



Eclipse RCP 4

Inter Enterprise

Toulouse du 20 au 23 octobre 2015

OPC 15 INT FOR EAP 04 A

Table des matières



Objectifs	7
I - Présentation	9
II - Introduction Eclipse	13
A. Présentation.....	13
III - Eclipse : Atelier de développement Java	15
A. Principales notions.....	15
B. Configuration IDE.....	17
IV - Architecture OSGI et Eclipse	23
A. Open Services Gateway initiative (OSGi)	23
B. Architecture Eclipse.....	31
1. Plugin.....	31
2. Feature.....	34
V - Introduction Eclipse 4 RCP	37
VI - Application Model	43
VII - Launch Configuration	49
VIII - Plugin Development Environment	53
IX - La création d'IHM dans Eclipse 4	65

A. SWT.....	65
B. Création de Part.....	74
C. Gestion des événements SWT.....	78
D. Window Builder.....	80
E. Injection et annotations.....	84
F. Contextes d'injection.....	93
G. API Annotation.....	101
H. JFace.....	104
I. JFace Ressources.....	112
J. API pour l'IHM.....	116
X - Points d'extension et Model UI	127
A. Application Model UI.....	127
B. Eclipse 4 commands.....	131
C. Extension Expression definition.....	141
D. Gestions des Opérations.....	145
E. Gestion des wizards.....	146
F. Gestion des préférences.....	149
XI - Model Fragment et Processor	157
XII - E4 Event Bus	163
XIII - API Interne Eclipse 4	167
A. API Eclipse (non IHM).....	167
B. Services Eclipse 4.....	171
XIV - Crée des points d'extension	177
XV - CSS Styling	187
XVI - Livraison Eclipse	191
XVII - Build Tycho Jenkins Eclipse	193
A. Maven Tycho.....	193
B. Jenkins.....	205
XVIII - Cahier d'exercices Eclipse 4	209

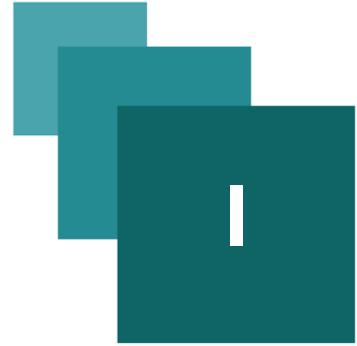
A. EAP 010 : Installation Eclipse 4.....	209
B. EAP 015 : Création d'une application Eclipse 4.....	210
C. EAP 30 : Création d'un Part dans Eclipse 4.....	211
D. EAP 032 : Utilisation du context Spy.....	213
E. EAP 035 : Création d'un Addon pour Rental.....	213
F. RCP 40 : TreeViewer.....	214
G. EAP 50 : Gestion de la sélection.....	214
H. RCP 052 : Drag And Drop.....	215
I. EAP 055 : Gestion des IAdaptables.....	215
J. EAP 60 : Création d'une perspective.....	215
K. EAP 70 : Ajout d'une commande.....	216
L. EAP 71 : Ajout de commande conditionnelle.....	216
M. EAP 80 : Ajout de pages de préférences.....	217
N. EAP 085 : Création de fragments de modèle.....	218
O. EAP 087 : Gestion d'événements avec EventBroker.....	219
P. EAP 090 : Parcours des extensions.....	219
Q. RCP 92 : Création d'un point d'extension.....	219
R. EAP 100 : Mise en place de la css.....	221
S. RCP 110 : Livraison Manuelle de l'application.....	221
T. BLD 030 : Livraison headless avec Tycho.....	222
U. BLD 100 : Jenkins, installation et tycho	223

Objectifs



- Comprendre l'architecture OSGi/Eclipse 4
- Connaitre les bonnes pratiques d'architecture
- Concevoir des plugins Eclipse
- Construire des IHM SWT/JFace
- Connaitre les principales API d'Eclipse
- Maitriser la distribution de l'application

Présentation



Tour de Table

- Société
- Attentes
- Connaissances

OPCoach

- SARL créée en juin 2009 <http://www.opcoach.com>¹
- Membre de la fondation Eclipse (Solution Member)
- Committer Eclipse (E4 tools, platform ui)
- **Olivier Prouvost**
 - Co fondateur Anyware Technologies
 - Expert XML, Java / Eclipse



Image 1

Activité Formation et Expertise

Formation sur mesure :

- RCP, Modeling, Build, Eclipse
des centaines de stagiaires formés

Expertise :

- réponse à appel d'offre
- aide à l'architecture
- audit développement
- expertise en direct sur skype (à l'heure)

Activité Emploi

Mise en relation sociétés et candidats pour des postes ou des missions

1 - <http://www.opcoach.com>



Image 2 Emploi

Références



Image 3

Site web

www.opcoach.com :

- inscription mailing list Eclipse sur la page d'accueil
- prochaines formations (mailing list dans le menu formation)
- boutique livres Eclipse
- rubrique emploi pour proposer un poste ou un CV Eclipse
- témoignages de formation (-> Merci de commenter en fin de formation)

The screenshot shows the OPCOACH website with several annotations:

- A red arrow points to the "Formations" menu item in the top navigation bar.
- A red arrow points to the "French, English and Spanish" language selection area in the top right.
- A red arrow points to the "Course plans and schedule" section, which features an image of two Eclipse RCP books.
- A red arrow points to the "Formation Eclipse RCP 4 Toulouse du 6 au 9 juillet 2015" section, which includes a "En savoir plus" button.
- A red arrow points to the "Mailing List" section under "Votre Expert Eclipse".
- A red arrow points to the "Inscrivez-vous pour rester informé sur Eclipse" sign-up form, which includes fields for "Prénom" and "Nom".

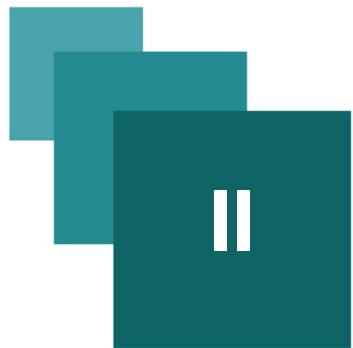
Image 4 site web

[Twitter](#)



@OPCoach_Eclipse

Introduction Eclipse



A. Présentation

Les différentes versions

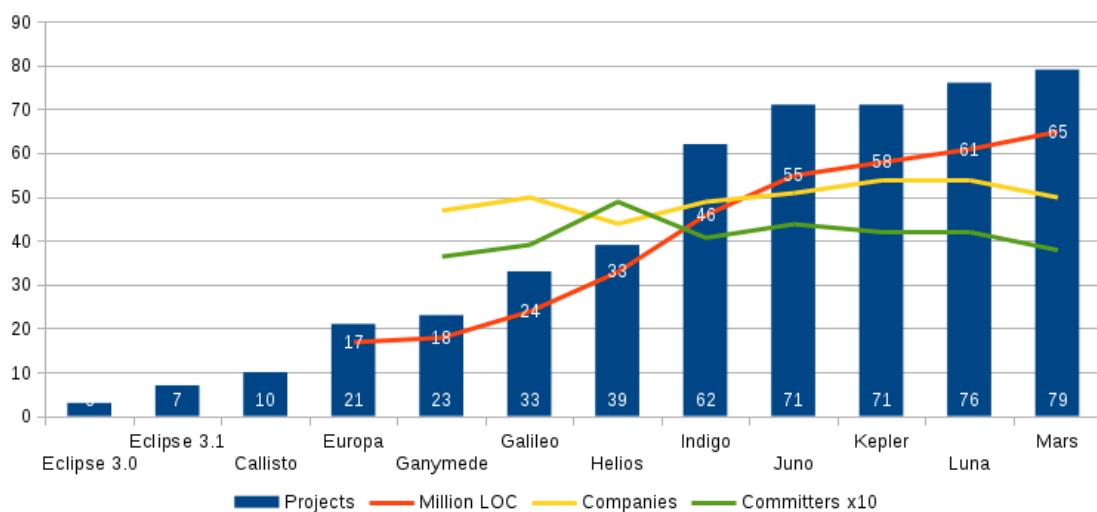


Image 5 Mars overview

Quelques chiffres (Juin 2015)

- 12 membres stratégiques (IBM, Itemis, Obeo, Oracle, Google, SAP, Redhat, Codenvy,...)
- une centaine de solutions membres (Airbus, Thales, Intel, BMW, Ebay, Github, OPCCoach ...)
- 79 projets et 65 millions de ligne de code dans Mars
- 600 committers actifs sur 1200 depuis le début.
- 12 années de livraison à l'heure
- trois conférences internationales annuelles (US,Europe,France)

Les points forts

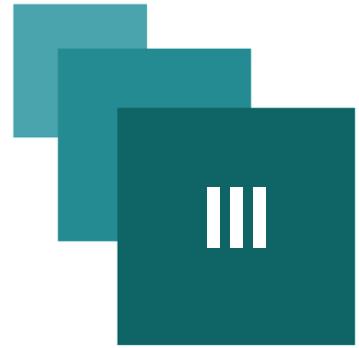
- Architecture RCP ouverte
- Adopté par tous les éditeurs de logiciels
- Environnement très riche
- Outilage
- Documentation abondante
- Gratuité

- Portable sur tous les OS
- Communauté (fondation, membres,...)
- Régularité et stabilité des livraisons (1/an)
- Licence d'utilisation (EPL)

Quelques liens sur Eclipse

Eclipse	http://www.eclipse.org ²
Wiki	http://wiki.eclipse.org/Main_Page ³
Bugzilla	https://bugs.eclipse.org/bugs ⁴
Press release	http://www.eclipse.org/org/press-release ⁵
Hudson	https://hudson.eclipse.org/hudson ⁶
Forums	http://www.eclipse.org/forums ⁷
Blog	http://planeteclipse.org ⁸
Conférences Eclipse	http://www.eclipsecon.org ⁹
Webinars, podcast	http://live.eclipse.org ¹⁰
Plugin central	http://marketplace.eclipse.org ¹¹
Git eclipse	http://git.eclipse.org ¹²

- 2 - <http://www.eclipse.org>
 3 - http://wiki.eclipse.org/Main_Page
 4 - <https://bugs.eclipse.org/bugs/>
 5 - <http://www.eclipse.org/org/press-release/>
 6 - <https://hudson.eclipse.org/hudson>
 7 - <http://www.eclipse.org/forums/>
 8 - <http://planeteclipse.org/>
 9 - <http://www.eclipsecon.org/>
 10 - <http://live.eclipse.org/>
 11 - <http://marketplace.eclipse.org>
 12 - <http://git.eclipse.org>



Eclipse : Atelier de développement Java

Principales notions

Eclipse est avant tout un IDE, multi langage

Eclipse définit des concepts communs

Ces concepts seront repris dans l'architecture RCP

A. Principales notions

Le vocabulaire de l'IDE Eclipse

- **Workbench** : IHM générale
- **Workspace** : ensemble des fichiers gérés
- **Préférences** : préférences globales (liées au workspace)
- **Propriétés** : paramètres d'une ressource du workspace
- **Wizard** : aide à créer, importer, exporter
- **Editeur** : associé à une 'outline' et à un type de fichier
- **Vue** : présentation de données sans notion d'édition
- **Perspective** : ensemble de vues organisées
- **Extension** : mécanisme permettant de contribuer à Eclipse

Extensions dans le workbench

On retrouve les extensions dans les différentes parties de l'IDE

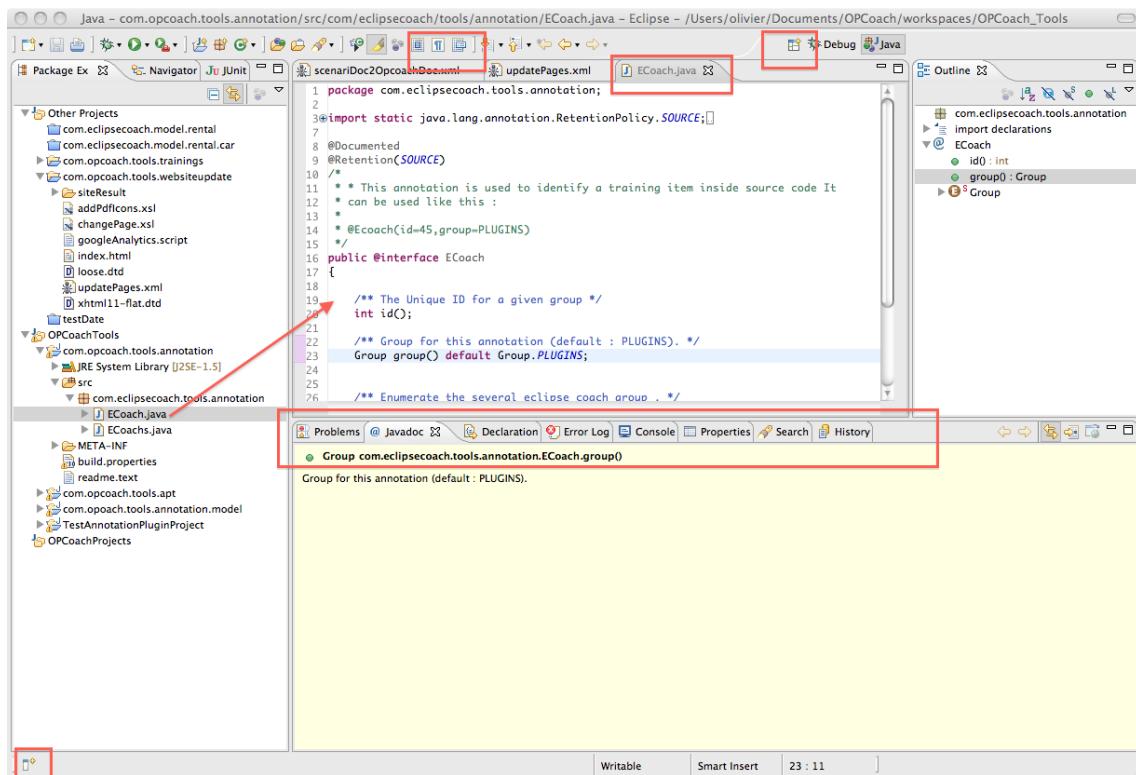


Image 6 Extensions in workbench

Extensions dans les préférences

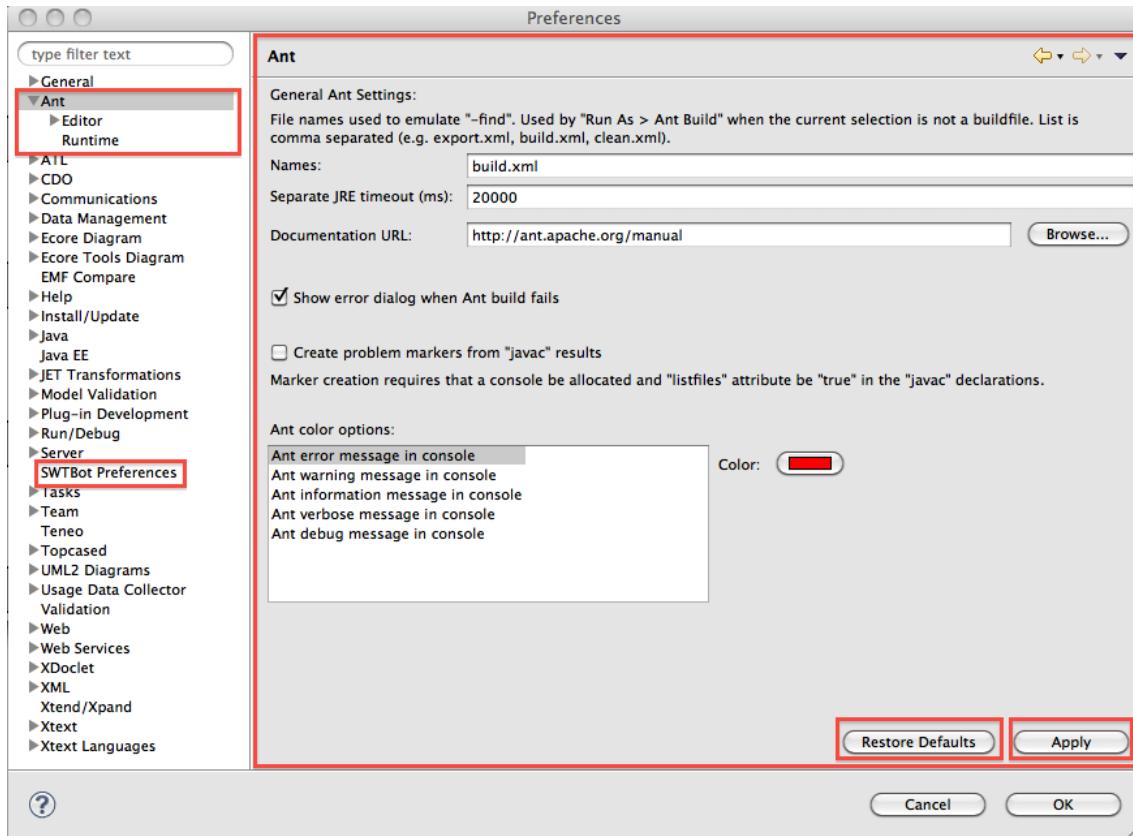


Image 7 Extensions in preference pages

Extensions dans les autres composants

Les extensions se trouvent également dans :

- les wizards export, import, new
- les pages de propriétés sur les ressources (projet, fichier...)
- les pages d'aide
- etc...

Remarque : Extensions et Eclipse 4

Avec la nouvelle architecture Eclipse 4, certaines extensions vont migrer dans l'application model

B. Configuration IDE

Les raccourcis indispensables

- **Ctrl Shift T** : Chercher une classe/Interface/Type
- **Ctrl Shift R** : Chercher une ressource (fichier, dossier)
- **Ctrl Shift A** : Chercher un artifact (point extension, ...)
- **Ctrl Shift G** : Chercher un texte dans tout le workspace

- **Ctrl Shift O** : Organize imports
- **Ctrl O** : Quick Outline
- **Alt Shift L** : extrait une variable locale
- **Ctrl 1 ('cmd' sur mac)** : Quick Fix
- **Ctrl 3 ('cmd' sur mac)** : Quick access
- **F3** : accès à une déclaration
- **F4** : accès au type hierarchie
- **Ctrl T** : Quick Hierarchy (2 ways)
- **Alt Shift F1** : plugin Spy (pour développement RCP)
- **Alt Shift F2**: plugin menu Spy (pour développement RCP)
- **Alt Shift F9**: ouvre l'éditeur de modèle d'application (spécifique e4)
- **Ctrl Shift F3** (ou Cmd) : open a model element (for EMF)

L'organisation du workspace : les working sets

- Ils permettent de regrouper les projets par groupe
- S'accèdent dans le navigator ou le package explorer (menu en haut à droite)
- Un projet peut appartenir à plusieurs working sets
- Un working set peut contenir des projets ou des fichiers

