

TUTORIAL FOR THE OCCIWARE CLOUDDESIGNER USING OZWILLO'S MODELED PLATFORM

CloudDesigner (eclipse) version downloaded on sept. 17, 2015 10:28:16

Content:

- Modeling & Designing using the Eclipse toolset
- Setting the model into the OCCI interface (erocci)
- Curl Generators
- Create a Launcher to execute it manually on other projects

• Modeling & Designing using the Eclipse toolset

Create new Cloud Designer Project (File > new > Project :: CloudDesigner > OCCI extension Project)

Project Name: org.ozwillo.data.model.dcp
OCCI Extension Name: org.ozwillo.data.model.dcp
OCCI Extension Scheme: <http://occiware.org/ozwillo/dcp>

New Project

Create an OCCI Extension project

Enter a project name

Project name:

☒ Use default location

Location:

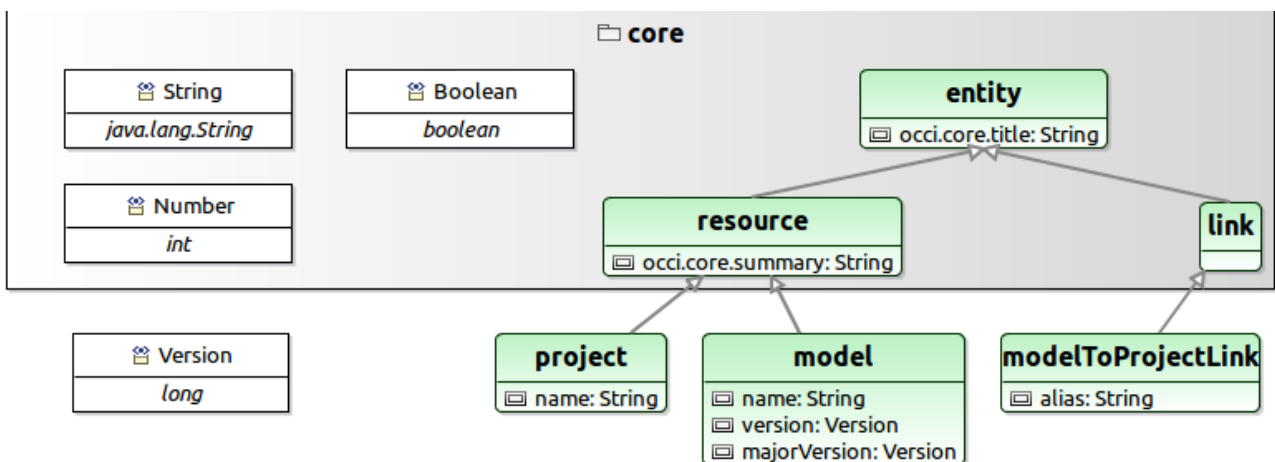
OCCI Extension name:

OCCI Extension scheme:

Referenced extensions:

