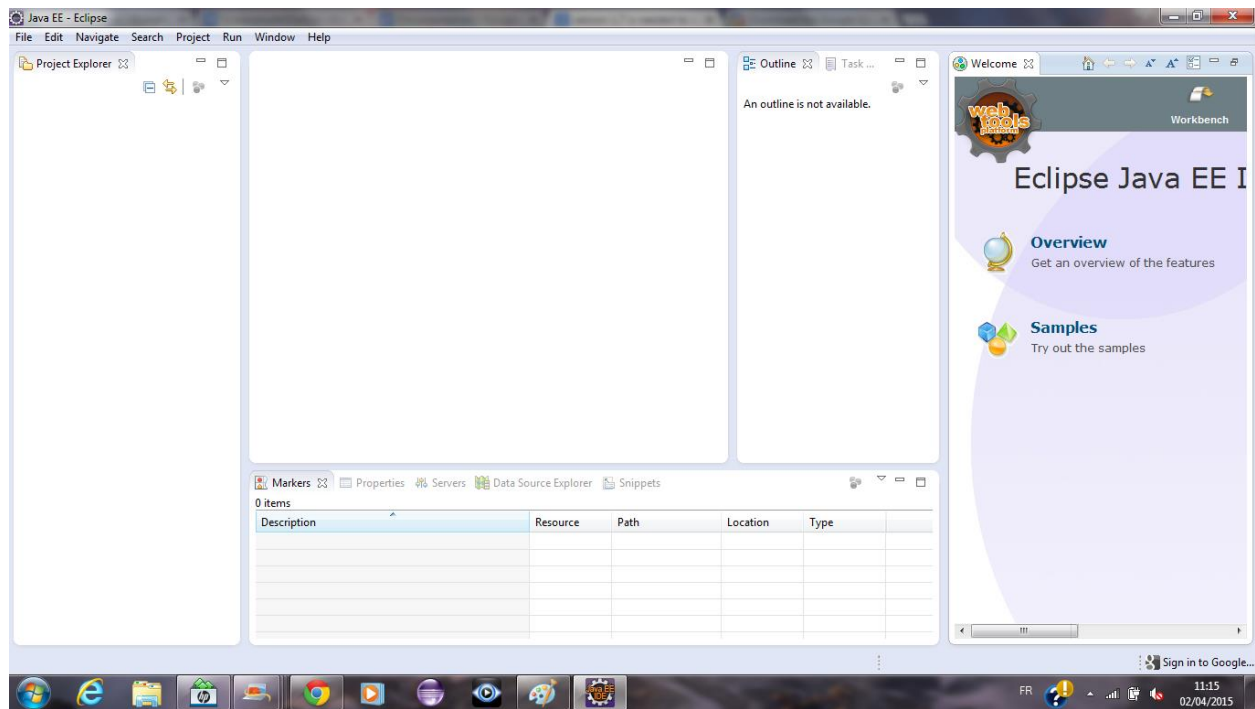


**Figure 4. Install Eclipse Kepler that we provided**



**Figure 5. Java 1.7 is required with the Google Plugin for Eclipse**

➔ Click ok (This is why we set up Java 1.7)



**Figure 6. The eclipse workspace is empty**

## **1. Use the default Eclipse workspace**

Eclipse installed by default a workspace: E:\Workspace.

In this case, close Eclipse, copy the workspace with the m3 project and replace the default workspace with this workspace.

Result expected:

## 2. Optional: Reference your own workspace

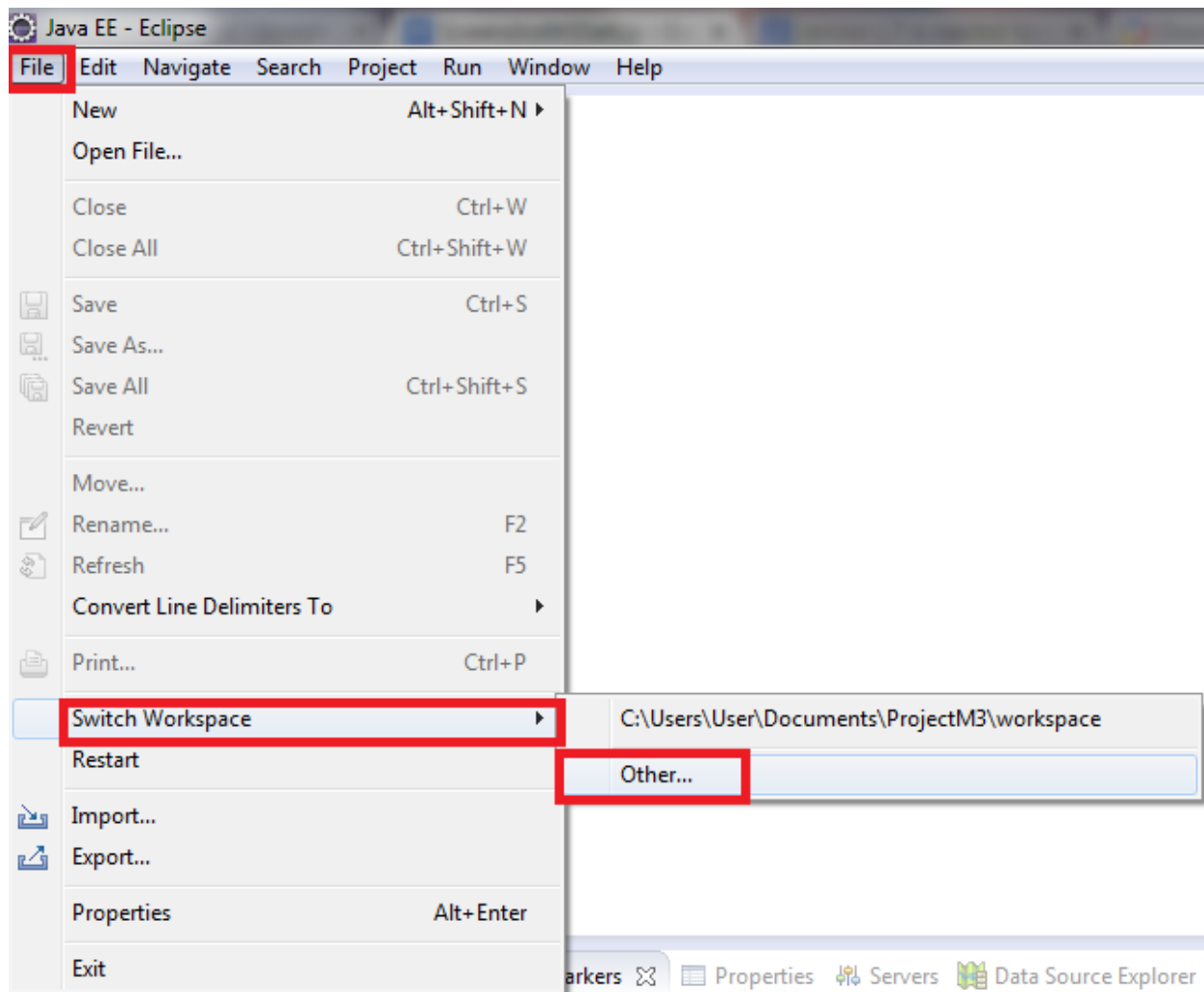


Figure 7. Switch workspace to reference the M3 project

We referenced the existing M3 workspace and not the one by default:

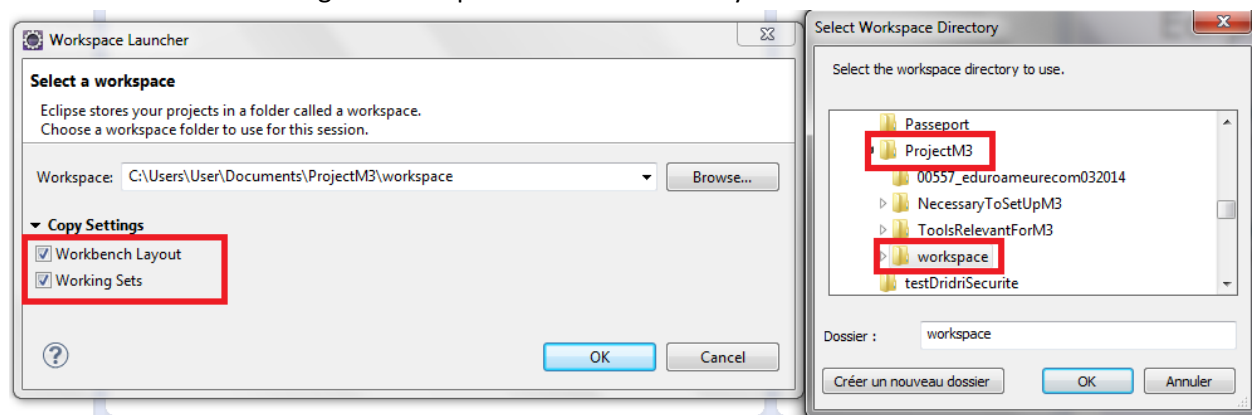


Figure 8. Reference the M3 project



### 3. Result expected: Check that you can see the M3 project

You should have the following screenshot:

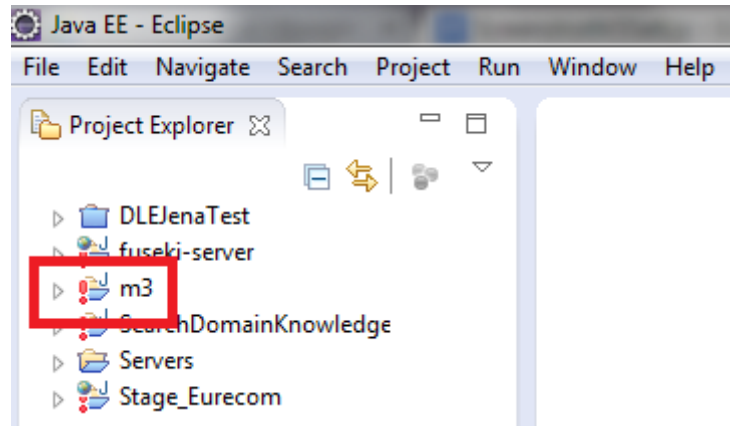


Figure 9. The M3 project in the Eclipse workspace

## IV. (Optional) Install the Google Plugin for Eclipse

In case you encounter some issues, you can follow this tutorial:

<https://cloud.google.com/appengine/docs/java/tools/eclipse>



**We do not remember why, but we have to change the default setting of Google:  
M3 project -> Right Click -> Google -> App Engine Settings  
→ Choose DataNucleus JPO/JPA version v1**