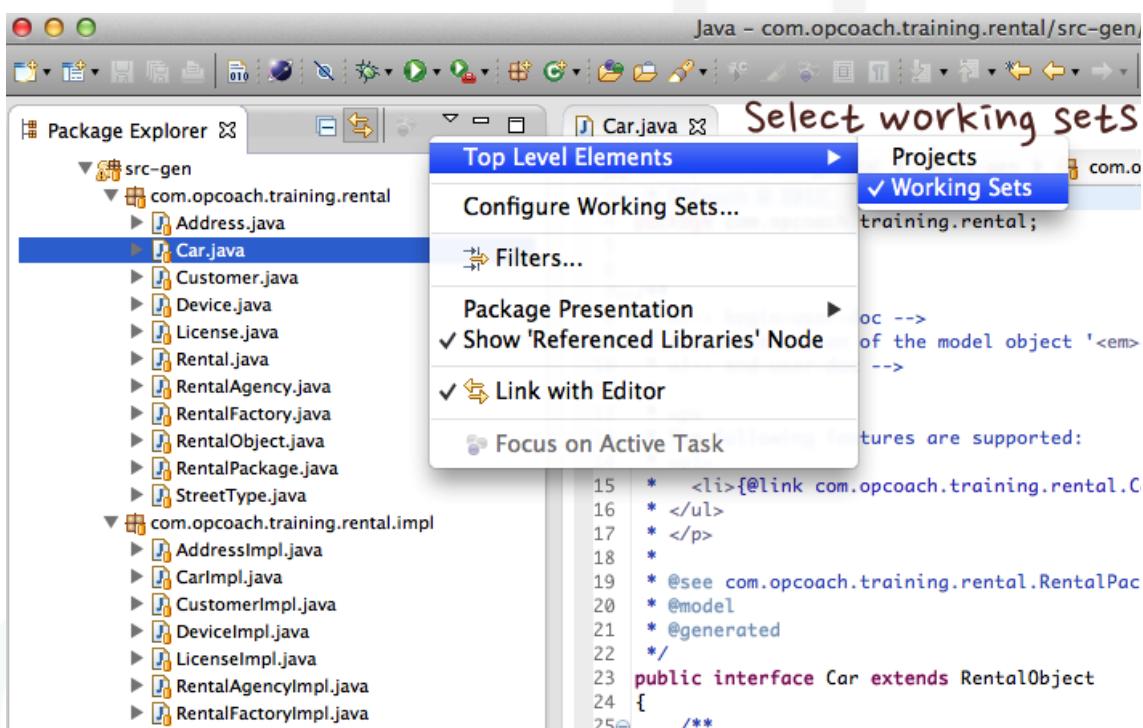


Image 8 Working sets menu

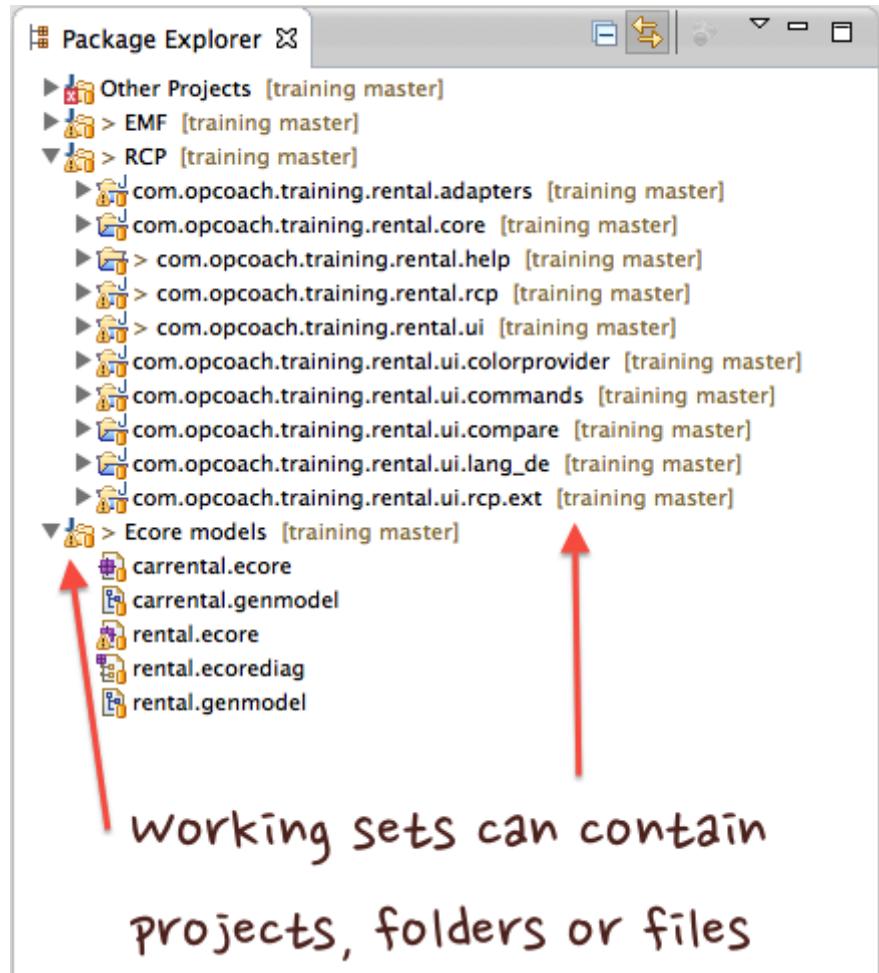


Image 9 Working sets

Les Team Project Sets

- Il s'agit d'un export particulier du composant Team
- Permet d'exporter un état CVS, SVN, GIT
 - Contient les url des repositories
 - Fixe les versions ou branches utilisées
- Peut se partager entre les membres d'une équipe
- Conserve les working sets
- Le travail est unifié dans l'équipe.

Target Platform

- Le développement de plugins Eclipse nécessite de connaître la target platform
- Par défaut elle est initialisée avec les plugins de l'Eclipse courant
- Elle est globale au workspace et peut se définir dans un fichier .target

- On peut changer de target platform dans les préférences :

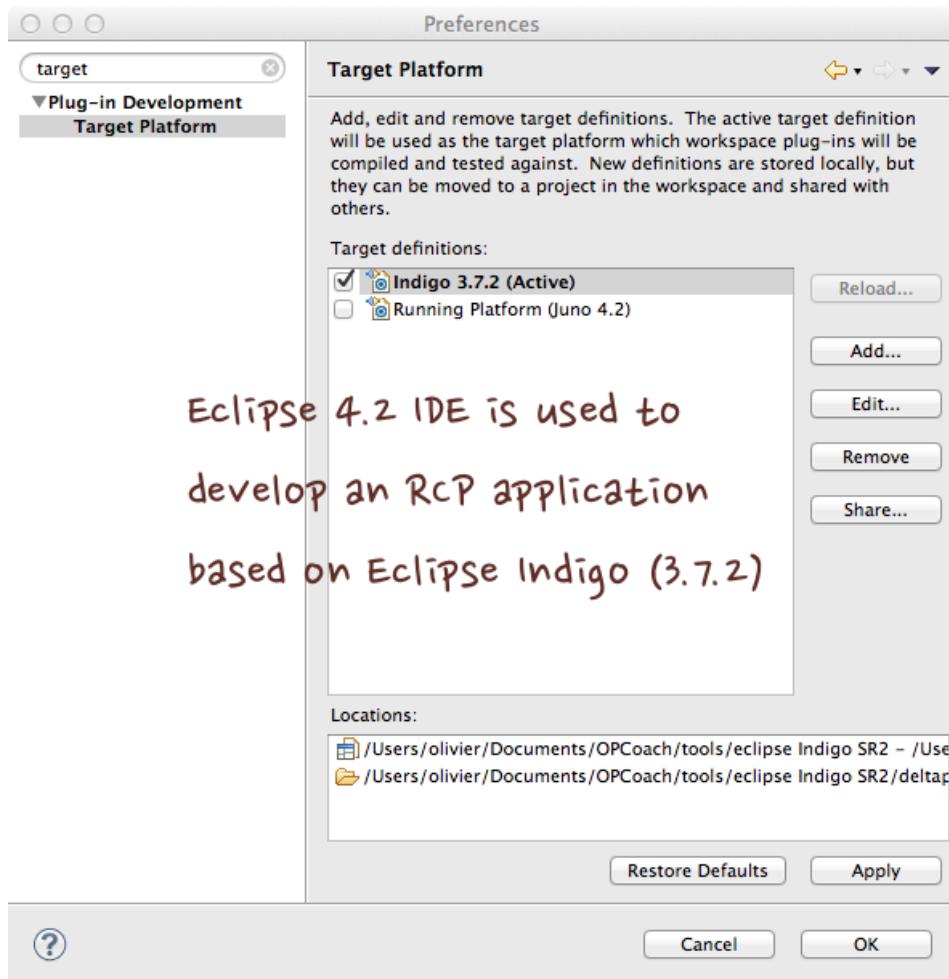


Image 10 target platform

Article

Un article pour mieux travailler en équipe :

<http://www.opcoach.com/2012/02/10-conseils-pour-travailler-en-équipe-avec-eclipse/>¹³

Git et eGit

eGit est le nouveau système de gestion de code source pour Eclipse :

- permet de se connecter à tout repository git
- très performant
- distribué (chaque poste possède une copie complète du repository)
- gère parfaitement les merge
- remplace svn haut la main !
- permet de travailler hors ligne
- permet de créer une branche instantanément
- change radicalement la manière de travailler

13 - <http://www.opcoach.com/2012/02/10-conseils-pour-travailler-en-équipe-avec-eclipse/>

Plus d'informations : <http://marklodato.github.com/visual-git-guide/index-en.html>¹⁴

Vue Git Repositories

La vue Git Repositories permet d'accéder à git :

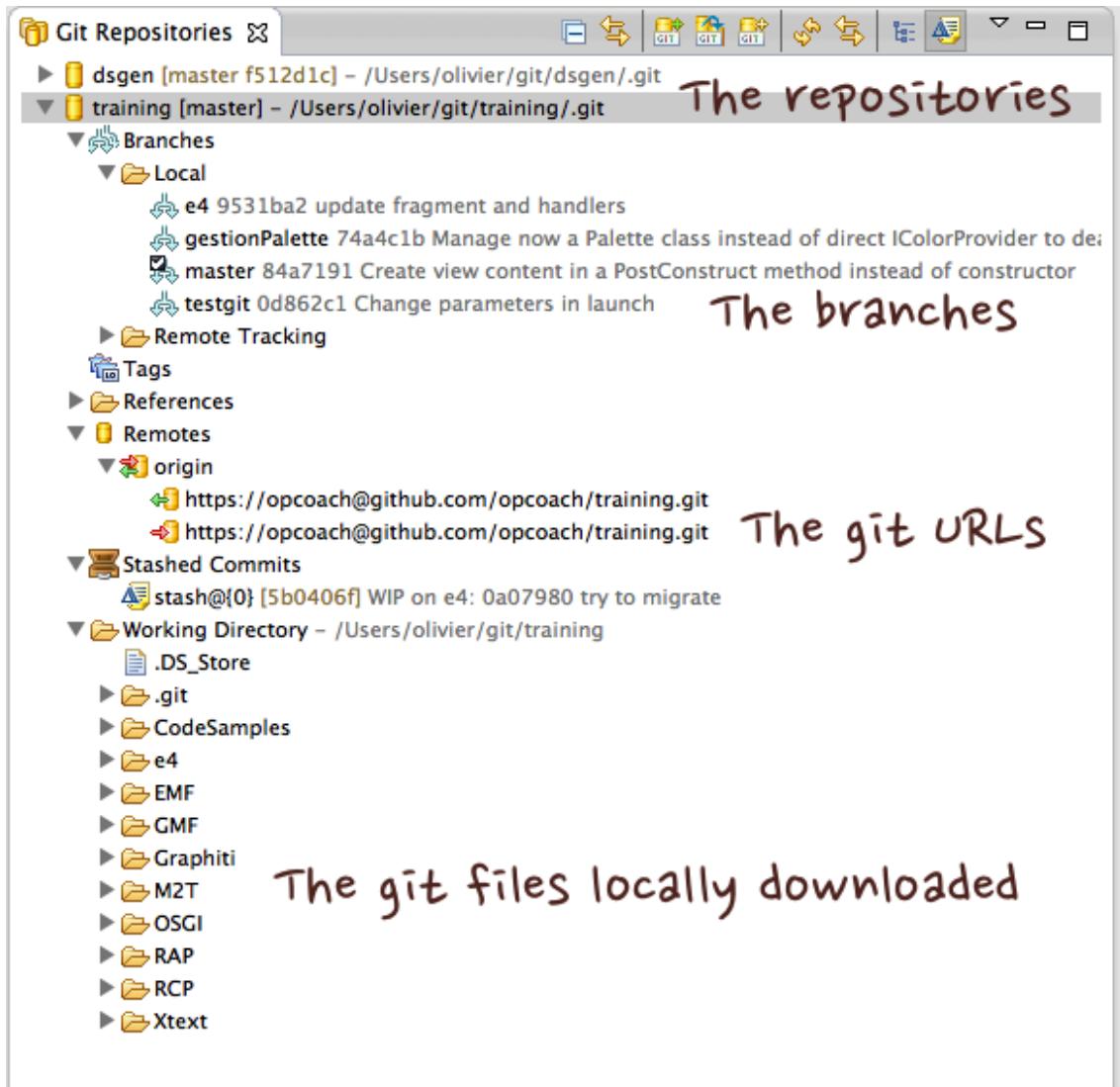


Image 11 Git Repositories

14 - <http://marklodato.github.com/visual-git-guide/index-en.html>

Architecture OSGI et Eclipse

IV

Open Services Gateway initiative (OSGi)

23

Architecture Eclipse

31

A. Open Services Gateway initiative (OSGi)

Pourquoi OSGi ?

Java semble modulaire

- packages
- classes
- méthodes
- différents jar

Tout est bien séparé lors du développement.

Mais que se passe t il quand on lance un programme java ?

Lancement Java

La jvm se lance avec un seul class loader.

La modularité s'est dissoute.

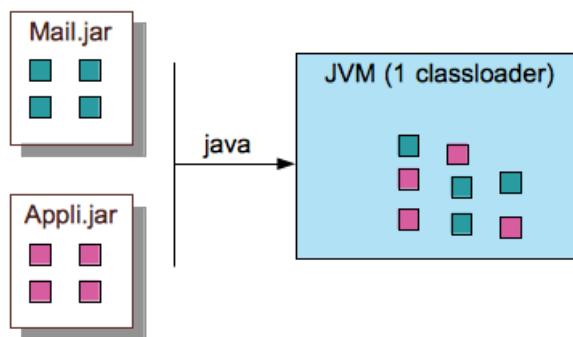


Image 12 Classic Java launch

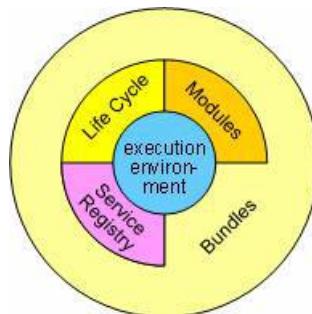
Les inconvénients du runtime java classique

- Toute classe publique dans un jar est accessible par toute autre classe
- La notion de version dans le manifest est sans effet au runtime
- Il est impossible de changer un jar 'à chaud'

- Deux versions d'un même jar dans deux versions différentes ne peuvent cohabiter
OSGi va répondre à ces problèmes

Présentation

- OSGi définit la notion de 'Bundle'
- OSGi gère les versions des bundles
- OSGi gère l'activation des bundles
- OSGi permet de gérer des services (Service Registry)



OSGi Framework

OSGI fournit un modèle de déploiement de modules logiciels java.

OSGi est une spécification gérée par la communauté OSGi Alliance : <http://www.osgi.org>¹⁵

Le bundle

Il est défini par un MANIFEST :

```
com.opcoach.message
Bundle-ManifestVersion: 2
Bundle-Name: Message Manager
Bundle-SymbolicName: com.opcoach.message
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: com.opcoach.message.MessageActivator
Bundle-Vendor: OPCoach
Bundle-RequiredExecutionEnvironment: J2SE-1.5
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ActivationPolicy: lazy
Export-Package: com.opcoach.message;uses:="org.osgi.framework"
```

Image 13 Manifest sample

- Il possède un ID unique
- Il est défini pour une version
- Il peut publier ou consommer des services
- Il possède son propre class loader
- Il peut être défini pour une plateforme spécifique (windows, linux...)

Relations avec les autres bundles

- Le bundle exporte les packages qu'il veut rendre visible
- Le bundle peut dépendre d'autres bundles

15 - <http://www.osgi.org>

```

1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Bundle-Name: %pluginName
4 Bundle-SymbolicName: com.opcoach.training.rental;singleton:=true
5 Bundle-Version: 1.0.1
6 Bundle-ClassPath: .
7 Bundle-Vendor: OPCoach
8 Bundle-Localization: plugin
9 Bundle-RequiredExecutionEnvironment: J2SE-1.5
10 Export-Package: com.opcoach.training.rental, ← visibility for the others
11 com.opcoach.training.rental.impl,
12 com.opcoach.training.rental.util
13 Require-Bundle: org.eclipse.core.runtime, ← others used bundles
14 org.eclipse.emf.ecore;visibility:=reexport,
15 org.eclipse.swt;bundle-version="3.5.1";visibility:=reexport
16 Bundle-ActivationPolicy: lazy

```

Image 14 Relations of a bundle

Comportement dynamique

Le cycle de vie du bundle est géré par le moteur OSGi :

- Il peut être démarré quand cela est nécessaire
- Il peut être arrêté à tout moment
- Il peut être remplacé par une version plus récente
- Deux versions différentes peuvent cohabiter
- Des nouveaux services peuvent être publiés à chaud

Bundle Fragment

- Le fragment est un complément d'un bundle (host bundle)
- Le fragment est utilisé pour :
 - gérer du code spécifique selon une plate forme
 - compléter un bundle avec des fichiers (i18n, ...)
 - écrire des tests (possède la visibilité sur le bundle)

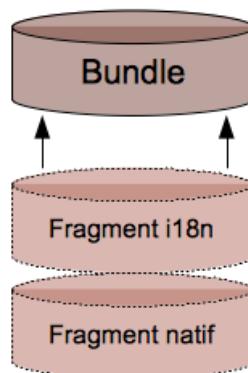


Image 15 Fragment

Le container

- OSGi est une spécification

- Equinox, Felix, Knoplerfish sont des implémentations
- Un bundle ne peut tourner que dans un container
- Exécuter un bundle revient à :
 - lancer un container
 - charger le bundle
 - démarrer le bundle
- Pour démarrer le container equinox :

```
java -jar org.eclipse.osgi_XXXX.jar -console
```

Moteur OSGi

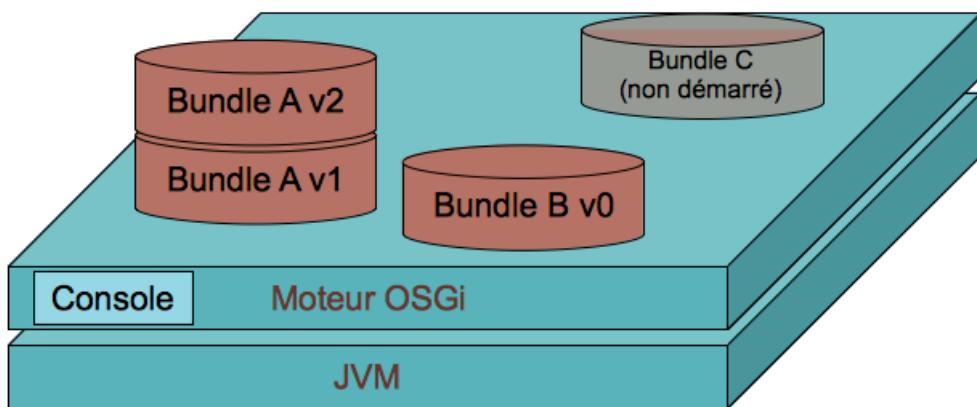


Image 16 OSGi engine

La console

Elle permet de contrôler les bundles installés et accepte des commandes :

- **help**
- **ss** : short status
- **headers i** : affiche les informations du bundle i
- **install path** : installe un bundle pointé par path
- **start i** : démarre le bundle i
- **stop i** : stoppe le bundle i

Exemple de console

The screenshot shows the Eclipse IDE's Console view with the following output:

```

osgi> ss
Framework is launched.

id      State      Bundle
0      ACTIVE     org.eclipse.osgi_3.5.0.v20090520
       Fragments=22, 26
12     ACTIVE     com.opcoach.notification_1.0.0.qualifier
22     RESOLVED   org.eclipse.persistence.jpa.equinox.weaving_1.1.2.v20090612-r4475
       Master=0
26     RESOLVED   javax.transaction_1.1.1.v200906161300
       Master=0

osgi> headers 12
Bundle headers:
  Bundle-ActivationPolicy = lazy
  Bundle-Activator = com.opcoach.notification.Activator
  Bundle-ManifestVersion = 2
  Bundle-Name = Notification
  Bundle-RequiredExecutionEnvironment = J2SE-1.5
  Bundle-SymbolicName = com.opcoach.notification
  Bundle-Vendor = OPCOACH
  Bundle-Version = 1.0.0.qualifier
  Export-Package = com.opcoach.notification;uses:="org.osgi.framework"
  Import-Package = org.osgi.framework;version="1.3.0"
  Manifest-Version = 1.0

```

Annotations in red boxes highlight specific bundle details:

- Bundle ID 12: `com.opcoach.notification_1.0.0.qualifier`
- Bundle Headers: `com.opcoach.notification`, `OPCOACH`, `1.0.0.qualifier`

Image 17 Console

Les états d'un bundle

- On peut les visualiser dans la console
- L'état est géré par le container du bundle