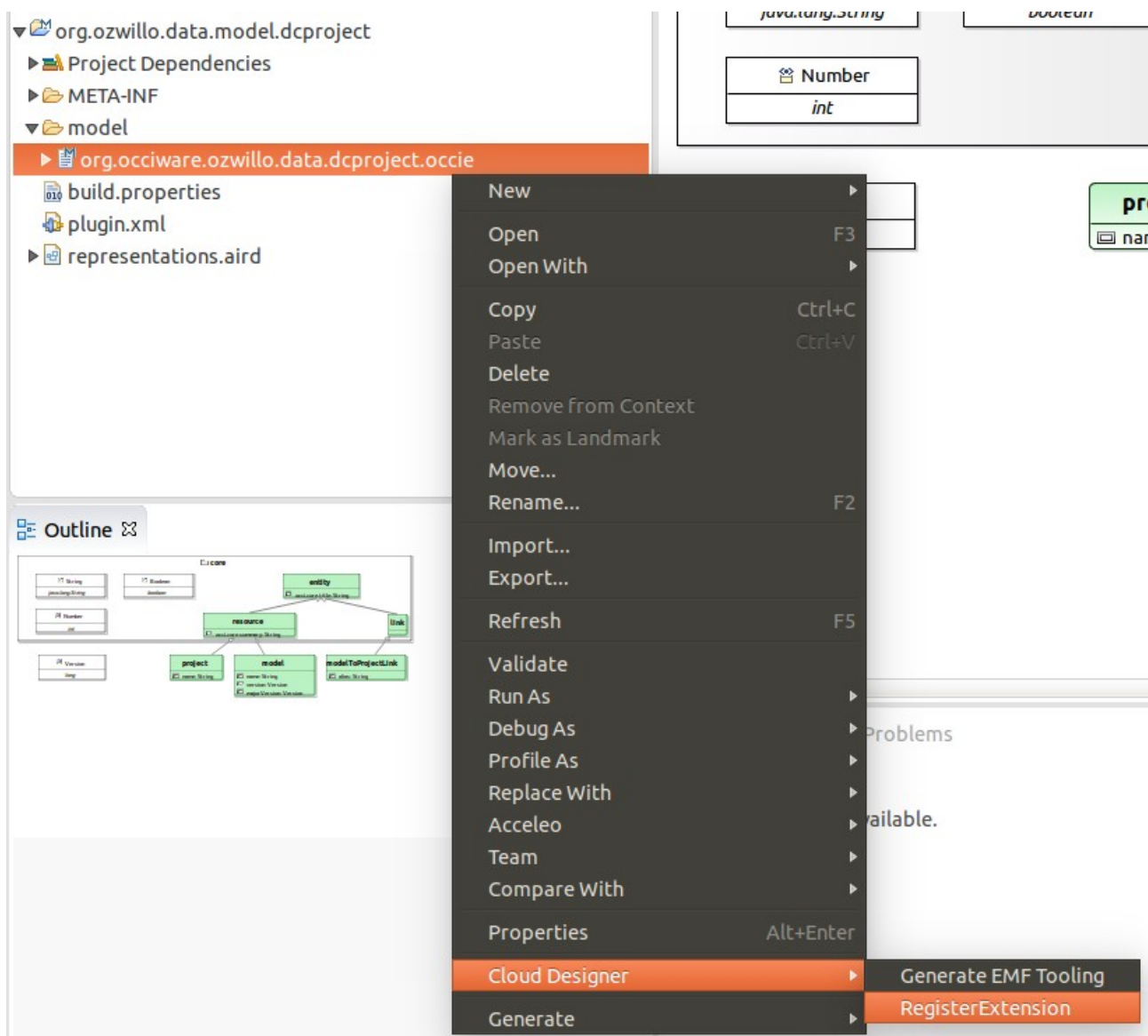
- ☐ <http://occiware.org/hypervisor#>
- ☒ <http://schemas.ogf.org/occi/infrastructure#>
- ☐ <http://occiware.org/docker#>

It will create a modeling project with a base (*.occie) file. Then add all required resources:



Once the resource has been modeled, it is necessary to register the extension to use it on other models.

- Right click on the extension file (*.occie) > Cloud Designer > Register Extension



Then its time to create the designer project.

- File > new > Modeling Project

- :: > Project name: org.ozwillo.data.designer

Click on the project and then we add a configuration file.

- File > new > other

- :: Cloud Designer > OCCi Configuration File

- :: File name: dcproject.occic

New OCCi Configuration

OCCI Configuration Model

Create a new OCCi Configuration model. Converts the container project to a modeling project if necessary.

Enter or select the parent folder:

org.ozwillo.data.designer

org.ozwillo.data.designer

org.ozwillo.data.model.dcproject

File name: dcproject.occic

Advanced >>

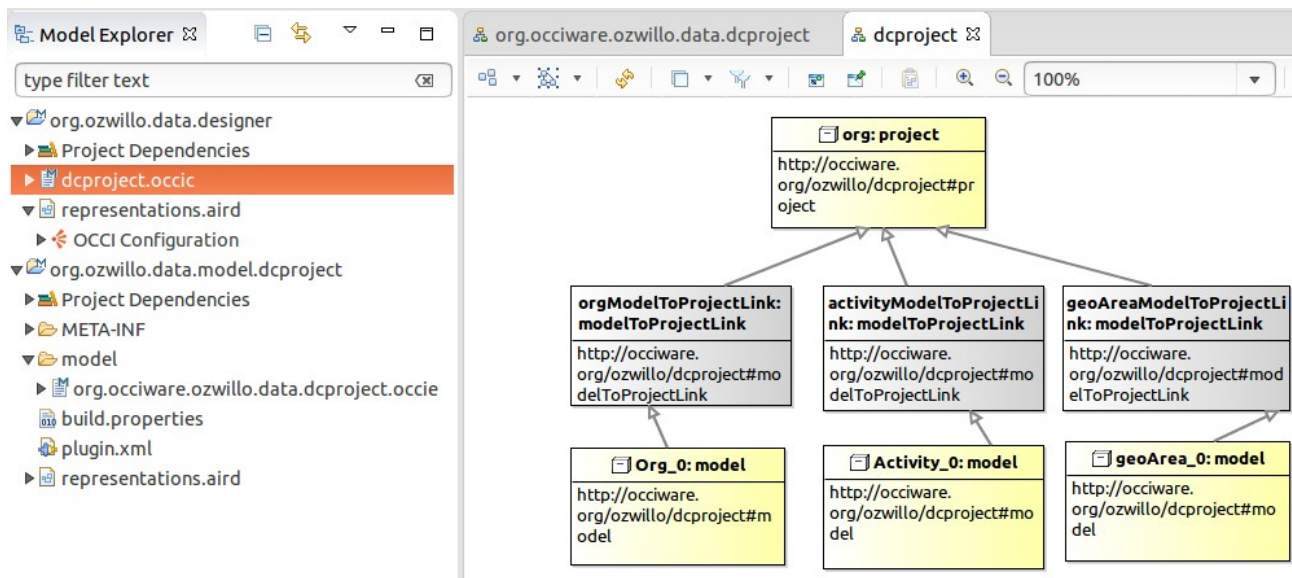
Referenced extensions:

- ☒ http://occiware.org/ozwillo/dcproject#
- ☐ http://occiware.org/hypervisor#
- ☐ http://schemas.ogf.org/occi/infrastructure#
- ☐ http://schemas.ogf.org/occi/core#
- ☐ http://occiware.org/docker#

< Back Next > Cancel Finish

It will be created a Resource Designer with resource defined in dcproject extension.