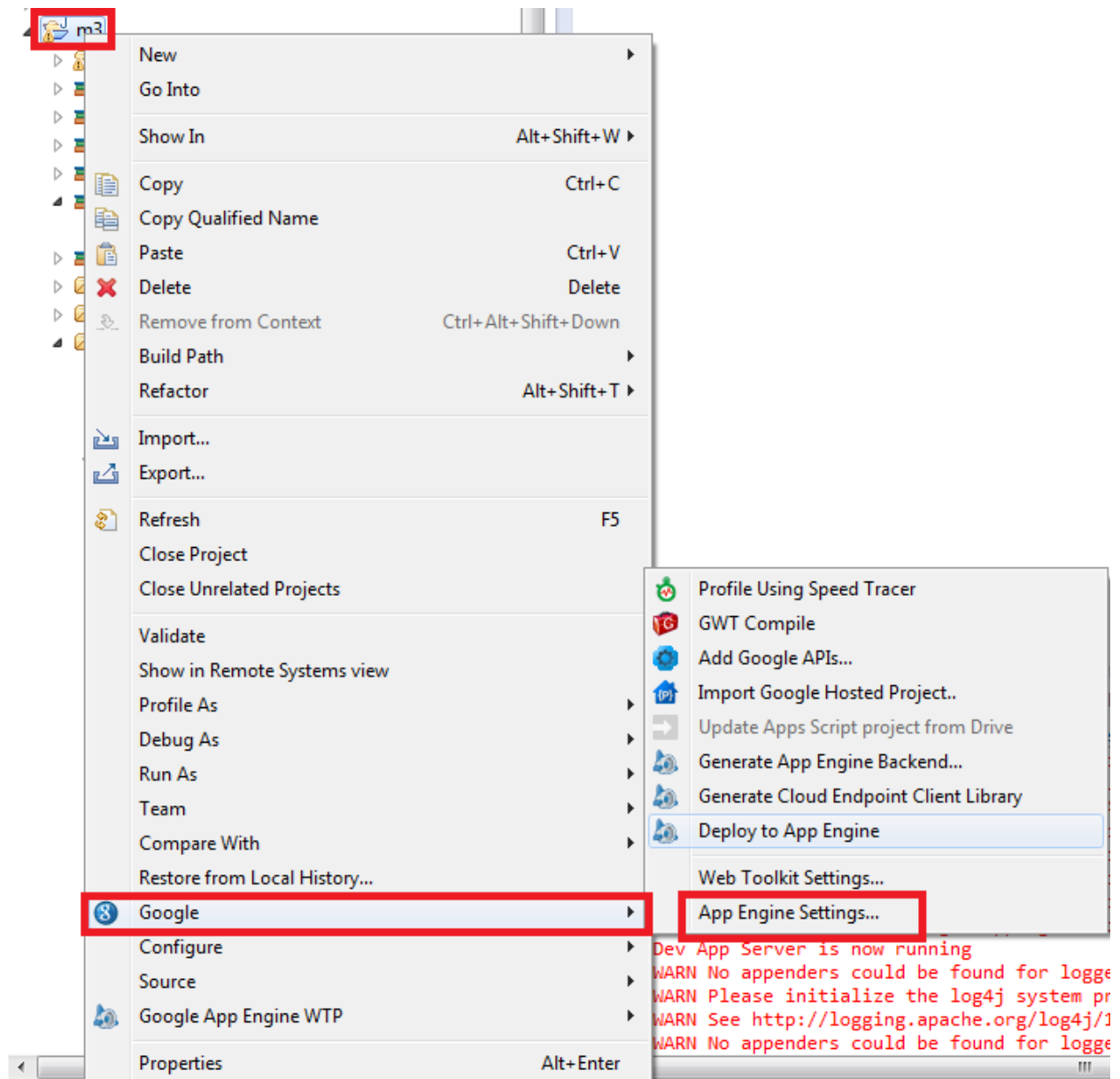


Figure 10. Google App Engine settings

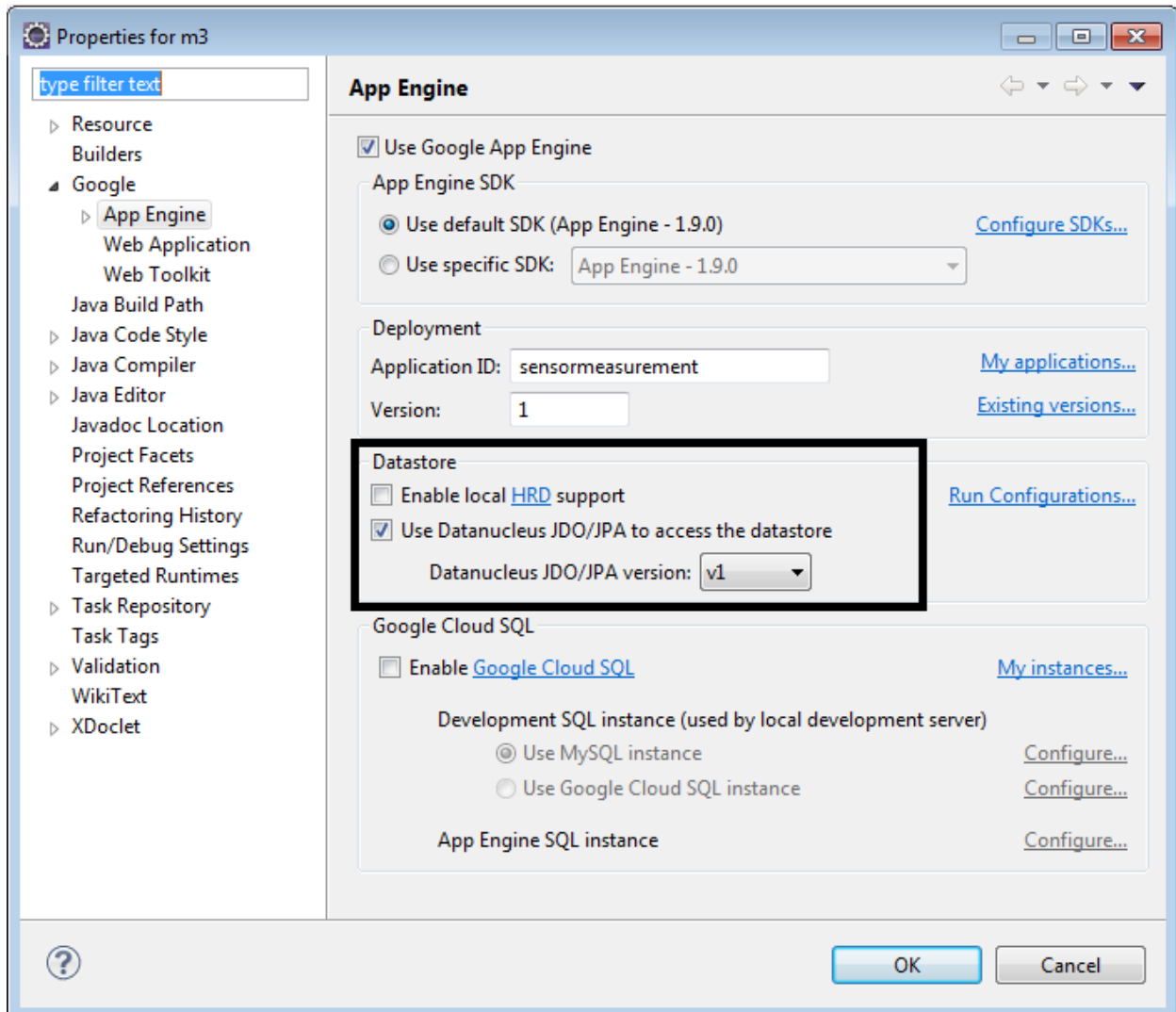
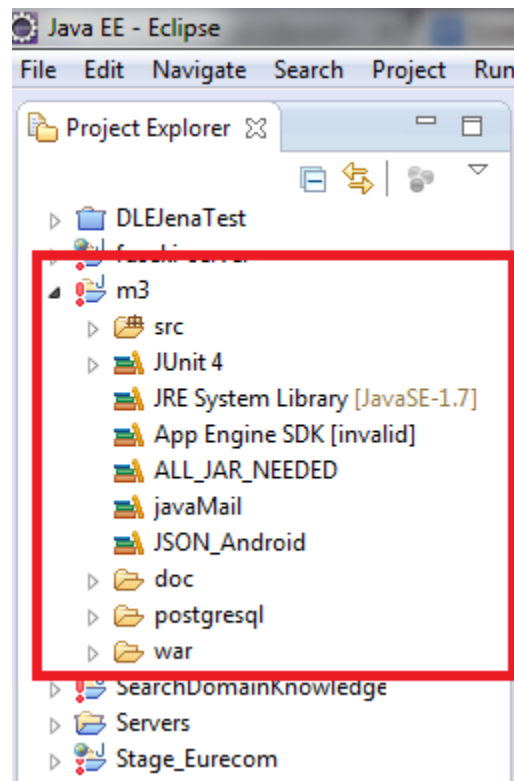


Figure 11. Check that the Datanucleus JPP/JPA is V1

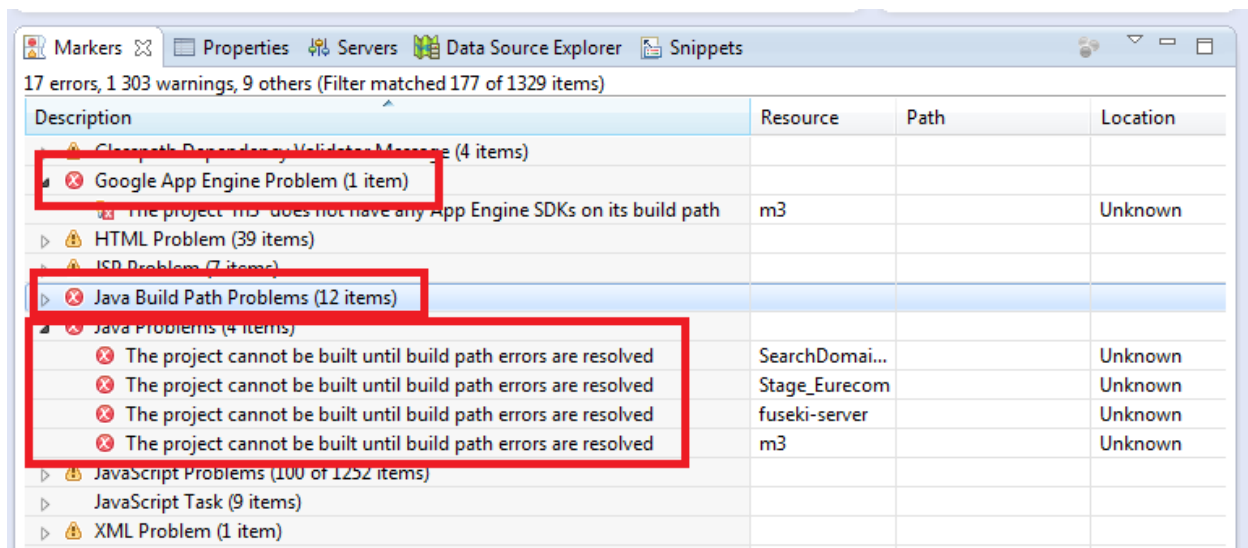
## Part II : Running the M3 project on localhost

### I. Discover the M3 project and run it



**Figure 12. Open the M3 project**

We have to fix the following issues:



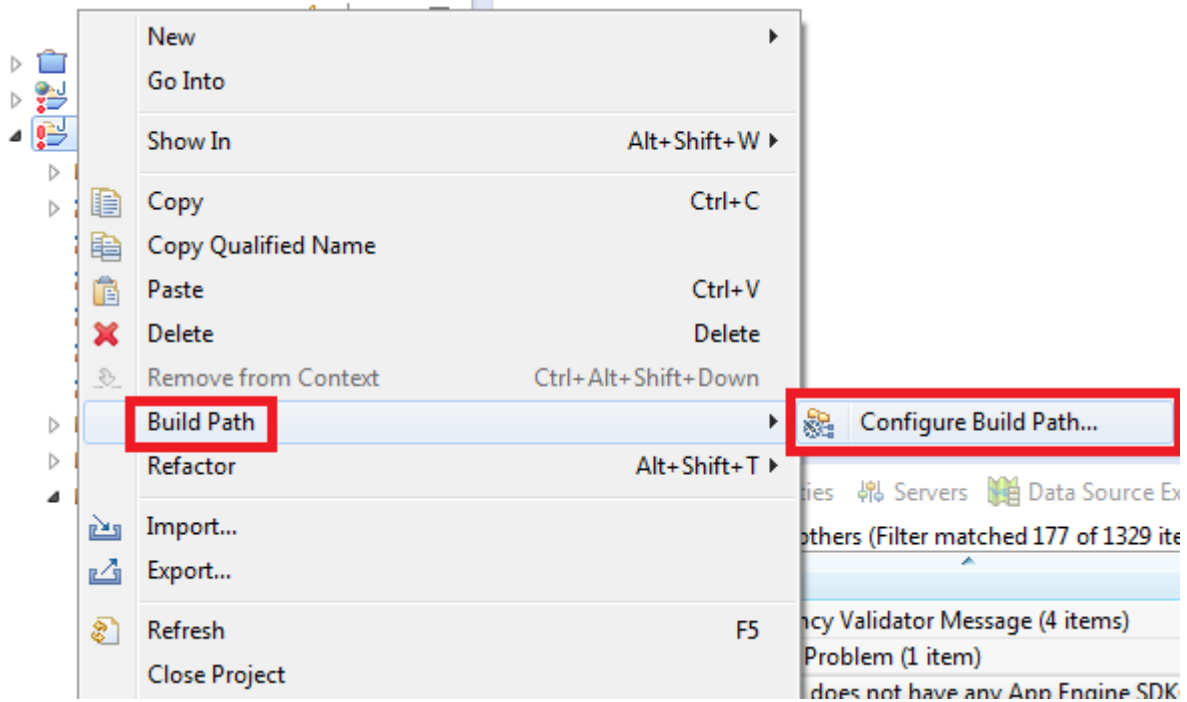
**Figure 13. Fix issues to run the M3 project**

## 1. Fixing Google App Engine Problem

Two solutions:

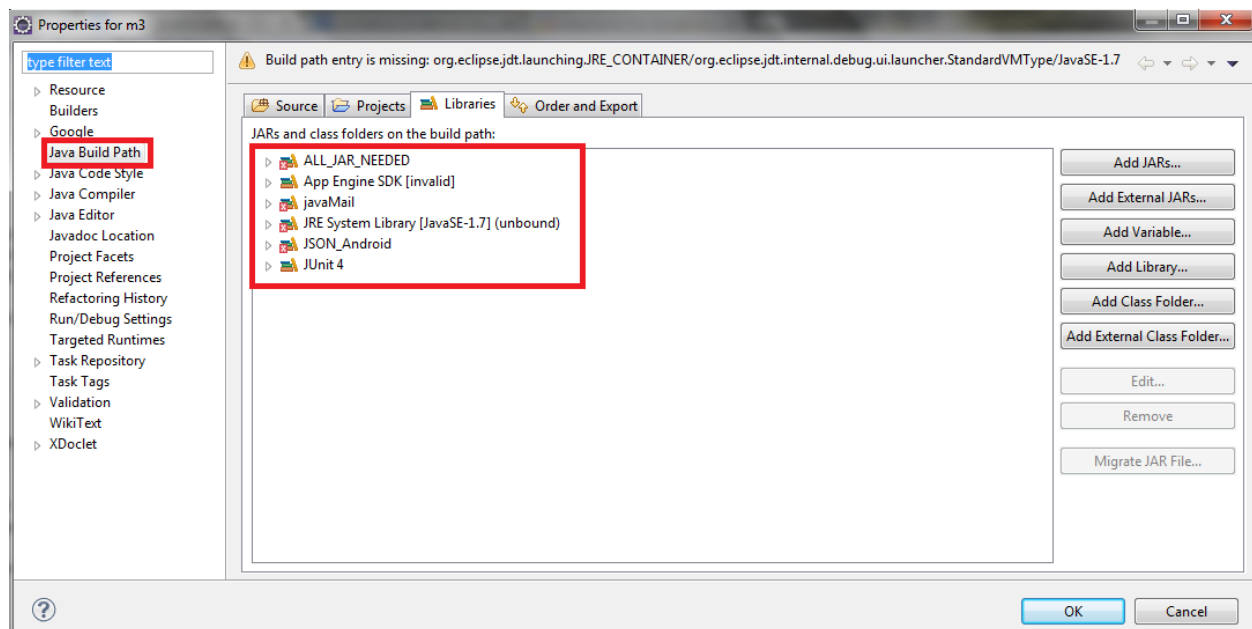
- 1) Copy/Paste the JAR directory that we provided in the M3 project
- 2) or Modify the Build Path

Modify the build path:



**Figure 14. Modify the build paths to reference the JARs required**

Modify the build path of all jar libraries:



**Figure 15. Add the JARs required**

## 2. Java Build Path Problems

Add all jars required, that are available in the JAR directory.

