

Light - IOT



Un cadre formel et outillé pour la
gestion de toute ressource en nuage

Christophe Gourdin



SOMMAIRE

ETAPE 1 : Objectifs	4
ETAPE 2 : Installation de Cloud Designer	5
Démarrage du programme LightServer	6
ETAPE 3 : Création du modèle d'extension	7
Création du kind « lampe »	8
Validez le diagramme avec OCL	8
Création d'un attribut occi.light.state	10
Création d'un type Enum	10
Création des actions sur la ressource « lampe »	12
Génération des classes représentant l'extension light	13
Autres vues du modèle d'extension light	15
ETAPE 4 : Génération de la spécification Alloy	16
ETAPE 5 : Validation formelle avec Alloy	17
ETAPE 6 : Génération de la documentation	18
ETAPE 7 : Génération du connecteur java	19
Génération du projet connecteur light.connector	19
Préparation du connecteur	21
Mise à jour du code du connecteur	22
Callbacks OCCl	22
Actions OCCl Light	23
ETAPE 8 : Génération des tests JUNIT	25
Implémentation du test unitaire d'une lampe	26
ETAPE 9 : Génération d'un designer dédié à l'extension Light	27
Génération du Light Designer	27
Création d'une configuration minimale de test	30
Lancement des actions (Create, switchOn, switchOff, Update, Delete)	32
ETAPE 10 : Erocci dbus java backend	34
Dépendance erocci-dbus-java	35
Appel du runtime erocci-dbus-java	36
Construction des jars light et light.connector	38
Référencement au classpath de Erocci-dbus-java	39
ETAPE 11 : Lancement erocci	41

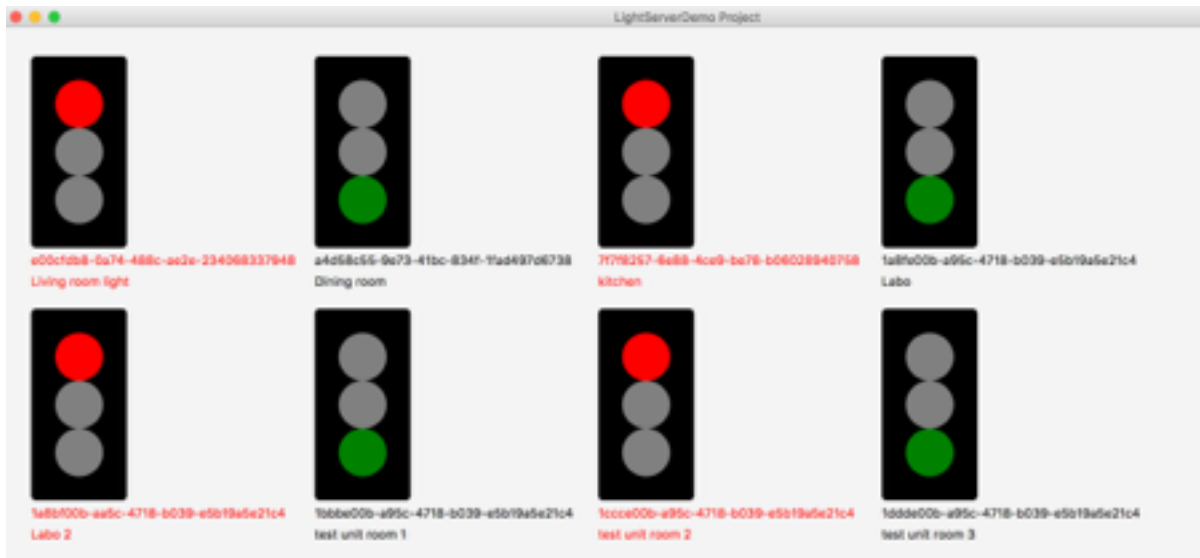
Lancement erocci	41
ETAPE 12 : Utilisation des lampes IOT via curl manuel	45
ETAPE 13 : Modélisation de la configuration	47
Création du projet LightIOTProject	47
Création d'une ressource lampe	51
ETAPE 14 : Génération spécification Alloy pour la configuration	53
ETAPE 15 : Vérification Alloy pour la configuration	53
ETAPE 16 : Génération des requêtes curl	54
ETAPE 17 : Lancement du script généré	56
ETAPE 18 : Interaction manuelle avec Erocci pour vérifier que la configuration est bien déployée	57
ETAPE 19 : Récupérer la configuration déployée via jOCCI	58
ETAPE 20 : Utiliser le connecteur jOCCI	58
Modification de la configuration	58
Créer/modifier/détruire des lampes et exécuter des actions	58

ETAPE 1 : Objectifs

Le principal objectif de ce tutorial est de prendre en main la chaîne d'outils OCCIware.

Pour matérialiser la prise en main, nous allons « OCCIfier » une application client / serveur Light IOT qui permet de manipuler des lampes en utilisant des opérations CRUD ainsi que des actions comme allumer une lampe et éteindre une lampe.

L'application client et server nous permet d'obtenir un aperçu comme ceci :



L'application client est téléchargeable à l'adresse :

<https://github.com/cgourdin/LightClientBuild/blob/master/lightclient.jar>

L'application serveur est téléchargeable à l'adresse :

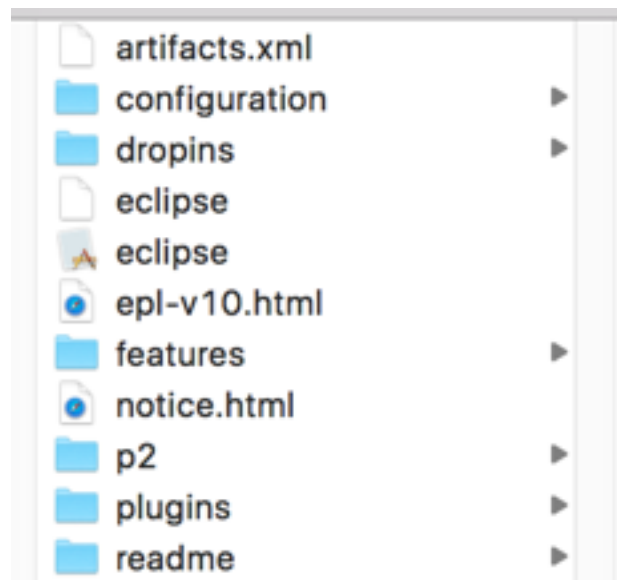
<https://github.com/cgourdin/LightServerInstaller>

Les applications LightClient et LightServer utilisent le protocole Websocket pour communiquer. LightClient sera utilisé comme un bibliothèque tandis que LightServer matérialisera les lampes.

ETAPE 2 : Installation de Cloud Designer

Installez Cloud Designer comme suit :

- Téléchargez Cloud Designer pour votre système d'exploitation via l'adresse :
 - <https://www.obeo.fr/download/occiware/>
 - Mac OSX : https://www.obeo.fr/download/occiware/products/org.occiware.clouddesigner.product-macosx.cocoa.x86_64.zip
 - Windows : https://www.obeo.fr/download/occiware/products/org.occiware.clouddesigner.product-win32.win32.x86_64.zip
 - Linux : https://www.obeo.fr/download/occiware/products/org.occiware.clouddesigner.product-linux.gtk.x86_64.zip
- Pour l'installer, il suffit de décompresser le fichier dans un répertoire :



Enfin **lancez** Cloud Designer, il suffit de double cliquer sur le fichier exécutable **eclipse**.

Dans le cas où l'on vous a fourni une VM :

Vous pouvez importer l'image dans virtual box, ceci est décrit dans cette page : https://docs.oracle.com/cd/E26217_01/E35193/html/qs-import-vm.html

La VM est préparée avec tous les outils, vous n'avez donc pas à installer de logiciels.

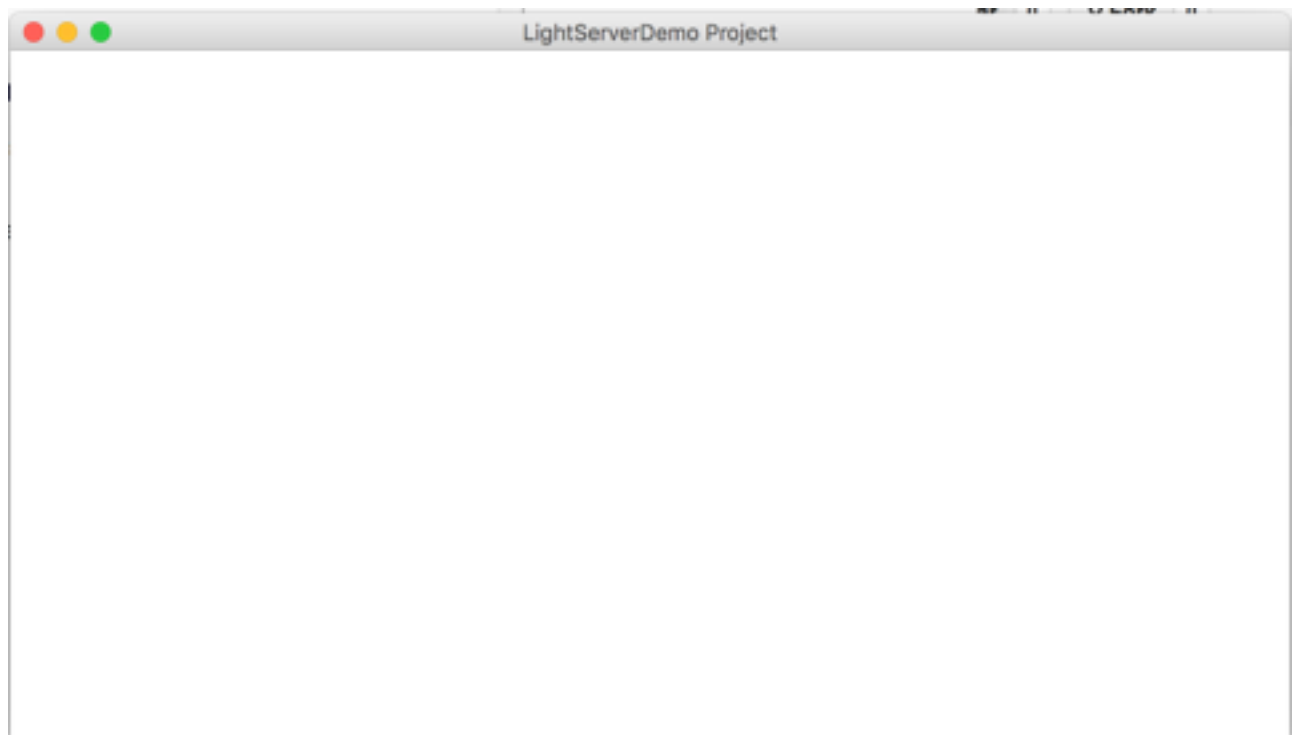
Démarrage du programme LightServer

Il y a deux façons de procéder :

- Soit vous installez le programme directement via un installateur fourni (ou <https://github.com/cgourdin/LightServerInstaller/tree/master>)
- Pré-requis avec les sources : installer gradle (dernière version) et clonez le repository : <https://github.com/cgourdin/LightServerDemo>

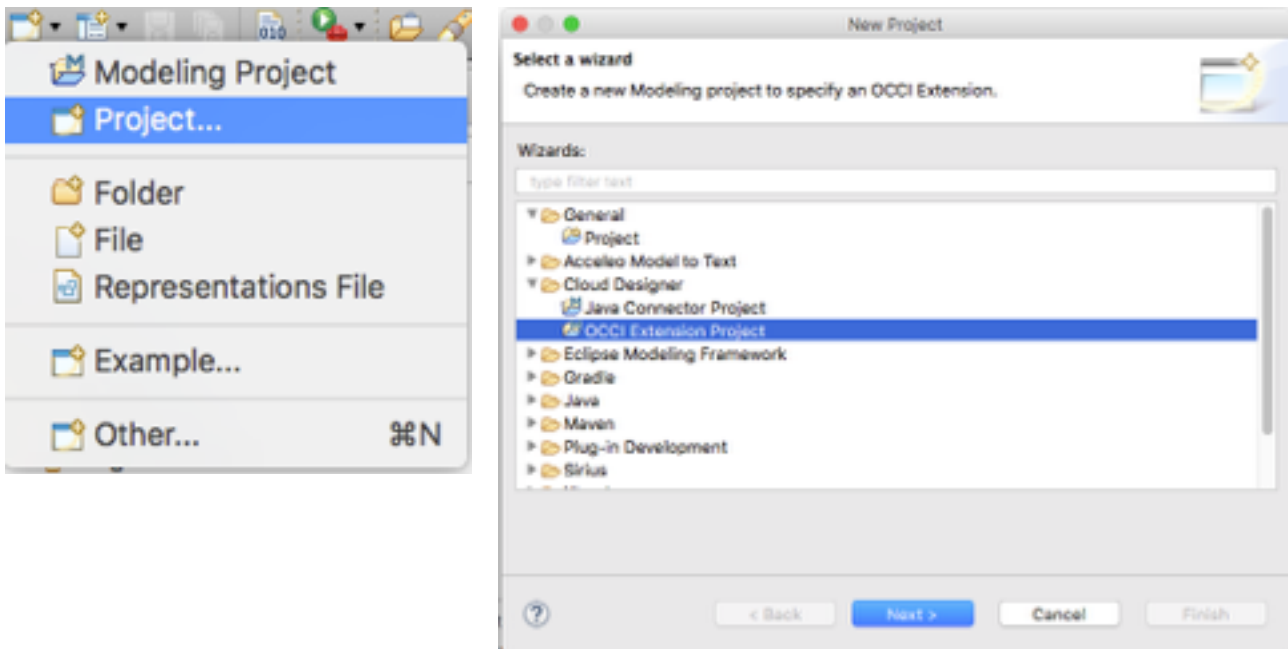
Pour lancer l'application, double cliquez sur son icône ou avec les sources exécutez la commande **gradle run**.

Une fenêtre blanche s'ouvre, à ce stade c'est normal :



ETAPE 3 : Création du modèle d'extension

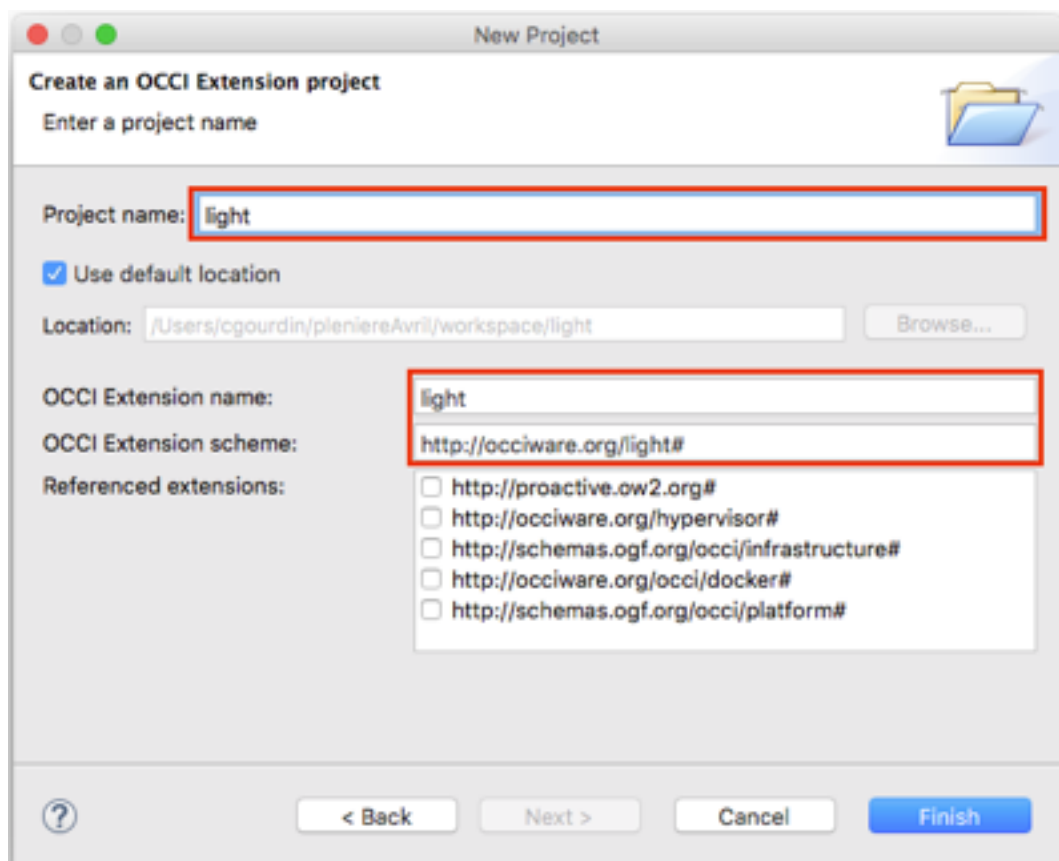
New project —> Sélectionner **Cloud Designer / OCCl Extension Project** puis cliquer sur **Next**.



Project name : **light**

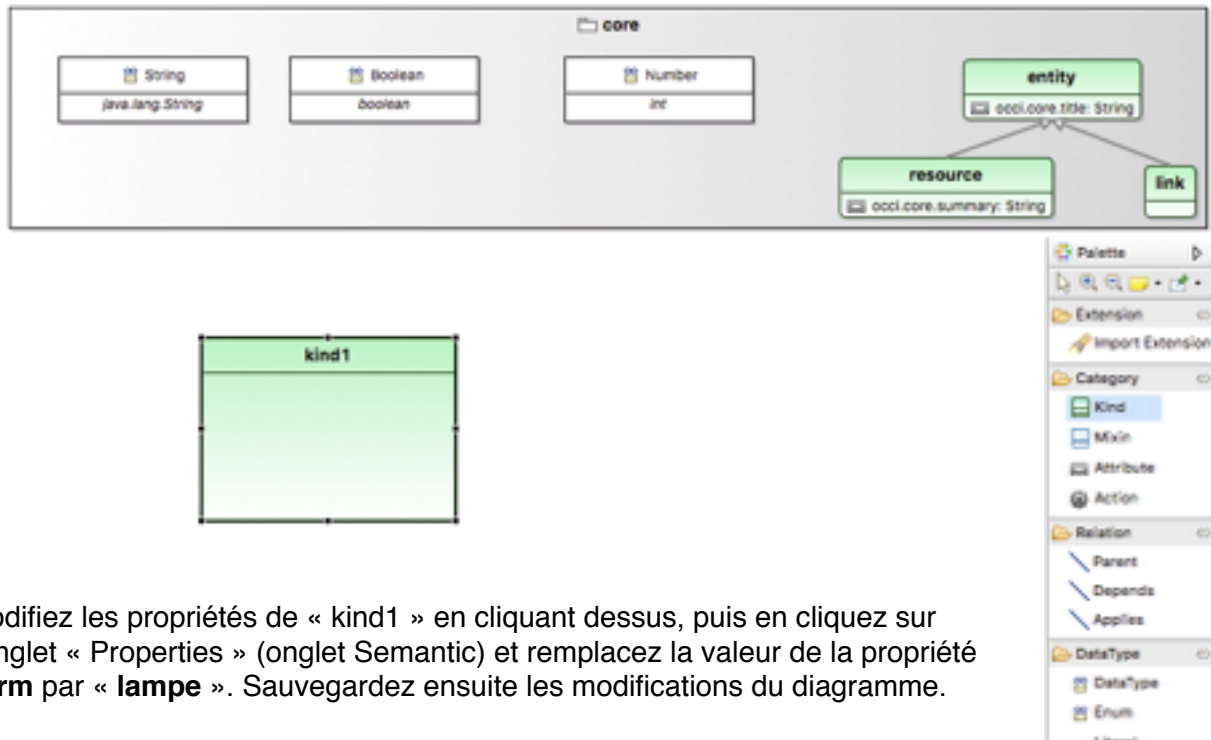
OCCI Extension name : **light**,

OCCI Extension scheme : <http://occiware.org/light#> , puis cliquez sur Finish.



Création du kind « lampe »

Dans la palette cliquez sur Kind et cliquez dans une zone libre du modeleur.



Modifiez les propriétés de « kind1 » en cliquant dessus, puis en cliquez sur l'onglet « Properties » (onglet Semantic) et remplacez la valeur de la propriété **Term** par « **lampe** ». Sauvegardez ensuite les modifications du diagramme.