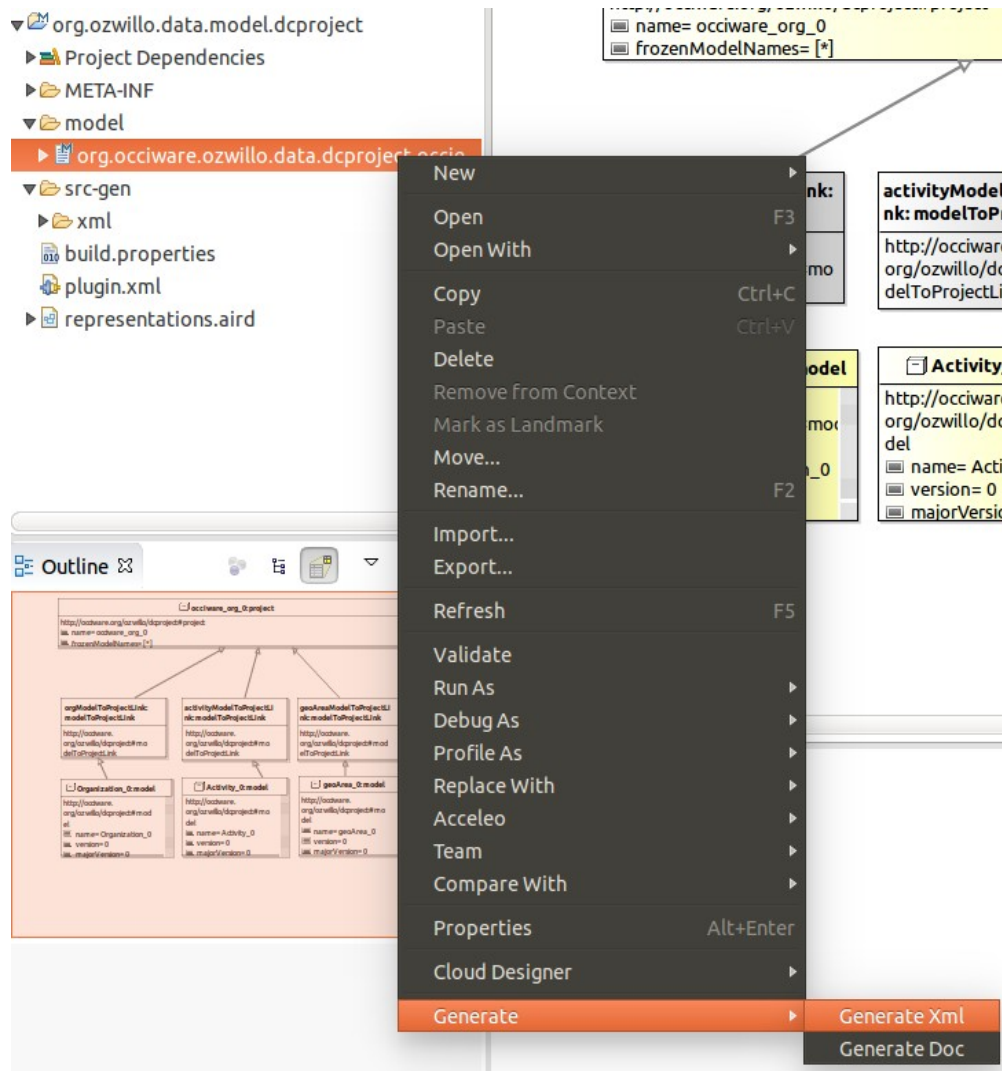
- Add the required resource defining its kind as dcproject#Project or dcproject#Model
- Use the Link category to link a dcproject#Model with its dcproject#Project, after manually link them it will be autogenerated a link resource
- Set the kind dcproject#modelToProjectLink on the link resource
- Set finally all the ids as needed



- **Setting the model into the OCCI Interface (erocci)**

In order to establish the model extension (occie – modeled resources) into erocci, first is necessary to create the xml file using the provided in-built XML generator.

- Right click on the « .occie » file > Generate > Generate Xml
- This will create a sub-directory called « src-gen » containing all the generated files



NB. There are issues on translating the Kind attributes' type into xml, and on setting the title. Once the XML files are generated, it is required to set manually the defined types... since the tested erocci version (docker-erocci described below) don't accept xsd: (should be xs:); also if other types are set, they are no set/converted in xml standard. For example, a kind attr defined as Long in a user-defined type (MajorVersion <::: java.Long) are not set at all, resulting in a line like `<occi:attribute name="frozenModelNames" type="" title="" />`.

Once the XML file of the modeled resources are ready, its time to setup & launch the erocci interface.

- Launch docker with the « organization » ecore conf

To launch the docker-erocci (erocci app preconfigured in an docker container), first it is required to have installed docker (<https://docs.docker.com/installation/ubuntu/linux/>) and git pull the project (<https://github.com/erocci/docker-erocci>).

After installing and pulling the project, its necessary to configure the docker container to set the configuration files to be installed at erocci execution. To configure it :

- copy (or create a symbolic linked to) the XML-fied conf file (occie) into /docker-erocci
- Modifying the file sys.config to change the next line by adding the xml-fied models (all included required xml files generated from the occie), like:

```
{occi_backend_mnesia, [{schemas, [{path, "/tmp/occi.xml"}, {path,
"/tmp/org.occiware.ozwillo.data.models.xml.xml"}]}]}
```

Then, the docker container can be launched, providing the xml to be uploaded into the container by using a command like:

```
sudo docker run --name="erocci_linked-data" -v `pwd`/sys.config:/tmp/sys.config -v
`pwd`/occi.xml:/tmp/occi.xml -v
`pwd`/org.occiware.ozwillo.data.models.xml:/tmp/org.occiware.ozwillo.data.models.
xml.xml -P -t -i erocci/erocci
```

NB. Check if there is no errors on loading the container along with erocci app & conf

- Posting resources to validate that the models where well defined/built

At this point, the container docker with the erocci application running should be ready with the configured resources modeled and deployed as REST services.