## II. (Optional) Install Jena inside Eclipse

In case you encounter some issues, you can follow the following tutorial:

<http://www.iandickinson.me.uk/articles/jena-eclipse-helloworld/>

## III. Run M3 on localhost

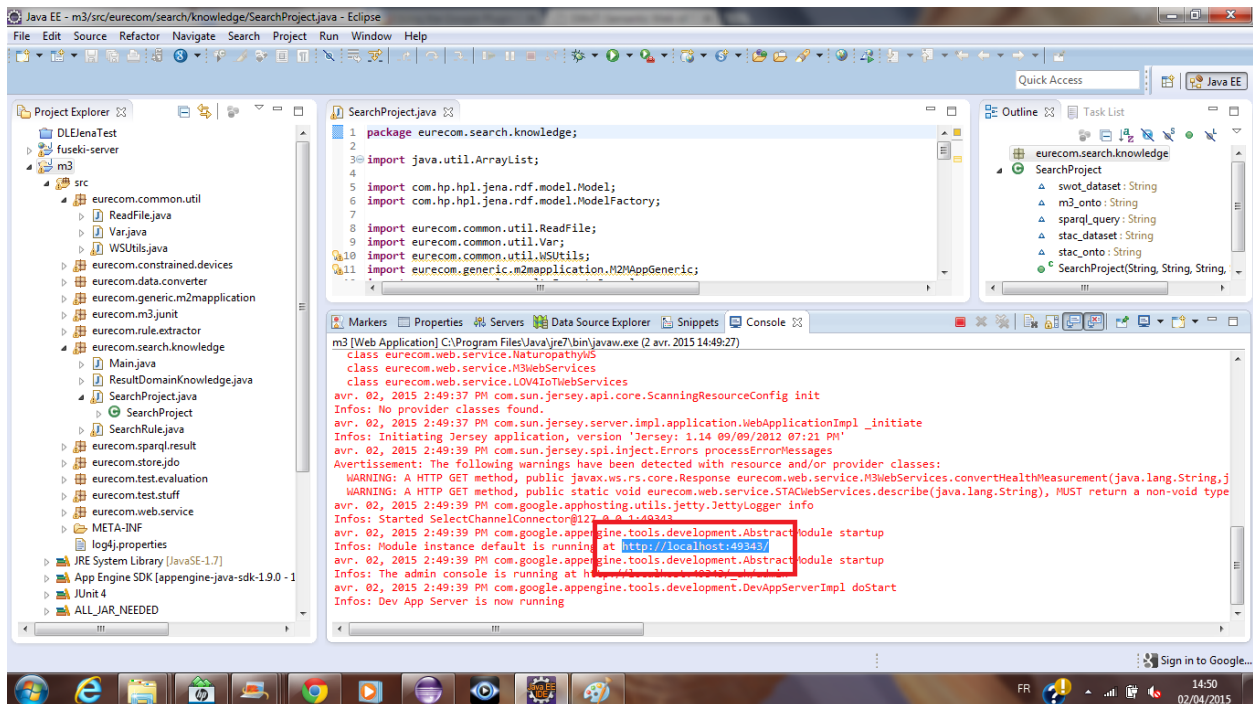
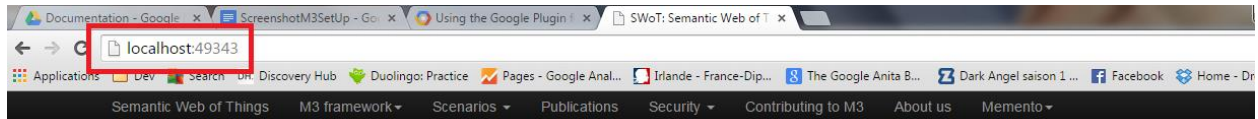


Figure 16. Get the localhost address

Copy paste the URL in red with the localhost address on the Browser (e.g., Chrome).



*Making smart discussions between things*

Machine-to-Machine Measurement (M3) is a framework to semantically annotate and easily interpret [Internet of Things \(IoT\)](#) data. M3 enables designing interoperable domain-specific or cross-domain [Semantic Web of Things \(SWoT\)](#) applications. M3 is composed of the following components:



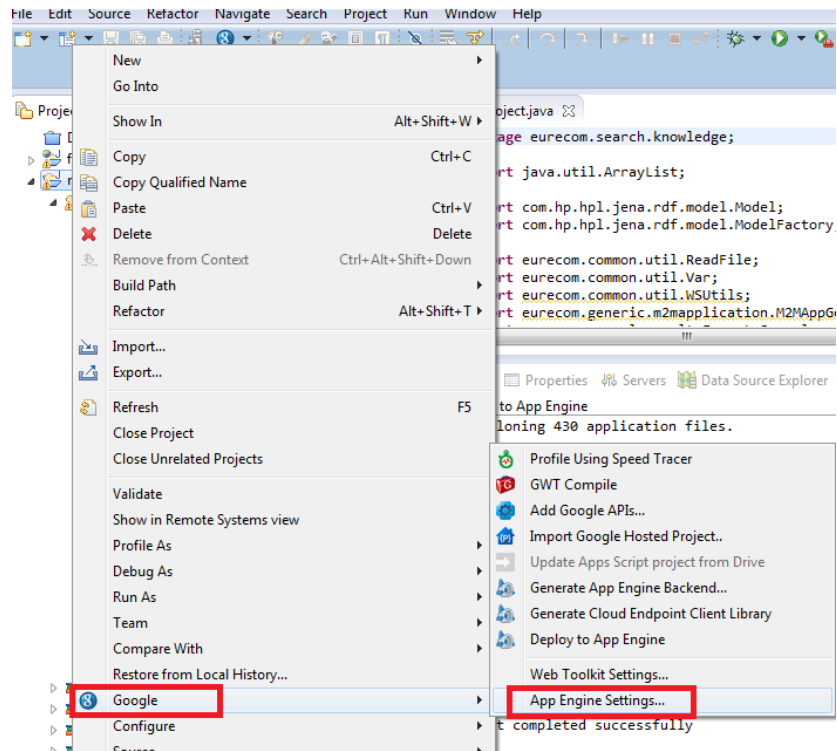
**Figure 17. M3 is running on localhost, it works!**

## Part III : Deploying M3 on the Web

Right click on the M3 project on Eclipse:

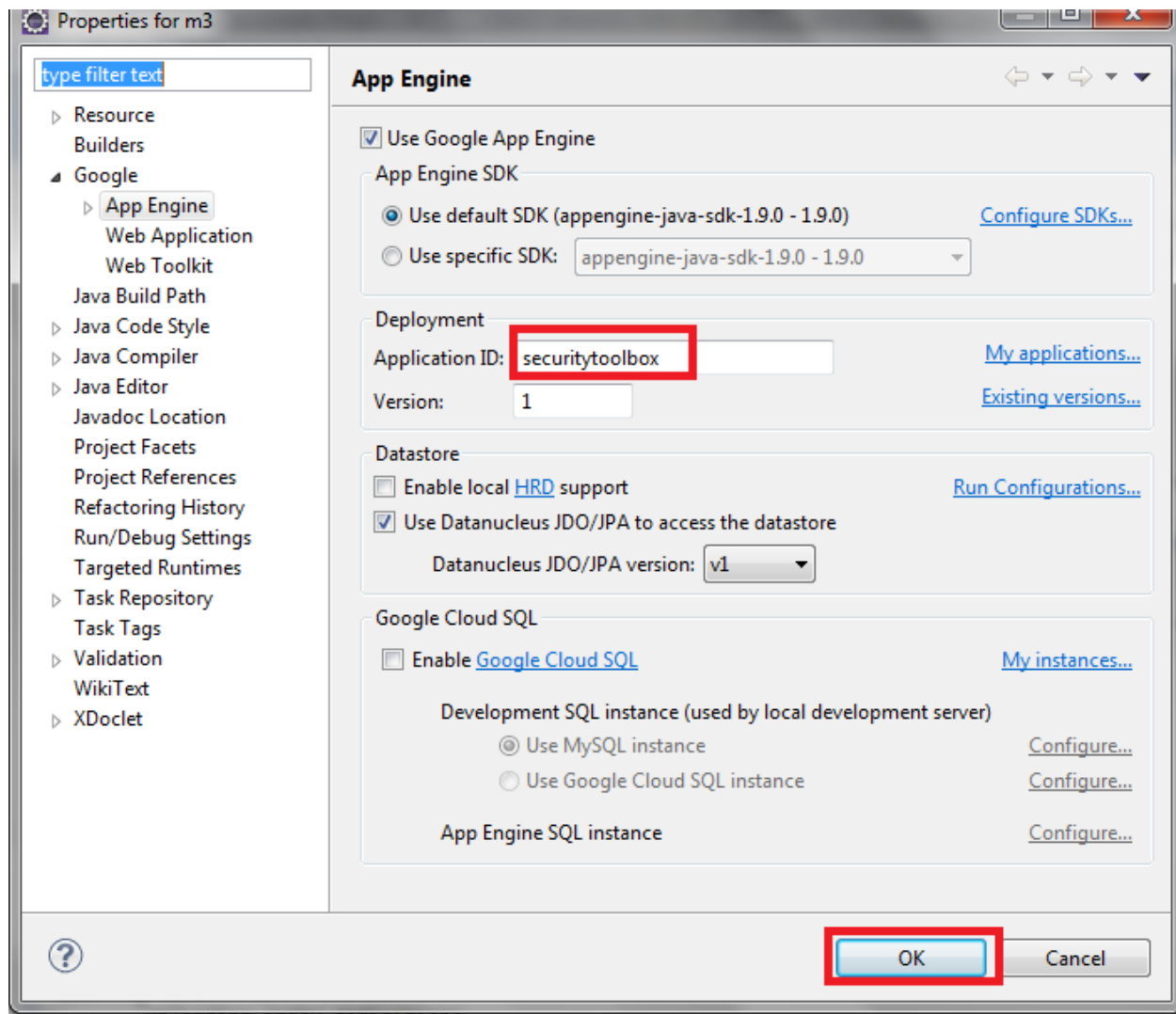
- ➔ Select Google
- ➔ Select App Engine Settings





**Figure 18. Check Google App Settings**

- ➔ You will have the following screen. In the Application ID enter **securitytoolbox**, this is the web site to try it.
- ➔ Click OK



**Figure 19. Indicate the name of the application where it will be deployed on the Web**

Then right click on M3 again:

- ➔ Select Google
- ➔ Select Deploy to App Engine

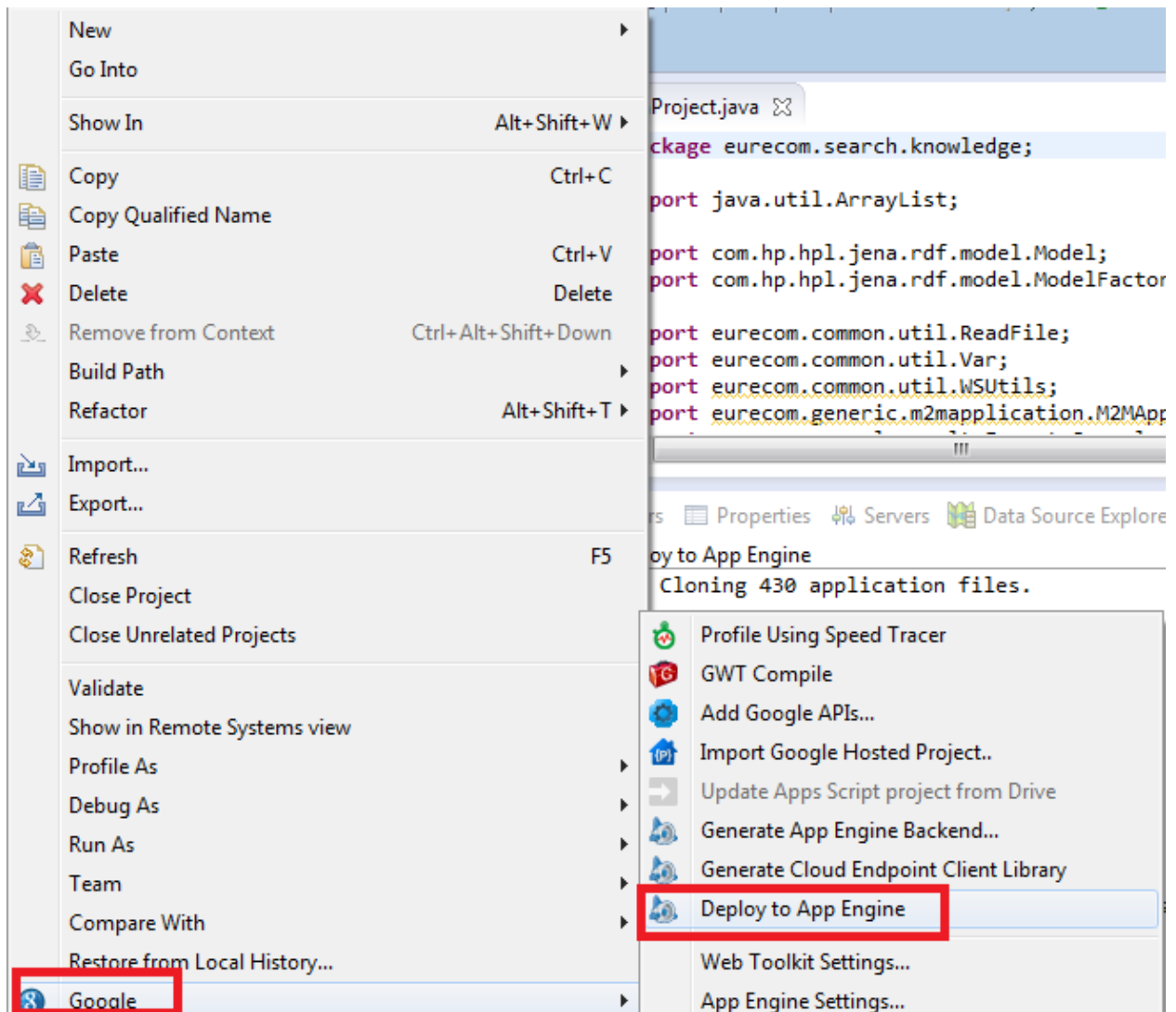


Figure 20. Deploy the M3 project on the web

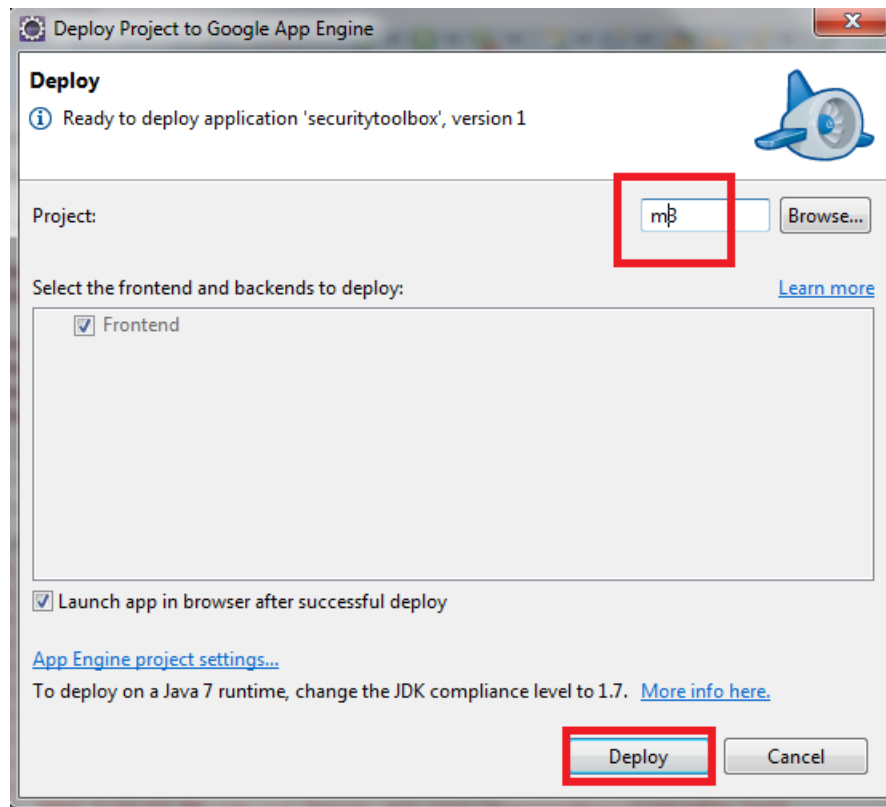


Figure 21. Confirm that you deploy the M3 project on the web

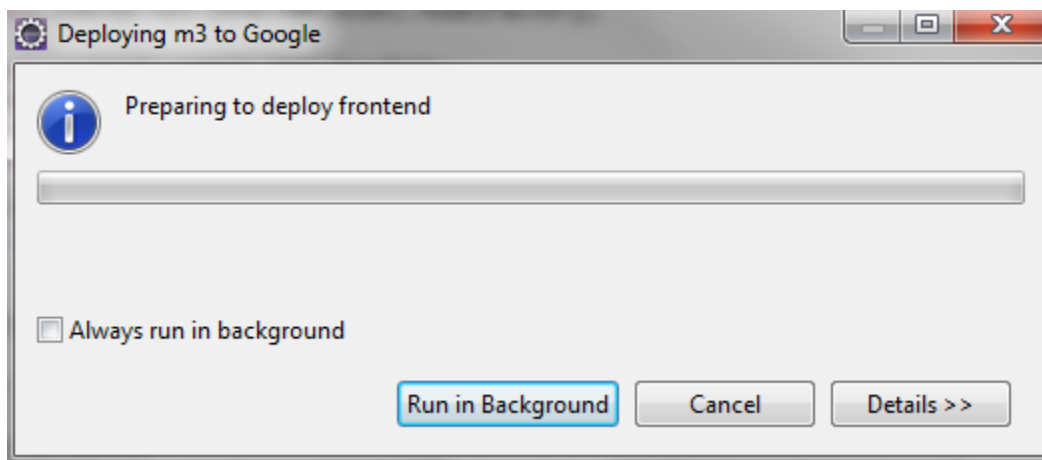


Figure 22. Wait, M3 is deploying on the web



Google will ask you to authenticate, you need to be authenticated on the project.

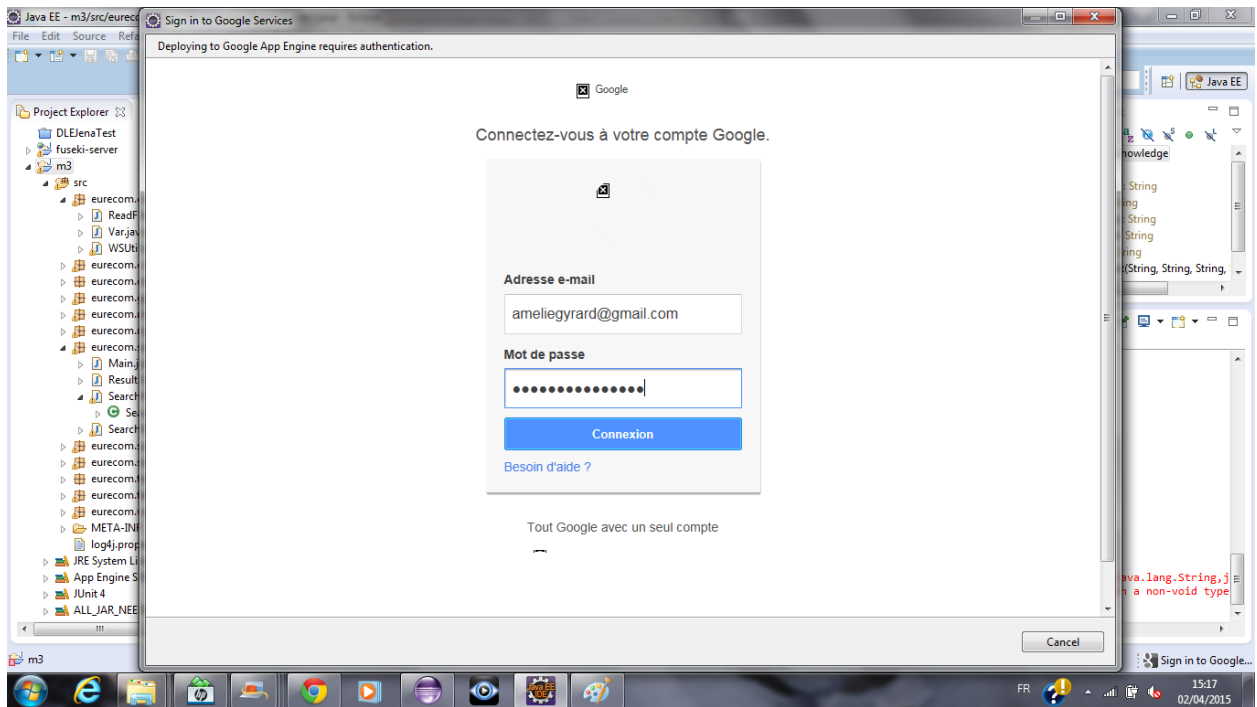


Figure 23. Authenticate yourself with Google App Engine

➔ Click on the button Accept:

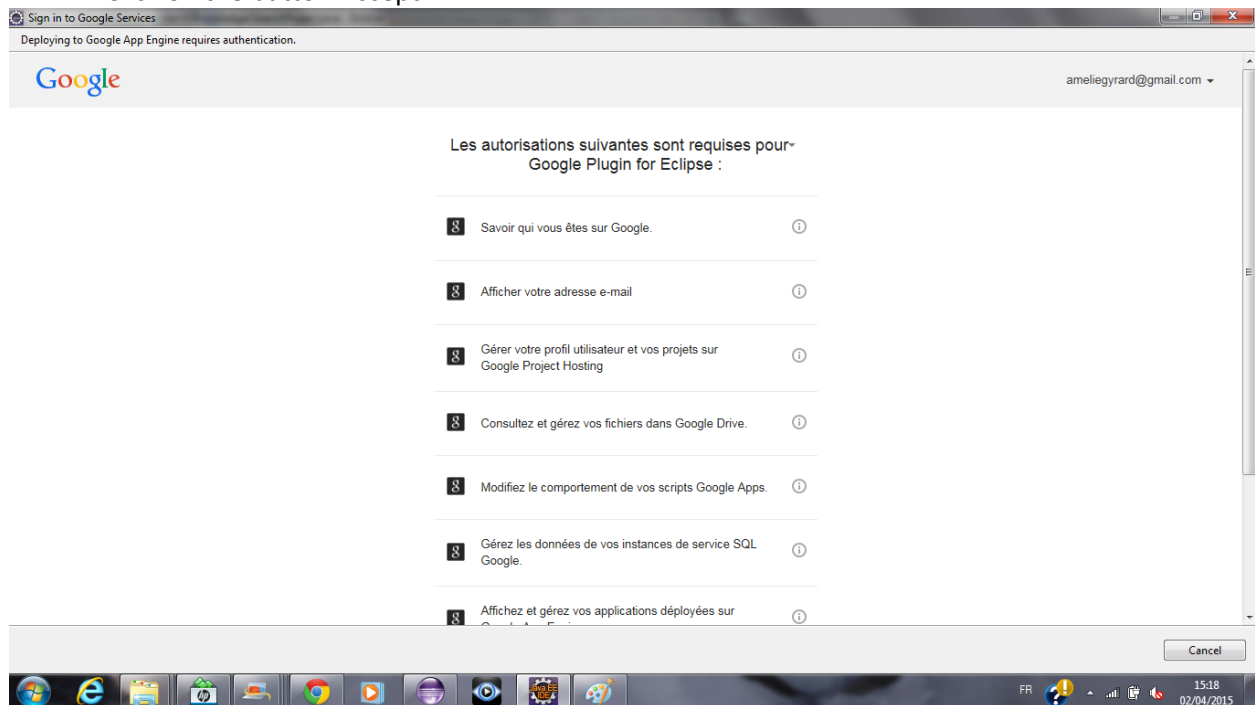


Figure 24. Accept the authentication on Google App Engine

Check that it works:



Machine-to-Machine Measurement (M3) is a framework to semantically annotate and easily interpret [Internet of Things \(IoT\)](#) data. M3 enables designing interoperable domain-specific or cross-domain [Semantic Web of Things \(SWoT\)](#) applications. M3 is composed of the following components:

				
Generating SWoT applications	Reusing domain knowledge	Interpreting IoT data	M3 Interoperable IoT Data & Domain Knowledge	Securing IoT applications

Figure 25. M3 deployed on the web

## I. Creating an application on app engine

We can deploy on the semantic-web-of-things.appspot.com web site.