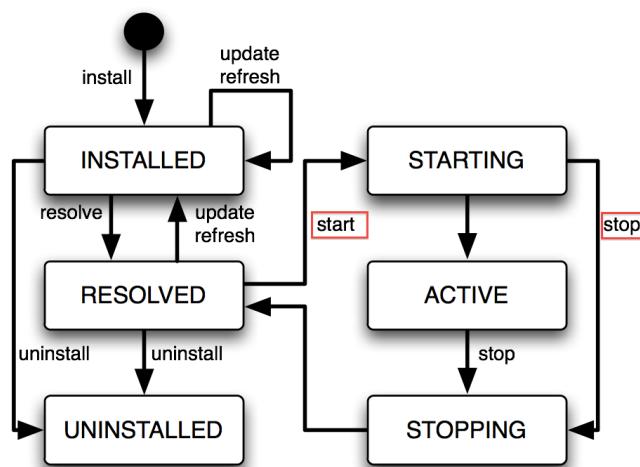


Image 18 Bundle states

L'activator

- L'activator est le code d'entrée du Bundle.
- Il implémente l'interface BundleActivator :

```
public interface BundleActivator {
    /**
     * Called when this bundle is started so the Framework can perform the
     * bundle-specific activities necessary to start this bundle. This method
     * can be used to register services or to allocate any resources that this
     * bundle needs.
     */
    public void start(BundleContext context) throws Exception;

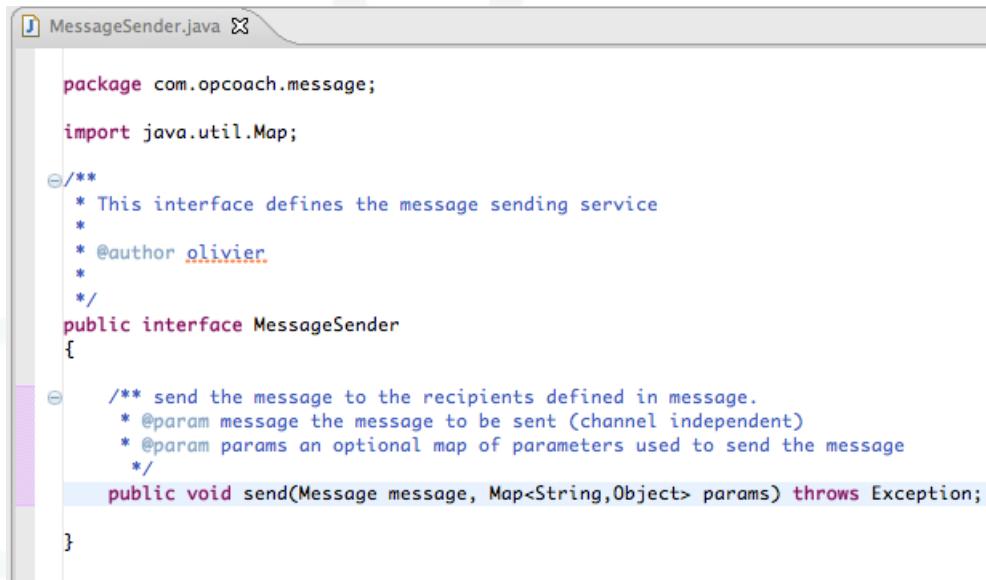
    /**
     * Called when this bundle is stopped so the Framework can perform the
     * bundle-specific activities necessary to stop the bundle. In general, this
     * method should undo the work that the <code>BundleActivator.start</code>
     * method started. There should be no active threads that were started by
     * this bundle when this bundle returns. A stopped bundle must not call any
     * Framework objects.
     */
    public void stop(BundleContext context) throws Exception;
}
```

Image 19 Bundle Activator

- L'activator peut contenir le code d'initialisation
- L'activator publie les services

Les services

- Un bundle peut publier des services
- Un service est matérialisé par une interface qui répond à un besoin



The screenshot shows a Java code editor window titled "MessageSender.java". The code defines a public interface named "MessageSender" with a single method "send". The method takes a "Message" object and a "Map<String, Object>" object as parameters, and throws an "Exception". The code is annotated with Javadoc comments describing the interface and its methods.

```
package com.opcoach.message;

import java.util.Map;

/**
 * This interface defines the message sending service
 *
 * @author olivier
 */
public interface MessageSender
{
    /**
     * send the message to the recipients defined in message.
     * @param message the message to be sent (channel independent)
     * @param params an optional map of parameters used to send the message
     */
    public void send(Message message, Map<String, Object> params) throws Exception;
}
```

Image 20 Service Interface

Enregistrement d'un service

Le service s'enregistre sur le BundleContext lors du start



```

package com.opcoach.message.mail;

import org.osgi.framework.BundleActivator;

public class MailMessageActivator implements BundleActivator
{
    public void start(BundleContext context) throws Exception
    {
        context.registerService(MessageSender.class.getName(),
                               new MailMessageSender(), null);
    }

    public void stop(BundleContext context) throws Exception
    {
    }
}

```

Image 21 Publication service

Utilisation d'un service

Le service s'utilise au moment voulu en interrogeant le bundleContext



```

package com.opcoach.message.test;

import org.osgi.framework.BundleActivator;

public class MessageTestActivator implements BundleActivator
{
    public void start(BundleContext context) throws Exception
    {
        // Get a service reference
        ServiceReference ref = context.getServiceReference(MessageSender.class.getName());
        // Get the service
        MessageSender sender = (MessageSender) context.getService(ref); ←
        Message m = new MessageImpl("Important Message");
        m.setText("Un nouveau message");
        // Use it !
        sender.send(m, null); ←
    }

    public void stop(BundleContext context) throws Exception
    {
    }
}

```

Image 22 Service usage

Attention: Disponibilité d'un service

Un service peut ne pas être disponible au moment où on le demande :

- bundle non chargé
- bundle non actif

- bundle non installé

Il faut tester la disponibilité du service ou utiliser le ServiceTracker qui permet d'attendre la disponibilité d'un service.

Services multiples

- Un service peut être rendu par plusieurs Bundles.
- Par exemple : le MessageSender peut être fourni par MailSender et SMSender
- On peut chercher les services et les traiter de manière spécifique en utilisant :

```
public interface BundleContext {  
  
    ...  
  
    public ServiceReference[] getAllServiceReferences(String clazz, String filter) throws InvalidSyntaxException;  
  
    ...  
}
```

Image 23 getAllServiceReferences

- Les références sont valides au moment de l'appel, mais rien ne dit que le service sera disponible ensuite !

Utilisation d'OSGi

Il faut raisonner en 'bundles' et fragments :

- un bundle de définition du 'modèle'
- des bundles clients qui utilisent le modèle
- des bundles spécifiques pour des services spécifiques
- des bundles wrappers de jar qui reexportent les packages
- des fragments pour le code natif ou l'i18n
- un ou plusieurs fragments de tests

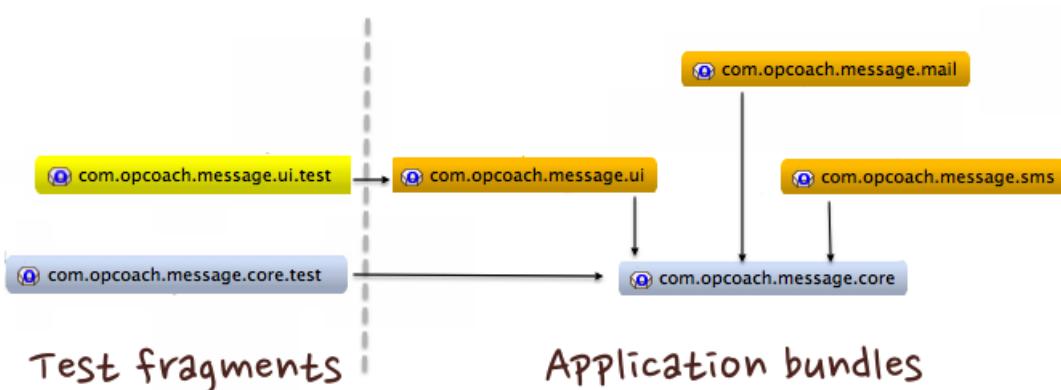


Image 24 OSGi architecture

B. Architecture Eclipse

Principe général

- L'architecture d'Eclipse est basée sur la notion de plugin.
- Un plugin est un composant logiciel élémentaire
- Les plugins sont gérés par un moteur (la Platform) qui les gère selon des règles d'optimisation.

1. Plugin

Plugin

Le plugin est une entité logicielle normalisée qui respecte la norme OSGi. Il est défini par deux fichiers :

- **META-INF/MANIFEST.MF** : fichier descriptif pour OSGi
- **plugin.xml** (optionnel) : fichier descriptif pour les extensions Eclipse

Pour simplifier la construction, un fichier est géré par Eclipse :

- **build.properties** : indique répertoire src, bin...

Un projet plugin est par défaut un projet java. Il contient donc automatiquement :

- Un fichier **.project**
- Un fichier **.classpath**

Vue logique du plugin

La vue logique (Package Explorer) représente le plugin selon son organisation :

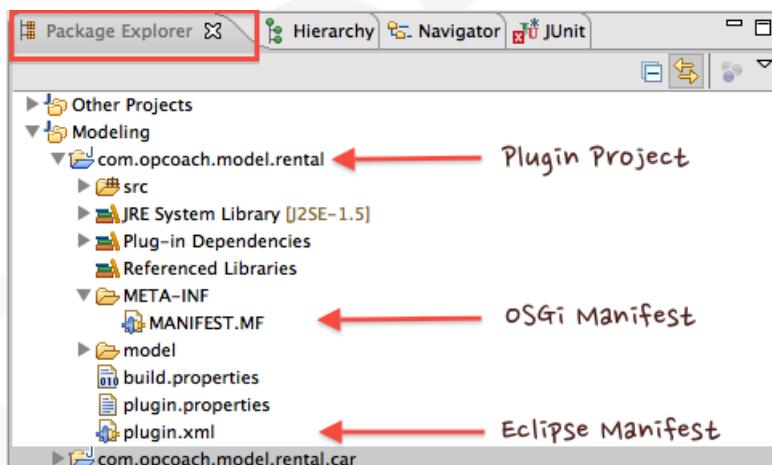


Image 25 Package explorer view

Vue physique du plugin

La vue physique (Navigator) représente le plugin dans le système de fichier :

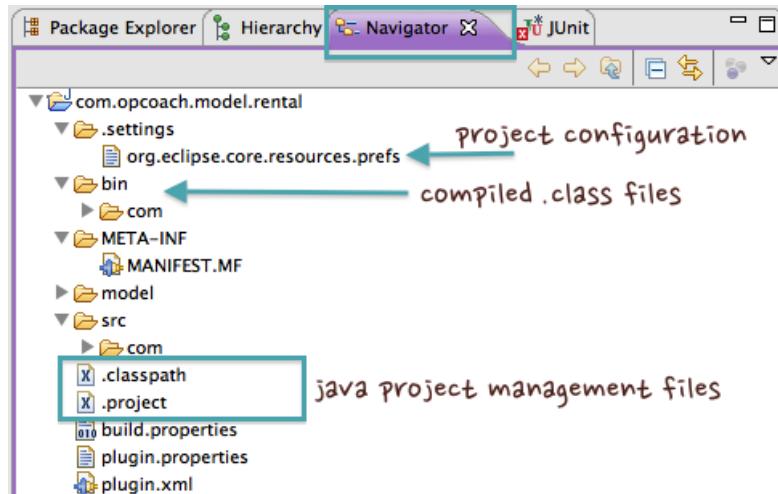
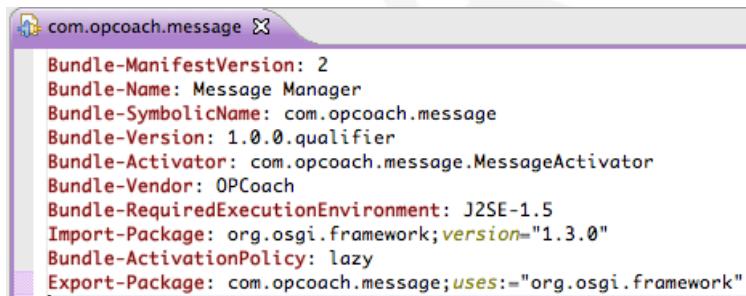


Image 26 Navigator view

Plugin vs Bundle OSGi

Le **MANIFEST.mf** contient les informations de base gérées par OSGi :



```

Bundle-ManifestVersion: 2
Bundle-Name: Message Manager
Bundle-SymbolicName: com.opcoach.message
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: com.opcoach.message.MessageActivator
Bundle-Vendor: OPCoach
Bundle-RequiredExecutionEnvironment: J2SE-1.5
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ActivationPolicy: lazy
Export-Package: com.opcoach.message;uses:="org.osgi.framework"

```

Image 27 Manifest sample

Plugin extension de Bundle

Le **plugin.xml** définit 2 notions supplémentaires : extensions et point d'extension :

- **Point d'extension** : définit un modèle d'un concept que le plugin sait gérer (un menu, une vue, un driver...).
- **Extension** : définit une instance du modèle décrit par le point d'extension (la vue Navigator, le driver Z, ...)

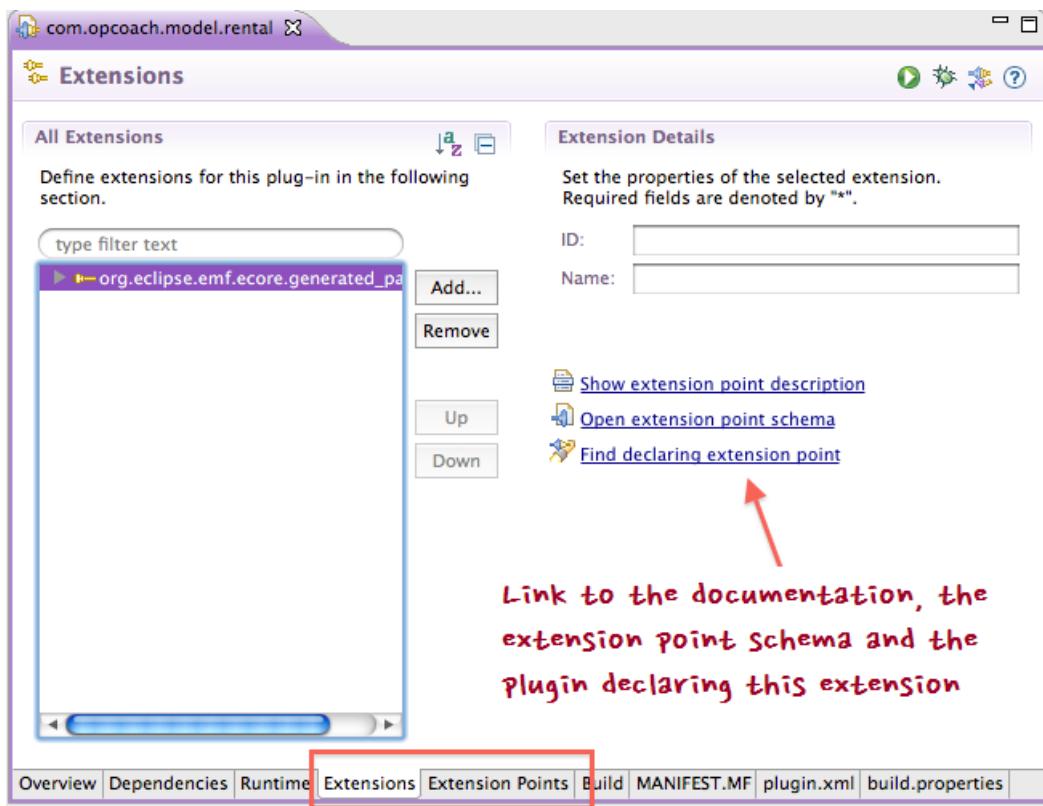


Image 28 Extensions management (plugin.xml editor)

Plugin Fragment

- Le fragment est un complément d'un plugin (host plugin)
- Le fragment est utilisé pour :
 - gérer du code spécifique selon une plate forme
 - compléter un plugin avec des fichiers de properties (i18n)
 - écrire les tests d'un plugin
 - isoler certaines parties du plugin :
 - séparer des parties non compatibles avec RAP par exemple
 - séparer des fonctionnalités optionnelles du plugins nécessitant un accès à du code privé
 - Le fragment a toute la visibilité sur le plugin

Architecture d'une application

Comme pour les bundles OSGi on modularise une application Eclipse en différents plugins :

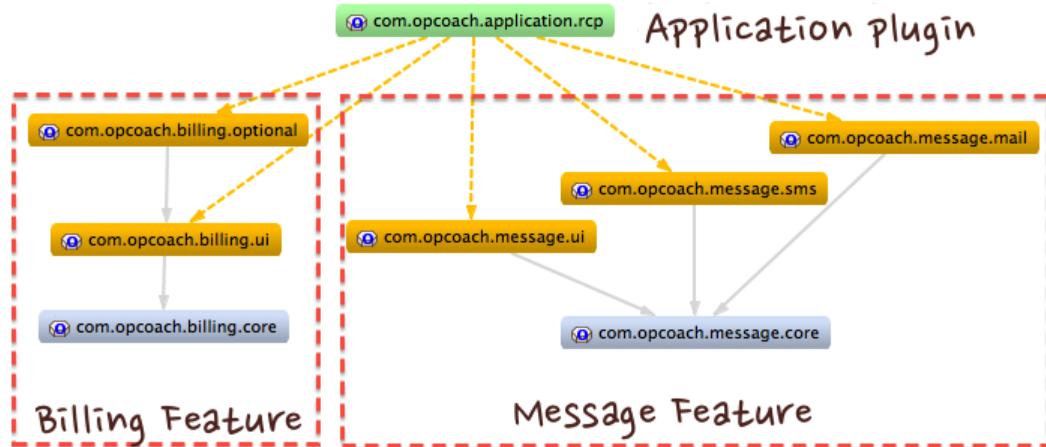


Image 29 Application Architecture

Architecture des tests

On crée un fragment par plugin à tester
Le fragment a ses propres dépendances

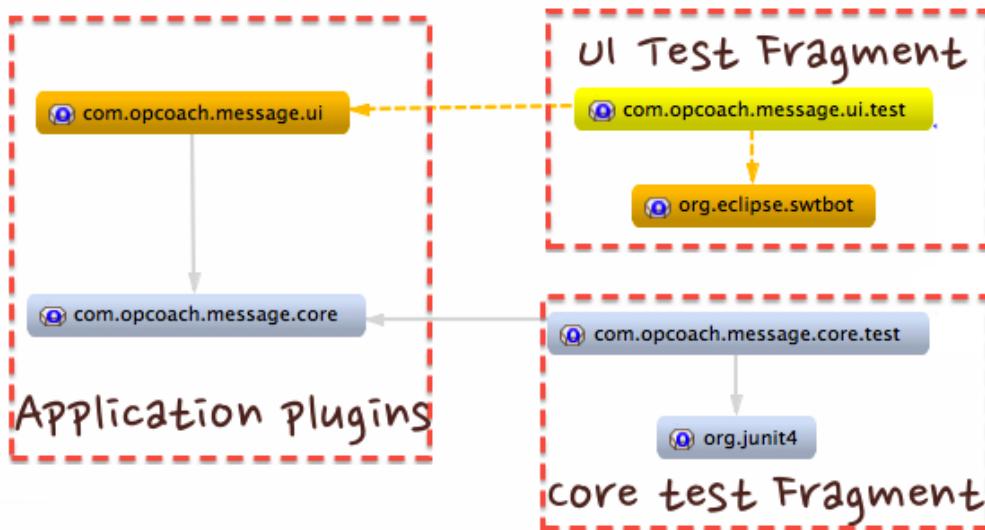


Image 30 Eclipse Test Architecture

2. Feature

Feature

- La feature est un ensemble de bundles, de plugins, de fragments ou de features
- Elle possède également : une licence, et la propriété d'être installable à distance
- Elle se crée dans un projet de nature 'feature', où l'on regroupe les plugins

- Elle gère les compatibilités de version des plugins qu'elle contient ou dont elle dépend.
- Elle peut être définie pour une plateforme en particulier

Vue logique du projet Feature

La feature est représentée dans une vue logique du package explorer

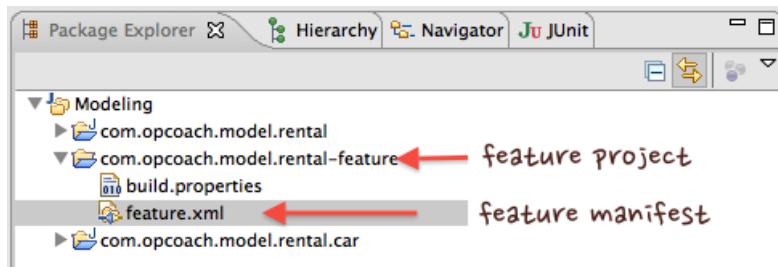


Image 31 Logical view of the feature

Edition de la feature

L'éditeur de feature permet d'indiquer :

- les plugins contenus dans la feature
- les données complémentaires : ID, description, texte de licence, etc...