To validate it, it is necessary to post resources (project/model/link configurations) into the erocci app (and that will be stored in a local database):

- to post resources, set the json values and tunnel it into the curl POST request :

```
echo '{ "resources": [ { "kind": "http://occiware.org/ozwillo/dcproject#project",
"attributes": { "name": "occiware_org_0" } } ] }' | curl -v -H 'content-type:
application/json' -X POST --data-binary @-
http://localhost:32771/collections/project/
```
- to validate the resources were posted correctly, launch the below command (which could give a list if "project" posted resources , change it to model if required):

```
curl -v -H 'accept: text/uri-list' http://localhost:32771/collections/project/
curl -v -H 'accept: text/uri-list' http://localhost:32771/collections/model/
```
- to get values :

```
curl -X GET http://localhost:32771/collections/project/XXXXXXX
```
- to delete values :

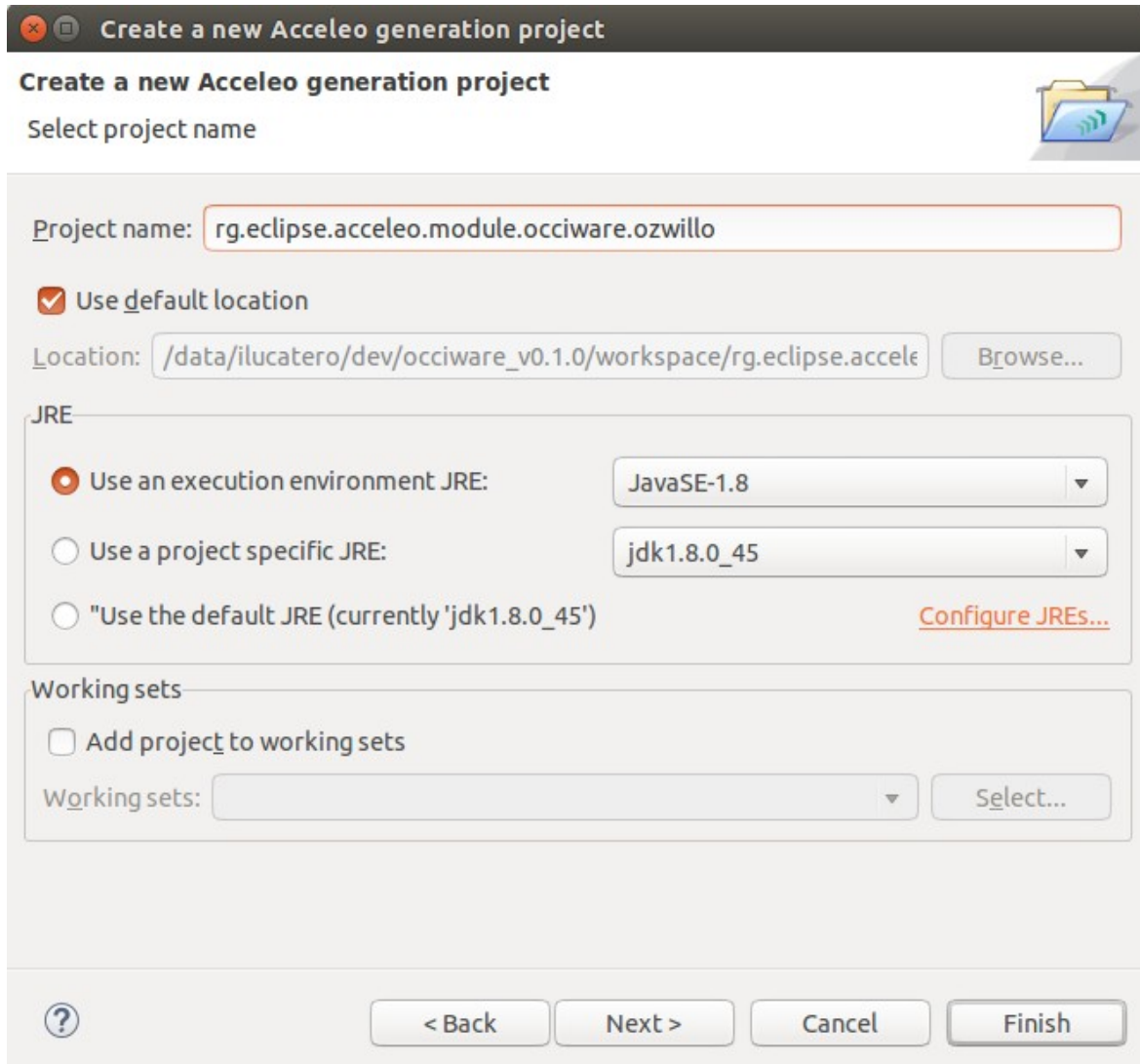
```
curl -X DELETE http://localhost:32771/collections/project/XXXXXXX
```