You will have to find another name (check availability button)

← <https://appengine.google.com>

Useful NewsInfo SWE SSN Conf Accueil - Events Project Job SW Dev Sensor SWOT Latex dataset PaperNotAvailable

Google app engine | [My Account](#) | [Help](#) | [Sign out](#)

## My Applications

« Prev 20 1-5 of 5 Next 20 »

Application	Title	Storage Scheme	Status
<a href="#">linkedopenrules</a>	linkedopenrules	High Replication	None Deployed
<a href="#">securitytoolbox</a>	securitytoolbox	High Replication	Running
<a href="#">semantic-web-of-things</a>	sensormeasurement	High Replication	Running
<a href="#">semanticdream</a>	semanticdream	High Replication	None Deployed
<a href="#">sensormeasurement</a>	sensormeasurement	High Replication	Running
<a href="#">Create Application</a>			

You have 20 applications remaining. « Prev 20 1-5 of 5 Next 20 »

© 2015 Google | [Terms of Service](#) | [Privacy Policy](#) | [Blog](#) | [Discussion Forums](#) | [Project](#) | [Docs](#)

← <https://appengine.google.com/start/createapp>

Google app engine [ameliegyrard@gmail.com](#) | [My Account](#) | [Help](#) | [Sign out](#)

## Create an Application

You have 20 applications remaining.

**Application Identifier:**

semantic-web-of-things .appspot.com

[Check Availability](#)

All Google account names and certain offensive or trademarked names may not be used as Application Identifiers.

You can map this application to your own domain later. [Learn more](#)

**Application Title:**

Semantic Web of Things

Displayed when users access your application.

**Authentication Options (Advanced):** [Learn more](#)

Google App Engine provides an API for authenticating your users, including Google Accounts, Google Apps, and OpenID. If you choose to use this feature for some parts of your site, you'll need to specify now what type of users can sign in to your application:

☒ **Open to all Google Accounts users (default)**

If your application uses authentication, anyone with a valid Google Account may sign in.

☐ **Restricted to the following Google Apps domain:**

e.g. foo.com

If your application uses authentication, only members of this Google Apps domain may sign in. If your organization uses Google Apps, use this option to create an application (e.g. an HR tracking tool) that is only accessible to accounts on your Google Apps domain. This option cannot be changed once it has been set.

☐ **(Experimental) Open to all users with an OpenID Provider**

If your application uses authentication, anyone who has an account with an OpenID Provider may sign in.

[Create Application](#)

[Cancel](#)

## II. Error App engine when deploying

Error when you want to deploy your application on app engine:

Another transaction by user ... is already in progress for this app and major version

```
E:\workspace\m3> "E:\eclipse-jee-kepler-SR1-win32-x86_64\eclipse\plugins\com.google.appengine.eclipse.sdkbundle_1.9.0\appengine-java-sdk-1.9.0\bin\appcfg" rollback war
Reading application configuration data...
Sep 29, 2014 9:23:57 AM com.google.apphosting.utils.config.AppEngineWebXmlReader readAppEngineWebXml
INFO: Successfully processed war\WEB-INF/appengine-web.xml
Sep 29, 2014 9:23:57 AM com.google.apphosting.utils.config.AbstractConfigXmlReader readConfigXml
INFO: Successfully processed war\WEB-INF/web.xml
Sep 29, 2014 9:23:57 AM com.google.apphosting.utils.config.IndexesXmlReader readConfigXml
INFO: Successfully processed war\WEB-INF\appengine-generated\datastore-indexes-auto.xml

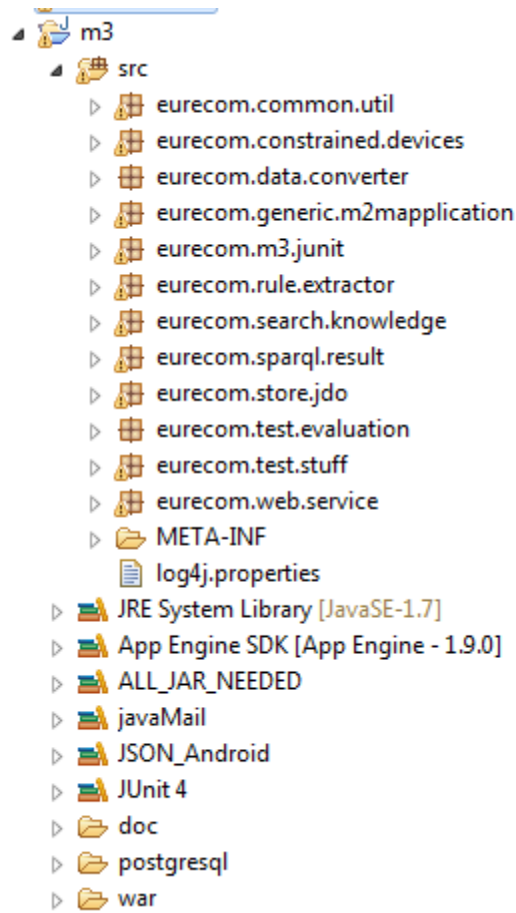
Beginning interaction for module default...
0% Rolling back the update.
Email: ameliiegyrard@gmail.com
Password for ameliiegyrard@gmail.com:
Success.
Cleaning up temporary files for module default...

E:\workspace\m3> "E:\eclipse-jee-kepler-SR1-win32-x86_64\eclipse\plugins\com.google.appengine.eclipse.sdkbundle_1.9.0\appengine-java-sdk-1.9.0\bin\appcfg" rollback war
wa./appcfg.sh rollback /home/workspace/vchat/war
```

## Part IV : Understanding the M3 project

### I. M3 framework overview



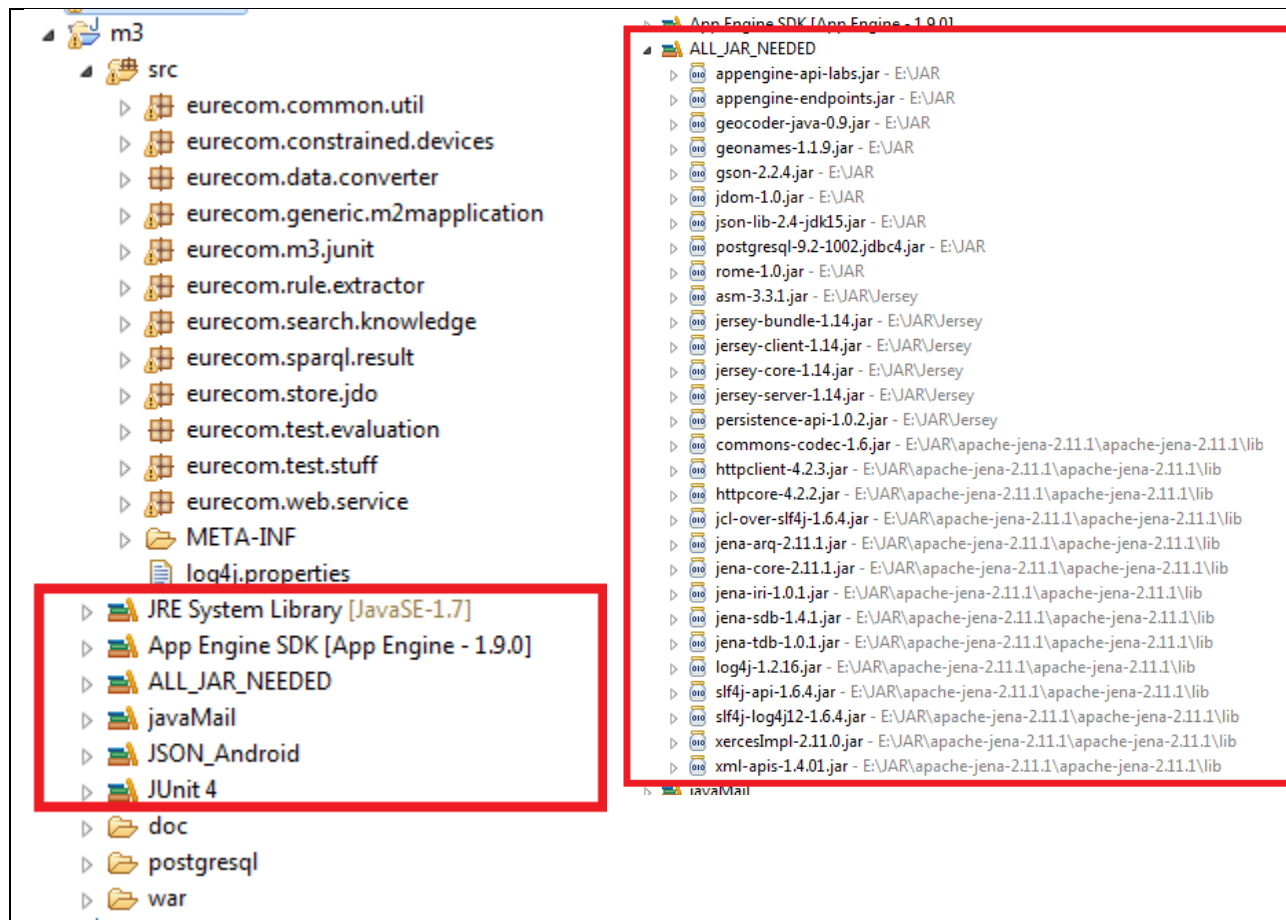


- WAR/CSS
- WAR/dataset
- WAR/html
- WAR/images
- WAR/javascript
- WAR/ont: all ontologies
- WAR/publication
- WAR/RULES: ruleM3NewType.txt is used in the M3 Converter to add the right sensor, measurement, unit or domain type
  - LinkedOpenRules\*.txt: M3 rules implemented according to the Jena syntax compliant with the Jena reasoned, they are classified by domains (e.g., weather,, environment, health, Home)
  - ruleM3NewType.txt used by the M3 converter to convert SenML<sup>1</sup> sensor data in RDF m3 sensor data
  - WAR/RULES/OTHER: files are from domain experts and just shared in the ontology.html web page
- WAR/SPARQL all SPARQL queries
- WAR/WEB-INF

---

<sup>1</sup> <http://www.ietf.org/archive/id/draft-jennings-senml-10.txt>

## II. Libraries (Jars) required



To run the M3 project, you will need the following libraries:

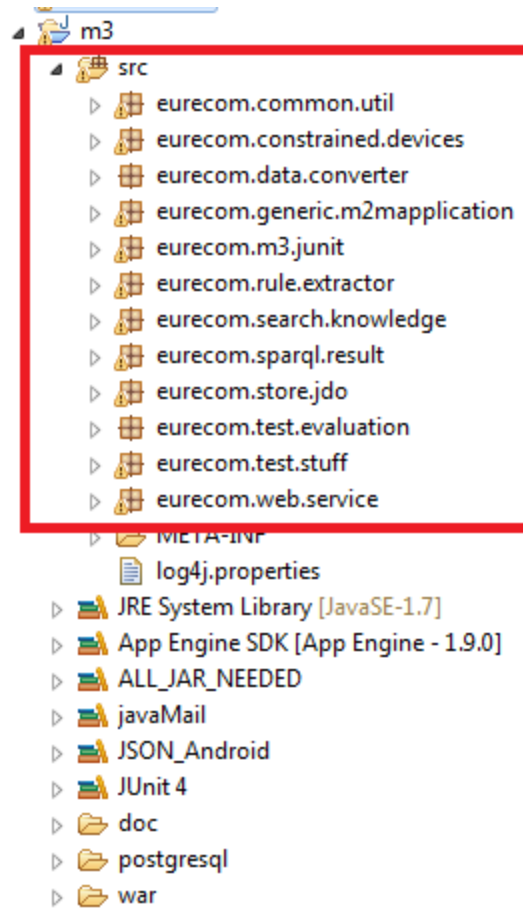
- **JRE System library**, should be the Java 1.7
- **App engine SDK** [App engine - 1.9.0] used to host our project online.
- **ALL\_JAR\_NEEDED** is comprised of
  - **Jersey 1.14** to develop RESTful web services in JAVA. These web services are then called in HTML web pages through AJAX and Javascript.
  - **Jena 2.11**, a framework uses to build semantic web applications. In the M3 project, more specifically, semantic web of things applications. We use the jena reasoner, the sparql engine, etc.
  - **Geonames**<sup>2</sup> used in the restaurant scenario. From a location (longitude and latitude), we can get additional information such as city, country, etc.
- **javaMail** to send emails, used for the LOV4IoT tool
- **JSON\_Android** used to convert sensor data, semantically annotate them and return it in JSON
- **JUnit4** to develop java unit tests

<sup>2</sup> <http://www.geonames.org/>

### III. Descriptions of packages

The M3 project is comprised of 12 packages:

- **eurecom.common.util**: some useful functions such as read a file, query a web service, etc.
- **eurecom.constrained.devices**: code tested to adapt the M3 framework to Android powered devices. For instance, we used JSON instead of XML, etc.
- **eurecom.data.converter**: to semantically annotate SenML data with the M3 nomenclature
- **eurecom.generic.m2mapplication**: to try to find a way to develop generic applications using the same reasoning process.
- **eurecom.m3.junit**: some Junit tests to test the M3 framework. Otherwise, the M3 framework has been tested manually with demos.
- **eurecom.rule.extractor** (work in progress): some tests to extract rules from the ontology catalogue called LOV4IoT.
- **eurecom.search.knowledge**: used to look for all projects using a specific sensor
- **eurecom.sparql.result**: to execute the SPARQL query and get the results in different formats.
- **eurecom.store.jdo**: to store data in the Google datastore
- **eurecom.test.evaluation**: used to evaluate the software performances
- **eurecom.test.stuff**: to test new technologies, new tools to integrate, etc.
- **eurecom.web.service**: all web services implemented in the M3 framework



## 1. eurecom.common.util

Var.java: Java class where all the static variable are set. Add to this there are sorted by what they are used for. Furthermore, all the URL are saved here.