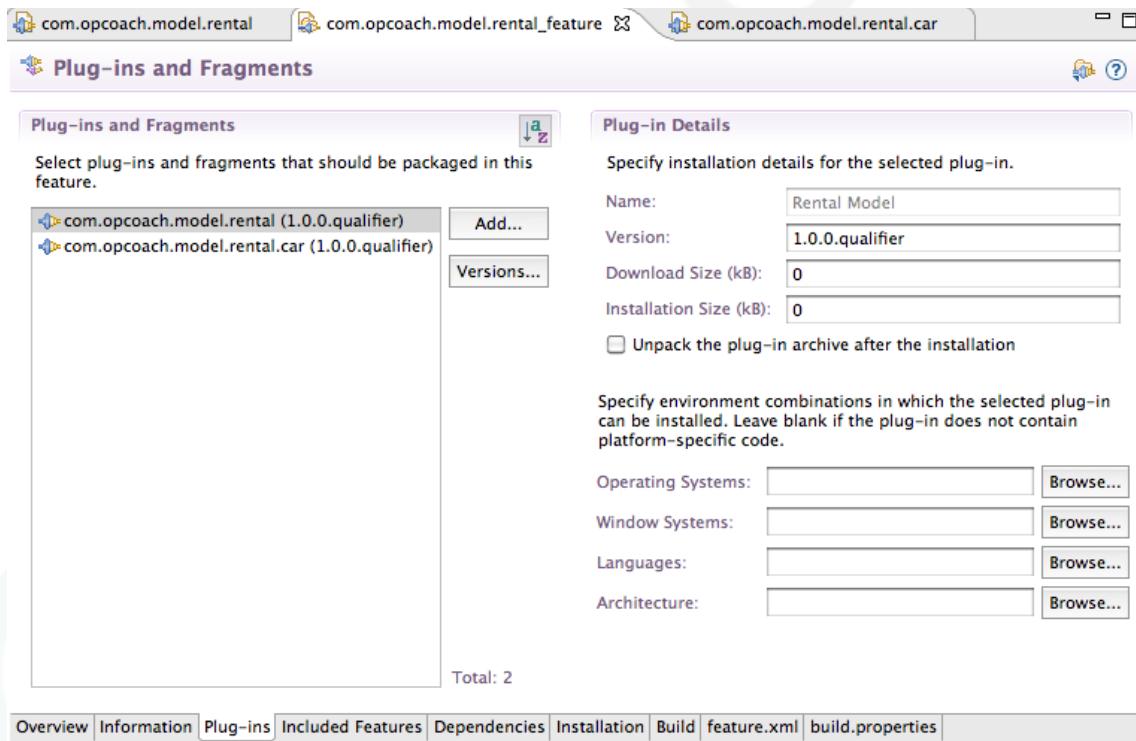
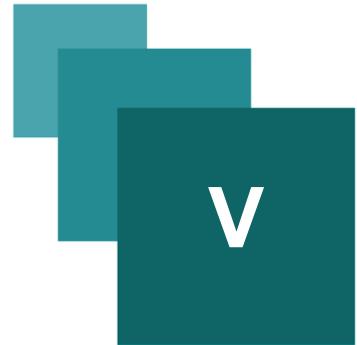


Image 32 Feature editor

Introduction Eclipse 4 RCP



Les raisons du changement

Les avantages d'Eclipse 3.X / RCP :

- architecture basée sur OSGi
- outillage avancé (JDT, PDE, EMF, ...)
- définition de point extensions
- open source and EPL license

Les raisons du changement

Les inconvénients d'Eclipse 3.X :

- une souche logicielle de 8 ans d'age et plus
- beaucoup de points d'extensions et d'extensions dispersés
- une forte liaison au framework (héritage, dépendances...)
- API peu uniformes
- Beaucoup de singltons
- Les mécanismes d'injection ne sont pas implémentés
- Pas de séparation du contenu et de l'apparence (pas de renderer)
- Difficulté de 'styler' l'application
- Pas d'utilisation des services OSGi

Architecture Eclipse 4

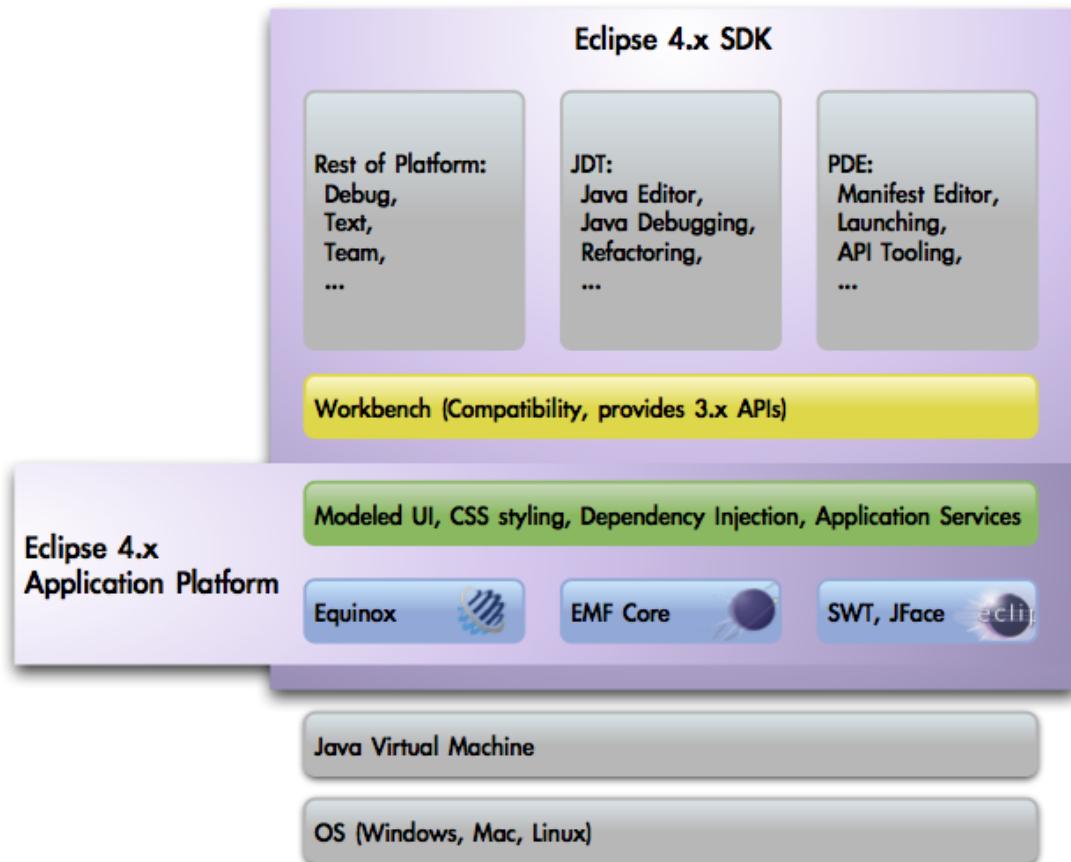


Image 33 e4 Architecture

E4 tooling / Eclipse Mars

Depuis la version Mars d'Eclipse, une partie du tooling est livrée en standard

- Editeur de modèle
- Templates de code java

- Wizards spécifiques dans la catégorie Eclipse 4 :

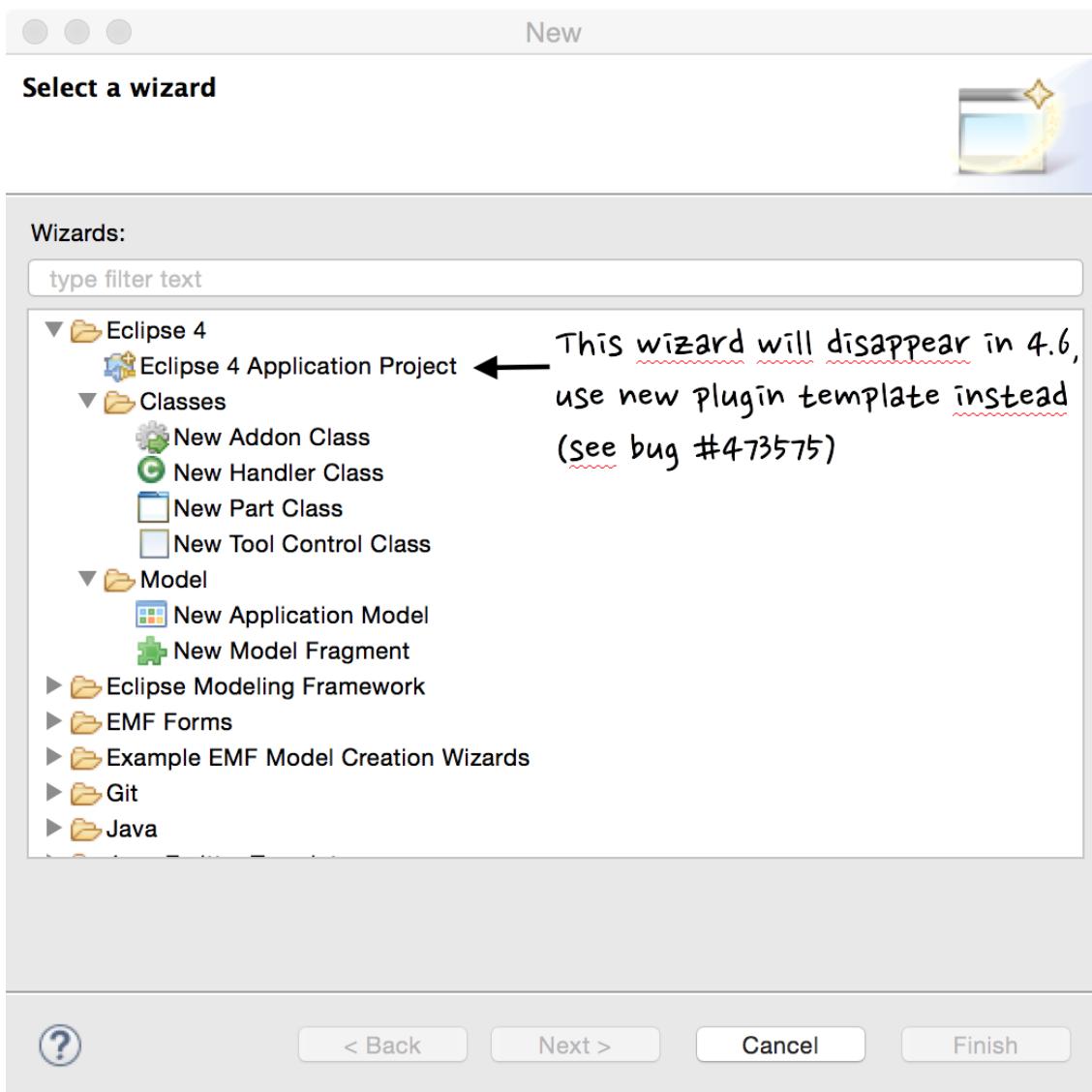


Image 34 e4 wizards

Création d'un projet Eclipse 4

Depuis la version Eclipse Mars, on crée un projet Eclipse 4 comme un plugin traditionnel.
Il faut sélectionner :

- This plugin will make contributions to UI
- Would you like to make a RCP application

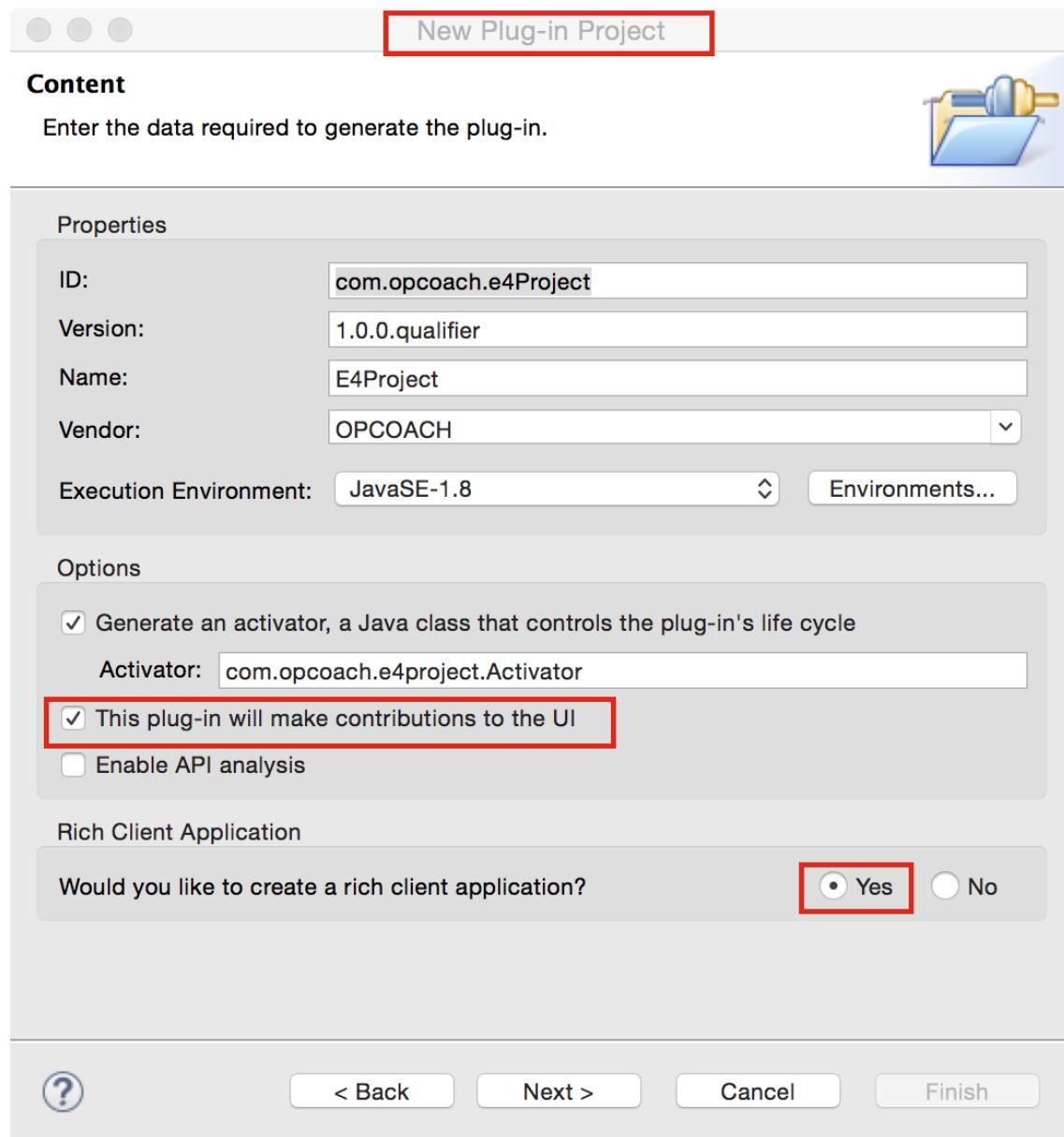


Image 35

Puis sélectionner le template 'Eclipse 4 RCP Application'

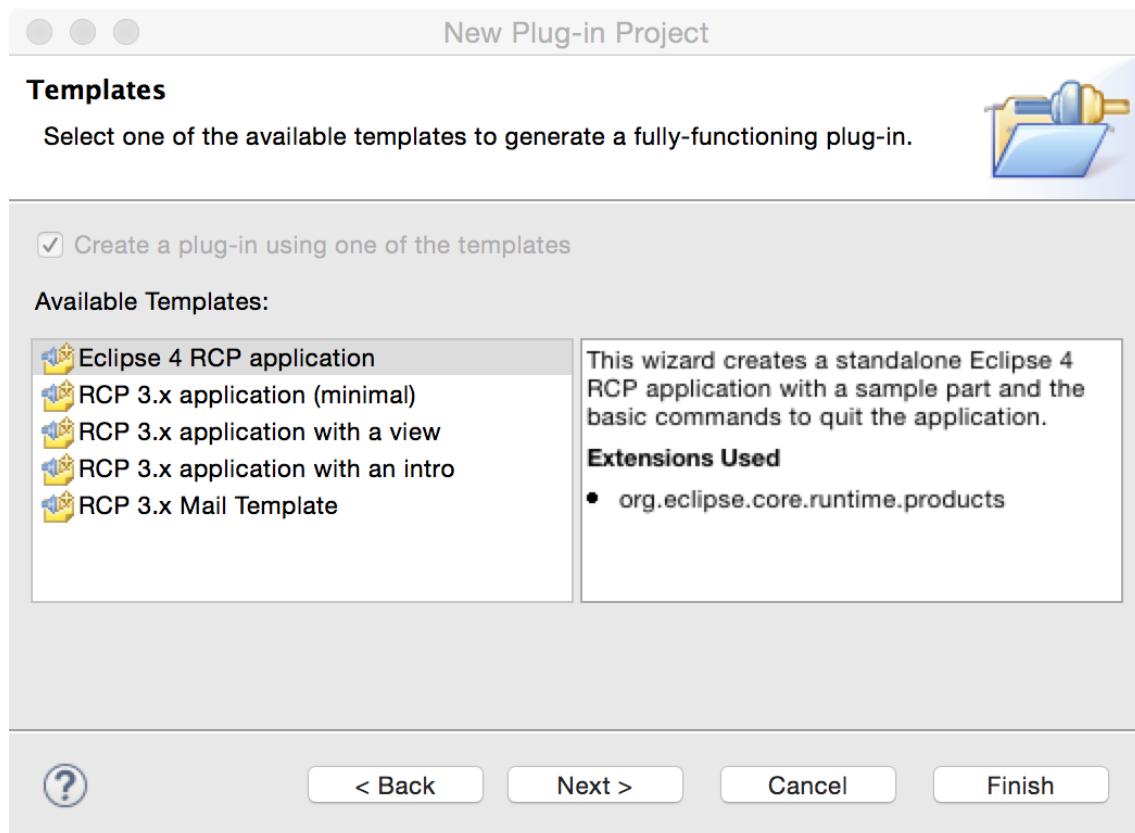


Image 36

E4 Spies

- Les E4 spies sont très utiles pour mettre au point l'application
- Ils permettent de consulter et de naviguer dans :
 - le modèle d'application
 - les contextes d'injection
 - les événements de l'Event Broker
 - les CSS
 - ...
- Eclipse Mars n'est pas livré avec les E4 spies
- Ils seront bientôt installés par défaut
- Pour les installer télécharger le zip sur :
 - <http://www.opcoach.com/e4-tools-and-spies-pour-eclipse-mars/>¹⁶
 - <http://download.eclipse.org/e4/downloads>¹⁷ (si mis à jour)

Télécharger le zip et l'installer dans Eclipse :

- Menu Help -> Install New Software..
- 'Add..', 'Archive..'

16 - <http://www.opcoach.com/e4-tools-and-spies-pour-eclipse-mars/>

17 - <http://download.eclipse.org/e4/downloads>

Installer ensuite tous les spies :

Name	Version
<input checked="" type="checkbox"/> ▾  Eclipse 4 - All Spies	
<input checked="" type="checkbox"/>  Eclipse 4 - All Spies (Incubation)	0.1.0.v20150508-0800
<input type="checkbox"/> ▶  Eclipse 4 - Bundle Spy	
<input type="checkbox"/> ▶  Eclipse 4 - Context Spy	
<input type="checkbox"/> ▶  Eclipse 4 - CSS Spy	
<input type="checkbox"/> ▶  Eclipse 4 - Event Spy	
<input type="checkbox"/> ▶  Eclipse 4 - Model Spy	
<input type="checkbox"/> ▶  Eclipse 4 - Preference Spy	

Image 37 E4 tooling

Quelques liens complémentaires sur E4

- Le forum E4 : <http://www.eclipse.org/forums/index.php/f/12/>¹⁸
- Le wiki E4 : <http://wiki.eclipse.org/Eclipse4/RCP>¹⁹
- Le tutorial de Jonas : <http://eclipsesource.com/en/info/eclipse-4-tutorial/>²⁰
- Le tutorial de Lars : <http://www.vogella.com/articles/EclipseRCP/article.html>²¹
- Le bugzilla Eclipse : <http://bugs.eclipse.org>²²

Exercice

EXO EAP 010 : installer Eclipse et les E4 spies

18 - <http://www.eclipse.org/forums/index.php/f/12/>

19 - <http://wiki.eclipse.org/Eclipse4/RCP>

20 - <http://eclipsesource.com/en/info/eclipse-4-tutorial/>

21 - <http://www.vogella.com/articles/EclipseRCP/article.html>

22 - <http://bugs.eclipse.org>

Application Model



Structure d'une interface

Toute interface d'application contient des éléments récurrents :
des vues, perspectives, commandes

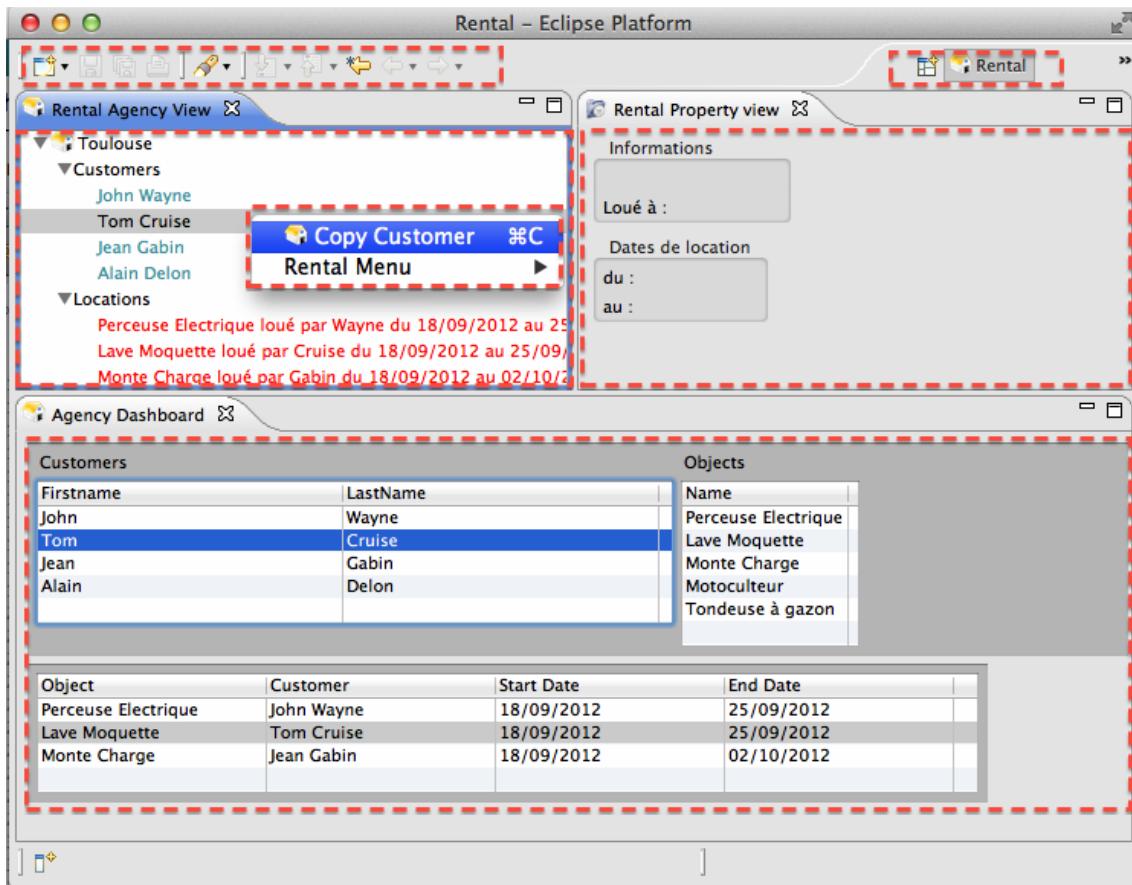


Image 38 UI anatomy

De Eclipse 3 à Eclipse 4

Eclipse 3 :