After validating that the model is up and running, and that the resource configurations are set correctly, its time to develop the a curl generator that will generate all the REST request to set configured resources (occic) into erocci (similar to the previous tests but massively and automated).

- **Curl Generators**

The following steps allow to create an Acceleo generator based on the OCCl metamodel:

- File > New > Acceleo Project
- Select a project name (or use the default one, just ensure it does not already exists)
- Click "Next"



Create a new Acceleo generation project

Select project name

Project name:

☒ Use default location

Location:

JRE

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ "Use the default JRE (currently 'jdk1.8.0_45') [Configure JREs...](#)

Working sets

☐ Add project to working sets

Working sets:

In the next window you have to set the generator options :

Create a new Acceleo generation project

Create a new Acceleo generation file

This wizard creates new mtl files which can be opened in the Acceleo editor.

Module Files

- generate

Parent Folder: ata.model.dcp/ src-gen **Browse...**

Module Name: generate

Metamodel URIs: http://schemas.ogf.org/occi

Template Name: generateElement

Type: Configuration

☒ Template ☐ Query

☐ Generate documentation

☒ Generate file

☒ Main template

< Back **Next >** **Cancel** **Finish**

- Click on finish. This will open the Acceleo perspective.
- Modify the *.mtl file to generate a SH file containing the curl posts for all resources (project, model and link) using Acceleo scripting (<http://www.lifl.fr/~dumoulin/enseign/2010-2011/pje/cours/2.generationCode/2.acceleo-mtl.pdf>).

The code for this generator should load the occic file and parse all its listed resources, then for each resource should validate the type of kind (project, model or link) and process it respectively (creating a curl post for the project, other for the model and other for the link including the attributes like names, versions, etc.).