WSUtils.java: Class behaviour to read ontology, data measurement, and SPARQL results. Moreover it implements the behaviour to execute a web service remotely. This class contains noticeable and useful functions:

- `convertXmlToJson(<string>)` and `convertJsonToXml(<string>)` : do like their name, before to use this object there is a pre-processing, the JSON (or the Xml) have to be send to the function in a String type.
- `queryWebService(<string> url, enum)` : execute remotely a web service, the first argument is the URL of the web service, the second is an enumeration (JSON or Xml), it depends if you want to use an Xml or JSON web service. Return the result of the request (<string>)
- `readXMLFile(<string> filename)` : read an xml file and transform the data in to RDF data, return the data object of type Measurements (see package `eurecom.data.convert`).
- `readFile(optional <model> (from jena library), <string> filename)` : if model is send to the function, read ontologies otherwise read a file).

## 2. eurecom.constrained.devices

To query sensor data provided by the raspberry pi

Java-json.jar used in eurecom.constrained.devices package

java json android parser for sensor data provided by the raspberry pi

## 3. eurecom.data.converter

- Domain.java: Class which transforms XML data to RDF data. Have a public ArrayList containing the measurements, getter and setter of the nameZone.
- ConvertSensorDataToM3.java: Behaviour of the conversion of java object (Measurements) into RDF data and of the other way. Can create also some instance object into RDF (type of object: Measurement, FeatureOfInterest, and Sensor).
- Measurement.java && Measurements.java: Objects measurements, describe the object that is used to transform Xml data into RDF data. Mostly composed by variables and their getter/setter.
- *Note:* A Domain has an ArrayList of Measurements (with an S) and Measurements has an ArrayList of Measurement. Where Measurement is just a data (30 °C for instance), Measurements is a bunch of data (all the data from one sensors) and FeatureOfInterest could be the domain (Temperature).

## 4. eurecom.generic.m2mapplication

Coordinates.java: Object to describe a coordinate on earth. Composed by a longitude and a latitude.

LOVClient.java: One function executeSparql(<string> sparql) return InputStream.

VariableSparql.java: Object that describe variable used in a sparql request.

M2MAppGeneric.java: Main behaviour of the application. Can load the dataset of the ontologies provide by the application. Execute the SPARQL queries and linked open rules.

Main function implemented:

- load<ontologyName>Dataset(): add the ontology in the model. There is one load for each kind of ontology.
- executeLinkedOpenRulesAndSparqlQuery(<string> SparqlQuery, ArrayList of variables of the query): prepare and call the function that can execute the SPARQL query.

M2MAppResto.java / M2MAppNaturopathy.java / M2MAppRecipe.java / M2MAppStac.java: These are an extension of M2MAppGeneric which have the same behaviour but with additional specific function that execute almost implemented SPARQL queries. A future goal is to make a real generic class which will make these classes depreciated.

Place.java: Same as Coordinates, object that describe a place in the world. However there is no getter or setter.

WlboxApi.java: Class that allows to get the hierarchy classes of a certain model. Could be depreciated.

## 5. eurecom.sparql.result

ExecuteSparqlGeneric.java: Java class that implements all type of SPARQL queries (delete or find some labels, replace/update). Concerning queries, you can specify (using a function) how to store the value of the query.

ExecuteSparql.java: Same as ExecuteSparqlGeneric.java but it uses to define one and only one SPARQL request, the object define must not execute other SPARQL request.

ExecuteSparqlHealth.java / ExecuteSparqlRecipe.java / ExecuteSparqlRestaurant.java /

ExecuteSparqlRecipeNaturopathy.java: Implement the SPARQL request function associated to an ontology, should be deleted and use a more generic function in ExecuteSparqlGeneric.java instead.

SparqlResultGeneric.java, SparqlResultRecipe.java, SparqlResultRecipeNaturopathy.java,

SparqlResultRestaurant.java: Describe the object result of a query, composed of variables, getters and setters.

VariableSparql.java: Object that describe variable used in a SPARQL request.

## 6. eurecom.store.jdo

GenericJDO: This class manipulates the measurements object get, create or delete it.

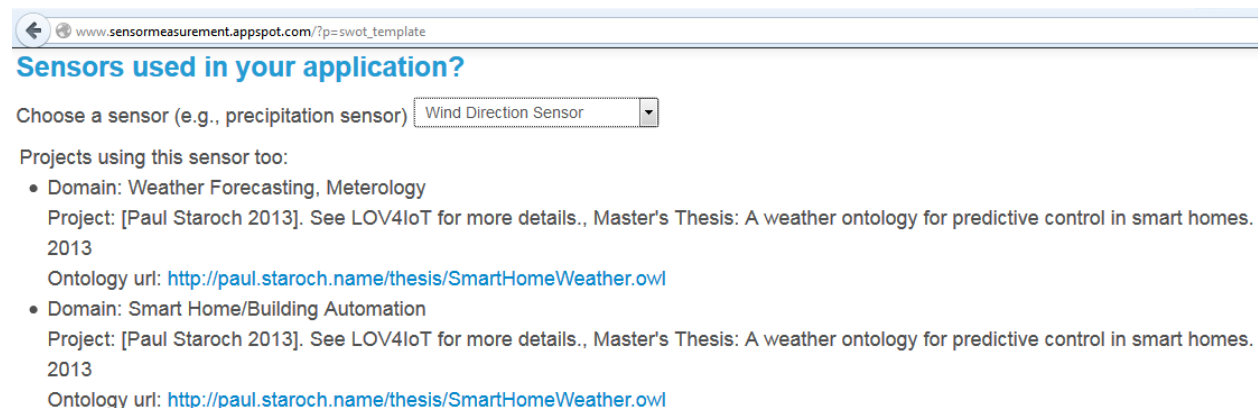
Util.java: Composed of useful function to manipulate entities and especially to get JSON strings.

JDO Java Data object, datastore compatible with google to store data (e.g., M3 rdf sensor data)

## 7. eurecom.search.knowledge

The code is used in this web page<sup>3</sup> through web services.

According to a specific sensor given in the drop-down list, we can retrieve all projects using this sensor.



www.sensormeasurement.appspot.com/?p=swot\_template

### Sensors used in your application?

Choose a sensor (e.g., precipitation sensor)

Projects using this sensor too:

- Domain: Weather Forecasting, Meterology  
Project: [Paul Staroch 2013]. See LOV4IoT for more details., Master's Thesis: A weather ontology for predictive control in smart homes. 2013  
Ontology url: <http://paul.staroch.name/thesis/SmartHomeWeather.owl>
- Domain: Smart Home/Building Automation  
Project: [Paul Staroch 2013]. See LOV4IoT for more details., Master's Thesis: A weather ontology for predictive control in smart homes. 2013  
Ontology url: <http://paul.staroch.name/thesis/SmartHomeWeather.owl>

## 8. eurecom.web.service

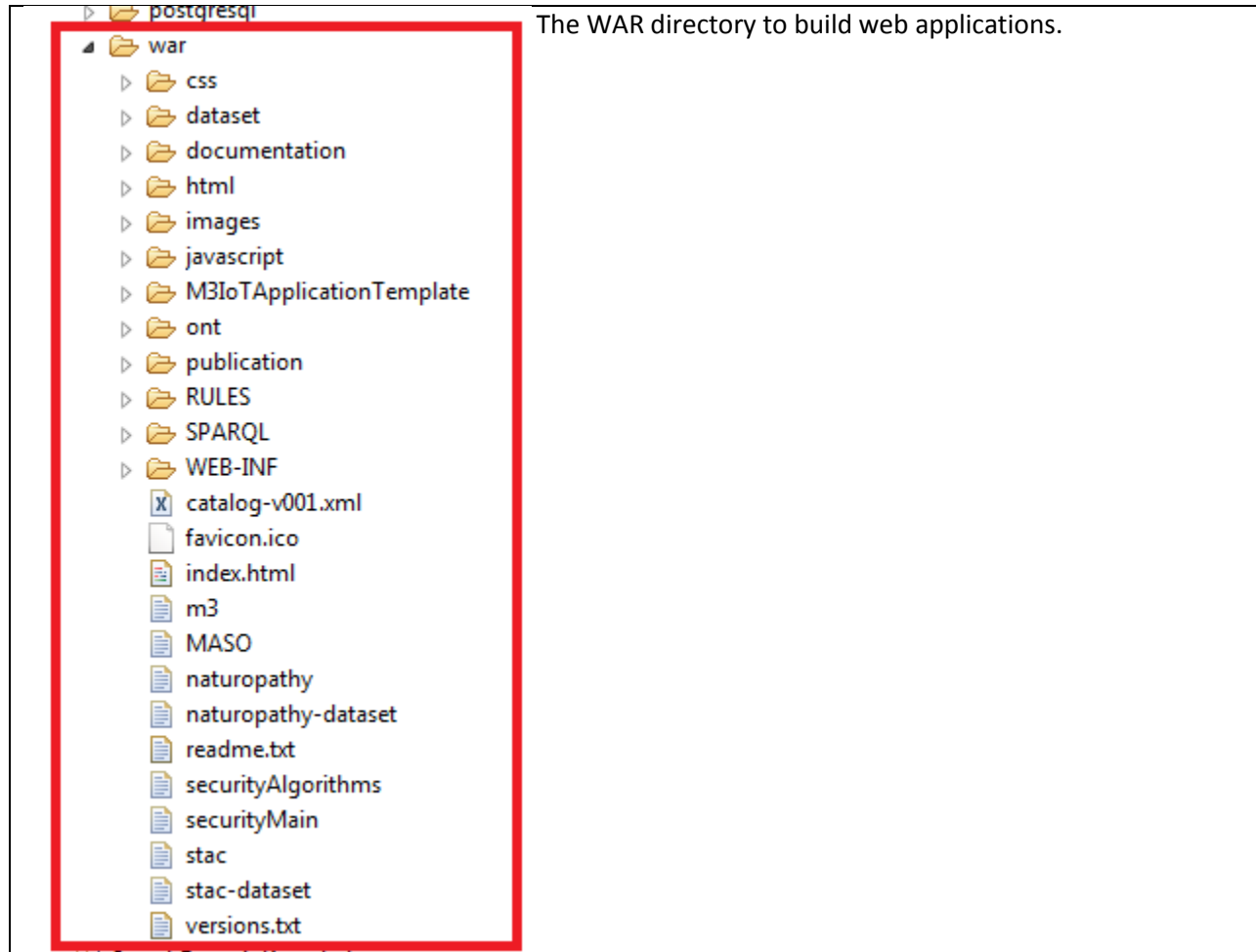
All the classes here are Web Services. There is one class for each type of ontology. Before every method, there are some important parameters:

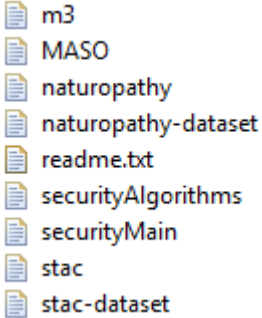


- @GET: Indicate the kind of protocol used (here is Get by opposition of POST for instance).
- @Path("String"): This is the URL where the method is called. This URL is relative.

<sup>3</sup> [http://www.sensormeasurement.appspot.com/?p=swot\\_template](http://www.sensormeasurement.appspot.com/?p=swot_template)

- @Consumes(MediaType): What kind of string this method takes.
- @Produces(MediaType): What kind of string this methods produces (It is often a JSON string or Xml).

## IV. The WAR directory



	<div data-bbox="548 226 727 394"></div> <div data-bbox="748 226 1438 405"><p>At the beginning of the project, we had the following ontologies and datasets directly in the WAR file. We keep these ontologies and datasets since they have been referenced in research articles. We do not remove them to avoid 'error 404: page not found'.</p></div> <div data-bbox="540 405 1312 478"><p>But now, all modifications of ontologies and datasets are in the corresponding directory.</p></div> <div data-bbox="548 485 764 646"></div> <div data-bbox="795 548 1409 621"><p><b>TO DO:</b> A solution would be to use the PURL tool<sup>4</sup> to avoid such issues!</p></div>
---	---

## 1. WAR/CSS

All CSS file for the web application user interface.

## 2. WAR/dataset

All datasets such as sensor data, sensor data semantically annotated with M3, domain knowledge bases, etc.

## 3. WAR/documentation

All documentation to understand the project or use the tools.

## 4. WAR/html

All HTML web pages used for the user interface.

## 5. WAR/images

All images used in the projects.

## 6. WAR/javascript

The HTML web pages will send AJAX queries to web services and get the results.