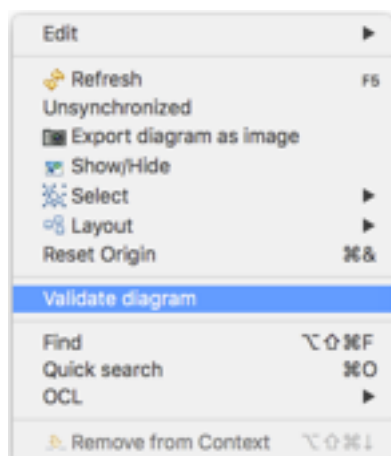
The screenshot shows the 'Properties' panel for 'Kind kind1'. The 'Semantic' tab is selected. The table below lists the properties and their values.

Property	Value
Kind kind1	
Entities	
Parent	
Scheme	http://occlware.org/light#
Term	kind1
Title	

Validez le diagramme avec OCL

Au préalable, désélectionnez le kind « lampe ».

Cliquez droit sur une zone libre du diagramme puis cliquez sur Validate diagram.



En résultat, une erreur OCL : OCL indique que le kind lampe a besoin d'un kind parent.



Ce kind est une ressource, il doit donc avoir comme **parent** une **ressource**.

Dans la palette cliquez sur parent et sélectionnez le kind « **lampe** » puis le kind « **resource** ».

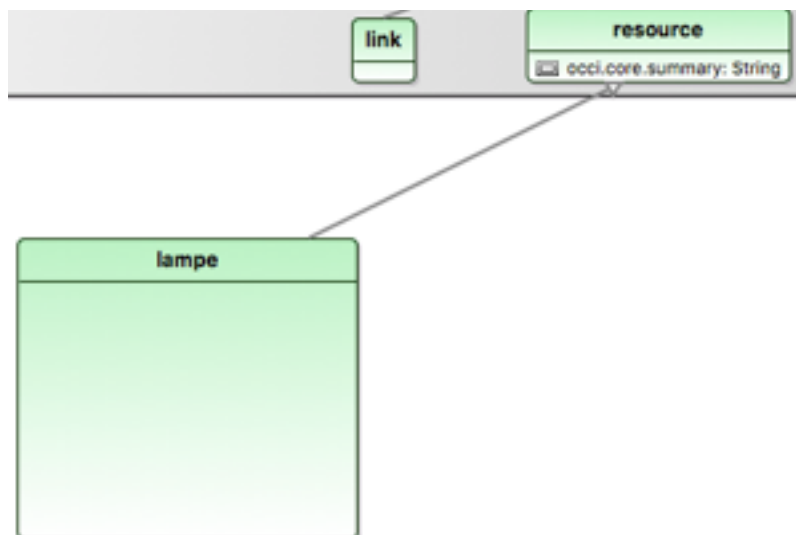
Cela permet de donner le lien de parenté suivant : le kind « **lampe** » a comme parent « **resource** ». Donc les lampes seront des ressources OCCI.



On s'aperçoit que la ressource « **lampe** » est toujours en **erreur** grâce à l'icône en haut à droite d'attention.

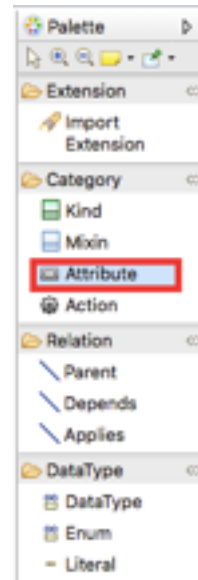
Relancez la validation OCL en cliquant droit sur une **zone libre du workbench** et cliquez sur **Validate diagram** comme précédemment.

Nous obtenons comme résultat : votre diagramme est valide.



Création d'un attribut occi.light.state

Dans la palette cliquez sur Attribute puis cliquez sur lampe.



Modifiez la propriété name : occi.light.state

Modifiez la propriété mutable : true

Modifiez le terme : State

Création d'un type Enum

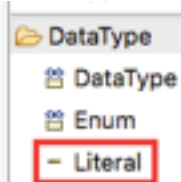
Ce type représentera l'état d'une lampe.

Nous allons lui donner un type **Enum** avec comme valeurs possible **on** et **off**.

Dans la palette, cliquez sur Enum puis sur une zone libre du workbench. Nommez le **State**.



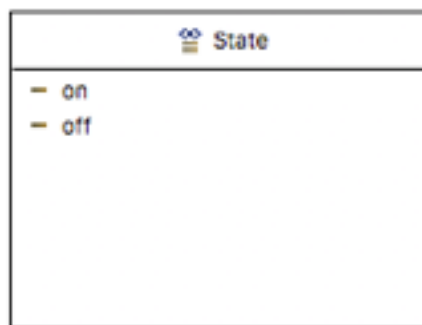
Cliquez dans la palette sur Literal puis sur l'Enum State



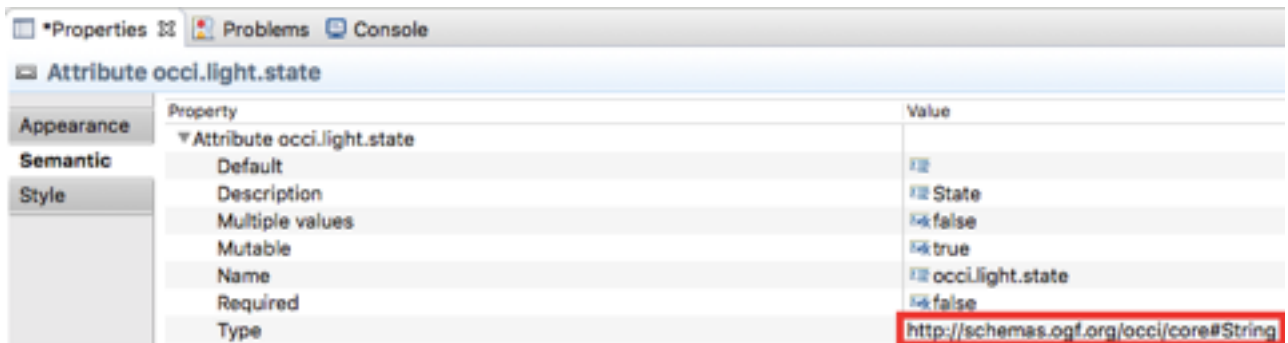
Modifiez la propriété name du literal en lui donnant comme valeur « **on** »

Ajoutez à l'enum **State** un literal « **off** » de la même manière que précédemment.

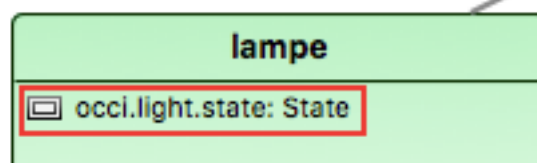
Sauvegardez le diagramme.



Revenons à notre ressource « lampe », cliquez sur l'attribut **occi.light.state**, modifiez son type en cliquant dans l'onglet **propriété** puis dans la colonne **Value** de la propriété **Type**, une liste apparaît puis sélectionnez le type **Enum** que vous venez de créer.
Le type Enum « **State** » doit apparaître dans la liste avec comme **schema** <http://occiware.org/light#State>.



En résultat vous devez obtenir :



Création des actions sur la ressource « lampe »

Une lumière peut être éteinte ou allumée via l'interrupteur **on** / **off**.

La lumière peut donc avoir deux états : **on** et **off**.

Nous allons donc créer deux actions : **switchOn** et **switchOff**.

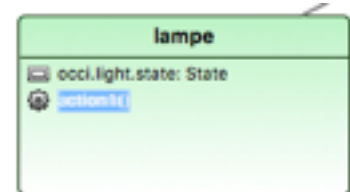
Pour créer une action sur une ressource, il faut cliquer dans la palette sur « **Action** », puis cliquez sur la ressource :

Modifiez les propriétés de action1():

Term : switchOn

Title : Turn on the light

Notez que la propriété Scheme est remplie automatiquement.



Property	Value
▼ Action switchOn	
Scheme	http://occiware.org/light/lampe/action#
Term	switchOn
Title	Turn on the light

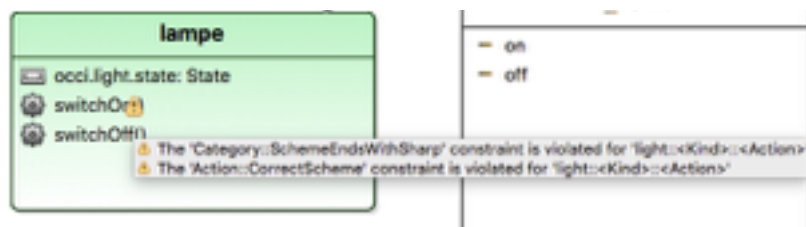
Créez une seconde action avec comme propriétés :

Term : switchOff

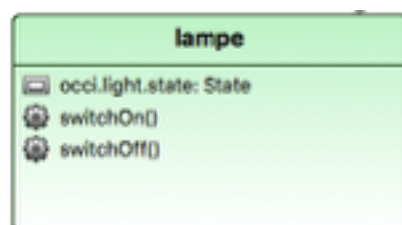
Title : Turn off the light

Changez le schéma d'une action en remplaçant la valeur de propriété Scheme par `http://test#action`, puis réalisez la validation OCL en cliquant droit sur une zone libre du workbench puis Validate diagram.

Vous obtenez ceci :



Remplacez la valeur de propriété Scheme par `http://occiware.org/light/lampe/action#` puis relancez la validation OCL en cliquant droit sur une zone libre du workbench puis Validate diagram.

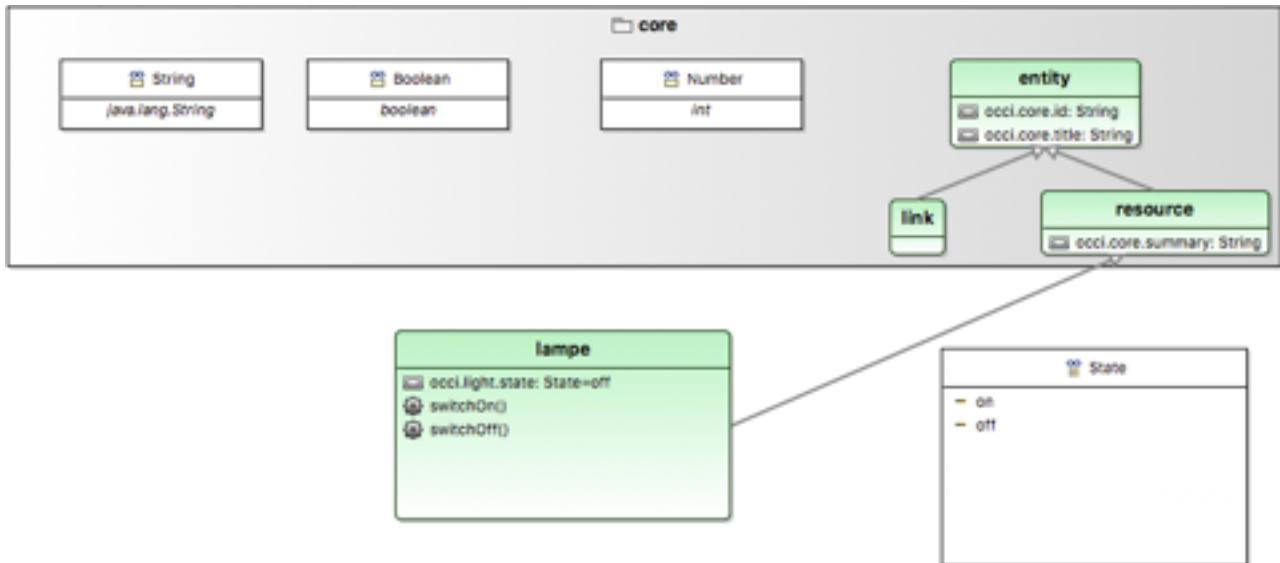


Le modèle est terminé !

Sauvegardez le modèle en cliquant sur l'icône :

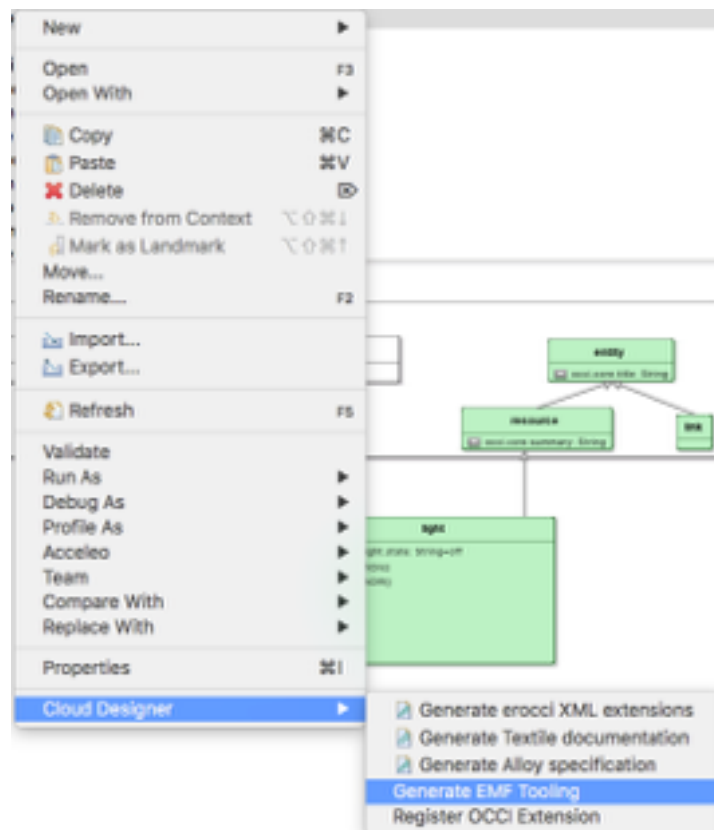


Votre modèle d'extension « **light** » doit ressembler à ceci :



Génération des classes représentant l'extension light

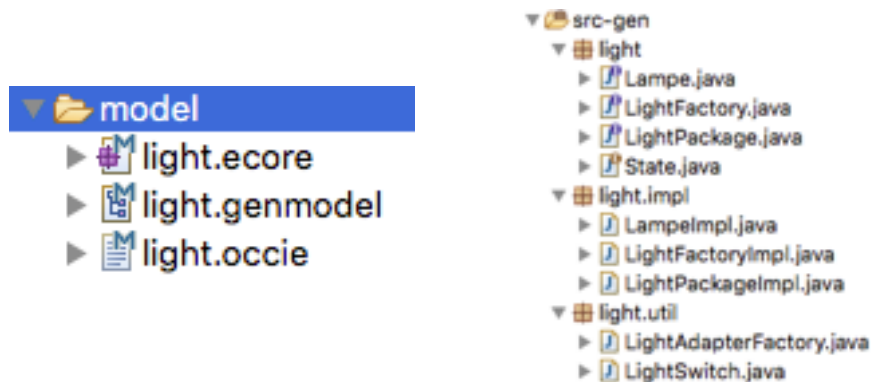
La génération des classes du connecteur permet d'utiliser EMF (Eclipse Modeling Framework) et par là exploiter les objets propres à notre modèle d'extension. Cliquez droit sur le fichier **light.occie** puis **Cloud designer** —> **Generate EMF Tooling**.



Cela va créer 2 fichiers dans le répertoire model, **light.ecore**, **light.genmodel** ainsi que les sources représentant le modèle programmatique de l'extension dans le répertoire **src-gen**.

Des interfaces pour le modèle sont générées dans **src-gen** :

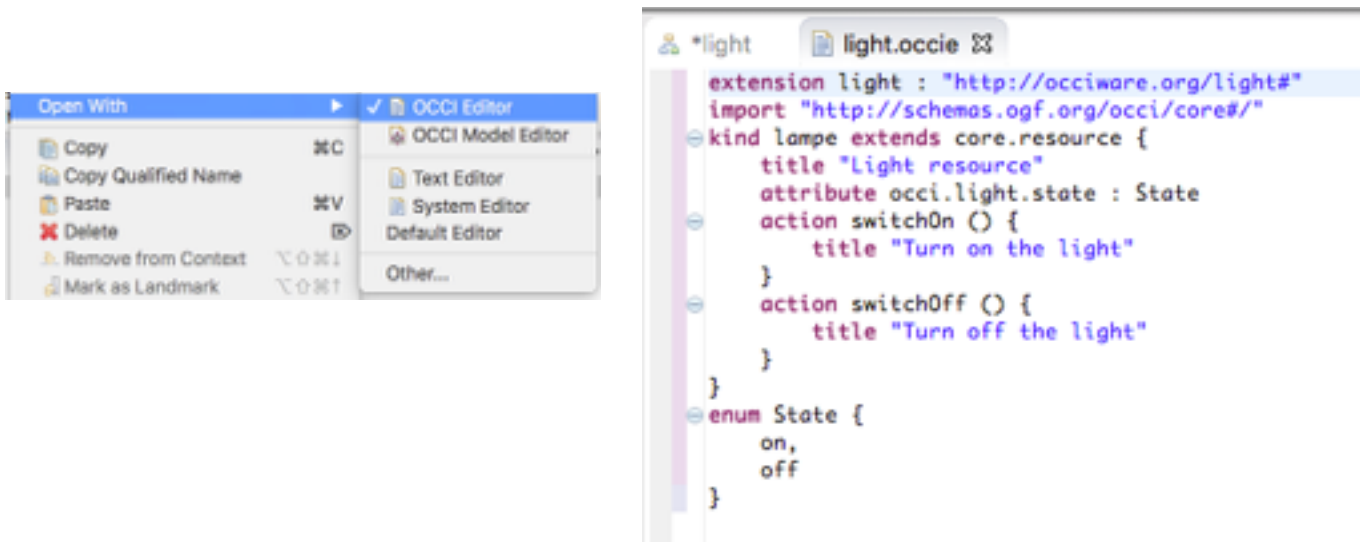
- package **light** :
 - Light.java est une interface représentant une ressource de type lumière
 - LightFactory.java représente la factory qui va permettre de mapper l'interface Light lors de l'instanciation d'une nouvelle ressource de type « Light ».
- LightPackage.java représente tous les types, attributs et méthodes de votre extension.
- package **light.impl** : représente toute les classes d'implémentation des classes d'interfaces ci-dessus.
- package **light.util** : Contient des classes utilitaire pour notre extension.