- Tous ces composants sont définis par points d'extension répartis dans différents plugins
- Les menus principaux, la perspective principale sont définis par différentes classes (advisors)

Eclipse 4 :

- Le modèle d'application assemble tous ces concepts :

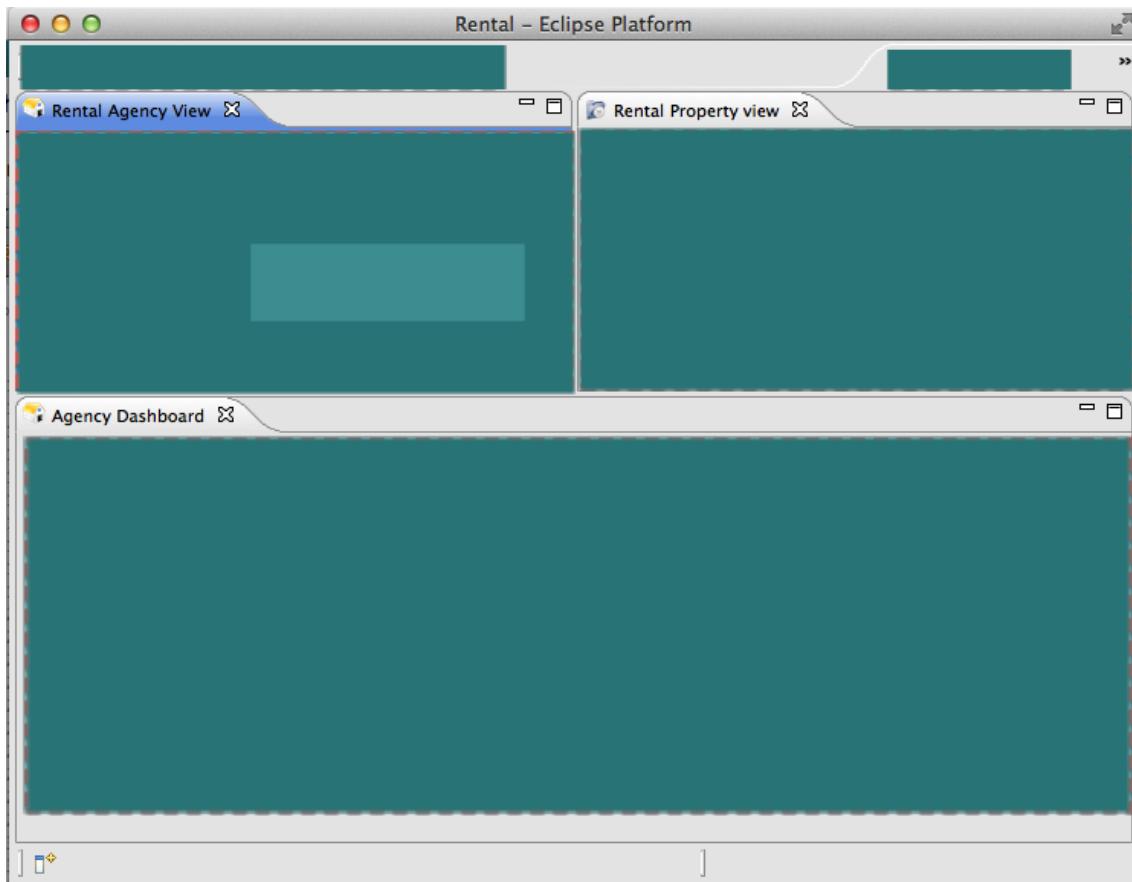


Image 39 UI in application model

Le modèle d'application E4

- C'est un modèle global qui rassemble les points d'extensions habituels :
 - vue, perspective, menus (visuels)
 - commande, handlers, key bindings (non visuels)
- Il décrit simplement la structure de l'IHM sans détailler son contenu
- Sa structure est définie par un méta modèle décrit en Ecore
- Il peut s'éditer avec un éditeur dédié
- Il peut être modifié et l'ihm est remise à jour
- Ce modèle est indépendant de l'affichage
 - un 'renderer' spécifique permet de l'afficher (swt et javafx)
- Les classes référencées dans le modèle d'application sont alors des POJOS annotés

E4 Project

Le modèle d'application se trouve dans le projet e4

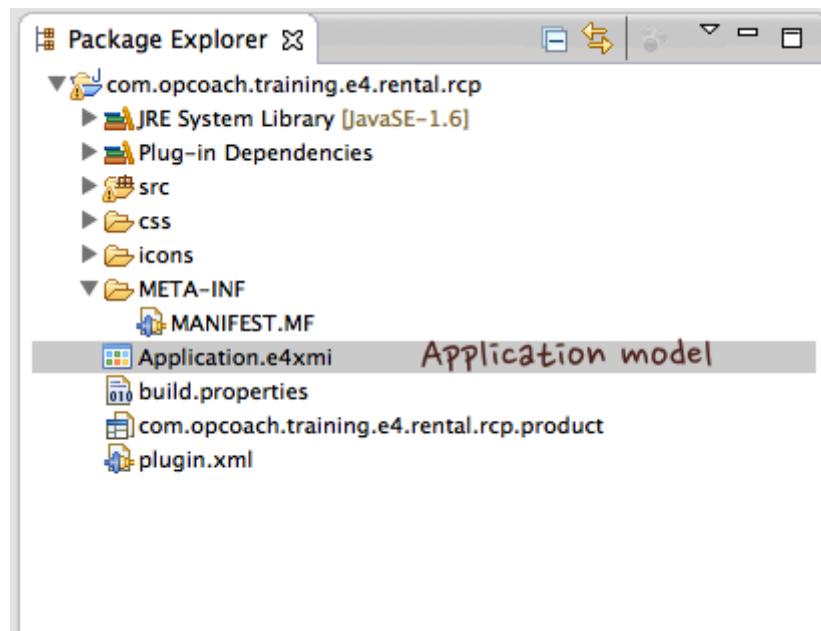


Image 40 e4 project structure

Le modèle d'application

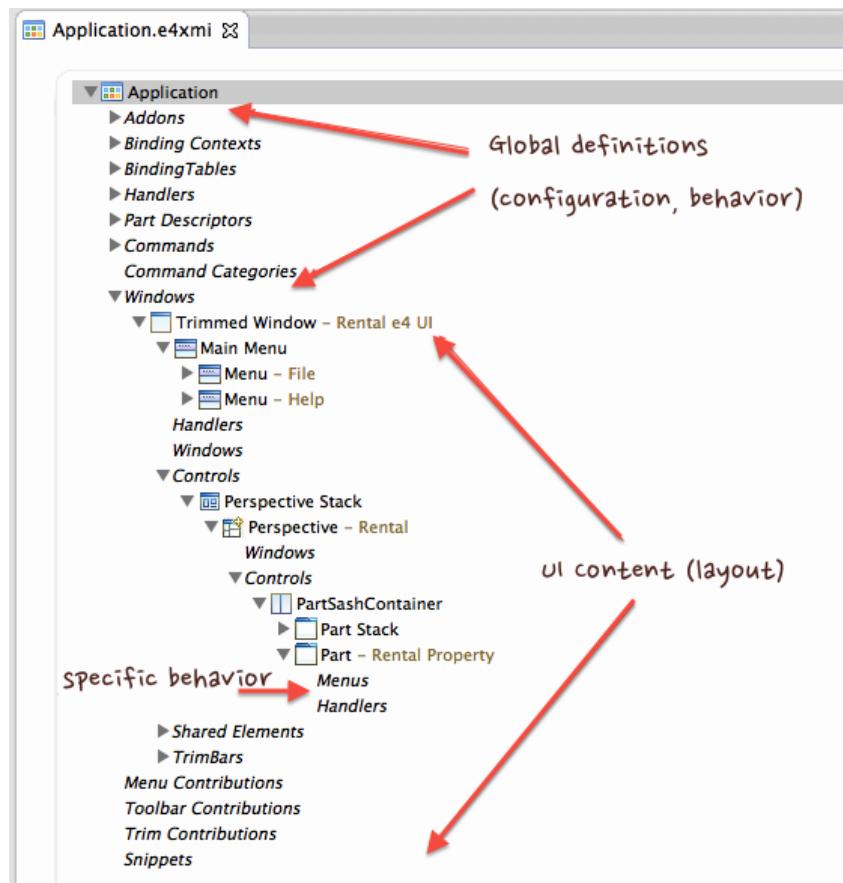


Image 41 Application Model

Les URI dans le modèle d'application

Le modèle d'application référence des classes ou des ressources (icônes...)

Identification d'une classe :

- bundleclass://bundleID/package.classname
- bundleclass://com.opcoach.training.e4.rental.ui/com.opcoach.training.rental.e4.ui.views.AgencyView

Identification d'une resource :

- platform:/plugin/bundleID/path/filename
- platform:/plugin/com.opcoach.training.e4.rental.ui/icons/Agency.png

Produit e4

Le modèle d'application doit être référencé dans le point d'extension product :
`org.eclipse.core.runtime.products`

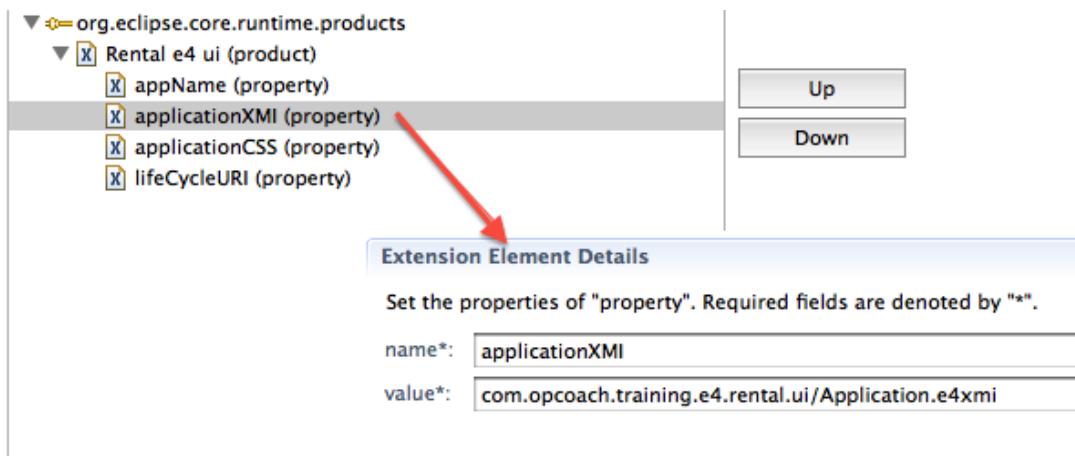


Image 42 product extension point



Remarque : Application model et vues 3.X

- Les points d'extension de vue, perspective, etc.. définis en 3.X sont traduits dans l'application model par la couche de compatibilité
- l'IDE Luna 4.4 n'a pas été entièrement porté et fonctionne avec la couche de compatibilité

Méta modèle du modèle d'application

Le modèle d'application est défini par un modèle.ecore :

- Cf : org.eclipse.e4.ui.model.workbench/model/UIElements.ecore

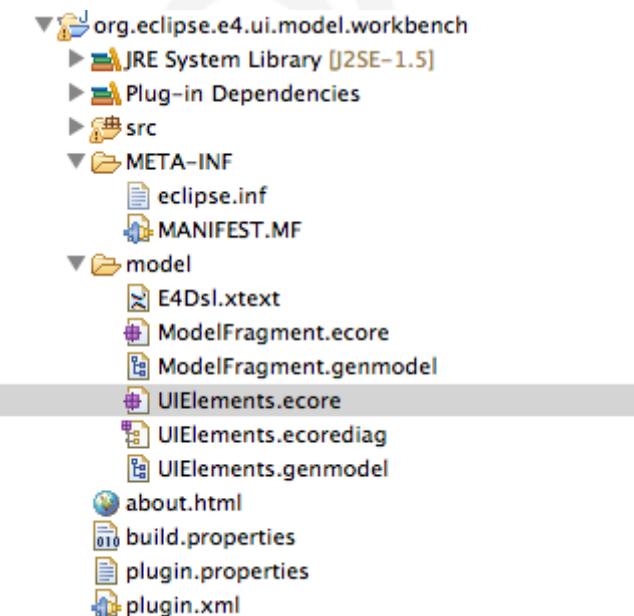


Image 43 Model Application project

Les différentes classes du modèle

UIElements définit de nombreuses classes.

Les classes principales pour l'IHM visible sont :

- **MApplication** : le modèle d'application
- **MTrimmedWindow** : une fenêtre de l'application
- **MPart** : une partie de l'IHM (vue, éditeur)
- **MPerspective** : une perspective (facultatif)

Chacune de ces classes possède son propre contexte d'injection.

L'éditeur de modèle

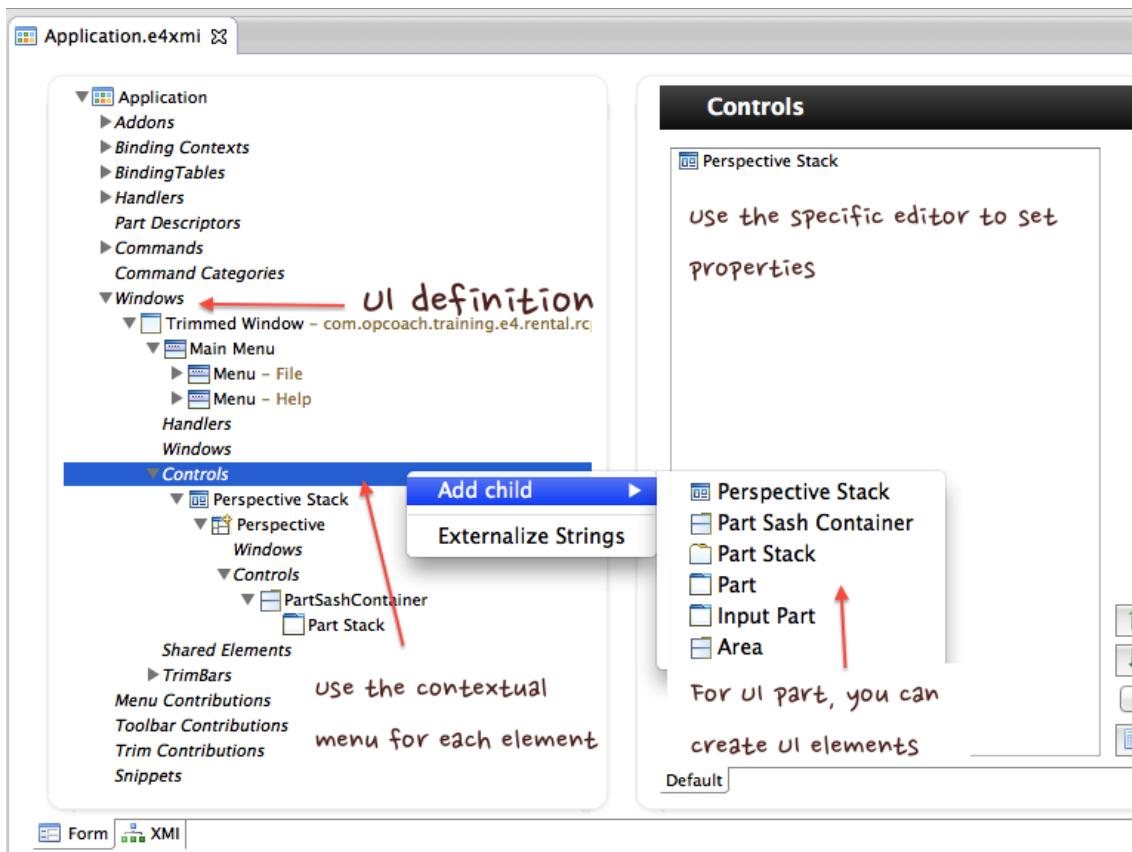


Image 44 Model editor

La visualisation du modèle au runtime

- A tout moment on peut voir le modèle d'application réel
- Raccourci : Alt Shift F9
- Il faut mettre le plugin [org.eclipse.e4.tools.emf.liveeditor](#) dans la launch configuration
- On peut éditer le modèle et voir les changements

Launch Configuration

VII

Introduction

- Les launch configurations permettent de lancer une application
- Pour les applications Eclipse, elle définissent également leur contenu
- On accède aux launch configurations dans le menu Run->Run Configurations...

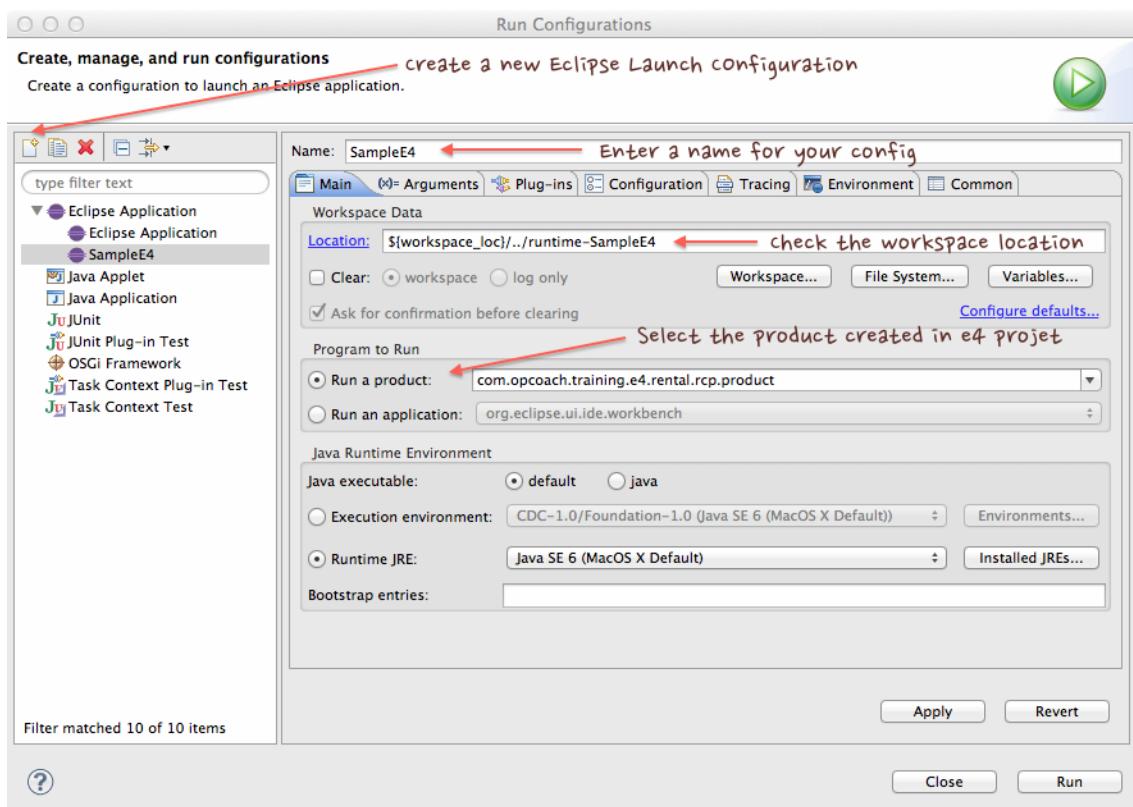


Image 45 launch config

Le lancement d'un plugin

- Pour lancer un plugin il faut exécuter une autre instance d'application Eclipse.
- On utilise les **launch configurations** de type 'Eclipse'
- Dans l'onglet 'plugins' (3e onglet) :
 - On choisit les plugins principaux du workspace de développement à lancer
 - On ajoute les plugins requis
- L'espace de travail (workspace) indiqué dans le premier onglet est utilisé. Il contiendra :
 - la configuration et le cache des plugins choisis (répertoire .metadata)
 - les fichiers utilisateurs éventuels

- Pour supprimer le cache on peut ajouter (2e onglet) :
 - **-clean** : efface la copie des plugins dans le .metadata
 - **-clearPersistedState** : efface le cache du modèle d'application (seulement Eclipse 4)