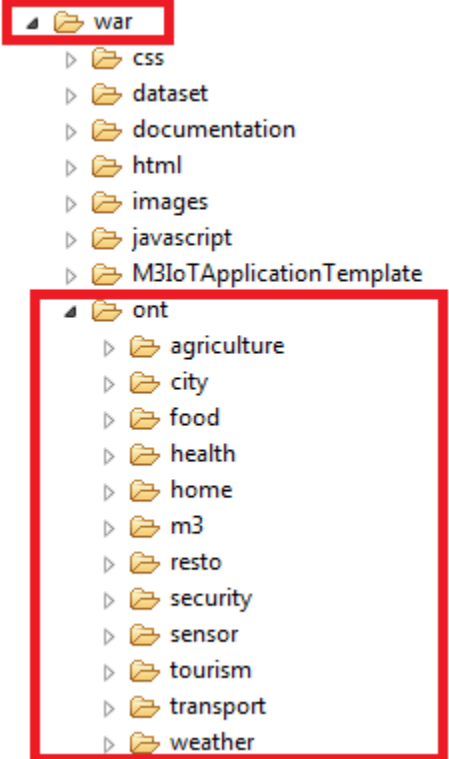
The result is parsed in JavaScript and displayed on HTML web pages.

---

<sup>4</sup> <https://purl.oclc.org/docs/index.html>.



## 7. WAR/ont

	<p>All new ontologies should be in this directory. They are classified by domains.</p> <p>Some old ontologies are still in WAR/WEB-INF/ontologies. The less interesting ones, that we did not move to the new directory.</p> <p>In WAR/ont/m3: you will find the M3 interoperable domain ontologies.</p>
--	--

## 8. WAR/publication

All publications related to this thesis to understand better the M3 project. You will find slides, research articles, participation to standards, etc.

## 9. WAR/RULES

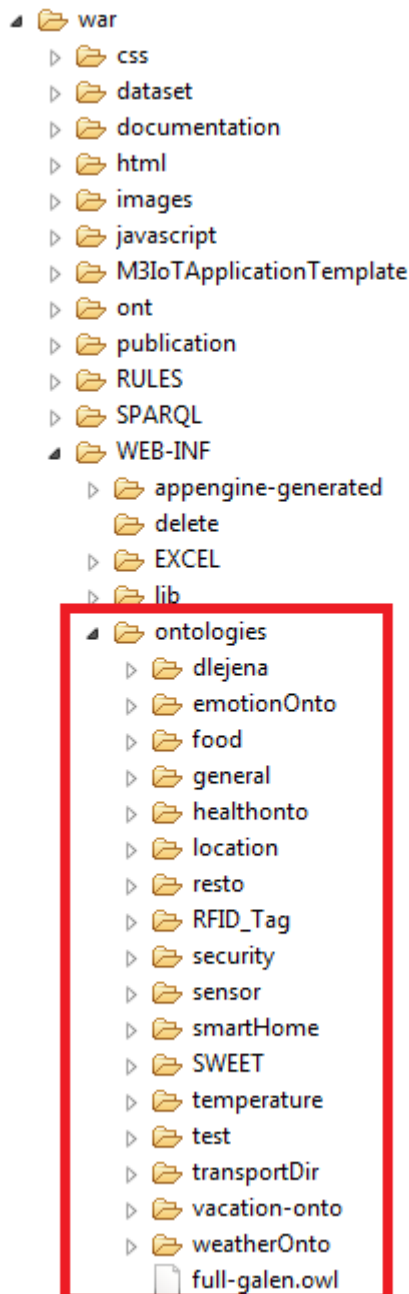
You will find in this directory all the interoperable rules that we designed.

This is the Sensor-based Linked Open Rules (S-LOR) tool. The SWoT generator has predefined-templates to build semantic-based IoT application. The templates will referenced these pre-defined set of rules classified by domains.

## 10. WAR/SPARQL

You will find in this directory all SPARQL queries.

## 11. WAR/WEB-INF/ontologies



Why do we have several directeroy related to ontologies?

Be careful: Ontologies and datasets duplicated  
We are agree this is a total mess! Some ontologies required to be shared online, but if we move all ontologies and datasets in the same place, we will have issues with path and namespace already defined in previous ontologies and referenced in the LOV4IoT dataset!

WAR/ont: we added this directory to host the ontology that we found online

WEB-INF/ontologies/: old repository with all ontologies that we found but not necessarily use them.



**Solve this issue with PURL<sup>5</sup> or URL redirection?**  
If if we migrate the ontologies in the other directory, we will have error page not found in some web

pages.

**TO DO:** delete WEB-INF/ ontology and merge it with WAR/ontologies and WAR/datasets.

**BIG ISSUE:** change namespaces everywhere (ontologies, datasets, rules, sparql) to be accessible online (URI deferencable)

**URL** already used in publications links.

<sup>5</sup> <https://purl.oclc.org/docs/index.html>

# Part V : Understanding M3 web services

There is also the documentation to use the web services if required<sup>6</sup>.

Root path web service: <http://www.sensormeasurement.appspot.com/>

In the package `eurecom.web.service`, you will find all web services, implemented in Java using the Jersey<sup>7</sup> implementation.

	All web services names ended by WS in Java class
---	--

## 1. APIJsnWS Java class (deprecated)

The web services provided by this Java class were focused on returning the result in JSON.

After, I found the solution to return the result either as JSON or as XML by adding:

```
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public static Response nameFunction( @QueryParam(value = "format") String format) {
    if (format.equals("xml")){
        resultSparql = req.selectResultAsXML(var);
    }
    else if (format.equals("json")){
        resultSparql = req.selectResultAsJson(var);
    }
    return Response
        // Set the status and Put your entity here.
        .ok(resultSparql)
        // Add the Content-Type header to tell Jersey which format it should marshal the entity into.
        .header(HttpHeaders.CONTENT_TYPE, "json".equals(format) ? MediaType.APPLICATION_JSON : MediaType.APPLICATION_XML)
        .build();
}
```

**Figure 26. Code example to return the result either as JSON or XML**

## 2. LOV4IoTWS Java class

All web services related to the Linked Open Vocabularies for Internet of Things (LOV4IoT) dataset<sup>8</sup> to automatically count the number of ontologies in this dataset (e.g., by domains, by ontology status, etc.)

<sup>6</sup> <http://www.sensormeasurement.appspot.com/documentation/M3APIDocumentation.pdf>

<sup>7</sup> <https://jersey.java.net/>

<sup>8</sup> <http://www.sensormeasurement.appspot.com/?p=ontologies>

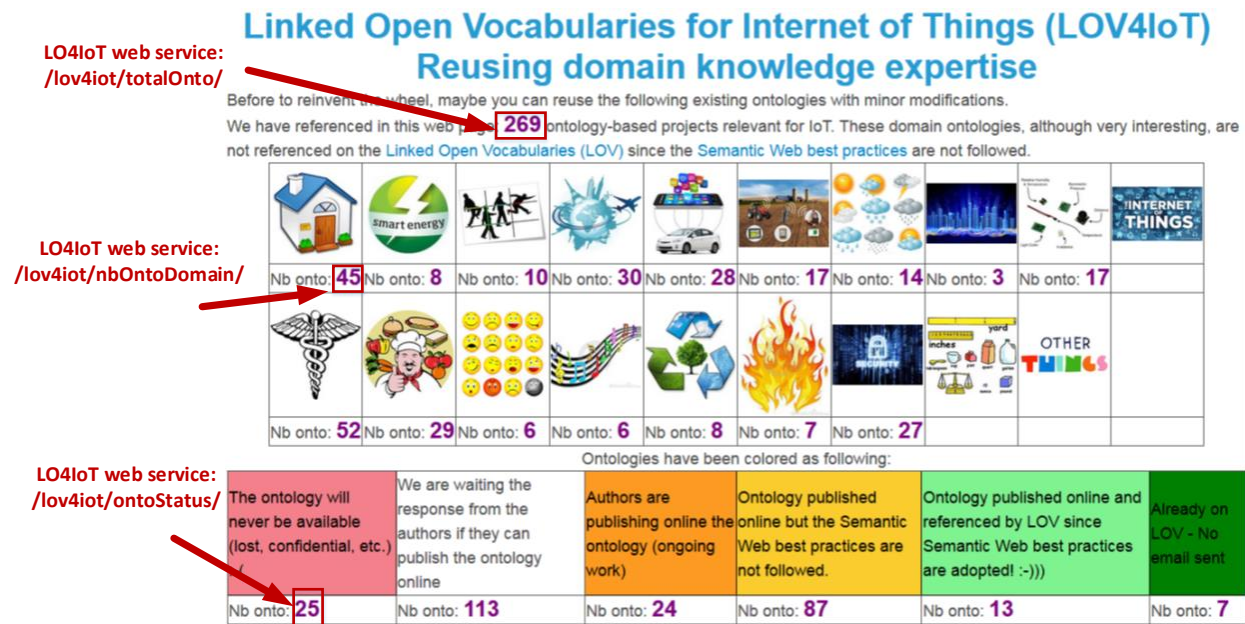


Figure 27. LOV4IoT web services

```
@GET
@Path("/totalOnto/")
@Produces(MediaType.APPLICATION_XML)
public Response getTotalNumberOntology() {
    //load the LOV4IoT dataset into the model
    Model model = ModelFactory.createDefaultModel();
    ReadFile.enrichJenaModelOntologyDataset(model, Var.LOV4IOT_DATASET_PATH);
    M2MAppGeneric m2mappli = new M2MAppGeneric(model);

    //SPARQL query
    ExecuteSparql sparqlQuery = new ExecuteSparql(model, Var.ROOT_SPARQL_LOV4IoT + "countTotalOntology.sparql");

    //no variable to replace in the SPARQL query
    ArrayList<VariableSparql> var = new ArrayList<VariableSparql>();
    String resultSparqlsenml = sparqlQuery.getResultAsXML(var);

    return Response.status(200).entity(resultSparqlsenml).build();
}
```

Figure 28. Example of the lov4iot/totalOnto: web service

### 3. M3JsonWS (deprecated)

The web services provided by this Java class were focused on returning the result in JSON.  
After, I found the solution to return the result either as JSON or as XML by adding:

These web services enable to query the M3 ontology to get:

- Get all M3 sensors (/json/m3/sensor)
- Get all M3 domains (/json/m3/featureOfInterest)
- Get all M3 measurement type (/json/m3/measurement)

### 4. M3WS

All web services related to the M3 nomenclature implemented in the ontology.  
Support new web services handling both XML and JSON format.

Should replace M3JsonWS and APIJsonWS Java classes:

- To semantically annotate sensor, IoT, M2M data (/m3/convert)
- Get all M3 sensors (/m3/subclassOf/sensor). This web service replaced M3JsonWS.
- Get all M3 domains (/m3/subclassOf/featureOfInterest). This web service replaced M3JsonWS.
- Get all M3 measurement type (/m3/subclassOf/measurement). This web service replaced M3JsonWS.

All web services related to the SWoT generator<sup>9</sup>:

- To look for templates (/m3/searchTemplate)
- To get the template (/m3/generateTemplate)
- To replace variables in the SPARQL query (/m3/getSparqlQuery)

The screenshot shows the web application interface for the Semantic Web of Things (SWoT) Generator. The browser address bar shows the URL: [www.sensormeasurement.appspot.com/?p=m3api](http://www.sensormeasurement.appspot.com/?p=m3api). The navigation bar includes links: Semantic Web of Things, M3 framework, Scenarios, Publications, Security, Contributing to M3, About us, and Memento. The main heading is "Semantic Web of Things (SWoT) Generator". Below the heading, a text description states: "The SWoT generator enables designing SWoT applications to interpret IoT data." The interface is divided into three steps:

- STEP 1: Search M3 Template**
  - 1. Choose a sensor (e.g., Light/Illuminance Sensor) [Wind Direction Sensor]
  - 2. Choose the domain where is deployed your sensor (e.g., Weather) [Agriculture, Smart farm]
  - 3. Search IoT Application Template => call web service: (/m3/searchTemplate) => call web service: (/m3/subclassOf/FeatureOfInterest)
- STEP 2: Choose M3 Template**
  - Choose an application template: [ ]
- STEP 3: Download M3 template**
  - Generate zip file => call web service: (/m3/generateTemplate)

Figure 29. M3 web services used in the SWOT generator

## 5. NaturopathyWS

All web services used for the restaurant scenario<sup>10</sup>:

- Suggest home remedies according to the body temperature (/naturopathy/sick)
- Deducing mood according to the external luminosity (/naturopathy/emotion\_luminosity/)
- Suggesting food according to the outside temperature (/naturopathy/seasonTemperatureFoodRecipe/)
- Deducing mood or diseases from:
  - heart beat (/naturopathy/emotion\_disease/HeartBeat)
  - skin conductance (/naturopathy/emotion\_disease/SkinConductance)
  - blood pressure (/naturopathy/emotion\_disease/BloodPressure)
- See the comments in the Java class for more web services

<sup>9</sup> <http://www.sensormeasurement.appspot.com/?p=m3api>

<sup>10</sup> <http://www.sensormeasurement.appspot.com/?p=naturopathy>

www.sensormeasurement.appspot.com/?p=naturopathy

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## Scenario Naturopathy: Suggest ingredients or recipes according to the weather, diseases, diets and emotions

### Suggesting home remedies according to body temperature

1. This scenario is based on: [M3 RDF health data](#)
2. M2M Aggregation Gateway (Convert Health Measurements into Semantic Data):
3. We deduce that the temperature corresponds to the body temperature.
4. We deduce that the person is sick.
5. We propose all fruits/vegetables according to this disease.
6. M2M Application: Temperature => Cold => Food: (Wait 10 seconds!)  => call web service /naturopathy/sick

**Figure 30. Naturopathy web services used in the naturopathy scenario**

## 6. RestaurantWS

All web services used for the restaurant scenario<sup>11</sup>.