Autres vues du modèle d'extension light

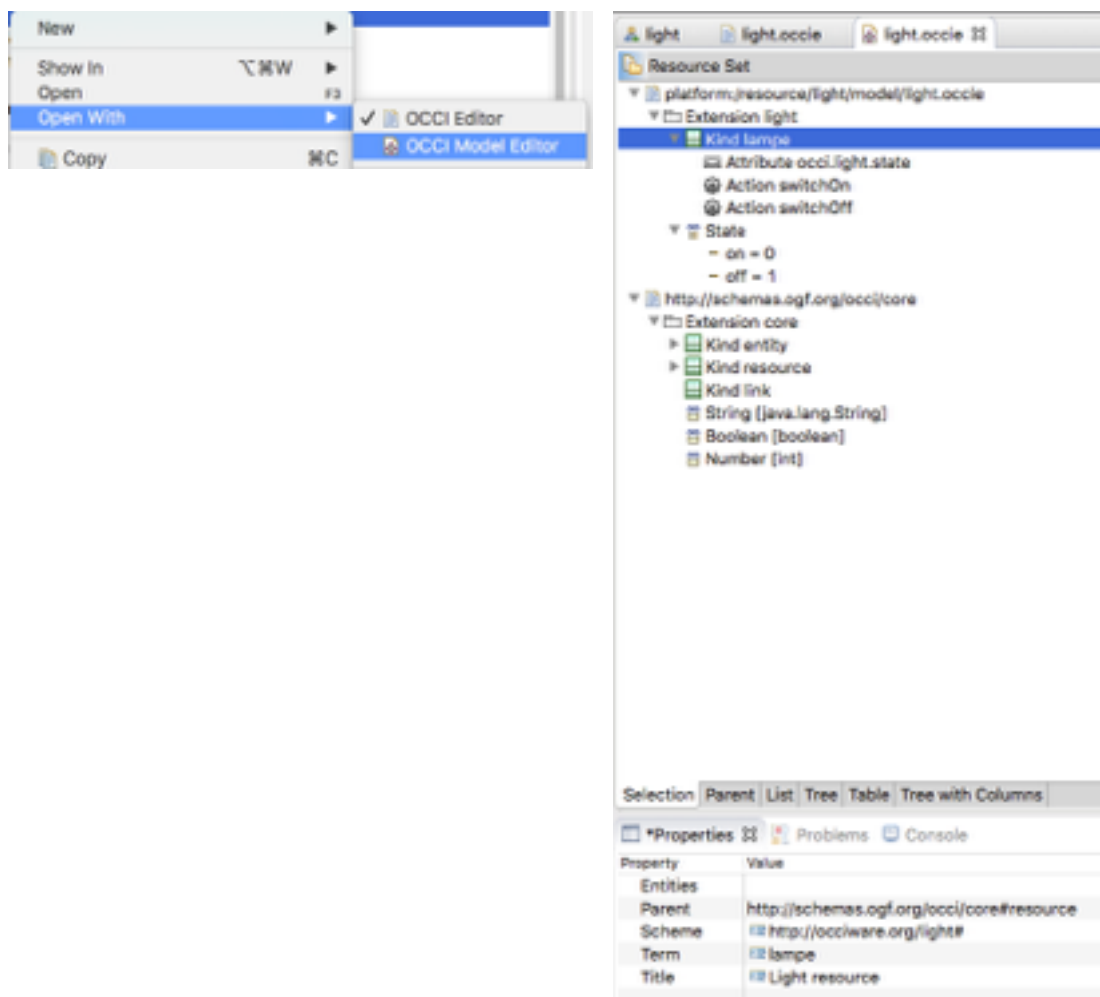
Editeur textuel OCCI :

Dans la partie Model Explorer clic droit sur le fichier **light.occie** puis Open With → OCCI Editor



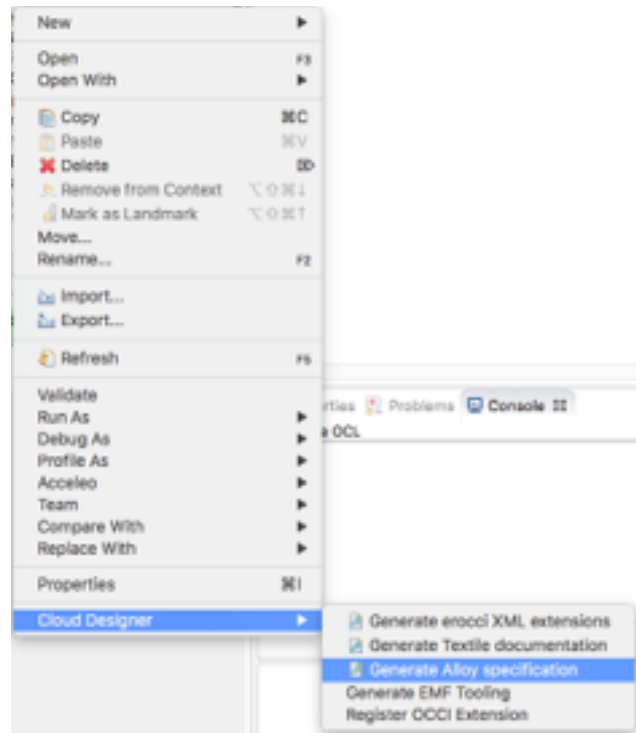
Vue arborescente OCCI :

Dans la partie Model Explorer clic droit sur le fichier **light.occie** puis Open With → OCCI Model Editor.

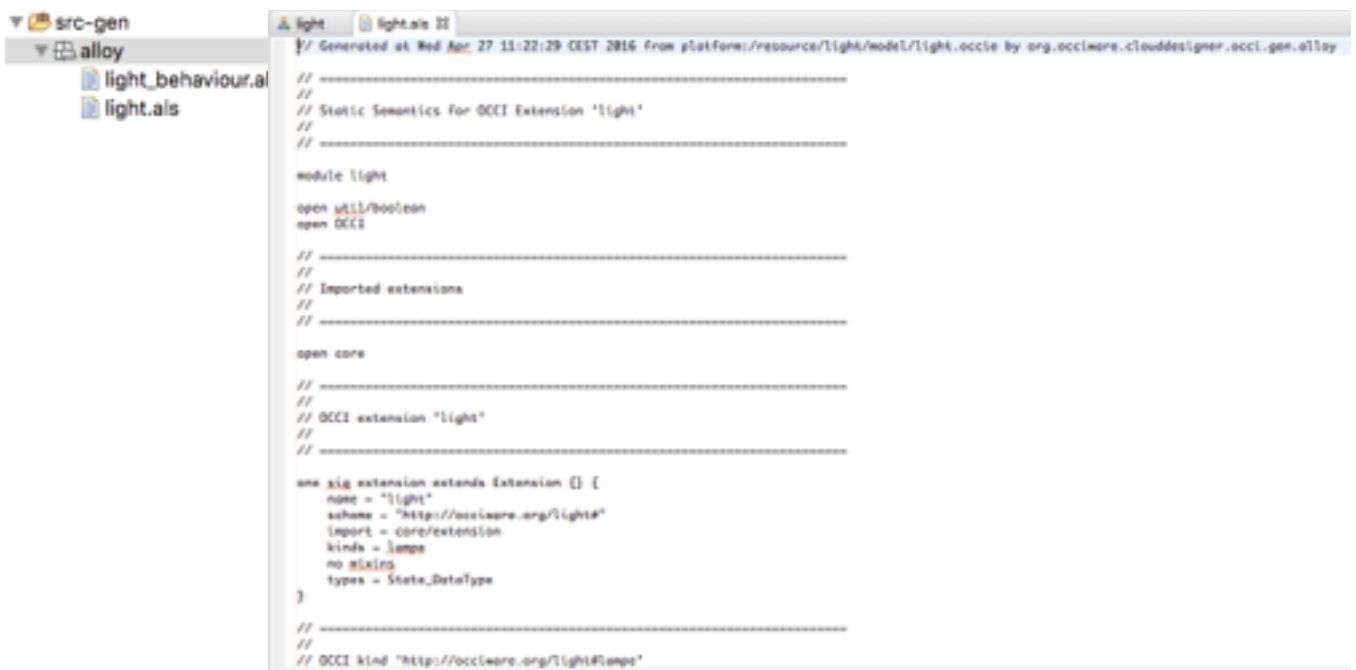


ETAPE 4 : Génération de la spécification Alloy

Pour réaliser la génération de la spécification **Alloy**, il faut cliquer droit sur le fichier **light.occie** (dans Model Explorer), sous menu **Cloud Designer** —> **Generate Alloy specification**.



Deux fichiers sont générés : **light_behaviour.als** et **light.als** dans le répertoire **src-gen/alloy/**.

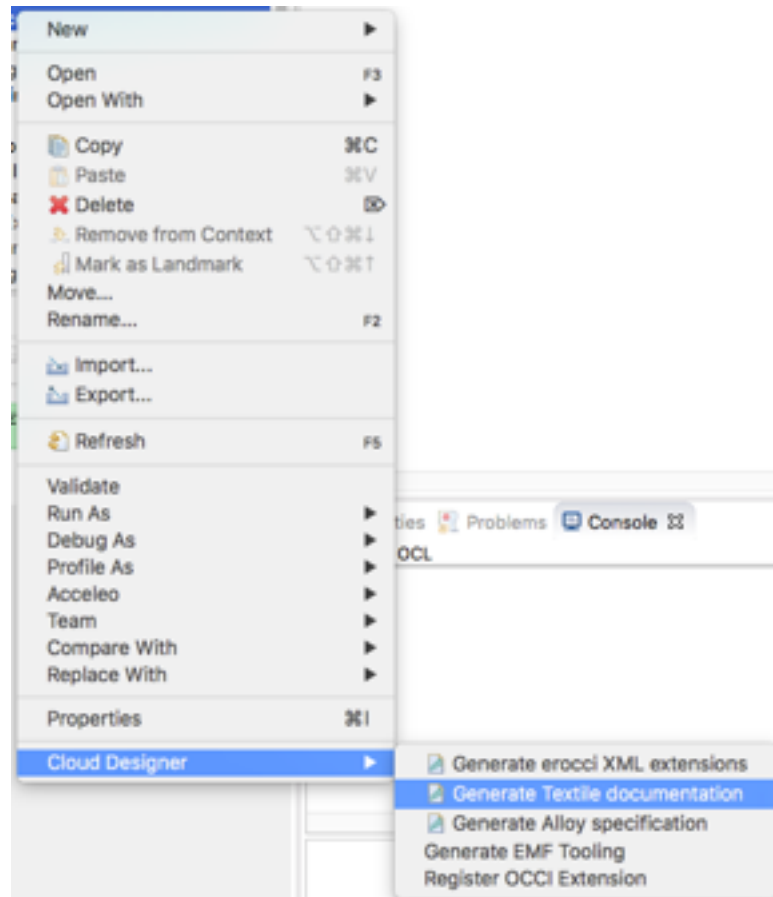


ETAPE 5 : Validation formelle avec Alloy

A compléter dans la future version du tutoriel

ETAPE 6 : Génération de la documentation

Pour générer la documentation de l'extension, il faut cliquer droit sur le fichier d'extension .occie (Model Explorer) puis sous menu **Cloud Designer** —> **Generate Textile documentation**. Cela va créer deux fichiers textile dans le répertoire src-gen —> **core.textile** et **light.textile**.



En double cliquant sur le fichier light.textile puis preview (en bas du workbench) on obtient :

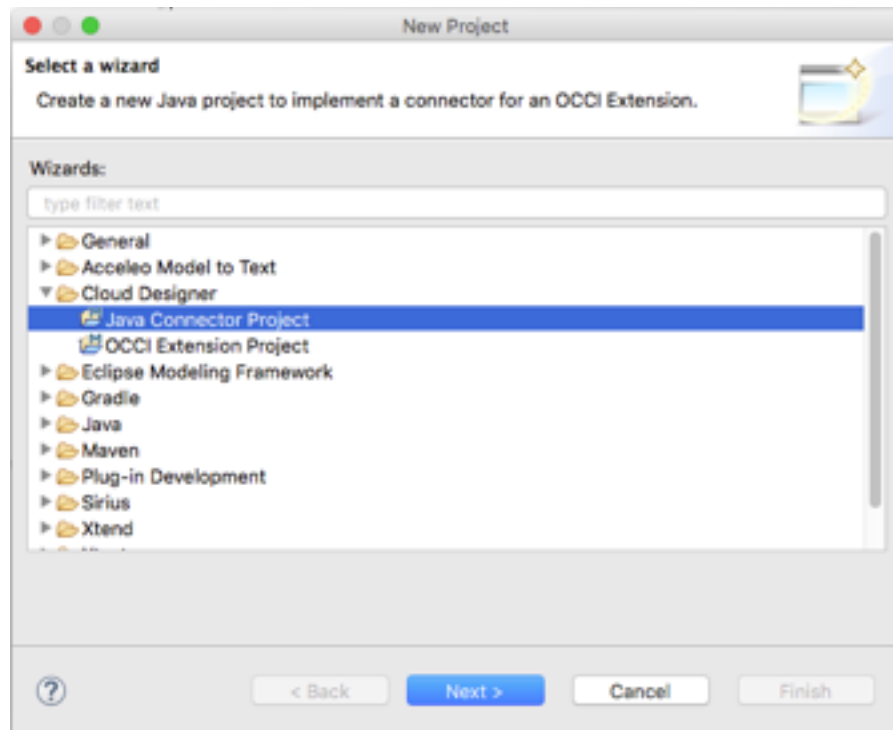


La documentation générée prend en compte les liens de parentés et génère les liens vers la documentation du parent.

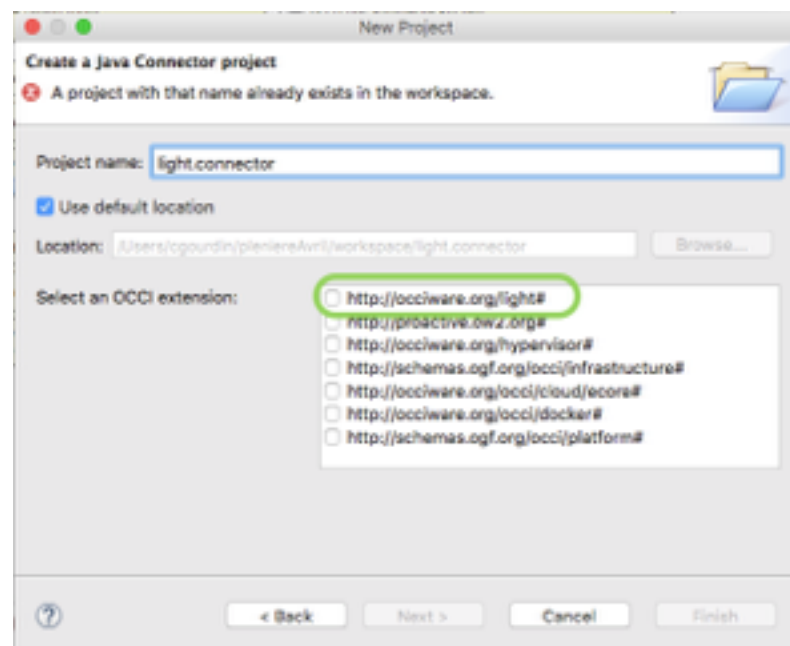
ETAPE 7 : Génération du connecteur java

Génération du projet connecteur light.connector

La génération du connecteur se fait via **CloudDesigner** → **New Project** → **Java Connector Project**.



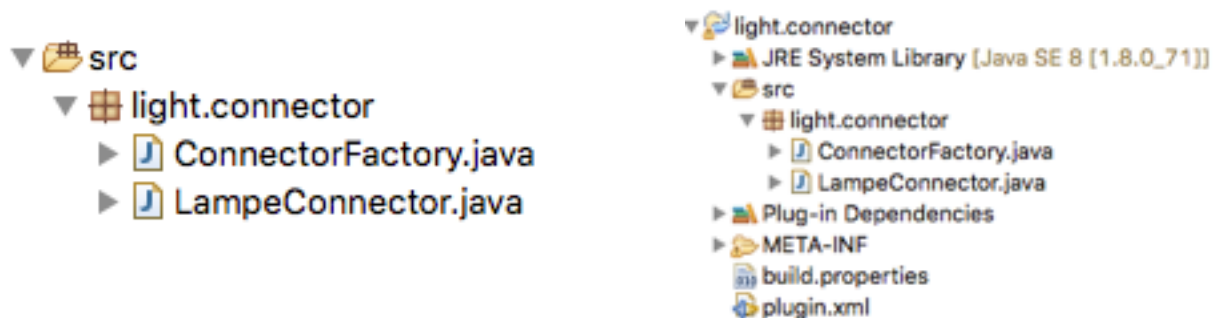
Cliquez sur **Next >** puis nommez le projet : **light.connector**



Créer un nouveau projet Java Connector avec l'extension **light#** :



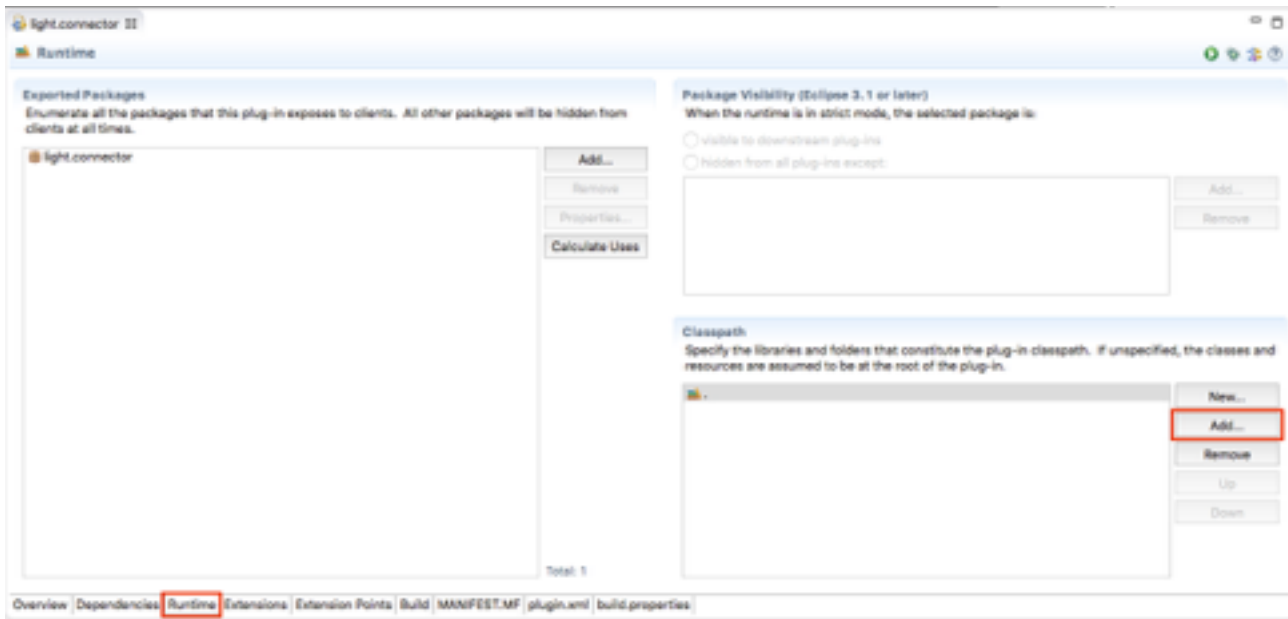
Sélectionnez l'extension **light#**, enfin cliquez sur le bouton **Finish**.
Le projet **light.connector** est créé avec les sources :



Préparation du connecteur

Ajoutez la bibliothèque client pour piloter le serveur LightServer :

- Télécharger le fichier **lightclient.jar** à l'adresse suivante :
 - <https://github.com/cgourdin/LightClientBuild/blob/master/lightclient.jar>
- Créez un répertoire **lib** dans projet **light.connector** et copiez **lightclient.jar** dans ce répertoire.
- Dans Cloud Designer, **ajoutez la bibliothèque au manifest pour le runtime et le build** :
 - Double cliquez sur le fichier **MANIFEST.MF** du répertoire **META-INF**, puis cliquez sur l'onglet **Runtime**
 - Cliquez sur le bouton **Add...**



- Cliquez sur **lightclient.jar** du répertoire **lib** puis **OK**
Cela va ajouter la bibliothèque au **classpath** déclaré dans le **manifest**.
- **Clean** and **build** sur le projet.



Mise à jour du code du connecteur

Mise à jour de la classe **LampeConnector** :

- Ajoutez la variable **LightClient** en privé global à la classe :

```
private LightClient lightClient = new LightClient("ws://localhost:8025/websocket/light");
```

Notez que l'adresse par défaut du serveur Light est **ws://localhost:8025/websocket/light**
LightClient est l'objet représentant l'API sur les light IOT. C'est un objet métier.

Callbacks OCCI

occiCreate() : Création d'une lampe dans l'application serveur LightServer.

Nous allons dans un premier temps récupérer les valeurs Id et location, ensuite exécuter l'action sur le serveur via la méthode **lightClient.createLight(String id, String location)**.