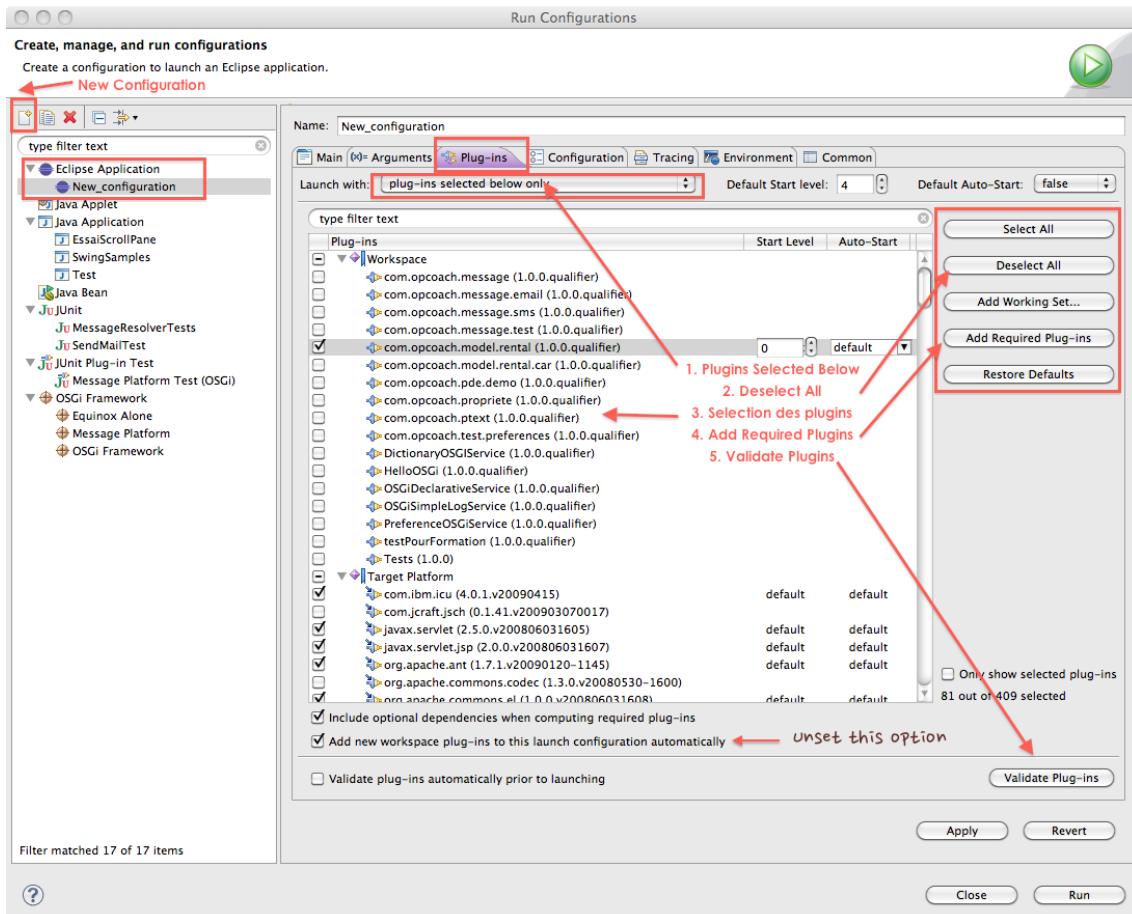


Image 46 Eclipse launch configuration



Attention: Gestion du renderer e4

Si vous obtenez cette erreur :

```

Problems @ Javadoc Declaration Console <terminated>
<terminated> SampleE4 [Eclipse Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (19 sept. 2012 14:24:52)
!ENTRY org.eclipse.e4.ui.workbench 4 0 2012-09-19 14:24:58.682 This error means that eclipse.platform is missing !
!MESSAGE Unable to create class 'org.eclipse.e4.ui.internal.workbench.swt.PartRenderingEngine' from bundle '9'
!STACK 0
org.eclipse.e4.core.di.InjectionException: java.lang.IllegalStateException: Could not create any rendering factory. Aborting ...
    at org.eclipse.e4.core.internal.di.MethodRequestor.execute(MethodRequestor.java:63)
    at org.eclipse.e4.core.internal.di.InjectorImpl.processAnnotated(InjectorImpl.java:857)
    at org.eclipse.e4.core.internal.di.InjectorImpl.inject(InjectorImpl.java:111)
    at org.eclipse.e4.core.internal.di.InjectorImpl.internalMake(InjectorImpl.java:318)
    at org.eclipse.e4.core.internal.di.InjectorImpl.make(InjectorImpl.java:240)
    at org.eclipse.e4.core.contexts.ContextInjectionFactory.make(ContextInjectionFactory.java:161)
    at org.eclipse.e4.ui.internal.workbench.ReflectionContributionFactory.createFromBundle(ReflectionContributionFactory.java:10)

```

Image 47 Renderer missing

Il faut ajouter manuellement le plugin de platform e4 (renderer) :

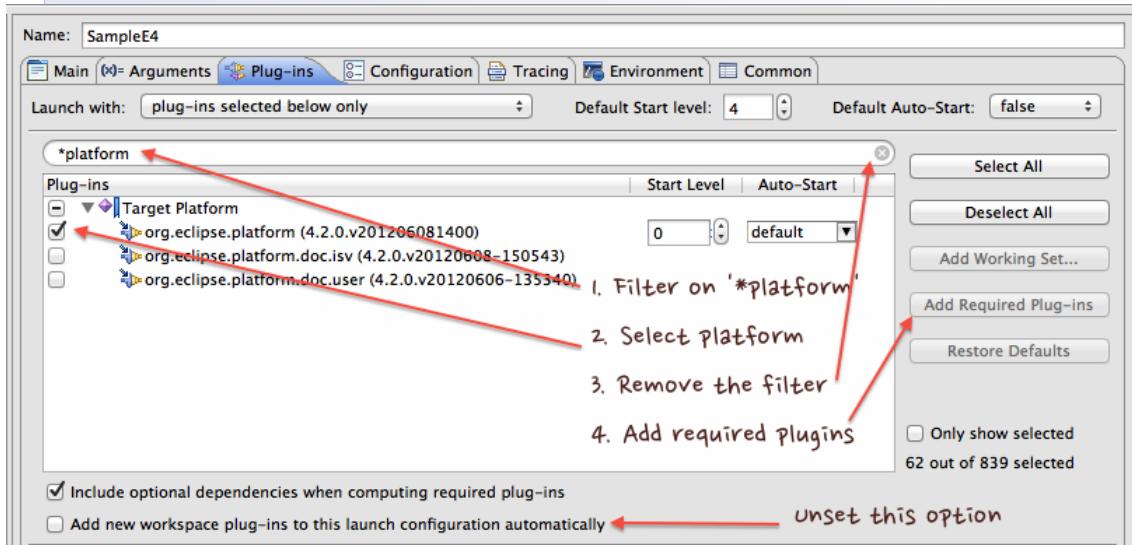


Image 48 Select the platform !

Partage de la launch configuration

La launch configuration peut être sauvee et partagée dans un projet

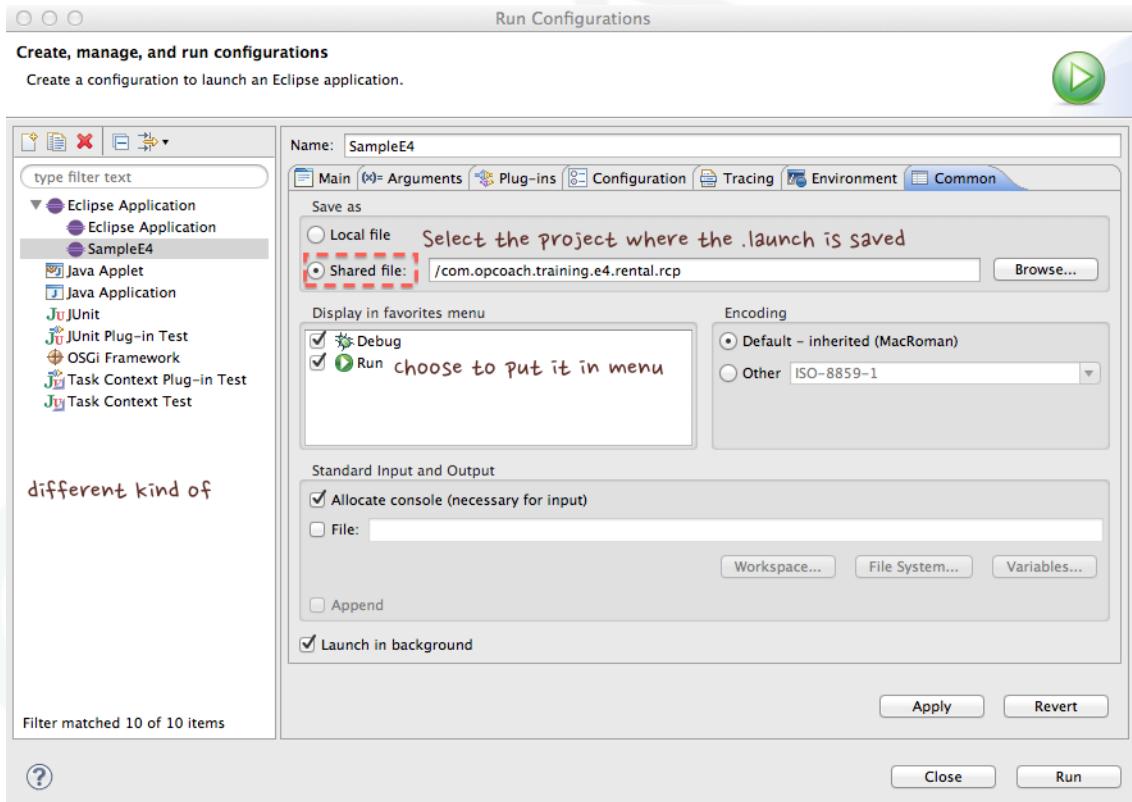


Image 49 Share your launch !

Exercice EAP 015

Création et lancement d'une application E4



Plugin Development Environment



Présentation

Le PDE est un ensemble de plugins pour aider à créer des plugins, features...

Il est bâti au dessus du JDT (Java Development Toolkit)

Les wizards proposés par le PDE

Le PDE propose des créations de projets ou de fichiers

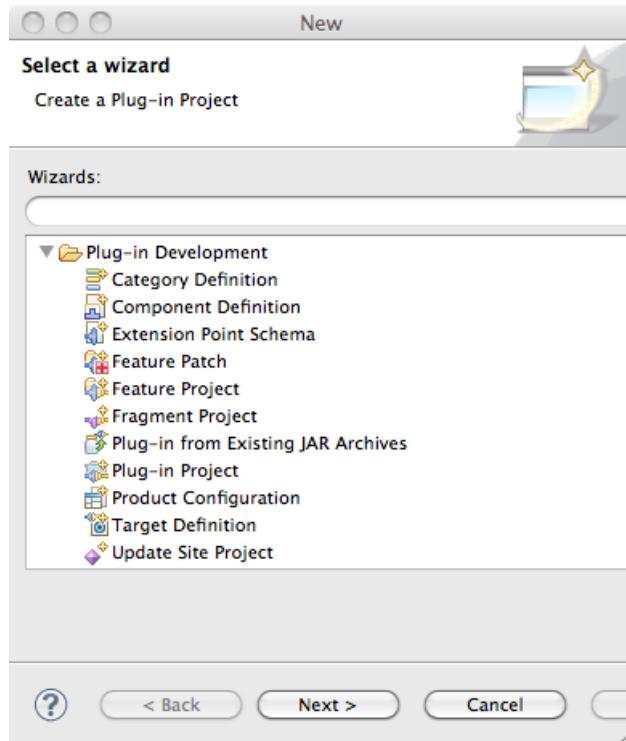


Image 50 PDE Wizards

Plugin Project

Le plugin project est utilisé à la fois pour :

- créer des plugins Eclipse
- créer des bundles OSGi

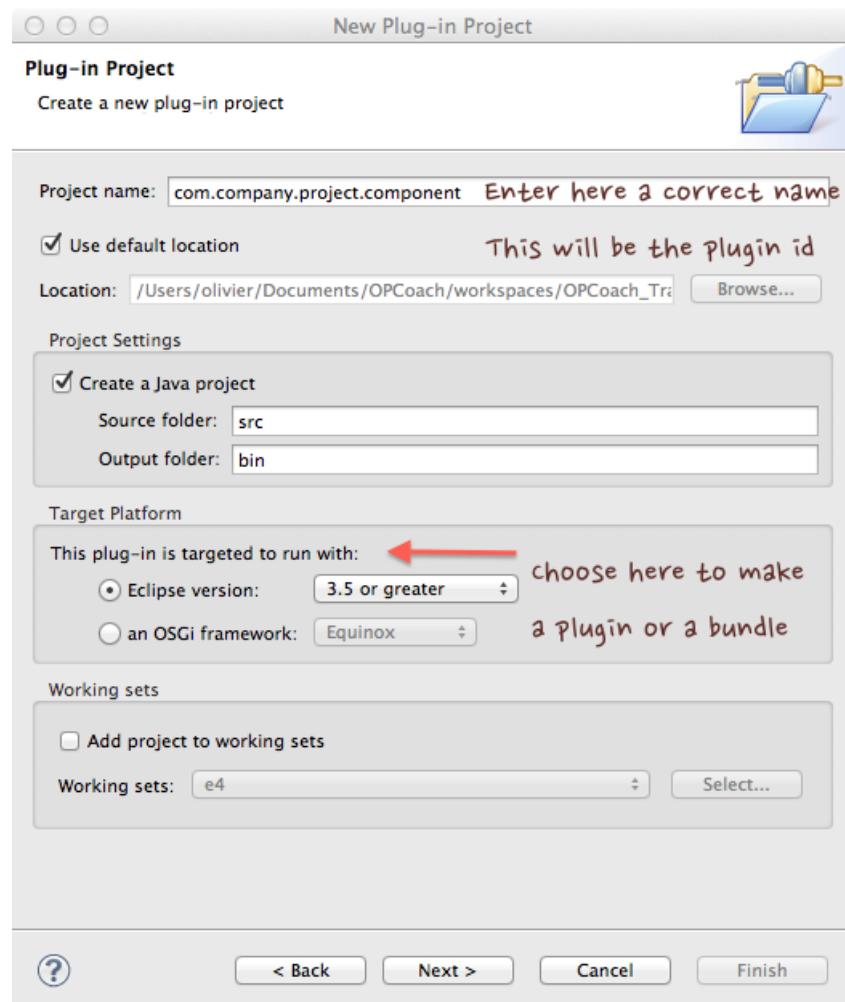


Image 51 PDE new plugin wizard, page 1

Choix d'une application RCP et autres paramétrages

Deuxième écran

- Version courante
- Activator
- Contribution à l'UI
- Analyse de l'API
- Structure d'appli RCP (non modifiable)

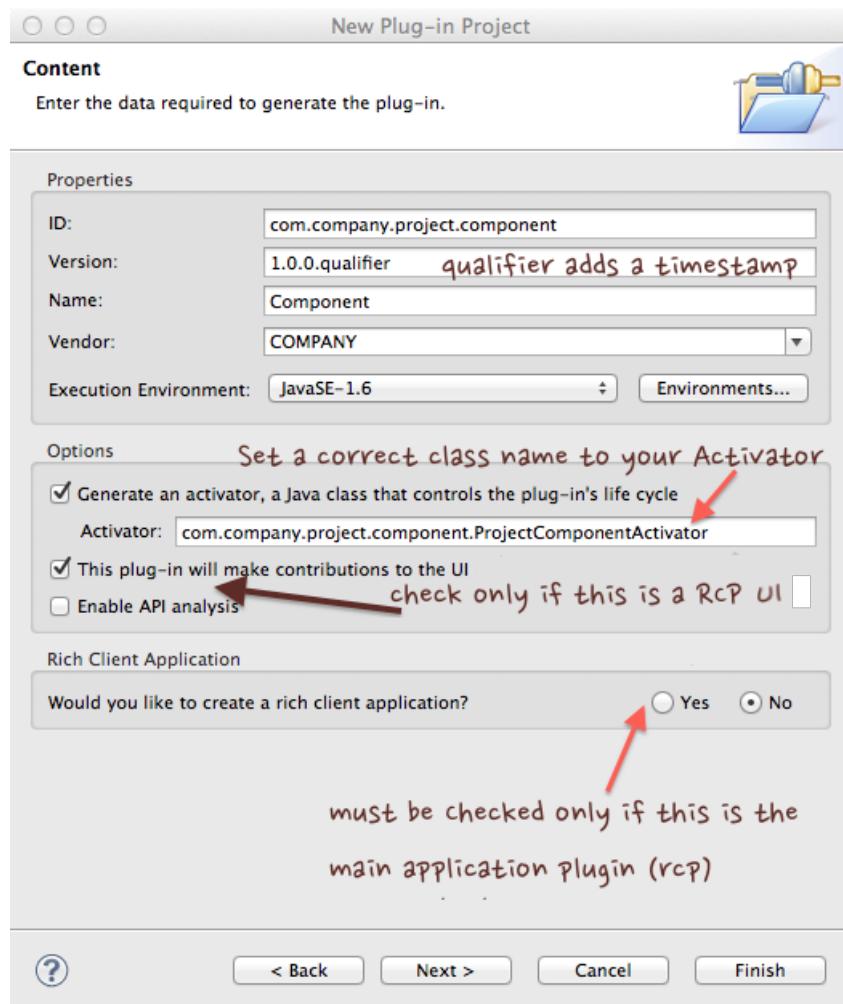


Image 52 new plugin wizard page 2

Overview du plugin

L'overview contient

- L'identité du plugin
- Des raccourcis sur des actions

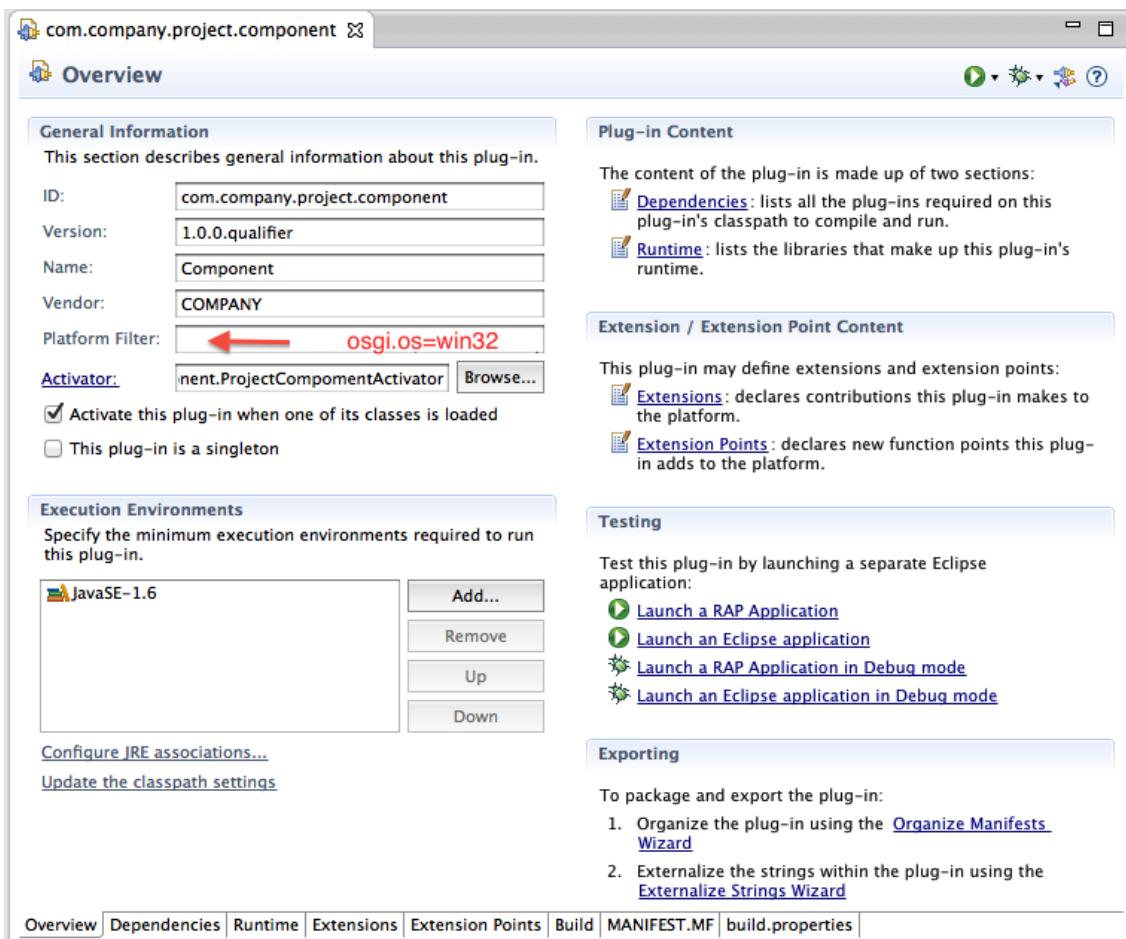


Image 53 PDE Overview

Dépendances du plugin

Les dépendances sont gérées :

- soit en direct
- soit en mode 'caché'
- soit sur des packages

On peut réexporter une dépendance (properties).