See Java Class comments.

This scenario does not work exactly in the same way that last scenarios (naturopathy, transport, tourism). It was our second scenario who helps us to design the semantic engine, to reuse the domain knowledge, etc.

## 7. STACWS

All web services used for security application<sup>12</sup>, called STAC (Security Toolbox: Attacks & Countermeasures):

- Get all technologies referenced in the STAC dataset (stac/subclassOf/Technology)
- Get all attacks related to a specific technology (e.g., stac/attack/SensorTechnology)
- Get all features related to a specific security mechanism (e.g., stac/hasFeature/SSH)

<sup>11</sup> <http://www.sensormeasurement.appspot.com/?p=restaurant>

<sup>12</sup> <http://www.sensormeasurement.appspot.com/?p=stac>

- Get all security mechanisms related to a specific technology (e.g., stac/techno/SensorTechnology)
- See Java class for all web services and their comments

The name of the technology should be referenced in the STAC ontology<sup>13</sup>!

www.sensormeasurement.appspot.com/?p=stac

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## STAC (Security Toolbox: Attack & Countermeasure)

### Technologies used in your application?

1. Choose a technology (e.g., WiFi Technology)  => call web service: /stac/subclassOf/Technology
2. Attacks related to this technology:  => call web service: /stac/attack/SensorTechnology
3. Wait (10 seconds!)
4. A tooltip is displayed for more information about a specific security mechanism
5. Click on a security mechanism (e.g., WPA2): => call web service: /stac/techno/SensorTechnology
6. Advantages and weaknesses are displayed (Feature):
7. Security properties satisfied are displayed:

Figure 31. STAC web services

## 8. SWOTWS

- Web service for the M3 converter (/swot/convert\_senml\_to\_rdf)
- Web service to get all rules associated to a specific sensor (/swot/convert\_senml\_to\_rdf)

www.sensormeasurement.appspot.com/?p=senml\_converter

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## M3 Converter (SenML to RDF)

### Copy/paste your SenML/XML data

Copy/paste your SenML/XML sensor data here (need to be surrounded by zone balise):

```
<zone name="health">
<senml
bn="urn:body:uuid:c68ad78b-09eb-4303-ae3c-d5d23149ee96">
<e n="blood pressure" t="0"
u="Pa" v="56"></e>
<e n="cholesterol" t="0"
u="g/L" v="5"></e>
<e n="heartbeat" t="0"
u="beet/m" v="155"></e>
```

=> call web service: /swot/convert\_senml\_to\_rdf

Figure 32. Web service to convert senML data to RDF according to the M3 nomenclature

<sup>13</sup> securitytoolbox.appspot.com/stac#



## Sensor-based Linked Open Rules (S-LOR)

### Sensors used in your application?

=> /m3/subclassOf/?nameClass=Sensor&format=xml

Choose a sensor (e.g., precipitation sensor)

Wind Direction Sensor

Rules using this sensor: => /swot/rule/WindDirectionSensor

- Rule: WestWind, IF m3:WindDirection greaterThan 225 m3:DegreeAngle AND lessThan 315 m3:DegreeAngle THEN WestWind  
Linked Open Rules URL: <http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt>
- Rule: SouthWind, IF m3:WindDirection greaterThan 135 m3:DegreeAngle AND lessThan 225 m3:DegreeAngle THEN SouthWind  
Linked Open Rules URL: <http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt>
- Rule: EastWind, IF m3:WindDirection greaterThan 45 m3:DegreeAngle AND lessThan 135 m3:DegreeAngle THEN EastWind  
Linked Open Rules URL: <http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt>
- Rule: NorthWind, IF m3:WindDirection (greaterThan 0 m3:DegreeAngle AND lessThan 45 m3:DegreeAngle) OR (greaterThan 315 m3:DegreeAngle AND lessThan 360 m3:DegreeAngle) THEN NorthWind  
Linked Open Rules URL: <http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt>

Figure 33. Web service to get rules related to a specific sensor



This Java class should be merged with M3WS in future versions or create new Java file SLORSW and M3ConverterWS and then update the user interface in HTML and JavaScript.

## 9. TourismWS

All web services used for the tourism scenario<sup>14</sup>:

- tourism/clothes\_weather/{nameMeasurement}: to suggest activities according to weather measurements
- tourism/activity\_weather/{nameMeasurement}: to suggest activities according to weather measurements
- /tourism/snowGarment/: to deduce snow, a more scenario involving two sensors at the same time

nameMeasurement is described in the M3 nomenclature<sup>15</sup>. For instance, it can be WeatherLuminosity, Precipitation, WeatherTemperature

See comments in the Java Class

<sup>14</sup> <http://www.sensormeasurement.appspot.com/?p=tourism>

<sup>15</sup> <http://www.sensormeasurement.appspot.com/documentation/NomenclatureSensorData.pdf>



www.sensormeasurement.appspot.com/?p=tourism

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## Tourism (Weather & Emotion & Activity & Transport)

### Suggesting activities according to the weather

1. This scenario is based on: [M3 RDF sensor data](#)
2. We deduce the weather outside.
3. We propose activities according to the weather. => [/tourism/activity\\_weather/Precipitation](#)
4. M2M Application (Temperature => weather => Activity): Activity & Temperature
5. M2M Application (Luminosity => weather => Activity): Activity & Luminosity
6. M2M Application (Precipitation => weather => Activity): **Activity & Precipitation**
7. M2M Application (Wind speed => weather => Activity): Activity & Wind Speed

Figure 34. Tourism web services

## 10. TransportWS

All web services used for the transportation scenario<sup>16</sup>:

- `tourism/safety_device_weather/{nameMeasurement}`: to suggest safety devices in the smart car according to weather measurements
- `/tourism/snow/`: to deduce snow, a more scenario involving two sensors at the same time

`nameMeasurement` is described in the M3 nomenclature<sup>17</sup>. For instance, it can be `WeatherLuminosity`, `Precipitation`, `WeatherTemperature`

See comments in the Java Class

www.sensormeasurement.appspot.com/?p=transport

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## Transport (Weather & Safety devices & Driver's state)

### Suggesting safety devices in the smart car according to the weather

1. This scenario is based on: [M3 RDF sensor data](#)
2. We deduce the weather outside.
3. We propose safety devices according to the weather. => [/transport/safety\\_device\\_weather/WeatherLuminosity](#)
4. M2M Application (Temperature => weather => Safety devices): Safety devices & Temperature
5. M2M Application (Luminosity => weather => Safety devices): **Safety devices & Luminosity**
6. M2M Application (Precipitation => weather => Safety devices): Safety devices & Precipitation

- Name=luminosity, Value = 5000.0, Unit=lx, InferType = Weather Luminosity, Deduce = Cloudy
- Name=luminosity, Value = 50000.0, Unit=lx, InferType = Weather Luminosity, Deduce = Sunny, Suggest= Switch on the sun visor

Figure 35. Transport web services

<sup>16</sup> <http://www.sensormeasurement.appspot.com/?p=transport>

<sup>17</sup> <http://www.sensormeasurement.appspot.com/documentation/NomenclatureSensorData.pdf>

## 11. Design a new web service

To design a new web service, take inspiration from all of these web services.

# Part VI : Adding a new SWoT template

Add a new template in the template dataset<sup>18</sup>:

- M3 is the prefix of the ontology.
- `<m3:hasM2MDevice rdf:resource="&m3;LightSensor"/>` means that the template is related to the light sensor which is already referenced in the M3 ontology
- `<m3:hasContext rdf:resource="&m3;Weather"/>` means that the template is related to the weather domain.
- `<m3:hasUrlOntology rdf:resource="&weather;"/>` the URL of the domain ontology required to build the Semantic Web of Things (SWoT) application
- `<m3:hasUrlDataset rdf:resource="&transport-dataset;"/>` the URL of the domain dataset required to build the Semantic Web of Things (SWoT) application
- `<m3:hasUrlSparql rdf:resource="&sparql;m3SparqlGeneric.sparql"/>` The URL of the SPARQL query
- `<m3:hasSparqlVariableinferTypeUri rdf:resource="&m3;WeatherLuminosity"/>` to replace variable in generic sparql queries (optionnal)
- `<m3:hasSparqlVariabletypeRecommendedUri rdf:resource="&transport;SafetyDevice"/>` to replace variable in generic sparql queries (optionnal)
- `<m3:hasUrlRule rdf:resource="&lorWeather;"/>` the URL of the Linked Open Rules dataset to get high level abstractions
- `<m3:hasUrlRule rdf:resource="&ruleM3Converter;"/>` the URL of the rule dataset to semantically annotate IoT data according to the M3 nomenclature and M3 ontology.

```
<m3:M2MApplication rdf:about="&m3;WeatherTransportationSafetyDeviceLight">
  <m3:hasContext rdf:resource="&m3;Weather"/>
  <m3:hasContext rdf:resource="&m3;Transportation"/>
  <rdfs:label xml:lang="en">Luminosity, Transportation and Safety Device</rdfs:label>
  <rdfs:comment xml:lang="en">IoT application to suggest safety devices according to the luminosity
  <m3:hasM2MDevice rdf:resource="&m3;LightSensor"/>
  <m3:hasUrlOntology rdf:resource="&m3;"/>
  <m3:hasUrlOntology rdf:resource="&weather;"/>
  <m3:hasUrlDataset rdf:resource="&weather-dataset;"/>
  <m3:hasUrlOntology rdf:resource="&transport;"/>
  <m3:hasUrlDataset rdf:resource="&transport-dataset;"/>
  <m3:hasUrlSparql rdf:resource="&sparql;m3SparqlGeneric.sparql"/>
  <m3:hasSparqlVariableinferTypeUri rdf:resource="&m3;WeatherLuminosity"/>
  <m3:hasSparqlVariabletypeRecommendedUri rdf:resource="&transport;SafetyDevice"/>
  <m3:hasUrlRule rdf:resource="&lorWeather;"/>
  <m3:hasUrlRule rdf:resource="&ruleM3Converter;"/>
</m3:M2MApplication>
```

**Figure 36. A SWoT template**

<sup>18</sup> [www.sensormeasurement.appspot.com/dataset/iot-application-template-dataset](http://www.sensormeasurement.appspot.com/dataset/iot-application-template-dataset)

# Part VII : Adding a new ontology in LOV4IoT



In the future, we will automatically build the HTML web page according to the LOV4IoT RDF dataset. This work is ongoing. Currently, we have to update the HTML web page and the RDF dataset when we want to reference a new ontology-based project.

## 1. HTML web page

Go to [m3/WAR/html/lov4iot.html](http://m3/WAR/html/lov4iot.html)

Look for the table related to the domain, add a new line with all columns required.

## 2. LOV4IoT RDF dataset

In the LOV4IoT RDF dataset<sup>19</sup>, add a new ontology-based project.

---

<sup>19</sup> <http://www.sensormeasurement.appspot.com/dataset/lov4iot-dataset>

```

<m3:M2MApplication rdf:about="PaulStaroch">
  <m3:hasContext rdf:resource="&m3;Weather"/>
  <m3:hasContext rdf:resource="&m3;BuildingAutomation"/>
  <rdfs:label xml:lang="en">[Paul Staroch 2013]. See LOV4IoT for more details.</rdfs:label>
  <rdfs:comment xml:lang="en">Master's Thesis: A weather ontology for predictive control
  <m3:hasM2MDevice rdf:resource="&m3;Thermometer"/> in smart homes. 2013</rdfs:comment>
  <m3:hasM2MDevice rdf:resource="&m3;PrecipitationSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;HumiditySensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;AtmosphericPressureSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;SolarRadiationSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;WindDirectionSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;WindSpeedSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;SunPositionDirectionSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;SunPositionElevationSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;CloudCoverSensor"/>
  <m3:hasUrlOntology rdf:resource="http://paul.staroch.name/thesis/SmartHomeWeather.owl"/>
  <m3:hasUrlRule rdf:resource="http://paul.staroch.name/thesis/SmartHomeWeather.owl"/>
  <lov4iot:hasOntologyStatus rdf:resource="&lov4iot;OnlineLOV"/>
  <dcterms:creator>
    <foaf:Person rdf:about="mailto:paul@staroch.name">
      <foaf:name>Paul Staroch</foaf:name>
    </foaf:Person>
  </dcterms:creator>
</m3:M2MApplication>

```

**Figure 37. An ontology-based project referenced in the LOV4IoT RDF dataset**

## Part VIII : Additionnal

### I. Jena TDB and Jena Fuseki (Internship Amira)

The project is available in the same BSCW directory

Install Jena TDB

Does not work with App Engine but the server tomcat

See documentation Internship

Install Jena Fuseki

Does not work with App Engine but the server tomcat

See documentationInternship

## II. Security issues to execute JavaScript with the project

to disable the browser security (javascript problem)

command name\_browser -disable-web-security

chrome -disable-web-security