Implémentation dans la méthode occiCreate (override) :

```
@Override  
public void occiCreate() {  
    LOGGER.debug("occiCreate() called on " + this);  
    // Create a light with the light api.  
    try {  
        lightClient.createLight(this.getId(), this.getSummary());  
        // The state is automatically set to off by default.  
    } catch (IOException | TimeoutException ex) {  
        LOGGER.error("Error while creating a light : " + ex.getMessage());  
    }  
}
```

occiDelete() : Suppression d'une lampe via l'application serveur LightServer, l'action est exécutée via l'api **lightClient.deleteLight(String id)**.

```
@Override  
public void occiDelete() {  
    LOGGER.debug("occiDelete() called on " + this);  
    try {  
        lightClient.deleteLight(this.getId());  
    } catch (IOException | TimeoutException ex) {  
        LOGGER.error("Error while removing a light : " + ex.getMessage());  
    }  
}
```

occiUpdate() : Met à jour le label « location » sur l'application serveur LightServer, l'action est exécutée via l'API **lightClient.updateLightLocation(String id, String location)**.

```
@Override  
public void occiUpdate() {  
    LOGGER.debug("occiUpdate() called on " + this);  
    try {  
        lightClient.updateLightLocation(this.getId(), this.getSummary());  
    } catch (IOException | TimeoutException ex) {  
        LOGGER.error("Error while removing a light : " + ex.getMessage());  
    }  
}
```

Actions OCCl Light

switchOn() : Allume la lumière sur le serveur LightServer. Cette action est une action OCCl. Elle est exécutée via l'api lightClient.switchOn(String id).

```
/**
 * Implement OCCl action:
 * - scheme: http://occiware.org/light/lampe/action#
 * - term: switchOn
 * - title: Turn on the light
 */
@Override
public void switchOn()
{
    LOGGER.debug("Action switchOn() called on " + this);

    // Lampe State Machine.
    switch(getState().getValue()) {

        case State.ON_VALUE:
            LOGGER.debug("Fire transition(state=on, action=\"switchOn\")...");
            break;

        case State.OFF_VALUE:
            LOGGER.debug("Fire transition(state=off, action=\"switchOn\")...");

            try {
                lightClient.switchOn(this.getId());
                this.setState(State.ON);
            } catch (IOException e) {
                LOGGER.error("Error while turning ON the light " + this.getId() + " on
location : " + location + " , message: " + e.getMessage());
            } catch (TimeoutException e) {
                LOGGER.error("Error while turning ON the light " + this.getId() + " on
location : " + location + " , message: " + e.getMessage());
            }
            break;

        default:
            break;
    }
}
```

switchOff() : Eteint la lumière sur le serveur LightServer. Cette action est une action OCCI. Elle est exécutée via l'api lightClient.switchOff(String id).

```
/**
 * Implement OCCI action:
 * - scheme: http://occiware.org/light/lampe/action#
 * - term: switchOff
 * - title: Turn off the light
 */
@Override
public void switchOff()
{
    LOGGER.debug("Action switchOff() called on " + this);

    // Lampe State Machine.
    switch(getState().getValue()) {

        case State.ON_VALUE:
            LOGGER.debug("Fire transition(state=on, action=\"switchOff\")...");

            try {
                lightClient.switchOff(this.getId());
                this.setState(State.OFF);
            } catch (IOException e) {
                LOGGER.error("Error while turning OFF the light " + this.getId() + " on location : " +
location + " , message: " + e.getMessage());
            } catch (TimeoutException e) {
                LOGGER.error("Error while turning OFF the light " + this.getId() + " on location : " +
location + " , message: " + e.getMessage());
            }
            break;

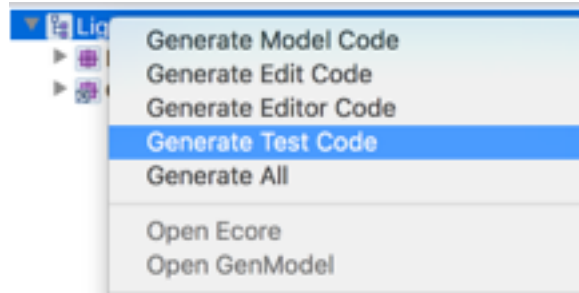
        case State.OFF_VALUE:
            LOGGER.debug("Fire transition(state=off, action=\"switchOff\")...");
            break;

        default:
            break;
    }
}
```

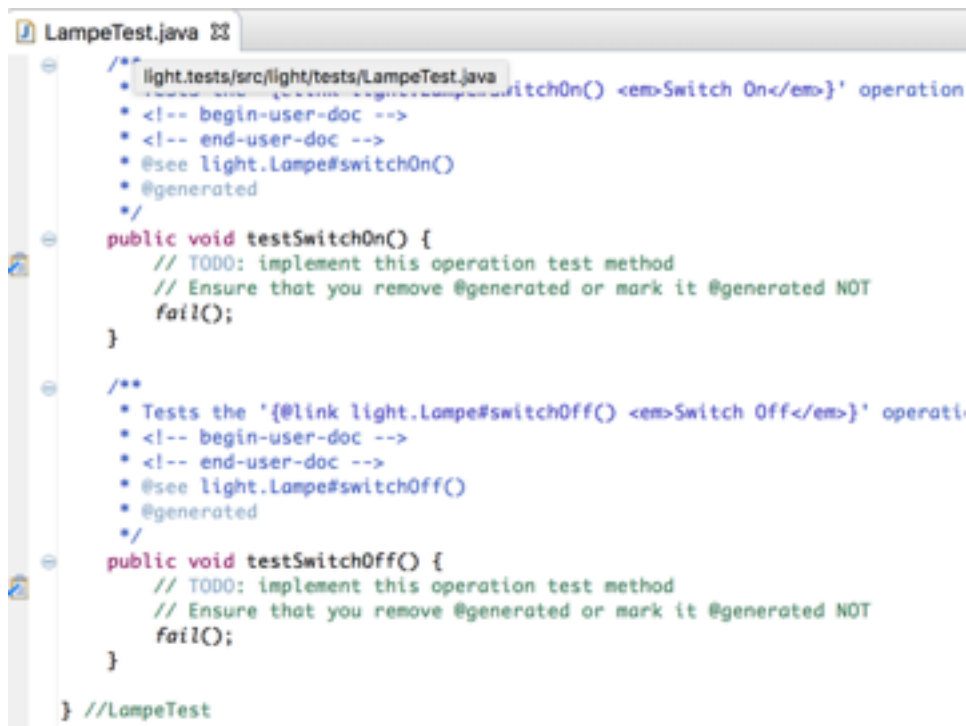
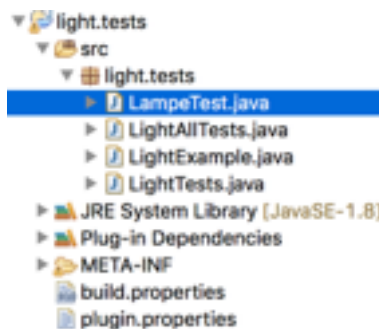

ETAPE 8 : Génération des tests JUNIT

Nous allons dans cette étape générer les tests unitaires de notre modèle d'extension **light**.

Pour réaliser cela, il faut double cliquer sur **light.genmodel** puis cliquez droit sur **light** de la vue **light.genmodel** (arborescente) puis cliquez sur **Generate test**.



Un nouveau projet est généré avec des classes de tests à compléter.



Implémentation du test unitaire d'une lampe

Dans LampeTest.java, ajoutez le code suivant dans la méthode **setUp()**, **tearDown()**, **testSwitchOff()**, **testSwitchOn()** :

```
@Override
protected void setUp() throws Exception {
    setFixture(LightFactory.eINSTANCE.createLampe());
    fixture.setId("light1");
    fixture.setSummary("kitchen");
    fixture.occiCreate();
}
```

```
@Override
protected void tearDown() throws Exception {
    fixture.occiDelete();
    setFixture(null);
}
```

```
public void testSwitchOff() {
    fixture.switchOff();
    assertTrue(fixture.getState() == State.OFF);
}
```

```
public void testSwitchOn() {
    fixture.switchOn();
    assertTrue(fixture.getState() == State.ON);
}
```

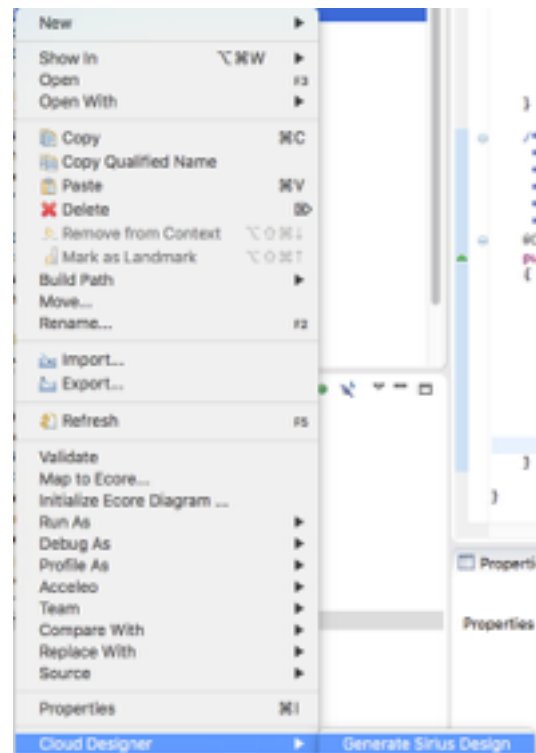
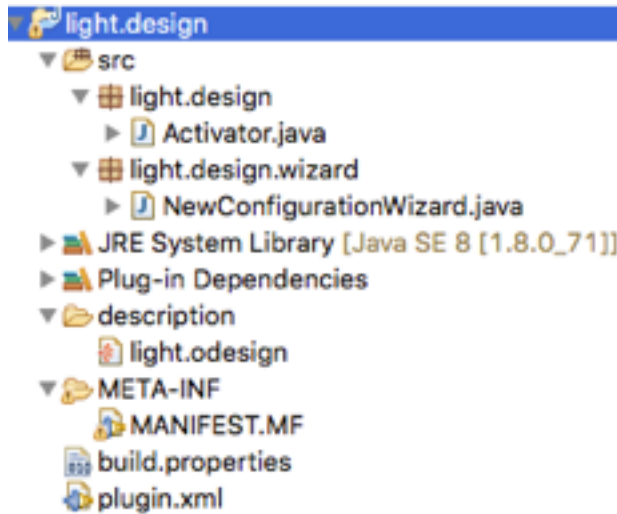
Pour lancer le test, sauvegardez, lancer **lightServer** s'il n'est pas lancé puis cliquez droit sur le fichier **LampeTest.java** -> **run as** -> **JUnit plugin test**

ETAPE 9 : Génération d'un designer dédié à l'extension Light

Génération du Light Designer

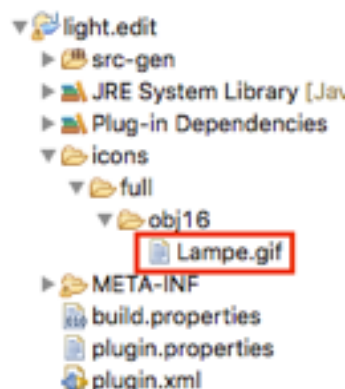
Dans le projet light, cliquez droit sur **light.ecore** puis cliquez sur Cloud Designer —> **Generate Sirius Design**

Un nouveau projet **light.design** est créé.

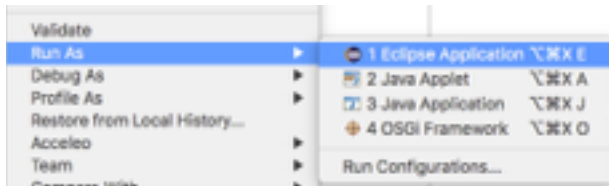


Afin de personnaliser votre designer, nous allons ajouter une icône représentant notre ressource

Dans le projet light.edit remplacez le fichier **Lampe.gif** par le nouveau **Lampe.gif** (sur la VM fournie, il est dans le répertoire **LightTutorial**) ou disponible sur github via l'adresse : <https://github.com/cgourdin/LightTutorial> .

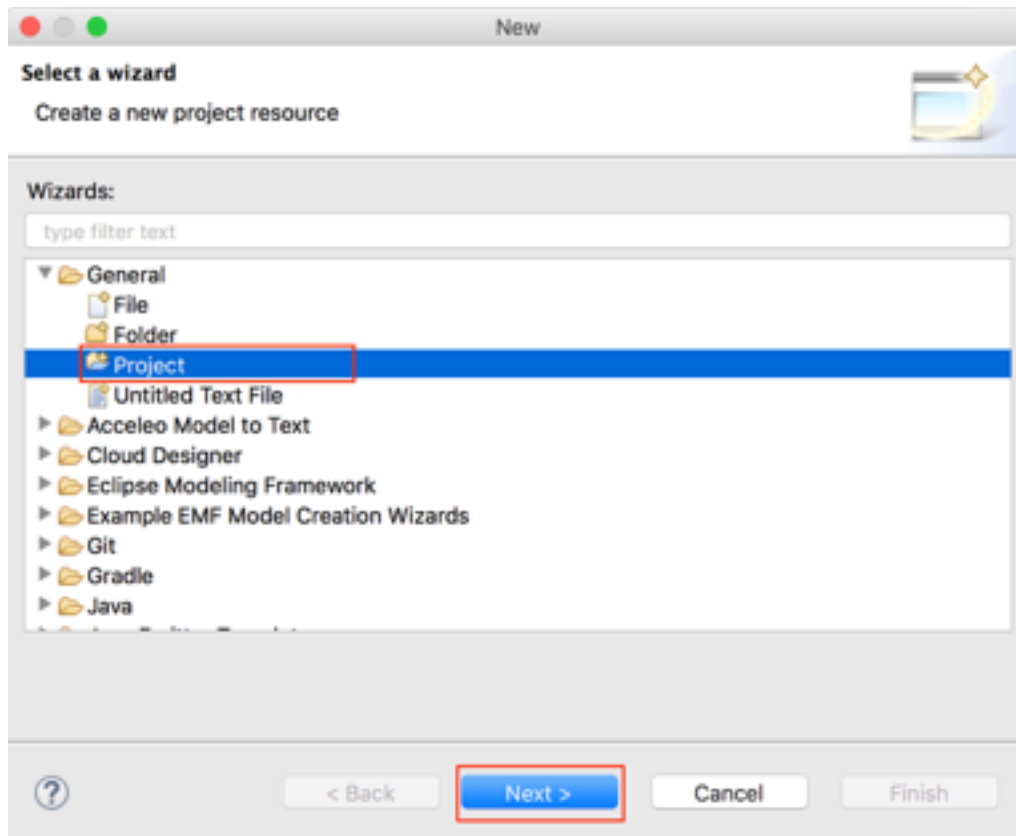


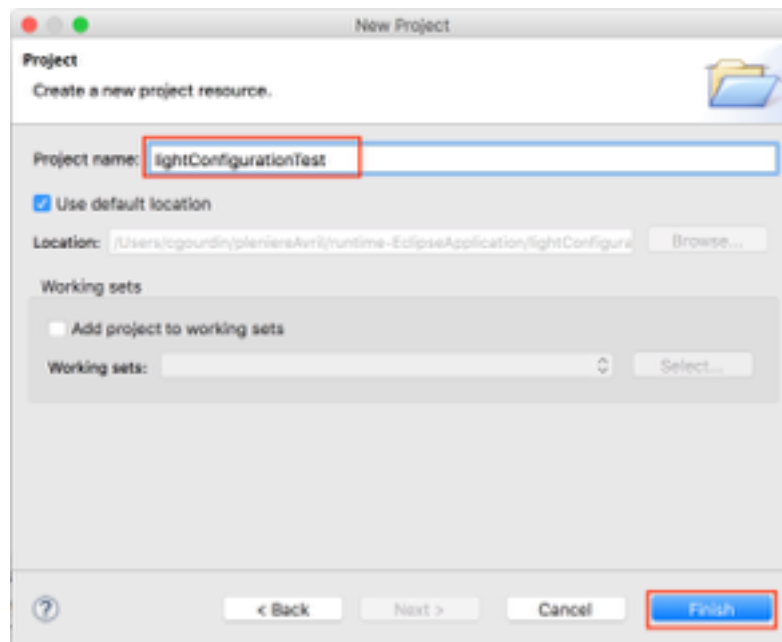
Cliquez droit sur **light.design** puis **Run As** —> **Eclipse Application**



Un nouvel environnement eclipse va s'ouvrir

Créez un nouveau projet (général) et donnez lui un nom, par exemple **lightConfigurationTest**.

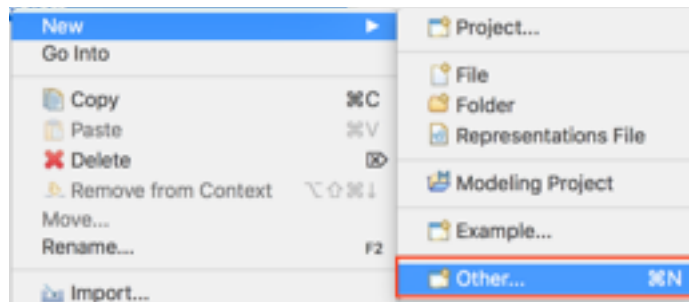




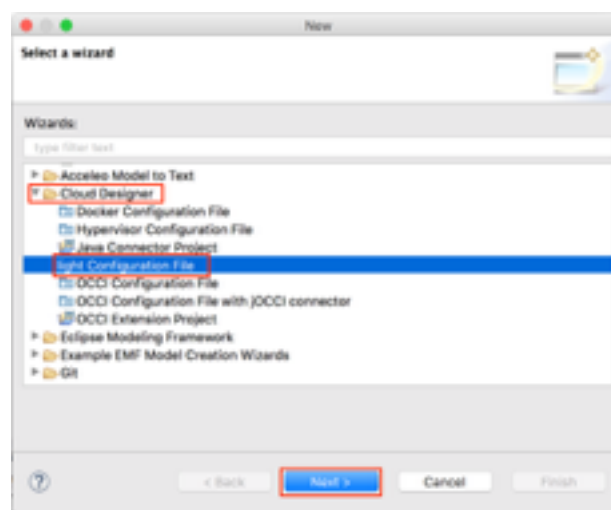
Enfin cliquez sur **Finish**.

Création d'une configuration minimale de test

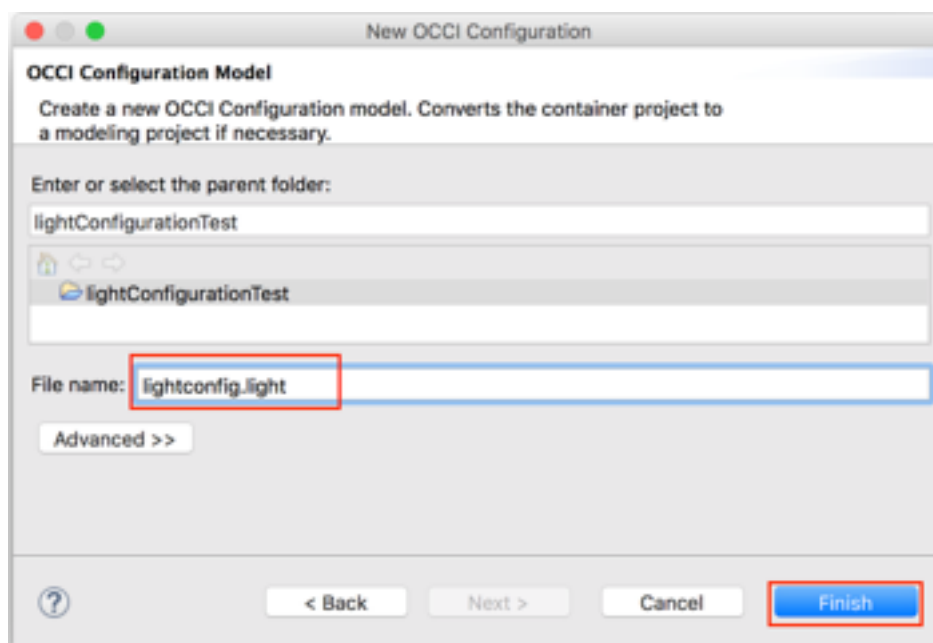
Créez un nouveau fichier de configuration « **light configuration** », cliquez droit sur une zone libre de Model Explorer puis **New** → **Other**



Déployez Cloud Designer puis cliquez sur **light Configuration File** puis cliquez sur **Next >**

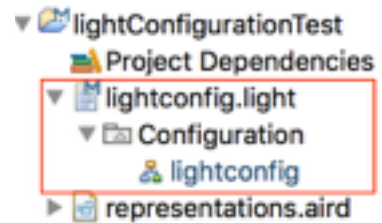


Nommez le fichier de configuration **lightconfig.light** puis cliquez sur **Finish**



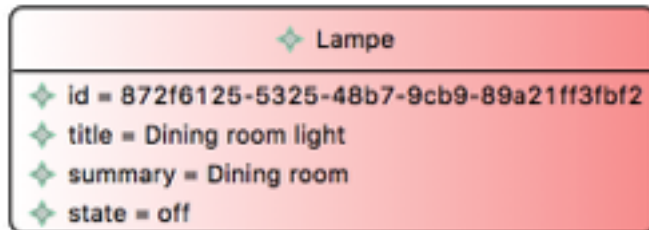
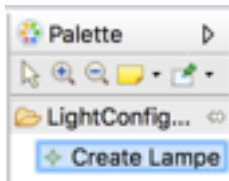
Vous devriez obtenir comme résultat dans Model Explorer :

Le fichier est normalement ouvert en mode designer.