Additionally, an important configuration to take into account is the properties files. In order to use the property files, you have to set it (adding as well the annotation `@notgenerated`) in the `getProperty()` method from the `generate.java` file. For example (properties file is named as « `default.properties` »):

```
/* @notgenerated */
@Override
public List<String> getProperties() {
    propertiesFiles.add("org.eclipse.acceleo.module.occiware.ozwillo.main.default");

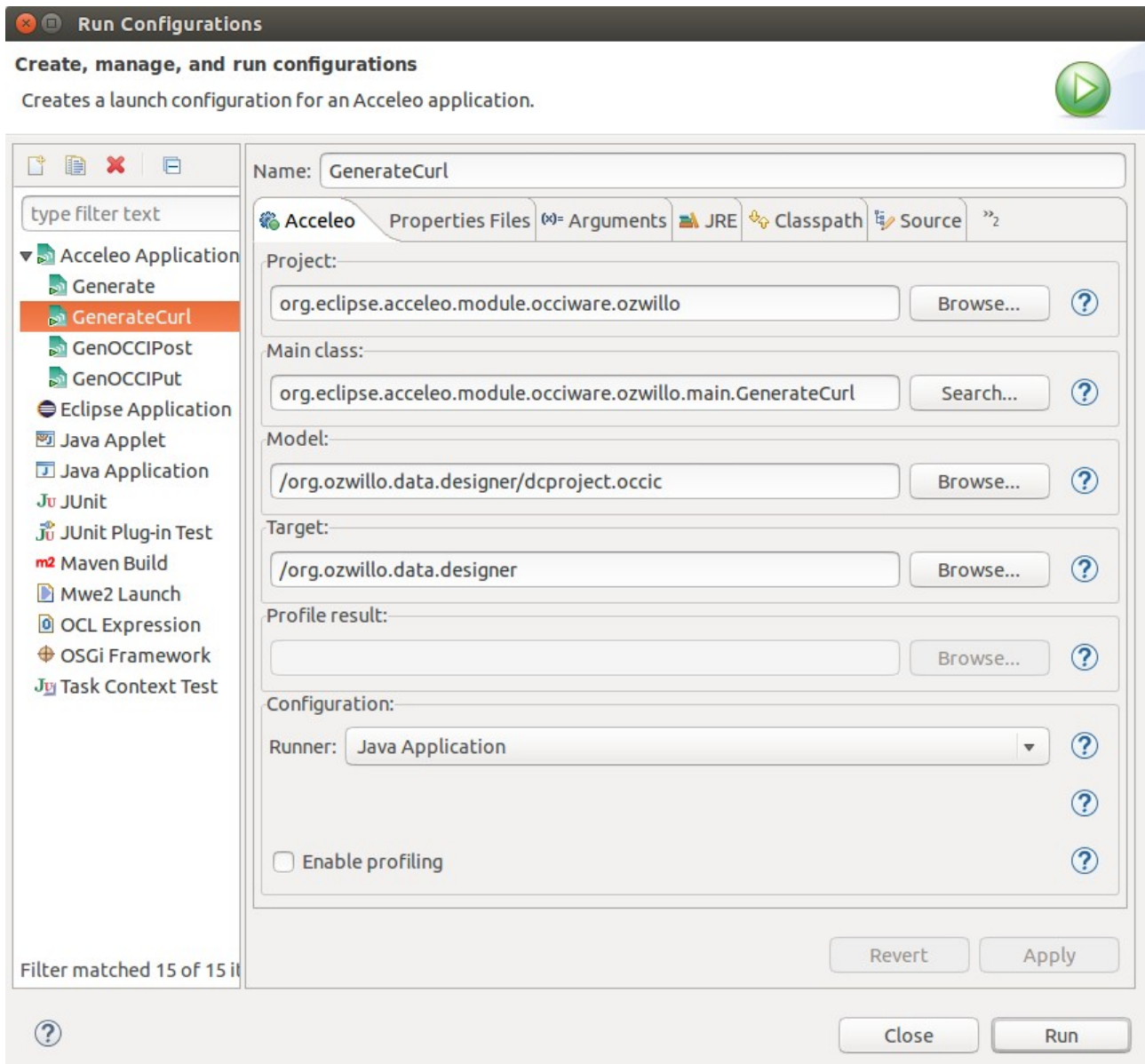
    return propertiesFiles;
}
```

NB. In the tested version, while retrieving and running the list of resources some values like "title" or "scheme" on the resources attributes were not found when processing the Acceleo list of resources/values.

Then to execute the generator manually do as next:

- In the new project, open the "generate.mtl" file from the navigator view (left panel). This file is located under the `src/` directory, in the `.main` package
- Set the generated file name. This can be a dynamic expression, e.g.: `file(extension.name, false, 'UTF-8')` or a static one: `file('configuration.xml', false, 'UTF-8')` (this example is on purpose, as a `Configuration` doesn't have any attribute usable to generate a distinct name)
- Inside of the "file" markups, write the generator: static text, dynamic text using ocl generation, templates calls.. Please refer to the Acceleo documentation for more informations on how to write a generator. The extension documentation generator can be used as an inspiration:
<https://github.com/occiware/ecore/blob/master/clouddesigner/org.occiware.clouddesigner.occi.gen.doc/src/org/occiware/clouddesigner/occi/gen/doc/main/generate.mtl>
- To test the generator, right-click on the "generate.mtl" file et select "Run as > Launch Acceleo application"
- Add the OCCl metamodel URI: click on the "+" button then select "<http://schemas.ogf.org/occi>"
- Select the input model (delete the "xmi" filter in the dialog in order to display all files), to use (the root type of the generator): `Extension` or `Configuration` (depending whether you want to generate from an `occie` or `occic` extension)
- And select the Target (project) directory where the generated files are going to be stored.
- Check the "Main template" & "Generate file" options
- Configuration : Runner => Java Application

- Then click "Run".



- Open the generated file in the target directory to check the result

```

[comment encoding = UTF-8 /]
[module genOCCIPost('http://schemas.ogf.org/occi')]

[comment http://www.lifl.fr/~dumoulin/enseign/2010-2011/pje/cours/2.generat

[template public main(configuration : Configuration)]
[comment @main/]

[file ('genOCCI_DCProjConf.sh', false, 'UTF-8')]
#!/bin/sh

SERVER=$1
LOG_RESP="genOCCI_DCProjConf.resp.log"

echo "Starting POSTting the configuration in erOCCI server : " $SERVER
#http://localhost:XXXX/collections/resourceName/
echo "*****"
[comment: Project/Model cUrl generation /]
[for (resource : Resource | configuration.resources)]
URL=$SERVER/collections/[resource.getCategory()]/
KIND='http://occiware.org/ozwillo/dcpjct#[resource.getCategory()]/'
ATTRIBUTES='{[resource.attributes.generateAttribute(resource.kind)]}'
echo "URL: " $URL
echo "ATTRIBUTES: " $ATTRIBUTES
echo "KIND: " $KIND
echo "Curl: " [resource.generateCurl()]/
echo "Resource: '"[resource.generateBinaryData()]/' "'
echo "POST: echo '"[resource.generateBinaryData()]/' "' | [resource.genera
echo '"[resource.generateBinaryData()]/' "' | [resource.generateCurl()]/ >>
echo "-----"
[/for]

echo ".....Fetch the resources....."
curl -v -H 'accept: text/uri-list' $URL
echo "Finish POSTting the confiruation in erOCCI server. Check responses in
echo "*****"
[/file]

[/template]

```

```

#!/bin/sh

SERVER=$1
LOG_RESP="genOCCI_DCProjConf.resp.log"

echo "Starting POSTting the configuration in erOCCI server
#http://localhost:XXXX/collections/resourceName/
echo "*****"
URL=$SERVER/collections/project/
KIND='http://occiware.org/ozwillo/dcpjct#project'
ATTRIBUTES='{ "name": "occiware_org_0", "frozenModelNames": "[*
echo "URL: " $URL
echo "ATTRIBUTES: " $ATTRIBUTES
echo "KIND: " $KIND
echo "Curl: " curl -v -H 'content-type: application/json' -;
echo "Resource: '"{ "resources": [ { "kind": ' "'$KIND', "
echo "POST: echo '"{ "resources": [ { "kind": ' "'$KIND',
echo '"{ "resources": [ { "kind": ' "'$KIND', "attributes
echo "-----"
URL=$SERVER/collections/model/
KIND='http://occiware.org/ozwillo/dcpjct#model'
ATTRIBUTES='{ "name": "Activity_0", "version": "0", "majorVersion
echo "URL: " $URL
echo "ATTRIBUTES: " $ATTRIBUTES
echo "KIND: " $KIND
echo "Curl: " curl -v -H 'content-type: application/json' -;
echo "Resource: '"{ "resources": [ { "kind": ' "'$KIND', "
echo "POST: echo '"{ "resources": [ { "kind": ' "'$KIND', "

```

- Create a Launcher to execute it manually on other projects

*** To see with Obeo how they integrate it within the tool ***

An example of how to create a launcher can be found in the next link :

https://wiki.eclipse.org/Acceleo/Getting_Started#Creating_a_UI_launcher