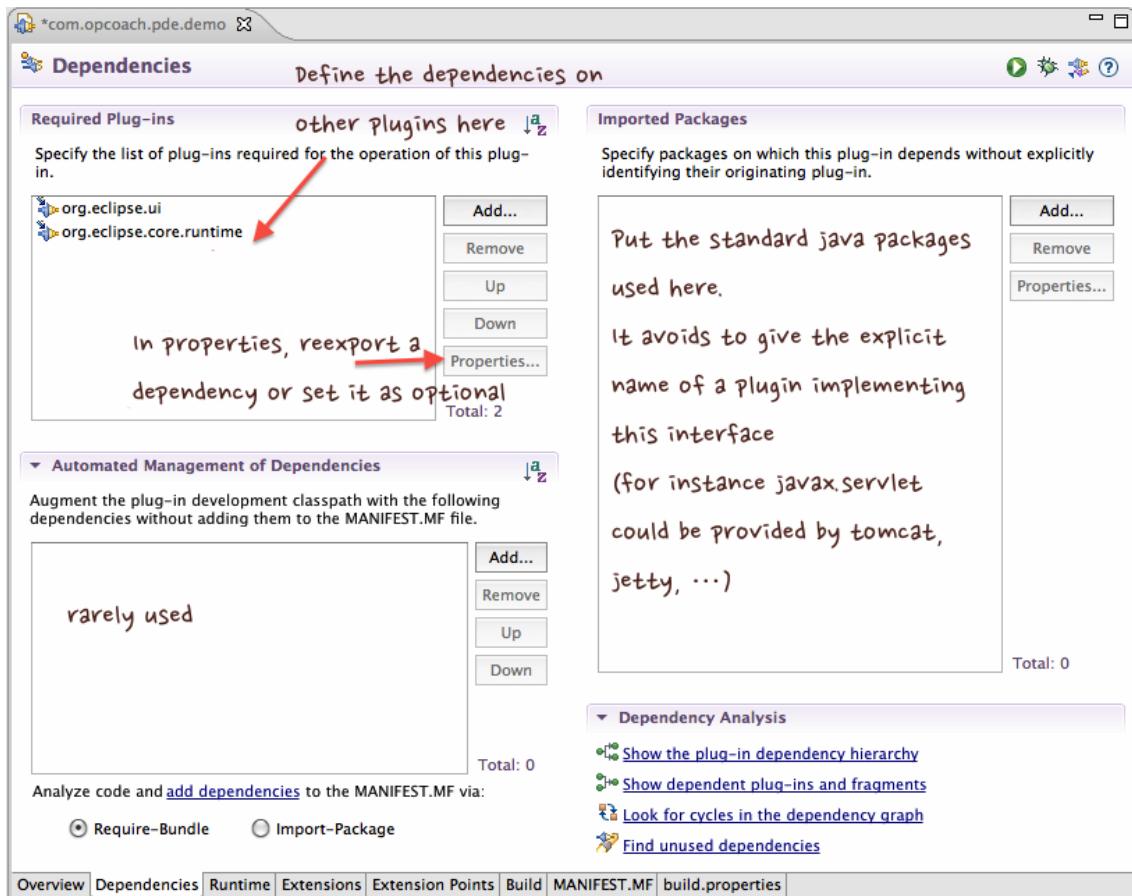


Image 54 Dependencies between plugins

Runtime du plugin

Le runtime permet de régler

- le classpath local du plugin/bundle
- les packages publiés à l'extérieur

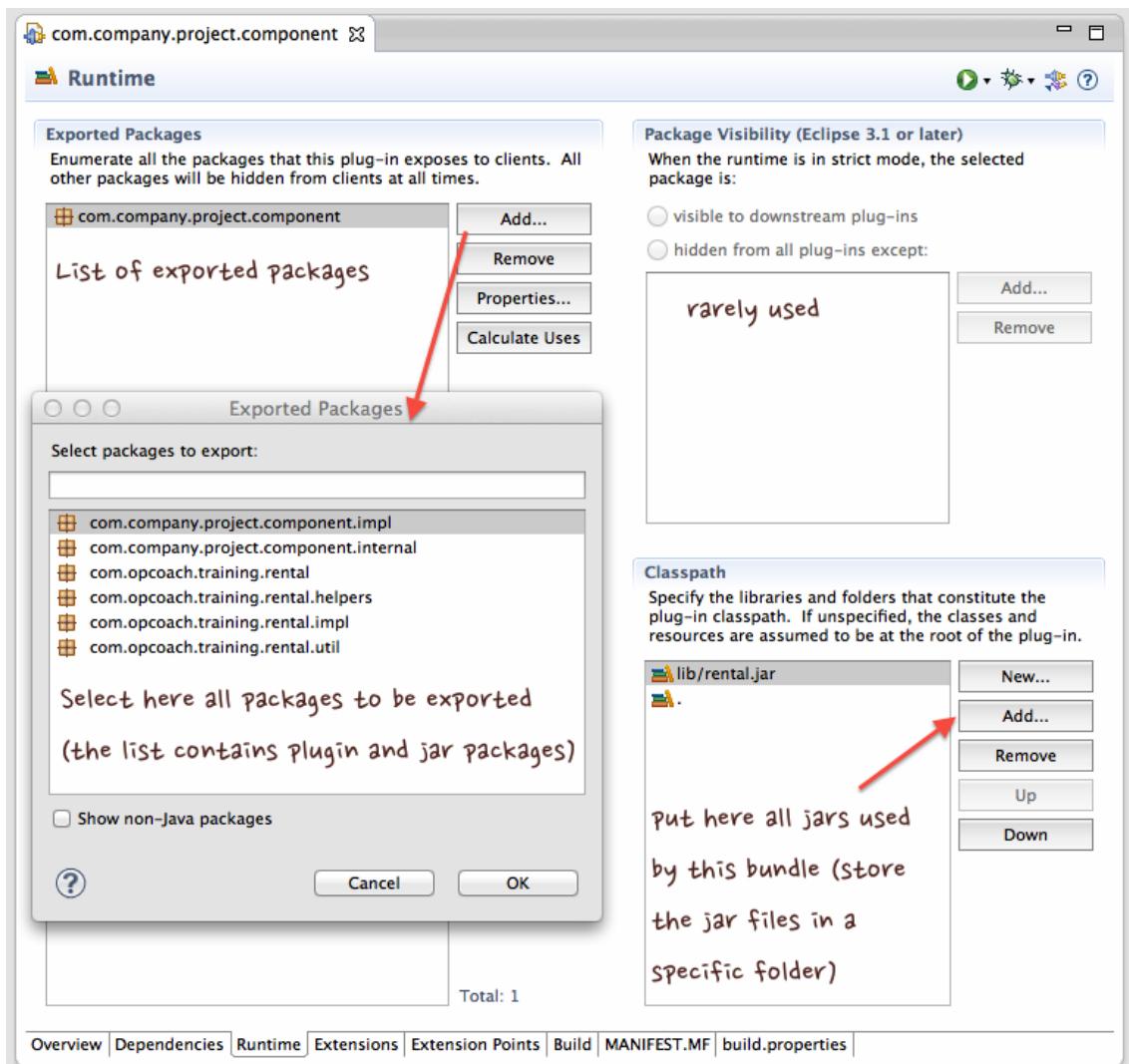


Image 55 PDE Runtime

Extensions du plugin

L'onglet des extensions :

- définit les paramètres des points d'extensions étendus
- peut étendre des points d'extension locaux
- met à jour les dépendances entre plugins
- offre des actions de navigation inter plugins

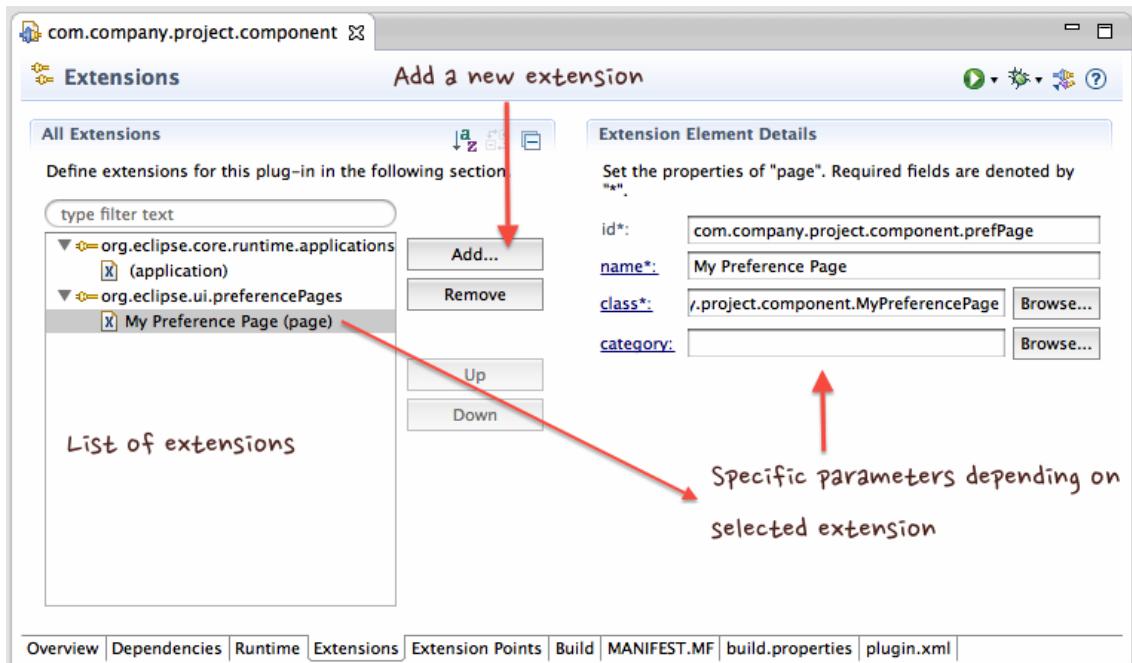


Image 56 Extensions

Points d'extension du plugin

Associe des noms de points d'extension aux schémas XML

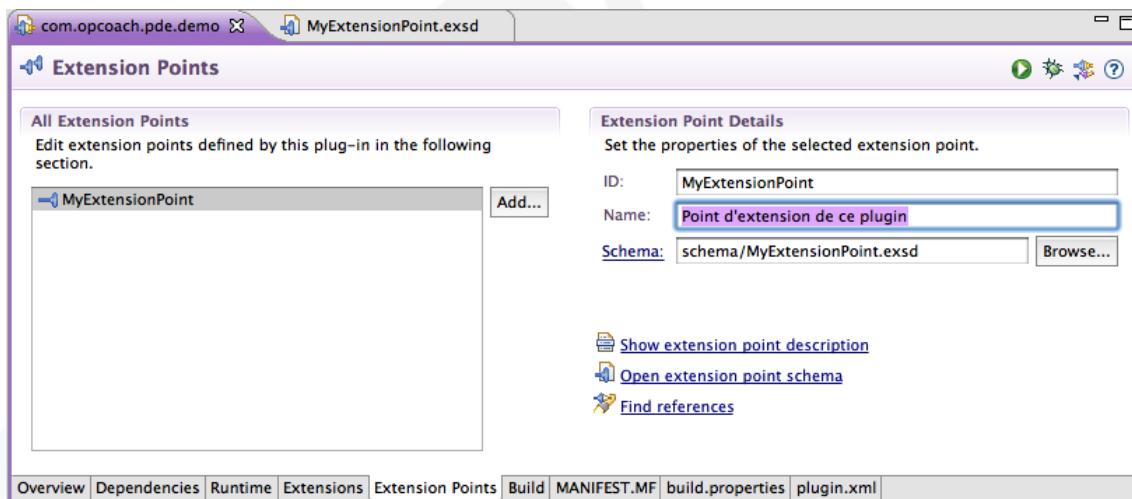


Image 57 Extension point definition

Build du plugin

L'onglet de build règle les paramètres pour le ant

- version binaire
- version source

Le ant est ensuite généré automatiquement

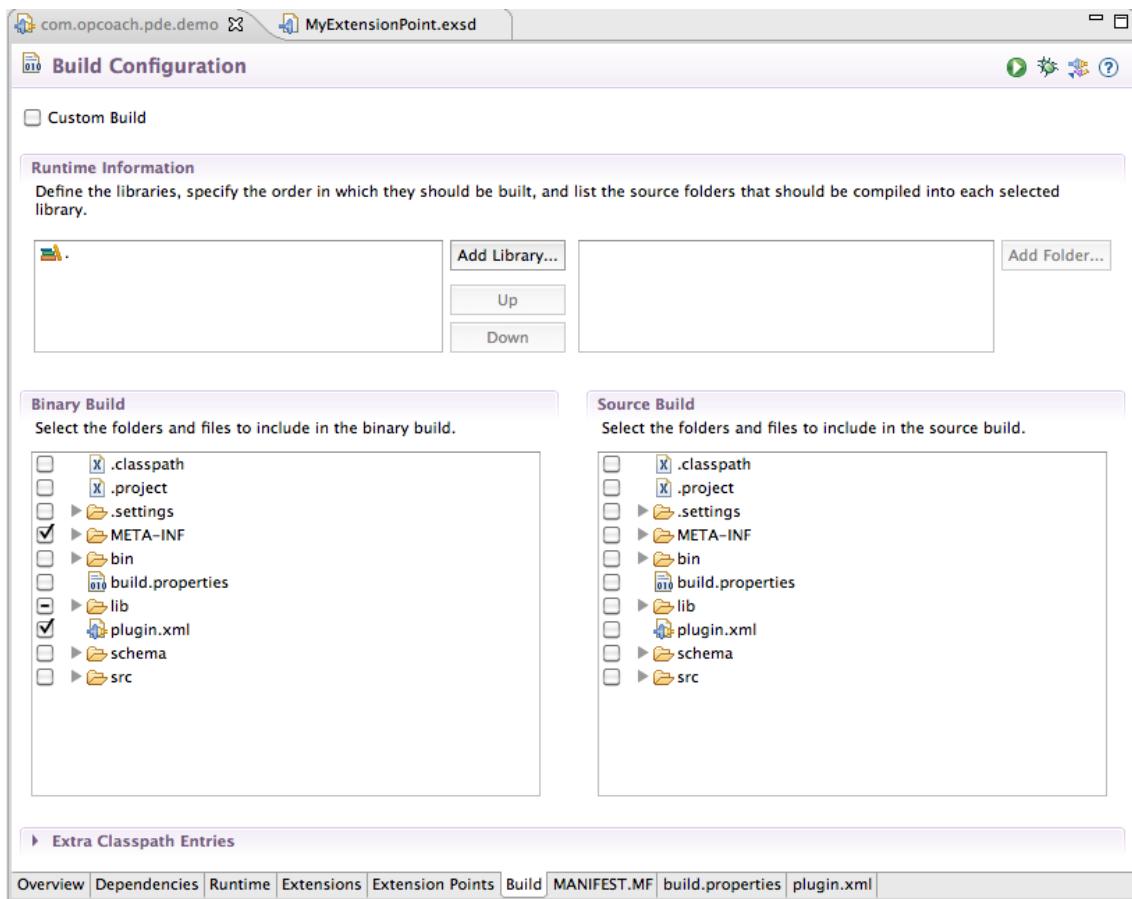
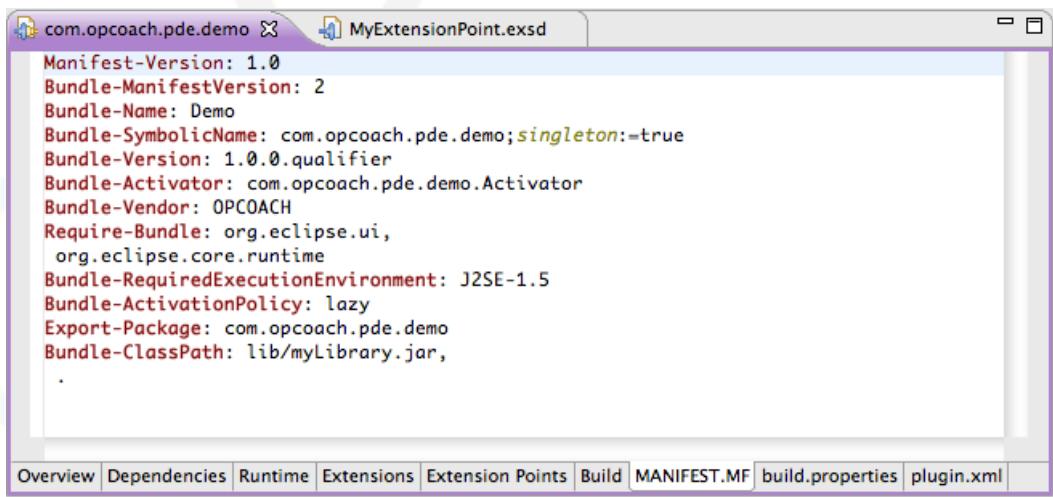


Image 58 Build configuration

Fichier MANIFEST.MF obtenu



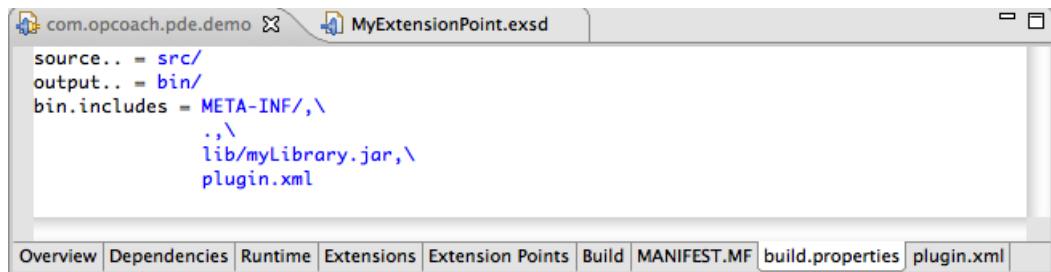
```

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Demo
Bundle-SymbolicName: com.opcoach.pde.demo;singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: com.opcoach.pde.demo.Activator
Bundle-Vendor: OPCOACH
Require-Bundle: org.eclipse.ui,
    org.eclipse.core.runtime
Bundle-RequiredExecutionEnvironment: J2SE-1.5
Bundle-ActivationPolicy: lazy
Export-Package: com.opcoach.pde.demo
Bundle-ClassPath: lib/myLibrary.jar,
.