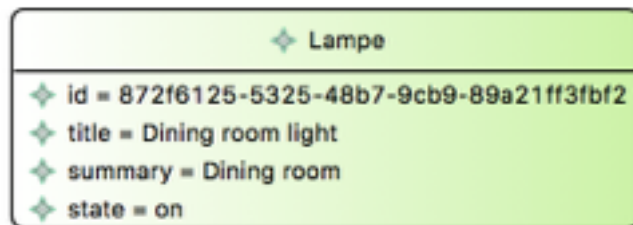
Création d'une lampe

Dans palette cliquez sur **Create Lampe** puis sur une zone libre du workbench.



Assignez les valeurs title et summary comme ci-dessus.

L'état par défaut est Off (0). Si vous changez la valeur state=on, notre lampe changera de couleur comme ceci :



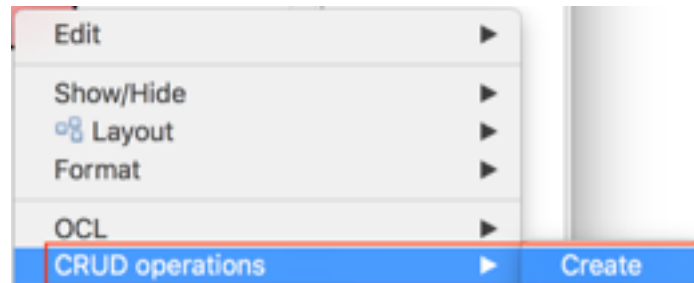
Repasser l'état à **Off**.

Lancement des actions (Create, switchOn, switchOff, Update, Delete)

Si ce n'est pas déjà fait lancez l'application **LightServer**.

Action de création :

Cliquez droit sur votre lampe puis **CRUD operations** —> **Create**

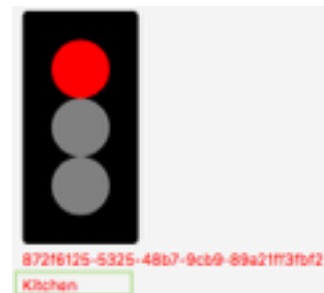


Normalement une **lampe** s'affiche sur l'application **LightServer** :



Action **Update** :

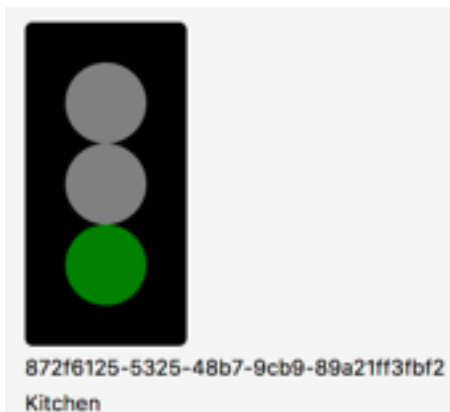
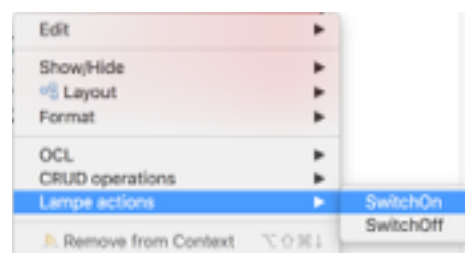
Changez la valeur de l'attribut summary en « **Kitchen** » puis Cliquez droit sur la lampe, **CRUD operations** —> **Update**
On s'aperçoit que **Kitchen** remplace la valeur **Dining room**.



Action **SwitchOn** :

Cliquez droit sur la lampe, **Lampe actions** puis cliquez sur **SwitchOn**.

La lampe s'allume dans **LightServer**



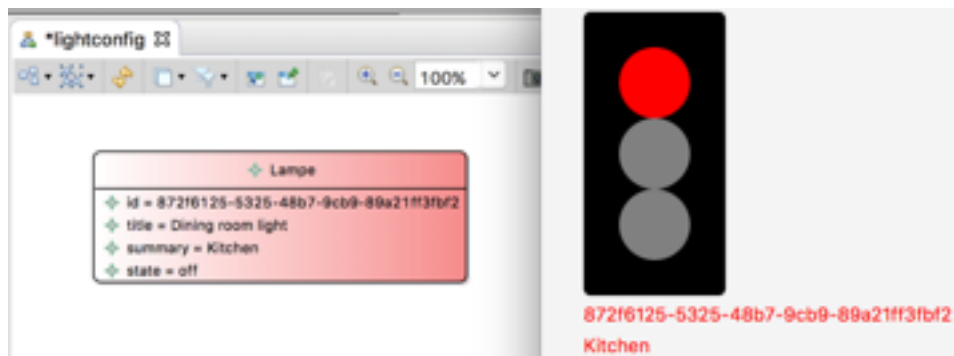
La lampe change d'état :



Action **SwitchOff** :

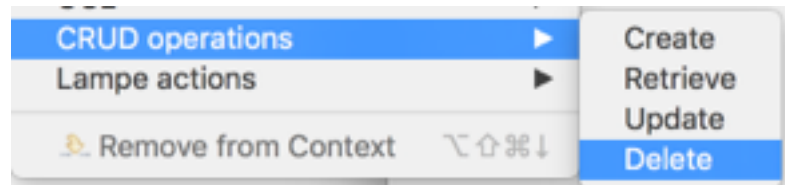
Cliquez droit sur la lampe, **Lampe actions** puis cliquez sur **SwitchOff**.

La lampe s'éteint dans **LightServer** :



Action de **suppression** :

Cliquez droit sur la lampe, **CRUD operations** puis cliquez sur **delete**. Normalement la lampe n'apparaît plus sur **LightServer**.

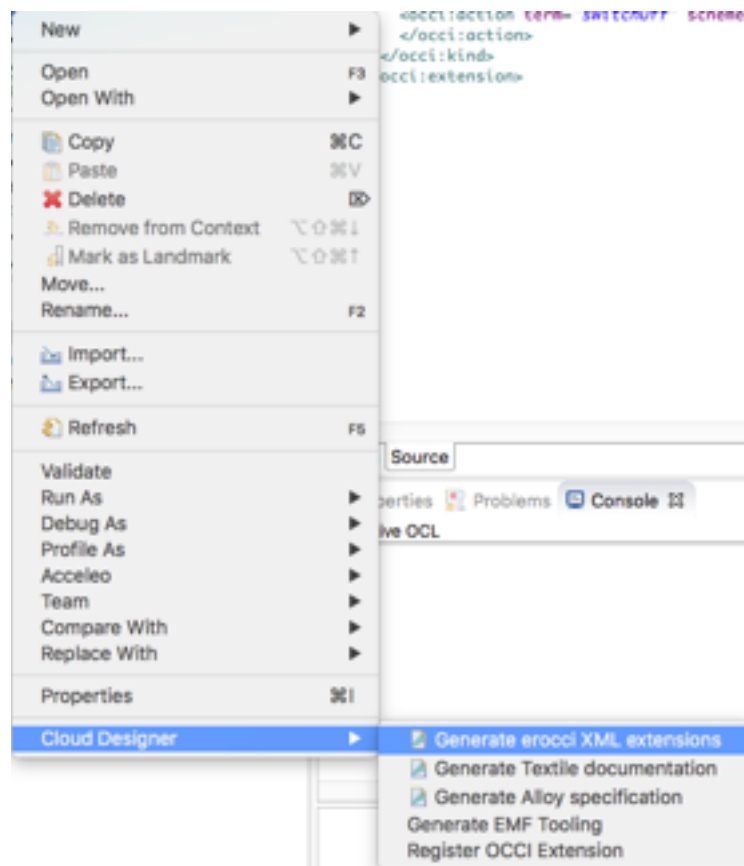


Nous avons ainsi testé les actions supportées par le connecteur **light.connector**.

Quittez l'application « **Light designer** », retour à **Cloud Designer**.

ETAPE 10 : Erocci dbus java backend

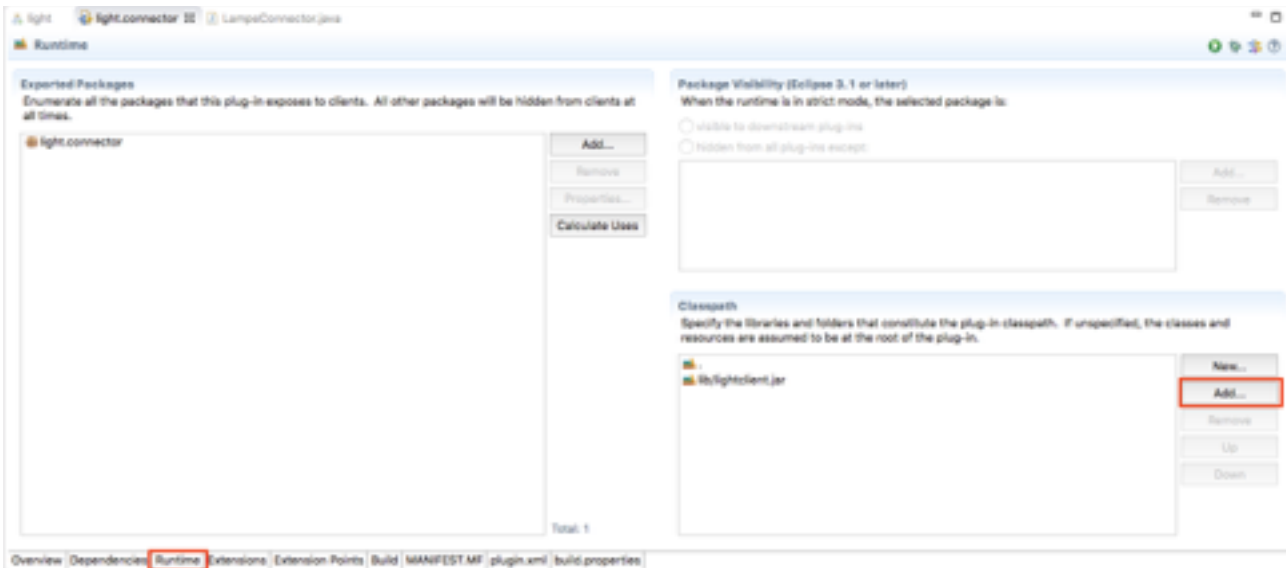
Pour générer le schema d'extension pour erocci, il faut cliquer droit sur le fichier d'extension .occie (Model Explorer) puis sous menu **Cloud Designer** —> **Generate erocci XML extensions**. Cela va créer un répertoire **erocci** dans **src-gen** avec les schemas XML. Son usage dans Erocci-dbus-backend est d'exposer le schéma à erocci pour qu'il puisse comprendre et valider les requêtes qu'il recevra pour la création des ressources comme pour l'exécution des actions (switchOn et switchOff).



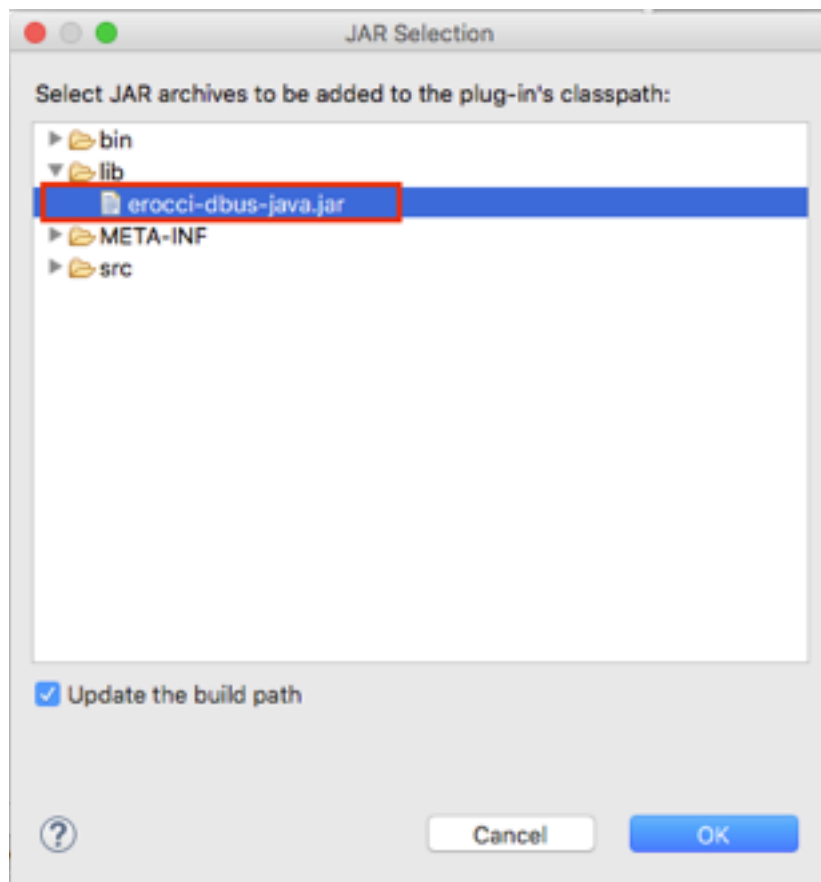
Dépendance erocci-dbus-java

Pour ajouter la dépendance **erocci-dbus-java.jar** au projet **light.connector** :

- Copiez la dépendance dans le répertoire **lib** du projet puis faite un **refresh** du projet

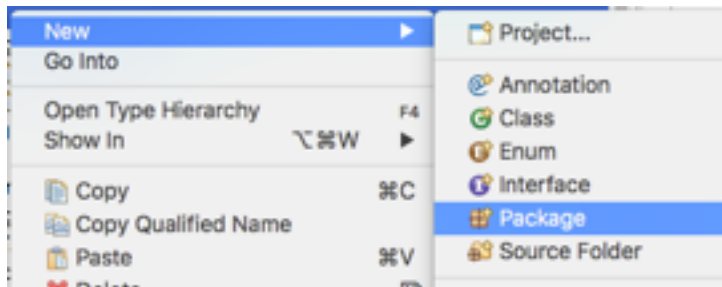


- Cliquez sur **Add** et déroulez le répertoire **lib** puis cliquez sur **erocci-dbus-java.jar**, enfin cliquez sur **OK**.

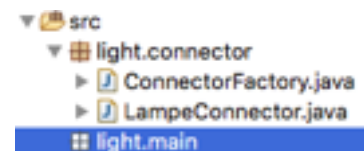
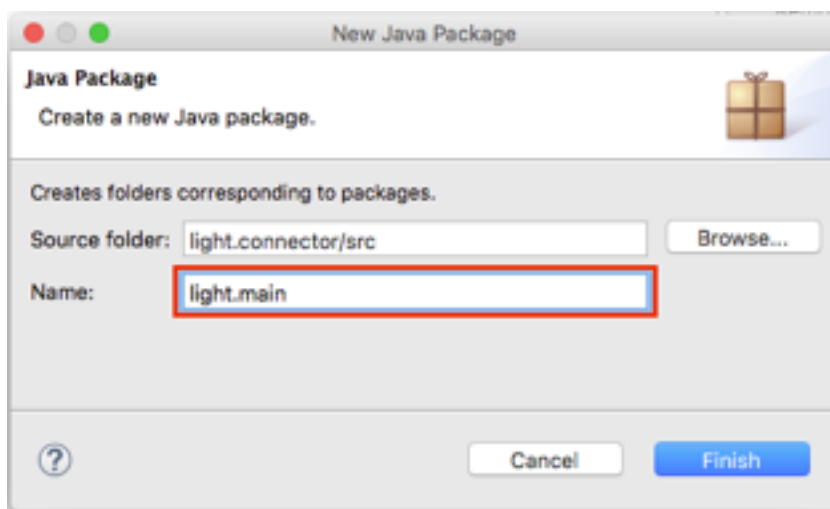


Appel du runtime erocci-dbus-java

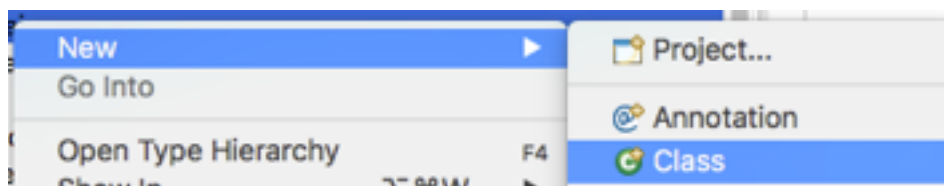
Créez un nouveau package **light.main** en cliquant droit sur le répertoire **src** du Model Explorer :



Saisissez le nom du package : **light.main** et cliquez **Finish**.

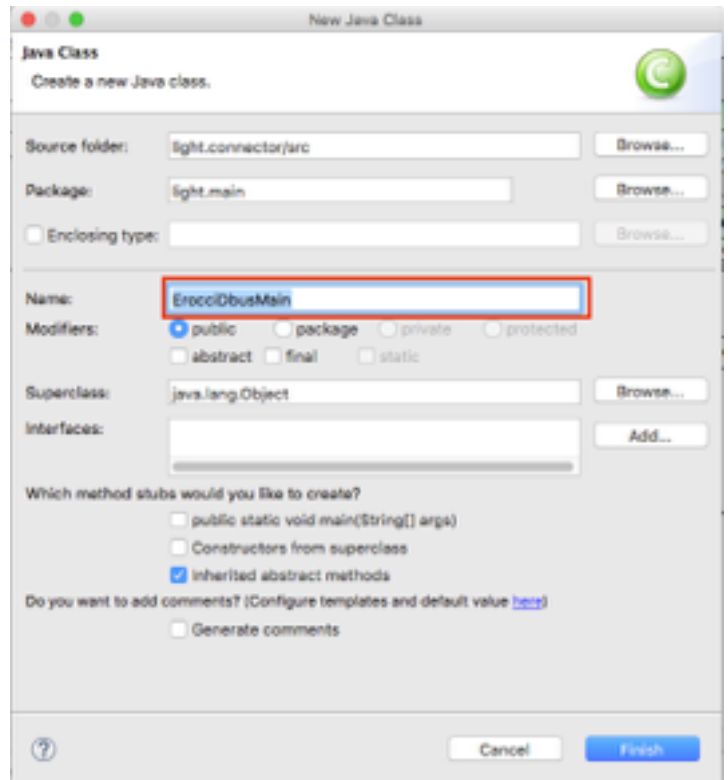


Cliquez droit sur **light.main** puis **New** —> **Class**



Donnez un nom à la classe : **ErocciDBusMain**

Cliquez sur **Finish**.



Ajoutez la méthode main comme suit :

`package light.main;`

`public class ErocciDBusMain {`

`public static void main(String[] args) throws Exception {`

`// Launch Erocci-dbus-java in the same context.`

`org.ow2.erocci.backend.BackendDBusService`

`.main(new String[]{"media/occiwareuser/homedisk/travail/workspace/light/`

`src-gen/erocci/light.xml"});`

`}`

`}`

Construction des jars light et light.connector

Afin que Erocci-dbus-java puissent charger les classes requises pour fonctionner avec l'extension Light, nous allons construire sous forme de jar les projets **light** et **light.connector**.

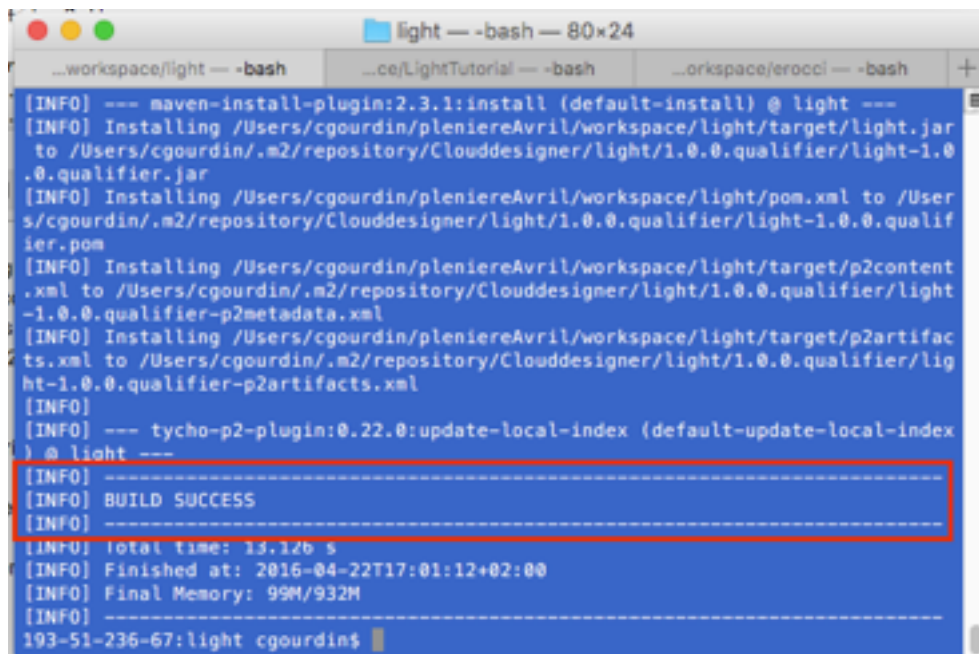
Attention, au préalable il faut récupérer les sources de CloudDesigner (partie ecore) via github : <https://github.com/occiware/ecore>

Construire le projet ecore avec la commande **mvn clean install** sur la racine du projet `org.occiware.clouddesigner.parent`, sinon à la construction du projet **light** une erreur de dépendance se lèvera.

Nous allons « maveniser » le projet :

- Téléchargez le fichier **pom.xml** pour light à l'adresse : <https://github.com/cgourdin/LightTutorial/blob/master/lightpom/pom.xml>
- Copiez le fichier dans le répertoire **racine** du projet **light**.
- Téléchargez le fichier **pom.xml** pour light.connector à l'adresse : <https://github.com/cgourdin/LightTutorial/blob/master/lightconnectorpom/pom.xml>
- Copiez le fichier dans le répertoire **racine** du projet **light.connector**.
- Dans un terminal exécutez la commande **mvn clean install** sur chaque projet (à la racine), en commençant par le projet **light**.

Vous devez obtenir comme message build success.



```
light - bash - 80x24
...workspace/light - bash
...ce/LightTutorial - bash
...orkspace/erocci - bash

[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ light ---
[INFO] Installing /Users/cgourdin/pleniereAvril/workspace/light/target/light.jar
to /Users/cgourdin/.m2/repository/Clouddesigner/light/1.0.0.qualifier/light-1.0
.0.qualifier.jar
[INFO] Installing /Users/cgourdin/pleniereAvril/workspace/light/pom.xml to /User
s/cgourdin/.m2/repository/Clouddesigner/light/1.0.0.qualifier/light-1.0.0.qualif
ier.pom
[INFO] Installing /Users/cgourdin/pleniereAvril/workspace/light/target/p2content
.xml to /Users/cgourdin/.m2/repository/Clouddesigner/light/1.0.0.qualifier/light
-1.0.0.qualifier-p2metadata.xml
[INFO] Installing /Users/cgourdin/pleniereAvril/workspace/light/target/p2artifac
ts.xml to /Users/cgourdin/.m2/repository/Clouddesigner/light/1.0.0.qualifier/lig
ht-1.0.0.qualifier-p2artifacts.xml
[INFO] --- tycho-p2-plugin:0.22.0:update-local-index (default-update-local-index
) @ light ---
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.126 s
[INFO] Finished at: 2016-04-22T17:01:12+02:00
[INFO] Final Memory: 99M/932M
[INFO] -----
193-51-236-67:light cgourdin$
```

Effectuez la même manipulation pour **light.connector**.

Référencement au classpath de Erocci-dbus-java

De retour dans **Cloud Designer**, cliquez droit sur **ErocciDbusMain.java** —> **Run As** —> **Run Configurations...**

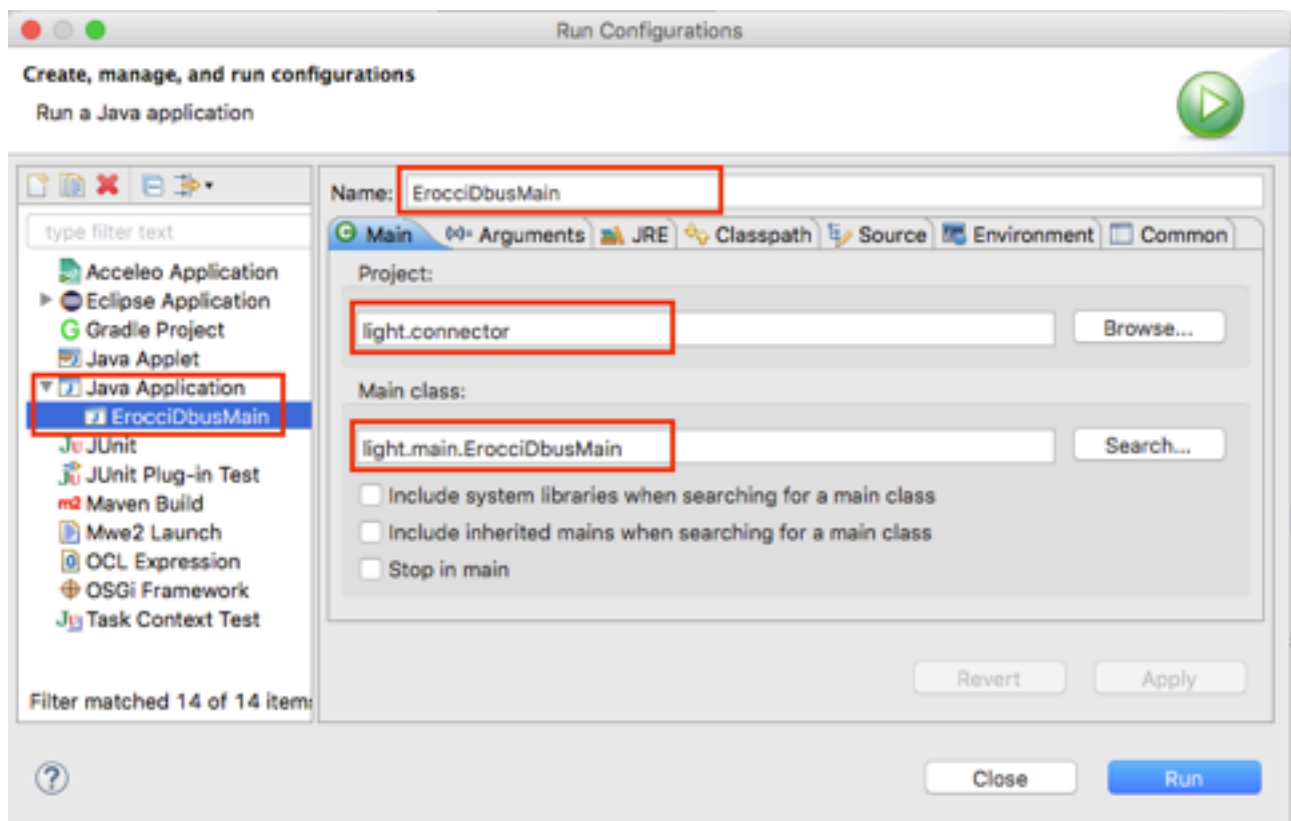


Cliquez sur **New_configuration** puis onglet **Main**.

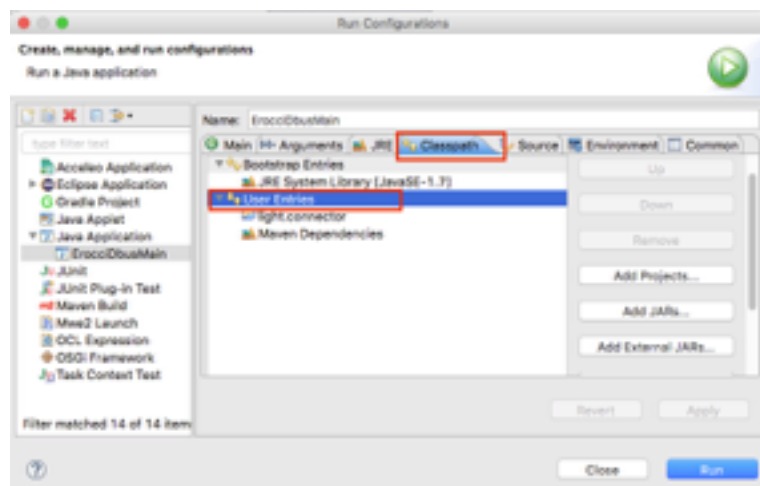
Renseignez un nom : **ErocciDbusMain**

Vérifiez que le projet est bien le votre « **light.connector** »

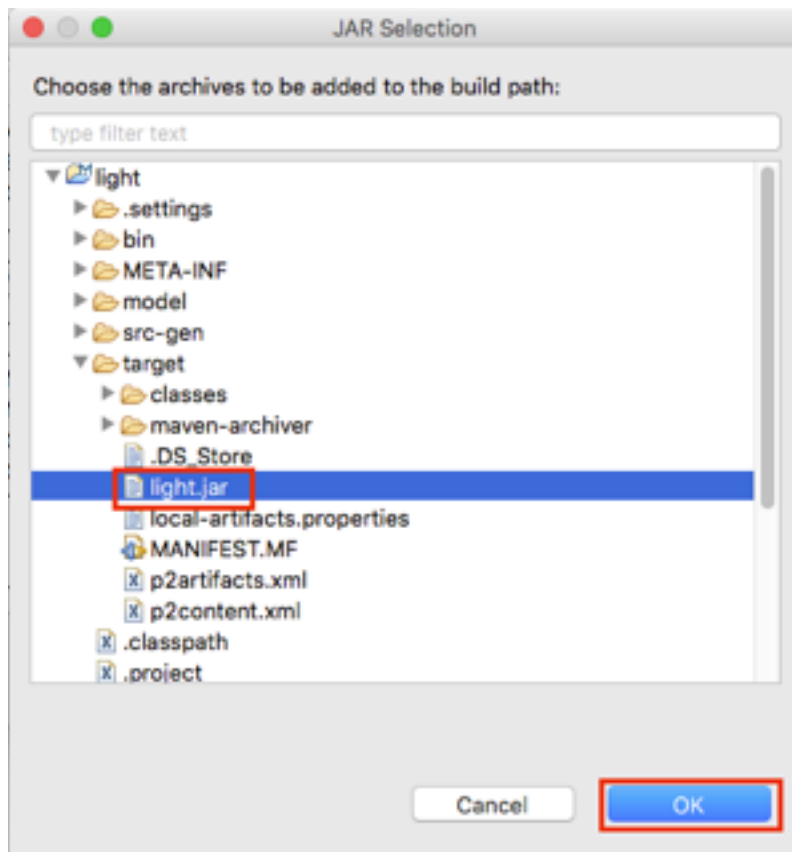
Renseignez la classe Main : **light.main.ErocciDbusMain**



Cliquez sur l'onglet **Classpath** puis sur **User entries** et cliquez sur le bouton **Add JARs...**



- Cliquez sur **Add JARs...**
- Cliquez sur **light.jar** du répertoire **target** du projet **light** puis sur **OK**



- Faites de même avec le projet **light.connector** (**lightconnector.jar**)
- Enfin cliquez sur **Run**, cela aura pour effet de lancer le main du connecteur qui lui même lancera **Erocci-dbus-java**, avec le **schema** xml pour **Erocci**.

Au lancement, la console indique des éléments importants :

INFOS: EMBED mode enabled

INFOS: Connected to dbus with unique name : :1.76

Cela permet de savoir que Erocci-dbus-java est bien connecté à DBus (mode Session).

Notez bien que DBus ne doit pas afficher de message d'erreur à ce niveau, si vous avez un message comme ceci : **AVERTISSEMENT: Error while connecting to DBUS !**, l'application ne communiquera pas avec Erocci.

ETAPE 11 : Lancement erocci

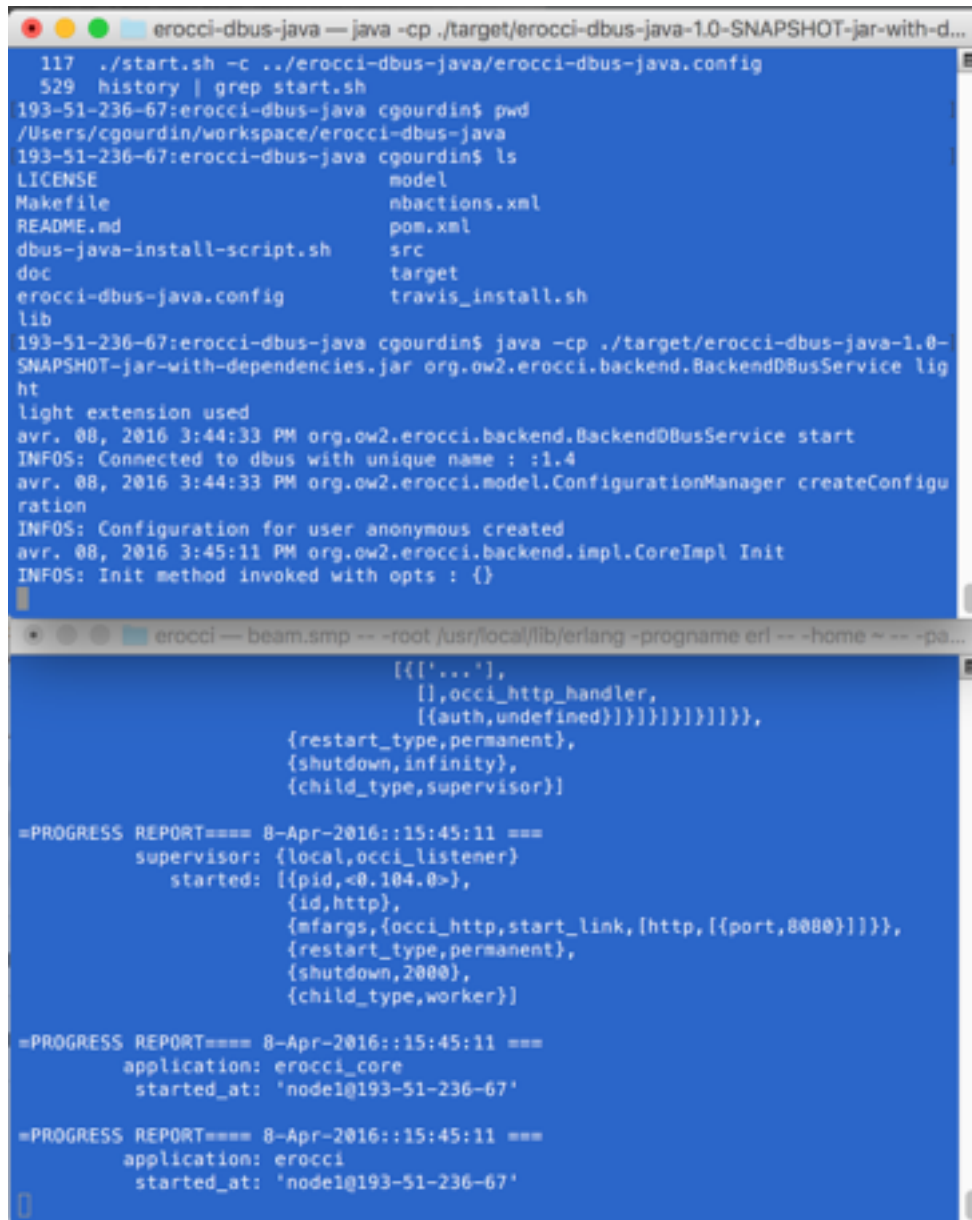
Lancement erocci

Positionnez vous dans le répertoire racine où vous avez installé **erocci**.

Modifiez la ligne suivante avec le chemin de votre installation de Erocci-dbus-java si nécessaire :

`./start.sh -c ../erocci-dbus-java/erocci-dbus-java.config`

Pour savoir si erocci s'est exécuté correctement (en haut backend dbus, en bas erocci):



```
erocci-dbus-java — java -cp ./target/erocci-dbus-java-1.0-SNAPSHOT-jar-with-d...
117 ./start.sh -c ../erocci-dbus-java/erocci-dbus-java.config
529 history | grep start.sh
193-51-236-67:erocci-dbus-java cgourdin$ pwd
/Users/cgourdin/workspace/erocci-dbus-java
193-51-236-67:erocci-dbus-java cgourdin$ ls
LICENSE                                model
Makefile                              nbactions.xml
README.md                             pom.xml
dbus-java-install-script.sh           src
doc                                    target
erocci-dbus-java.config                travis_install.sh
lib
193-51-236-67:erocci-dbus-java cgourdin$ java -cp ./target/erocci-dbus-java-1.0-
SNAPSHOT-jar-with-dependencies.jar org.ow2.erocci.backend.BackendDBusService lig
ht
light extension used
avr. 08, 2016 3:44:33 PM org.ow2.erocci.backend.BackendDBusService start
INFOS: Connected to dbus with unique name : :1.4
avr. 08, 2016 3:44:33 PM org.ow2.erocci.model.ConfigurationManager createConfigu
ration
INFOS: Configuration for user anonymous created
avr. 08, 2016 3:45:11 PM org.ow2.erocci.backend.impl.CoreImpl Init
INFOS: Init method invoked with opts : {}

erocci — beam.smp -- -root /usr/local/lib/erlang -programe erl -- -home ~ -- -pa...
[[['...'],
 [],occi_http_handler,
 [{auth,undefined}}]]]]]]],
 {restart_type,permanent},
 {shutdown,infinity},
 {child_type,supervisor}]

=PROGRESS REPORT=== 8-Apr-2016::15:45:11 ===
supervisor: {local,occi_listener}
started: [{pid,<0.104.0>},
 {id,http},
 {mfargs,{occi_http,start_link,[http,{port,8080}]}}],
 {restart_type,permanent},
 {shutdown,2000},
 {child_type,worker}]

=PROGRESS REPORT=== 8-Apr-2016::15:45:11 ===
application: erocci_core
started_at: 'node1@193-51-236-67'

=PROGRESS REPORT=== 8-Apr-2016::15:45:11 ===
application: erocci
started_at: 'node1@193-51-236-67'
```

Erocci a bien fait sa demande de **schéma**, pour vérifier que le schéma est pris en compte correctement, il faut remonter dans la console où erocci a été **lancé** jusqu'aux traces de register :



```
erocci -- beam.smp -- -root /usr/local/lib/erlang -programe erl -- -home ~ -- -pa...
=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Load extension: light v1

=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Load kind: http://occiware.org/light#light

=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Load attribute spec: occi.light.state

=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Load action spec: {occi_cid,<<"http://occiware.org/light/light/action#">>,
                  <<"switchOn">>,action}

=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Load action spec: {occi_cid,<<"http://occiware.org/light/light/action#">>,
                  <<"switchOff">>,action}

=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Loaded extension: light

=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Registering kind: {occi_cid,<<"http://occiware.org/light#">>,<<"light">>,kind} -
> {uri,

    undefined,

    [],

    [],

    undefined,

    "/collections/light/",

    []}

=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Registering action: {occi_cid,<<"http://occiware.org/light/light/action#">>,
                  <<"switchOn">>,action}

=INFO REPORT==== 8-Apr-2016::15:45:11 ===
Registering action: {occi_cid,<<"http://occiware.org/light/light/action#">>,
                  <<"switchOff">>,action}

=PROGRESS REPORT==== 8-Apr-2016::15:45:11 ===
    supervisor: {local,occi_store}
    started: [{pid,<0.96.0>},
              {id,testjava},
              {mfargs,
               {occi_backend,start_link,
```

Nous pouvons croire que le schéma a bien été pris en compte à partir de la trace

« **Load extension : light v1** »

Nous nous apercevons aussi que les **attributs** et **actions** (kind) sont chargés.

Une dernière vérification permet de contrôler la connexion DBUS entre les deux applications avec une simple requête **curl** : (remplacer localhost par une autre adresse si besoin)

curl -v -H "Accept: application/json" http://localhost:8080/collections/entity/

ou cette requête pour voir l'ensemble des ressources supportées par erocci :

curl -v -H "Accept: application/json" -X GET http://localhost:8080/-/

Le résultat doit donner :

```
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /-/ HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.43.0
> Accept: application/json
>
< HTTP/1.1 200 OK
< date: Mon, 25 Apr 2016 09:46:30 GMT
< content-length: 2266
< server: erocci OCCI/1.1
< content-type: application/json
< vary: accept
<
{
  "kinds" : [
    {
      "term" : "entity",
      "scheme" : "http://schemas.ogf.org/occi/core#",
      "class" : "kind",
      "title" : "Core Entity",
      "attributes" : {
        "occi.core.id" : {
          "mutable" : true,
          "title" : "Id",
          "required" : false,
          "type" : "string"
        },
        "occi.core.title" : {
          "mutable" : true,
          "title" : "Display name",
          "required" : false,
          "type" : "string"
        }
      },
      "location" : "http://localhost:8080/collections/entity/"
    },
    {
      "term" : "link",
      "scheme" : "http://schemas.ogf.org/occi/core#",
      "class" : "kind",
      "title" : "Core Link",
      "parent" : "http://schemas.ogf.org/occi/core#entity",
      "location" : "http://localhost:8080/collections/link/"
    },
    {
      "term" : "resource",
```

```

    "scheme" : "http://schemas.ogf.org/occi/core#",
    "class" : "kind",
    "title" : "Core Resource",
    "parent" : "http://schemas.ogf.org/occi/core#entity",
    "attributes" : {
      "occi.core.summary" : {
        "mutable" : true,
        "title" : "Summarising description",
        "required" : false,
        "type" : "string"
      }
    },
    "location" : "http://localhost:8080/collections/resource/"
  },
  {
    "term" : "lampe",
    "scheme" : "http://occiware.org/light#",
    "class" : "kind",
    "title" : "A light resource",
    "parent" : "http://schemas.ogf.org/occi/core#resource",
    "attributes" : {
      "occi.light.state" : {
        "mutable" : true,
        "required" : true,
        "type" : "string"
      }
    },
    "actions" : [
      "http://occiware.org/light/lampe/action#switchOn",
      "http://occiware.org/light/lampe/action#switchOff"
    ],
    "location" : "http://localhost:8080/collections/lampe/"
  }
],
"mixins" : [

],
"actions" : [
  {
    "term" : "switchOff",
    "scheme" : "http://occiware.org/light/lampe/action#",
    "class" : "action",
    "title" : "Turn the light off"
  },
  {
    "term" : "switchOn",
    "scheme" : "http://occiware.org/light/lampe/action#",
    "class" : "action",
    "title" : "Turn the light on"
  }
]
}

```

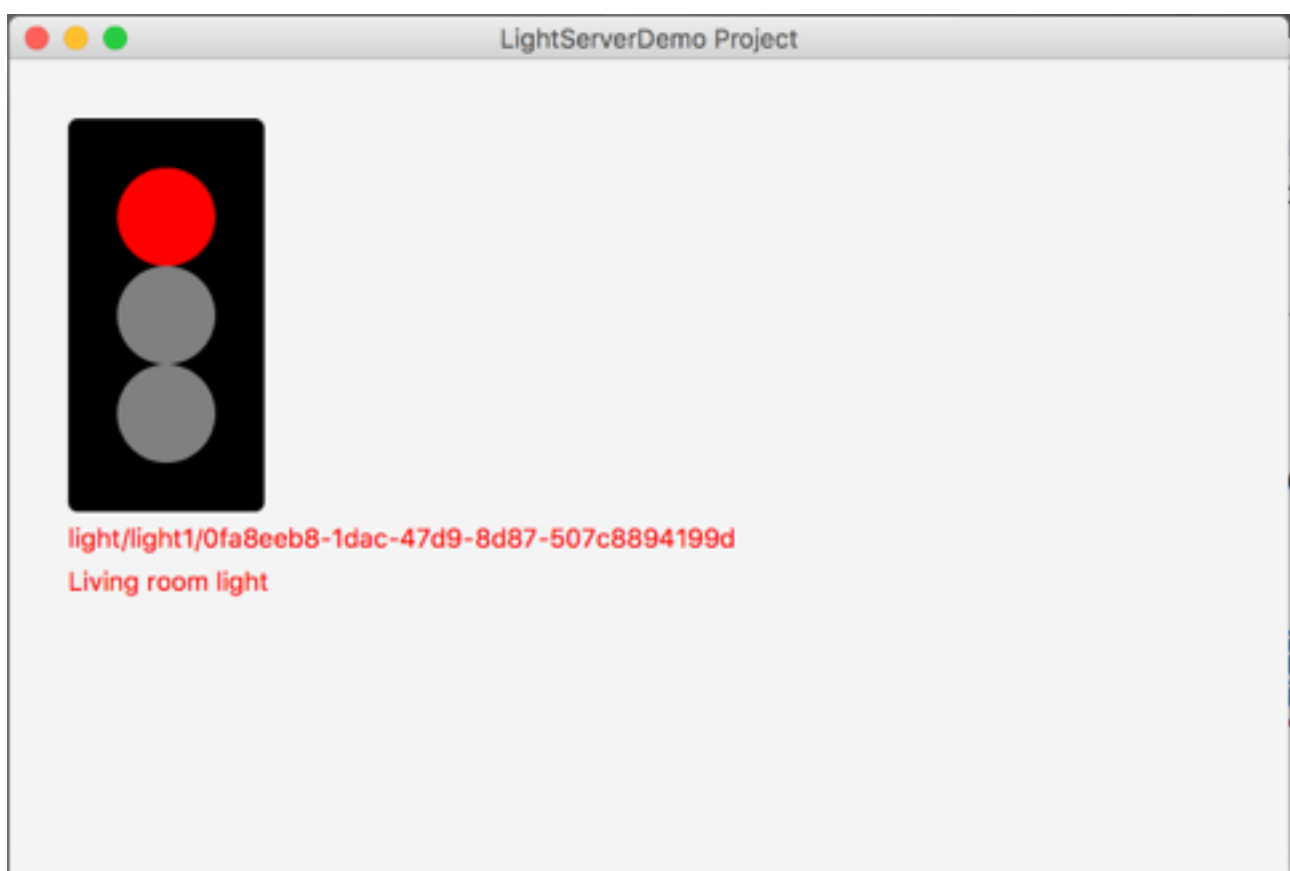
ETAPE 12 : Utilisation des lampes IOT via curl manuel

Lancement de la création d'une lampe dans un terminal, au préalable lancez l'application **LightServer** :

Nous allons créer une lampe avec l'identifiant : **light/light1/c1568df5-250f-45d4-ae1-ce053f186b14**

```
curl -v -s -i -X PUT http://localhost:8080/light/light1/c1568df5-250f-45d4-ae1-ce053f186b15 -  
H 'Content-Type: text/occi' -H 'Category: lampe; scheme="http://occiware.org/light#";  
class="kind";' -H 'X-OCCL-Attribute: occi.core.summary="Living room light"'
```

Normalement l'application **LightServer** doit matérialiser la ressource comme ceci :



Requête CURL pour l'action switchOn :

```
curl -v -X POST --header 'Content-type:text/occi' http://localhost:8080/light/light1/  
c1568df5-250f-45d4-ae1-ce053f186b15?action=switchOn -H 'Category: switchOn;  
scheme="http://occiware.org/light/lampe/action#"; class="action"'
```

Requête CURL pour l'action switchOff :

```
curl -v -X POST --header 'Content-type:text/occi' http://localhost:8080/light/light1/  
c1568df5-250f-45d4-ae1-ce053f186b15?action=switchOff -H 'Category: switchOff;  
scheme="http://occiware.org/light/lampe/action#"; class="action"'
```

Requête CURL pour la mise à jour, ici on modifie sa localisation matérialisée par l'attribut `occi.core.summary` :

```
curl -s -v -i -X POST http://localhost:8080/light/light1/c1568df5-250f-45d4-aee1-ce053f186b15 -H 'Content-Type: text/occi' -H 'Category: lampe; scheme="http://occiware.org/light#"; class="kind";' -H 'X-OCCL-Attribute: occi.core.summary="Kitchen"'
```

Lister les collections d'entité :

```
curl -v -H "Accept: application/json" http://localhost:8080/collections/entity/
```

Lister les entités dont la catégorie est **lampe** :

```
curl -v -H "Accept: application/json" -X GET http://localhost:8080/collections/lampe/
```

Détail d'une entité avec en sortie un format JSON :

```
curl -H "Accept: application/json" http://localhost:8080/light/light1/c1568df5-250f-45d4-aee1-ce053f186b15/
```

Cela donne comme résultat :

```
{
  "resources" : [
    {
      "kind" : "http://occiware.org/light#lampe",
      "id" : "/light/light1/c1568df5-250f-45d4-aee1-ce053f186b15",
      "attributes" : {
        "occi.core.id" : "http://localhost:8080/light/light1/c1568df5-250f-45d4-aee1-ce053f186b15",
        "occi.core.summary" : "Kitchen",
        "occi.light.state" : "on"
      }
    }
  ]
}
```

Supprimer une entité :

```
curl -H "Accept: application/json" -X DELETE http://localhost:8080/light/light1/c1568df5-250f-45d4-aee1-ce053f186b15
```

Les requêtes ci-dessus sont de bas niveau, pour la création des ressources et leur management nous pouvons effectuer toutes ces actions soit via Cloud Designer soit via JOCCI qui se chargera de communiquer avec Erocci .

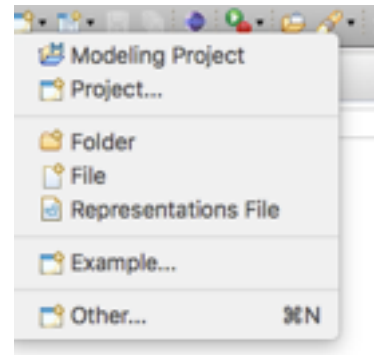
De plus les requêtes de création de ressource peuvent être générées et exécutées pour un usage ultérieur comme nous le verrons à l'étape suivante.

ETAPE 13 : Modélisation de la configuration

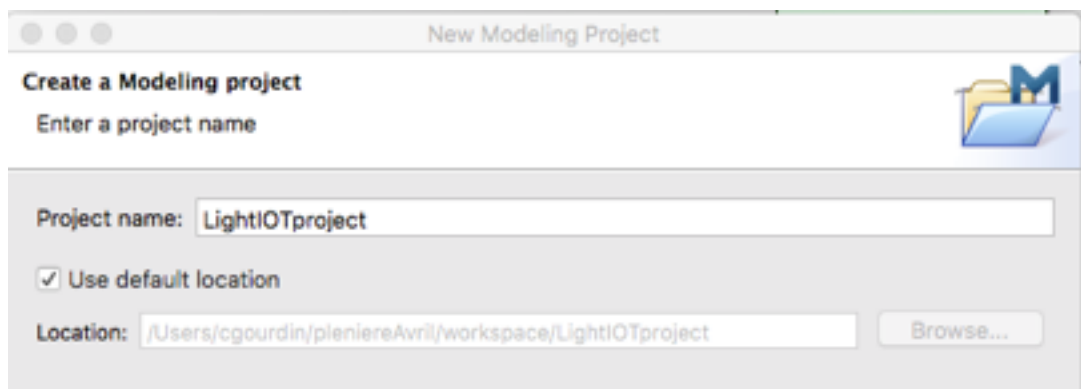
Création du projet LightIOTProject

Créez le projet **LightIOTProject** :

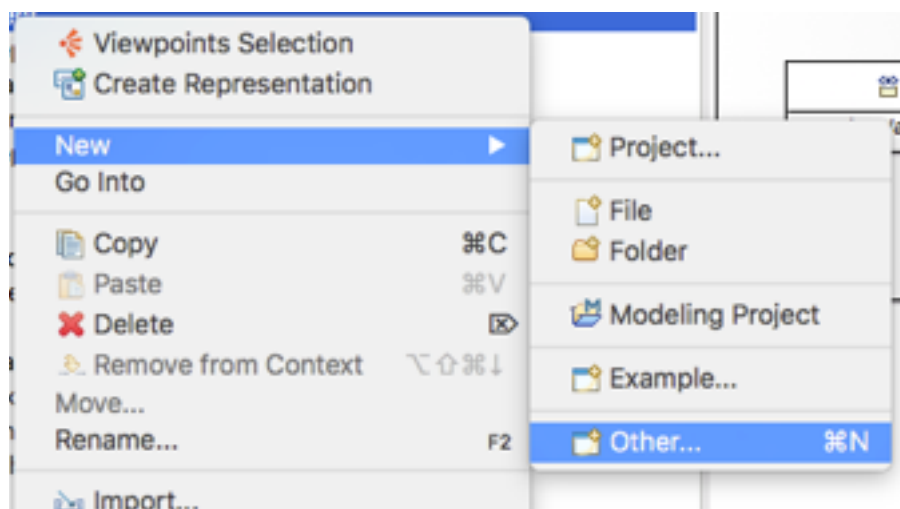
Clic sur l'icône représentant une flèche comme suit :



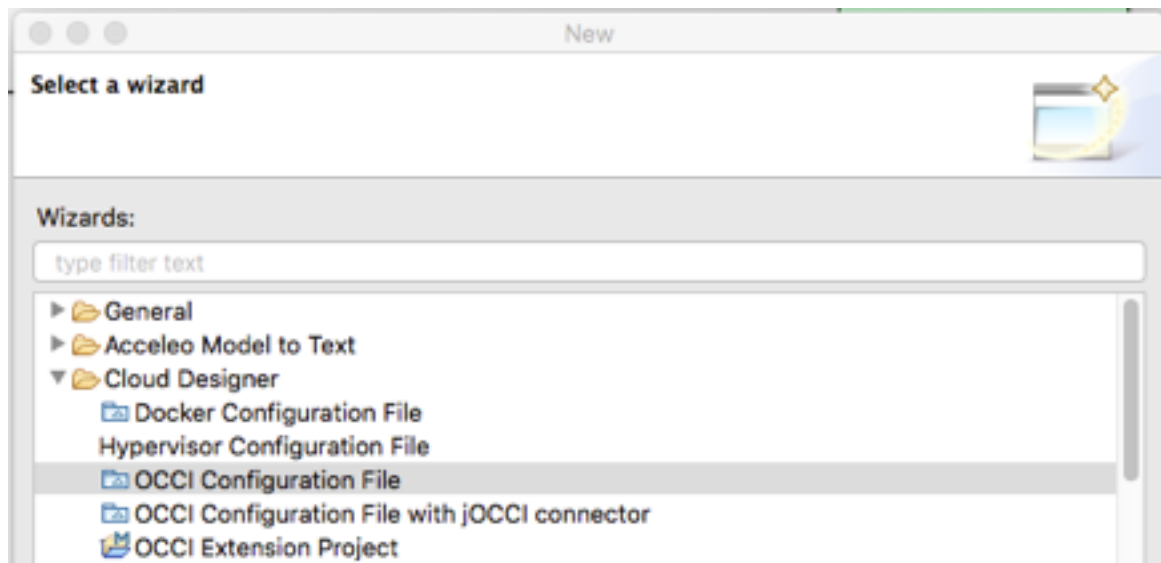
Cliquez sur « **Modeling Project** », saisissez le nom du projet puis cliquez sur **Finish**.



Cliquez droit sur le nom **LightIOTproject** :



Sélectionnez OCCl Configuration File : Cette configuration va nous servir à piloter la configuration via des requêtes curl générées.



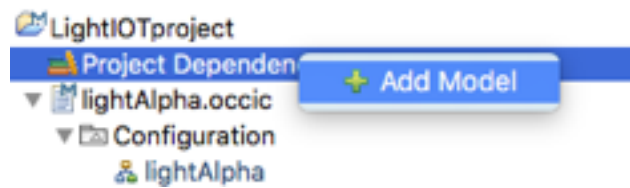
Cliquez sur **Next**.

Saisissez un nom de fichier pour la configuration : **lightAlpha.occic** par exemple.

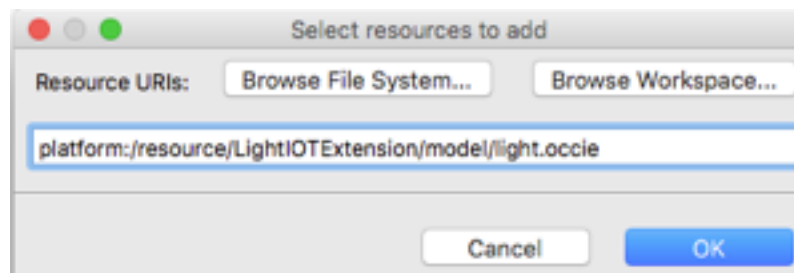
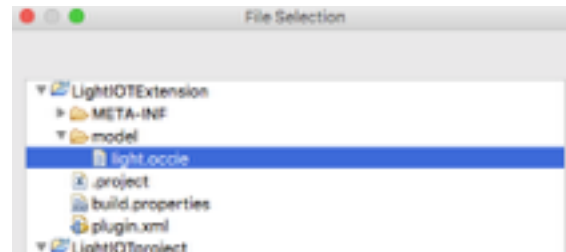
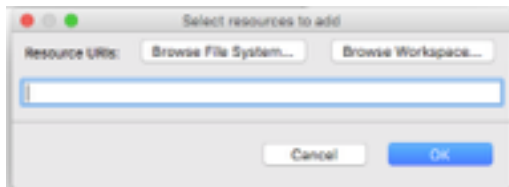
Sélectionnez l'extension « <http://occiware.org/light#> »

Cliquez sur **Finish**.

A faire uniquement si vous n'avez pas sélectionné d'extensions : cliquez droit sur « Project Dependencies » puis « add model »

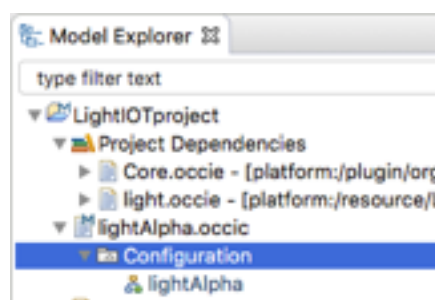


Sélectionnez Browse Workspace puis sélectionnez l'extension que vous avez modélisée (fichier.occie) puis validez.

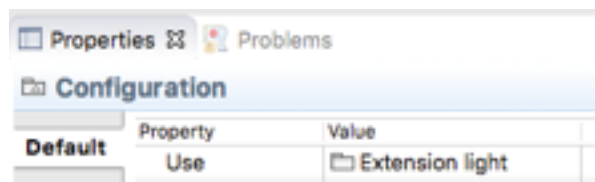
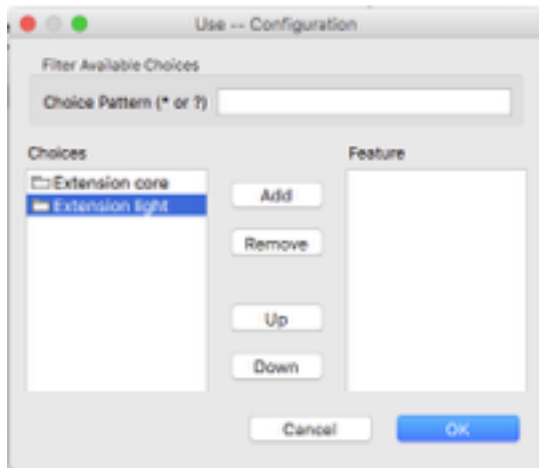
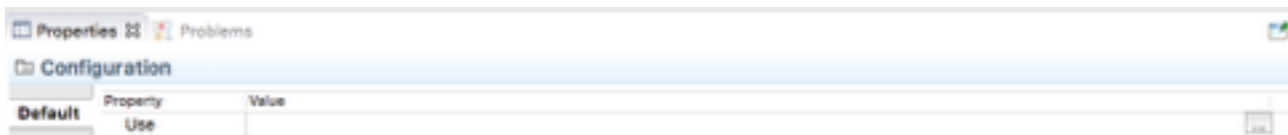


Une fenêtre « Viewpoints Selection » apparaît, cliquez sur ok.

Cliquez sur configuration :



Dans l'onglet **Properties**, propriété « **Use** », ajouter votre « **light extension** »

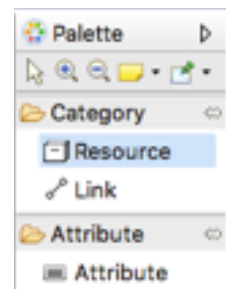
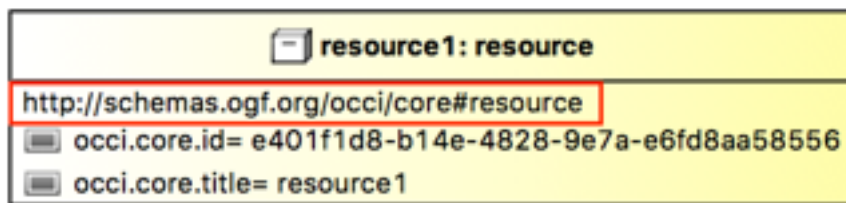


L'extension doit apparaître dans la liste des features (via le bouton add), puis cliquez sur ok.

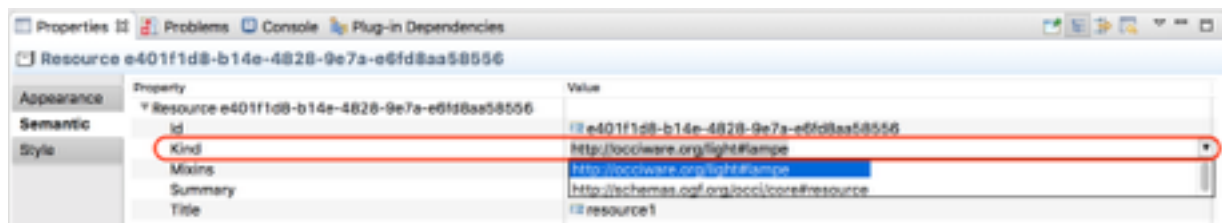
Création d'une ressource lampe

Dans la palette, cliquez sur « Resource » ensuite cliquez sur l'espace disponible du workbench pour créer une ressource.

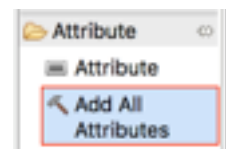
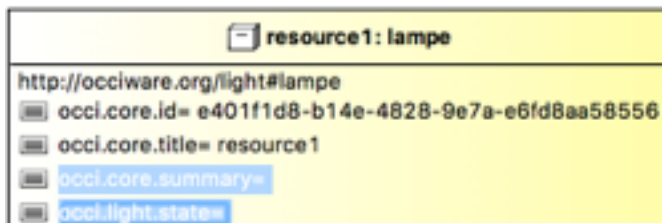
Une ressource comme celle-ci est créée :



Le problème est que son kind est « **Resource** » et non **lampe**, il faut lui assigner le bon kind. Pour ce faire, cliquez sur la ressource puis dans les propriétés changer son kind en sélectionnant votre « **lampe** » kind.



Dans la palette cliquez sur **Add All Attributes** ensuite sur votre ressource.



Cette opération permet d'ajouter tous les attributs non affichés et possibles d'une ressource.


























L'attribut occi.core.summary permet de définir un lieu où notre lampe est positionnée (dans notre cas d'usage). Modifiez sa propriété **Value** comme suit :
Value: Living room light

Créez de la même manière une deuxième ressource avec pour attribut :
Name : occi.core.summary
Value : Kitchen light

Créez de la même manière une troisième ressource avec pour attribut :
Name : occi.core.summary
Value : Bedroom light 1

Créez autant de ressources « light » que vous désirez.

Par exemple sur mon écran j'obtiens ceci :

<div> Dining room: lampe</div> <div>http://occiware.org/light#lampe  <code>occi.core.id= e401f1d8-b14e-4828-9e7a-e6fd8aa58556</code>  <code>occi.core.title= Dining room</code>  <code>occi.core.summary= Dining room</code>  <code>occi.light.state=</code></div>	<div><div> Kitchen room: lampe</div><div>http://occiware.org/light#lampe  <code>occi.core.id= 992190e5-8e5f-4c03-97f8-4aaf808ff442</code>  <code>occi.core.title= Kitchen room</code>  <code>occi.core.summary= kitchen</code>  <code>occi.light.state=</code></div></div>
<div><div> Bedroom light 1: lampe</div><div>http://occiware.org/light#lampe  <code>occi.core.id= 15195ede-e24a-4fc6-91a9-0ffa3182c53d</code>  <code>occi.core.title= Bedroom light 1</code>  <code>occi.core.summary= Bedroom 1</code>  <code>occi.light.state= off</code></div></div>	<div><div> Bedroom light 2: lampe</div><div>http://occiware.org/light#lampe  <code>occi.core.id= ca68e28f-cffb-4b85-9f13-15e04b97f4c9</code>  <code>occi.core.title= Bedroom light 2</code>  <code>occi.core.summary= Bedroom 2</code>  <code>occi.light.state=</code></div></div>
<div><div> Bedroom light child 3: lampe</div><div>http://occiware.org/light#lampe  <code>occi.core.id= 5acd6107-d459-4ca1-90a1-24186e4d1d0d</code>  <code>occi.core.title= Bedroom light child 3</code>  <code>occi.core.summary= Bedroom 3 light child</code>  <code>occi.light.state= off</code></div></div>	

ETAPE 14 : Génération spécification Alloy pour la configuration

A compléter dans une version ultérieure du tutoriel

ETAPE 15 : Vérification Alloy pour la configuration

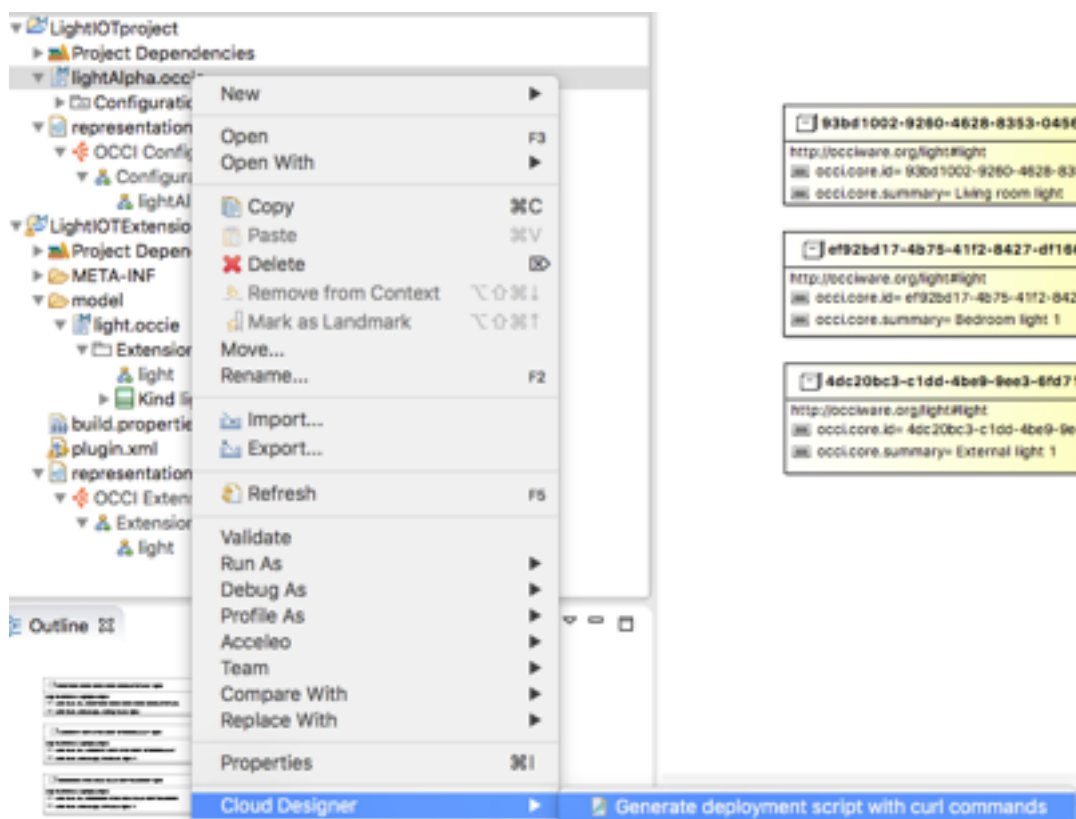
A compléter dans une version ultérieure du tutoriel

ETAPE 16 : Génération des requêtes curl

Les requêtes curl servent à réaliser les actions via le serveur HTTP Erocci (et le backend Erocci-dbus-java). Pour ce tutorial, Erocci va transférer au backend la totalité des informations pour chaque ressource, puis le backend exécute les actions dans le réel.

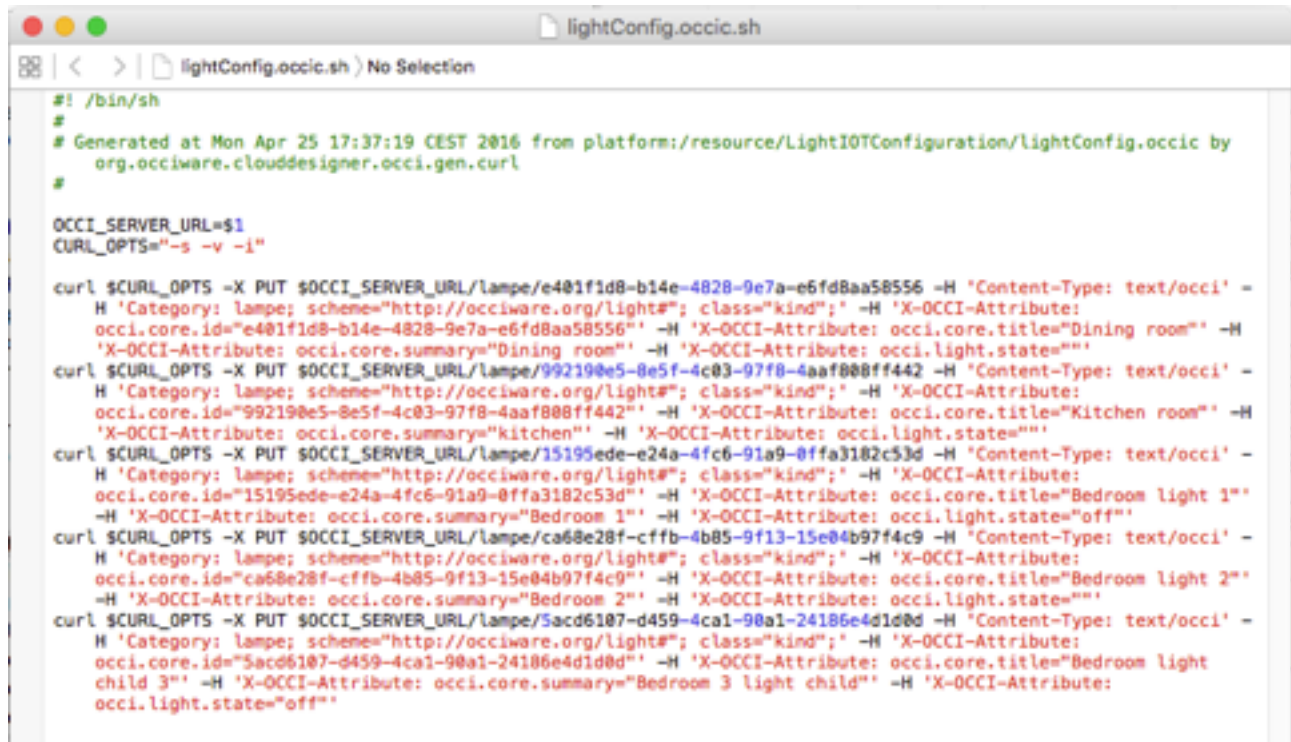
Pour ce faire, cliquez droit sur votre fichier de configuration (fichier .occic), sous menu « Cloud Designer » et cliquez sur « Generate deployment script with curl commands ».

Cela va créer un fichier shell avec toutes les requêtes curl pour créer les ressources via le backend Erocci-dbus-java.



Cela crée un répertoire src-gen/curl/ dans le projet et un fichier lightAlpha.occic.sh. Pour l'ouvrir : Clic droit sur le fichier et « open with » —> default editor.

Voici à quoi ressemble mon fichier :



```
lightConfig.occi.sh
lightConfig.occi.sh > No Selection

#!/bin/sh
#
# Generated at Mon Apr 25 17:37:19 CEST 2016 from platform:/resource/LightIoTConfiguration/lightConfig.occi by
# org.occiware.cloud designer.occi.gen.curl
#

OCCI_SERVER_URL=$1
CURL_OPTS="-s -v -i"

curl $CURL_OPTS -X PUT $OCCI_SERVER_URL/lampe/e401f1d8-b14e-4828-9e7a-e6fd8aa58556 -H 'Content-Type: text/occi' -
H 'Category: lampe; scheme="http://occiware.org/light#"; class="kind";' -H 'X-OCCI-Attribute:
occi.core.id="e401f1d8-b14e-4828-9e7a-e6fd8aa58556"' -H 'X-OCCI-Attribute: occi.core.title="Dining room"' -H
'X-OCCI-Attribute: occi.core.summary="Dining room"' -H 'X-OCCI-Attribute: occi.light.state=""'
curl $CURL_OPTS -X PUT $OCCI_SERVER_URL/lampe/992190e5-8e5f-4c03-97f8-4aaf808ff442 -H 'Content-Type: text/occi' -
H 'Category: lampe; scheme="http://occiware.org/light#"; class="kind";' -H 'X-OCCI-Attribute:
occi.core.id="992190e5-8e5f-4c03-97f8-4aaf808ff442"' -H 'X-OCCI-Attribute: occi.core.title="Kitchen room"' -H
'X-OCCI-Attribute: occi.core.summary="kitchen"' -H 'X-OCCI-Attribute: occi.light.state=""'
curl $CURL_OPTS -X PUT $OCCI_SERVER_URL/lampe/15195ede-e24a-4fc6-91a9-0ffa3182c53d -H 'Content-Type: text/occi' -
H 'Category: lampe; scheme="http://occiware.org/light#"; class="kind";' -H 'X-OCCI-Attribute:
occi.core.id="15195ede-e24a-4fc6-91a9-0ffa3182c53d"' -H 'X-OCCI-Attribute: occi.core.title="Bedroom light 1"'
-H 'X-OCCI-Attribute: occi.core.summary="Bedroom 1"' -H 'X-OCCI-Attribute: occi.light.state="off"'
curl $CURL_OPTS -X PUT $OCCI_SERVER_URL/lampe/ca68e28f-cffb-4b85-9f13-15e04b97f4c9 -H 'Content-Type: text/occi' -
H 'Category: lampe; scheme="http://occiware.org/light#"; class="kind";' -H 'X-OCCI-Attribute:
occi.core.id="ca68e28f-cffb-4b85-9f13-15e04b97f4c9"' -H 'X-OCCI-Attribute: occi.core.title="Bedroom light 2"'
-H 'X-OCCI-Attribute: occi.core.summary="Bedroom 2"' -H 'X-OCCI-Attribute: occi.light.state=""'
curl $CURL_OPTS -X PUT $OCCI_SERVER_URL/lampe/5acd6107-d459-4ca1-90a1-24186e4d1d0d -H 'Content-Type: text/occi' -
H 'Category: lampe; scheme="http://occiware.org/light#"; class="kind";' -H 'X-OCCI-Attribute:
occi.core.id="5acd6107-d459-4ca1-90a1-24186e4d1d0d"' -H 'X-OCCI-Attribute: occi.core.title="Bedroom light
child 3"' -H 'X-OCCI-Attribute: occi.core.summary="Bedroom 3 light child"' -H 'X-OCCI-Attribute:
occi.light.state="off"'
```

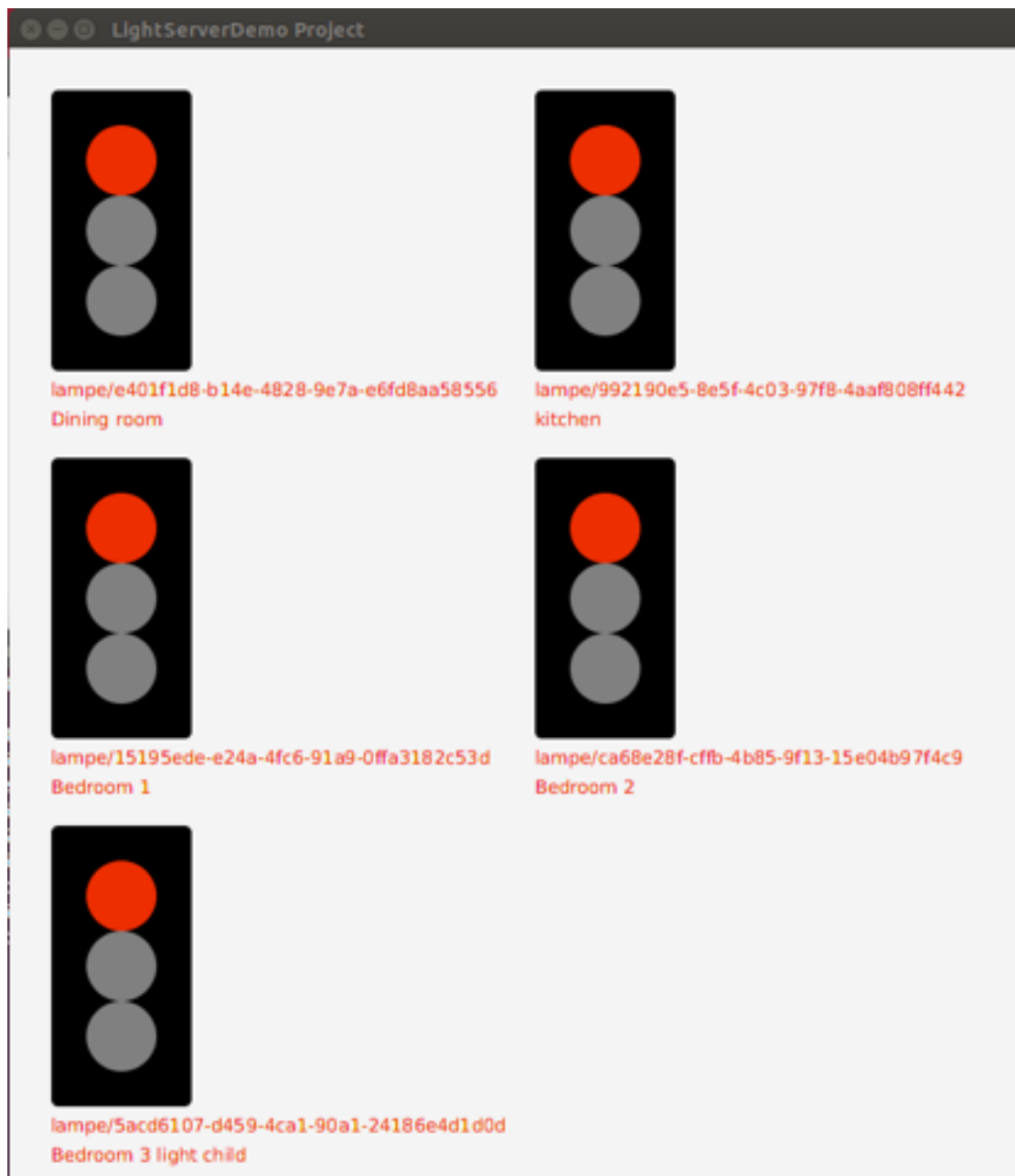
ETAPE 17 : Lancement du script généré

Au préalable, il faut lancer dans l'ordre :

- LightServer
- Erocci-dbus-java (via CloudDesigner —> projet light.connector)
- Erocci

Vous pouvez l'**exécuter** comme ceci : `./lightAlpha.occic.sh http://localhost:8080`

Le résultat doit donner montrer sur **LightServer** que vos lampes sont bien **créées sur LightServer**.



ETAPE 18 : Interaction manuelle avec Erocci pour vérifier que la configuration est bien déployée

Listez les collections de lampes avec cette requête :

```
curl -v -H "Accept: application/json" -X GET http://localhost:8080/collections/lampe/
```

Vous pouvez obtenir le détail d'une lampe avec un format JSON, exemple :

```
curl -H "Accept: application/json" http://localhost:8080/lampe/5acd6107-d459-4ca1-90a1-24186e4d1d0d/
```

où /light/light1/c1568df5-250f-45d4-ae1-053f186b15/ est à remplacer par l'identifiant de votre lampe.

En résultat :

```
{
  "resources" : [
    {
      "kind" : "http://occiware.org/light#lampe",
      "id" : "/lampe/5acd6107-d459-4ca1-90a1-24186e4d1d0d",
      "attributes" : {
        "occi.core.id" : "http://localhost:8080/lampe/5acd6107-d459-4ca1-90a1-24186e4d1d0d",
        "occi.core.summary" : "Bedroom 3 light child",
        "occi.core.title" : "Bedroom light child 3"
      }
    }
  ]
}
```

Allumage d'une lampe :

```
curl -v -X POST --header 'Content-type:text/occi' http://localhost:8080/lampe/5acd6107-d459-4ca1-90a1-24186e4d1d0d?action=switchOn -H 'Category: switchOn; scheme="http://occiware.org/light/lampe/action#"; class="action"'
```

ETAPE 19 : Récupérer la configuration déployée via jOCCI

A compléter dans la prochaine version.

ETAPE 20 : Utiliser le connecteur jOCCI

Modification de la configuration

Créer/modifier/détruire des lampes et exécuter des actions

A compléter dans la prochaine version.