```

Image 59 MANIFEST.MF

Fichier de build obtenu



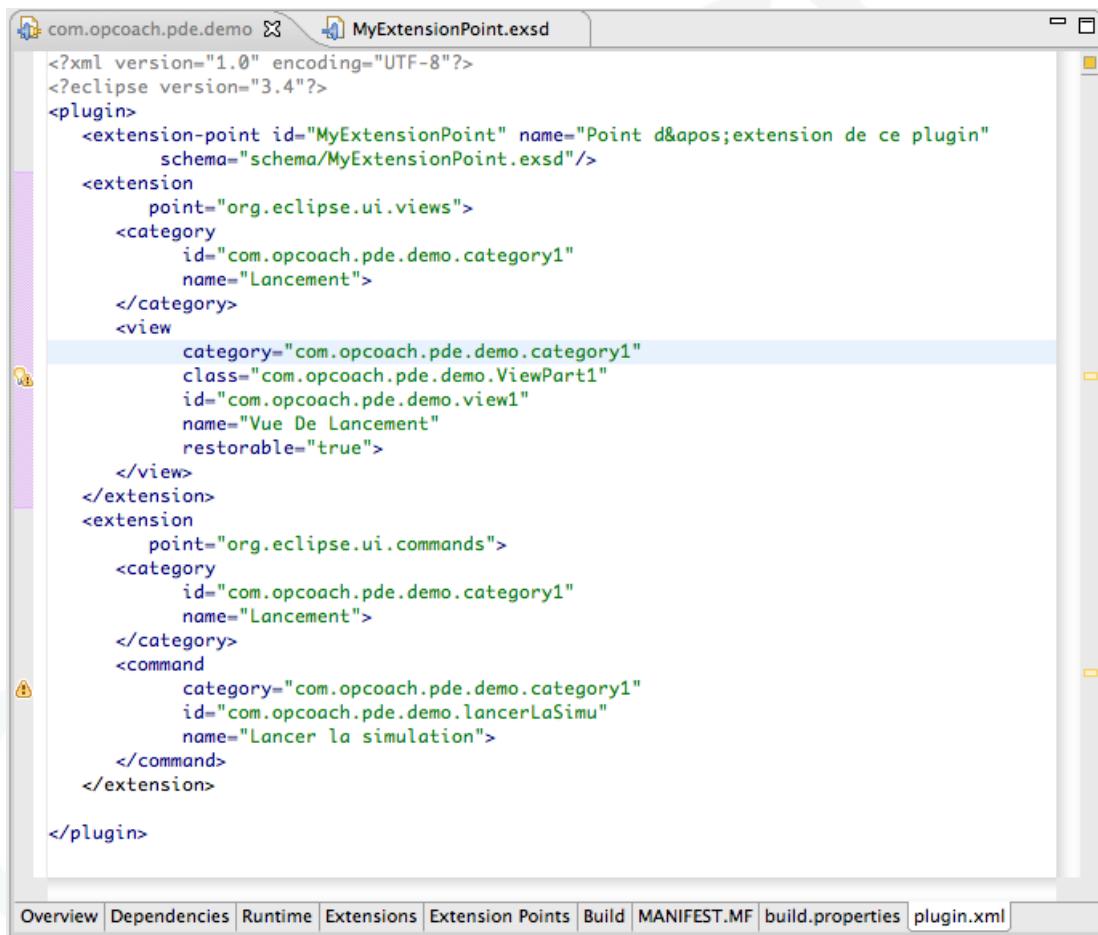
```

source.. = src/
output.. = bin/
bin.includes = META-INF/,\
               .,\
               lib/myLibrary.jar,\
               plugin.xml
  
```

Overview Dependencies Runtime Extensions Extension Points Build MANIFEST.MF build.properties plugin.xml

Image 60 build.properties

Fichier du plugin



```

<?xml version="1.0" encoding="UTF-8"?>
<eclipse version="3.4">
<plugin>
  <extension-point id="MyExtensionPoint" name="Point d'extension de ce plugin"
    schema="schema/MyExtensionPoint.exsd"/>
  <extension
    point="org.eclipse.ui.views">
    <category
      id="com.opcoach.pde.demo.category1"
      name="Lancement">
    </category>
    <view
      category="com.opcoach.pde.demo.category1"
      class="com.opcoach.pde.demo.ViewPart1"
      id="com.opcoach.pde.demo.view1"
      name="Vue De Lancement"
      restorable="true">
    </view>
  </extension>
  <extension
    point="org.eclipse.ui.commands">
    <category
      id="com.opcoach.pde.demo.category1"
      name="Lancement">
    </category>
    <command
      category="com.opcoach.pde.demo.category1"
      id="com.opcoach.pde.demo.lancerLaSimu"
      name="Lancer la simulation">
    </command>
  </extension>
</plugin>
  
```

Overview Dependencies Runtime Extensions Extension Points Build MANIFEST.MF build.properties plugin.xml

Image 61 plugin.xml

Internationalisation d'un plugin

- Se fait à l'aide du wizard accessible dans l'onglet overview
- Ne concerne que l'internationalisation des noms dans le plugin.xml
- Ne concerne pas le code source (généré avec source->Externalize Strings)
- Indiquer chaque valeur à traduire avec un %key
- **Ne pas traduire les ID !!**

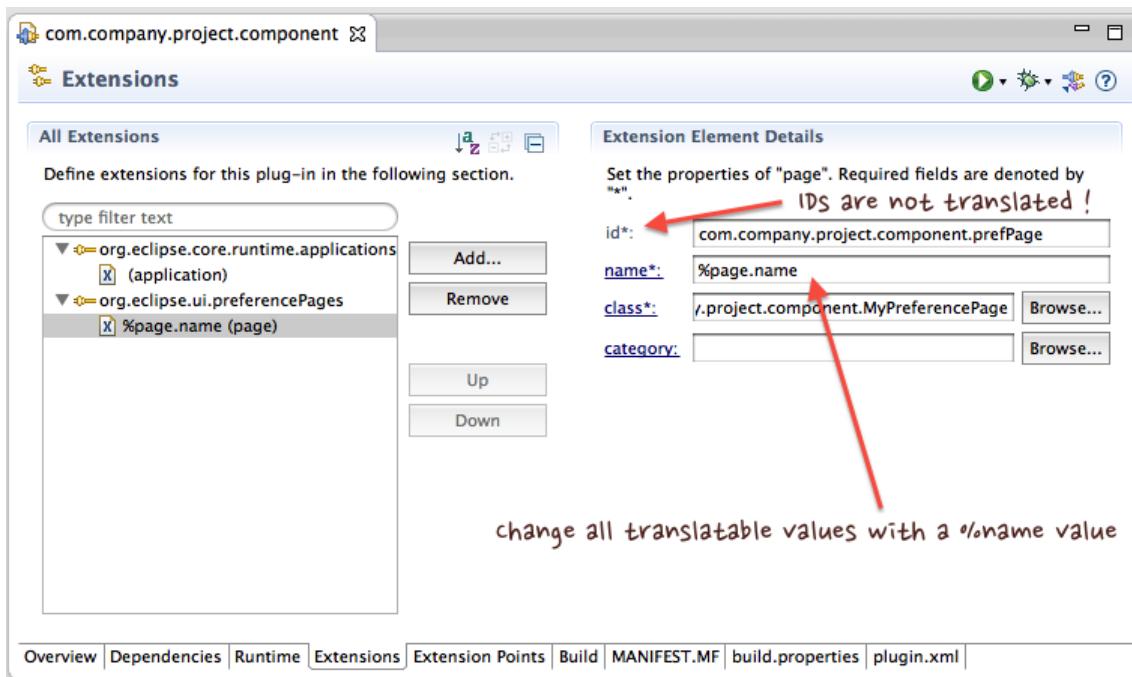


Image 62 Plugin translation

Le choix du fichier est fait par Eclipse selon la locale de lancement

- gérée par le paramètre -nl au lancement
- si pas de paramètre, locale par défaut.

Fichier properties de traduction

- Stocker les traductions dans :
 - OSGI-INF/l10n/bundle.properties : pour le fichier par défaut
 - OSGI-INF/l10n/bundle_en.properties : pour le fichier anglais
 - OSGI-INF/l10n/bundle_fr.properties : pour le fichier français
- Indiquer dans chaque fichier la valeur de key : key=texte
- On crée de préférence un fragment pour chaque langue

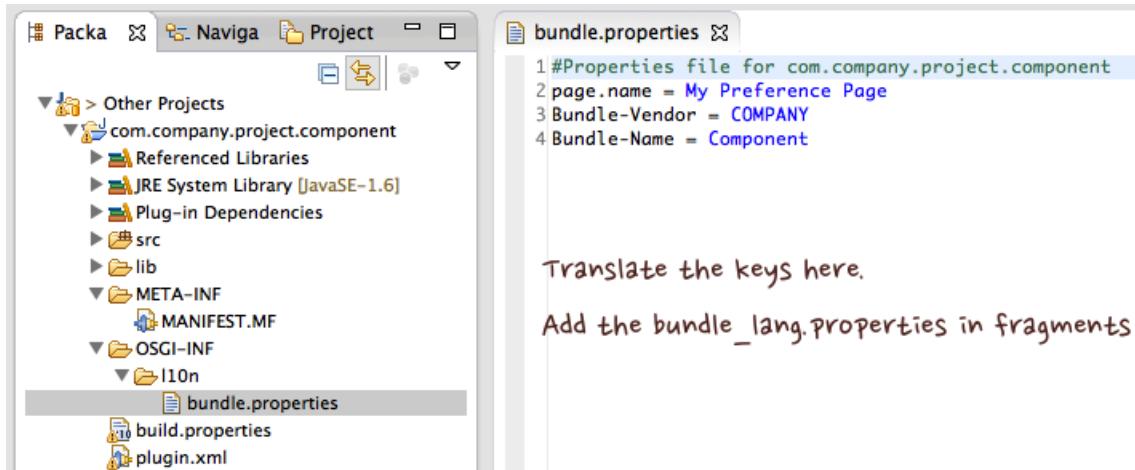


Image 63 bundle.properties file

Visualiser les dépendances entre plugins sous forme d'arbre

Le PDE propose une vue hiérarchique de base :

- Window->Show View-> Plugin Dependencies

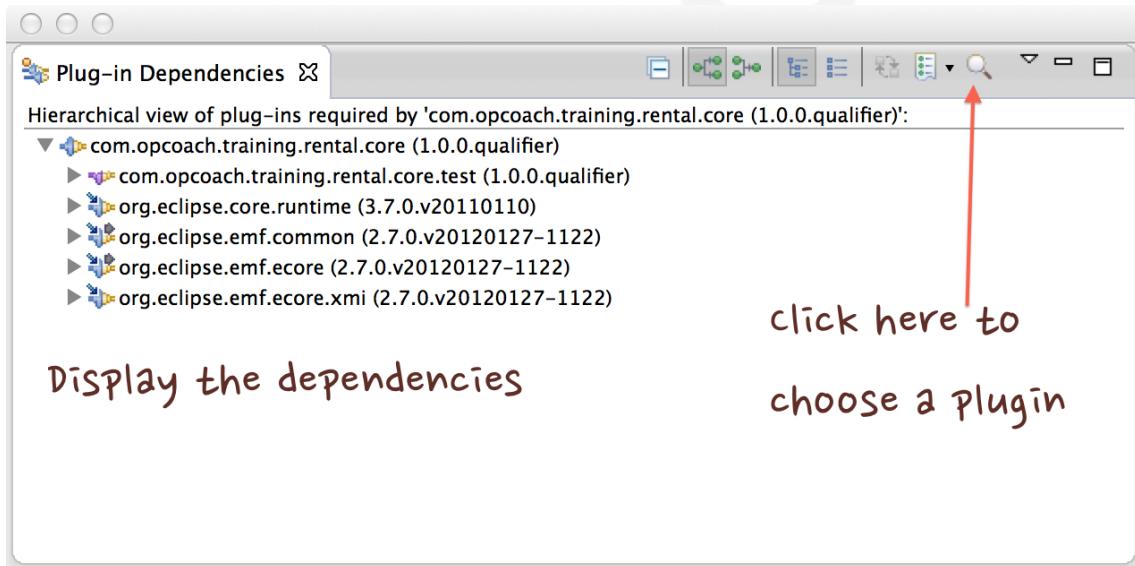


Image 64 Plugin tree dependencies

Visualiser les dépendances entre plugins sous forme graphique

Il faut installer le plugin de visualisation graphique à partir de l'update site :

- <http://download.eclipse.org/eclipse/pde/incubator/visualization/site>
- puis ouvrir la vue 'Graph Plugin Dependencies' :

Voir aussi l'article : <http://www.ibm.com/developerworks/library/os-eclipse-dependencyvisualization/>²³

23 - <http://www.ibm.com/developerworks/library/os-eclipse-dependencyvisualization/>

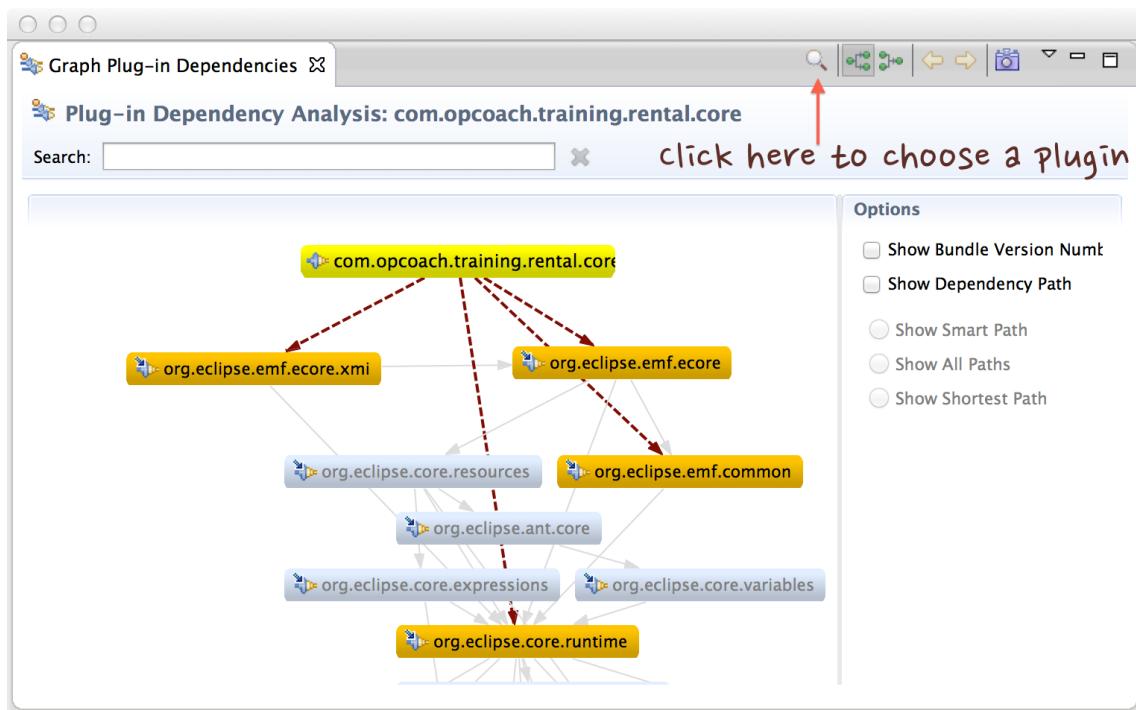


Image 65 Graphical dependencies

La création d'IHM dans Eclipse 4



SWT	65
Création de Part	74
Gestion des événements SWT	78
Window Builder	80
Injection et annotations	84
Contextes d'injection	93
API Annotation	101
JFace	104
JFace Ressources	112
API pour l'IHM	116

Objectifs

- Décrire les librairies Java/Eclipse dédiées aux IHM.
- Utiliser l'architecture RCP pour créer ses IHM

Eclipse fournit un ensemble de librairies permettant de développer des IHM de qualité.

- SWT : Standard Widget Toolkit
- JFace : apport du modèle MVC, Data Binding et lien avec Eclipse

Le développement des IHM Eclipse 4 est guidé par deux principes :

- ajout de composant d'ihm dans le modèle d'application
- utilisation des API d'IHM du workbench

A. SWT

Présentation

SWT est une librairie de widgets graphiques :

- efficace et performante
- simple
- reliée directement à l'OS :
 - (API + différentes implémentations)
 - restituant le look and feel natif

Pourquoi pas swing ?

- alternative peu professionnelle
- lent (à l'époque de la création de SWT)
- lourd à programmer

SWT Les implémentations

SWT fonctionne sur les os et window systems suivants :

- Windows XP, Windows Vista : win32, win64, wpf (.net)
- AIX, FreeBSD, Linux, HP-UX, Solaris: Motif et GTK+
- Mac OS X : Carbon et Cocoa (32 et 64 bits)
- QNX Photon
- Pocket PC
- RWT : implémentation pour les applications RAP (Remote Application Platform)

Un seul code compatible sur toutes les plateformes

Composition de SWT

SWT fournit 3 grandes composantes :

- des widgets graphiques
- des layouts
- un système d'événements graphiques.

Les widgets SWT

Tous les widgets swt sont décrits à cette adresse : <http://www.eclipse.org/swt/widgets/>²⁴

Chaque widget est décrit avec :

- présentation
- javadoc
- exemple de code

Widgets SWT (1)

24 - <http://www.eclipse.org/swt/widgets/>

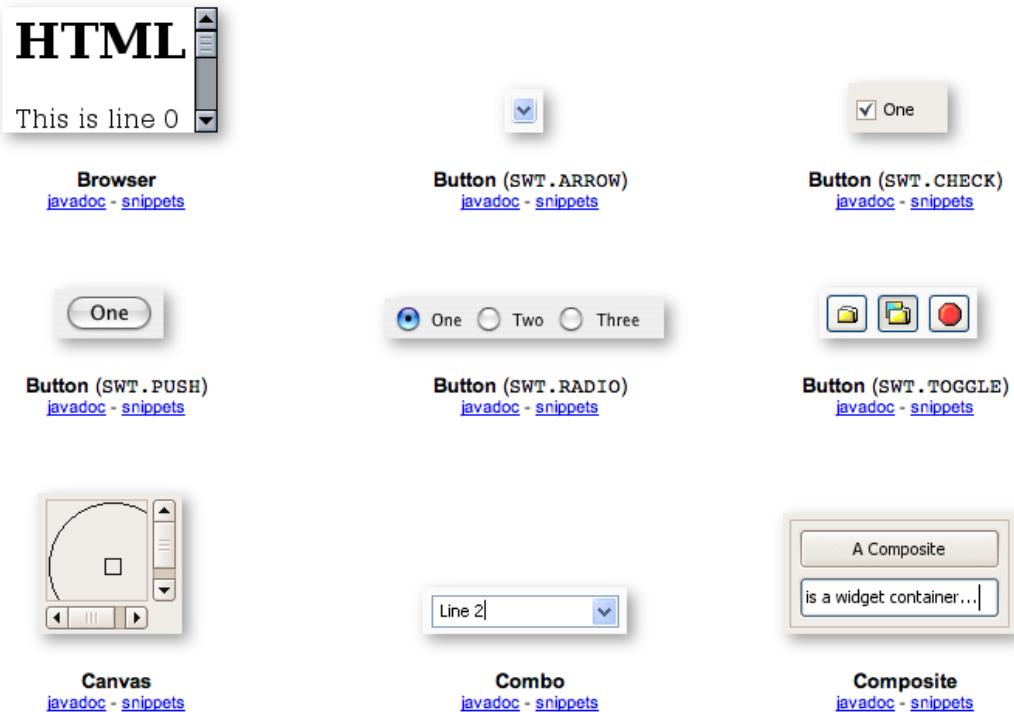


Image 66 SWT widgets 1

Widgets SWT (2)

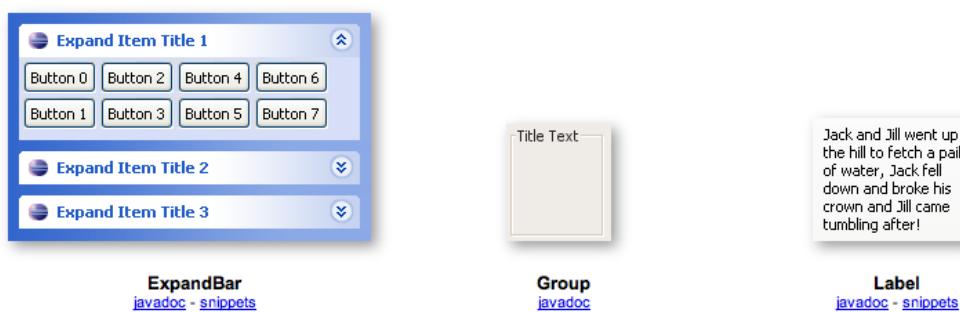
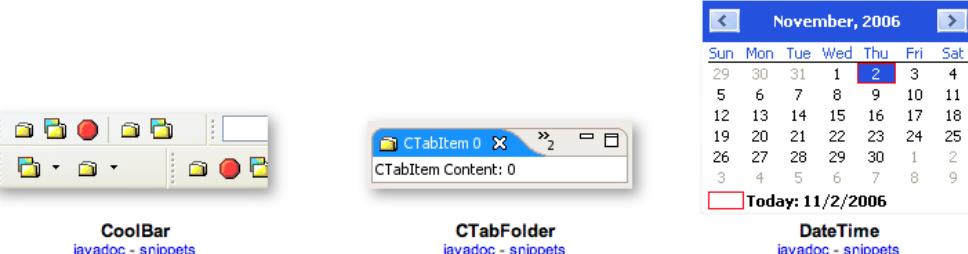


Image 67 SWT Widgets 2

Widgets SWT (3)

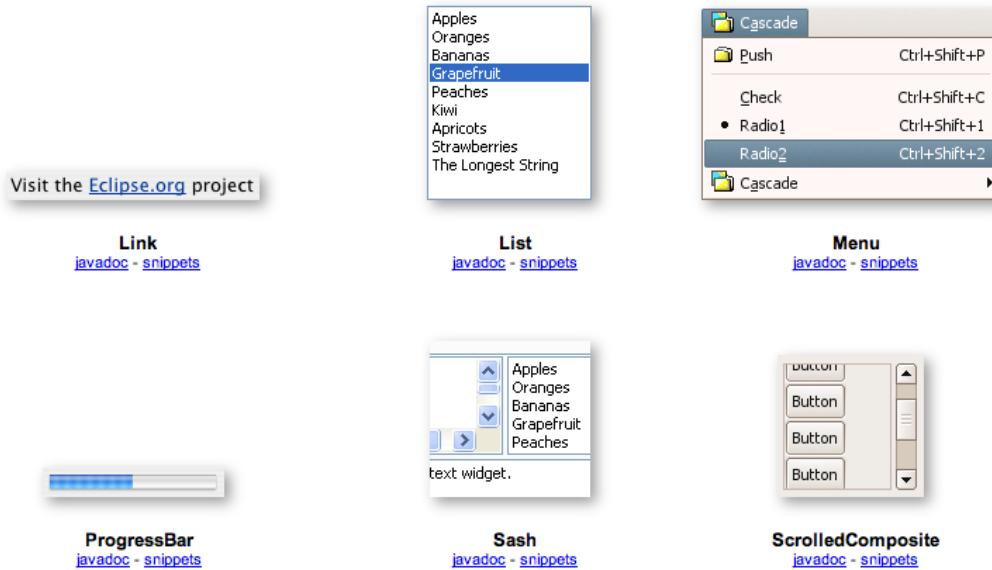


Image 68 SWT Widgets 3

Widgets SWT (4)

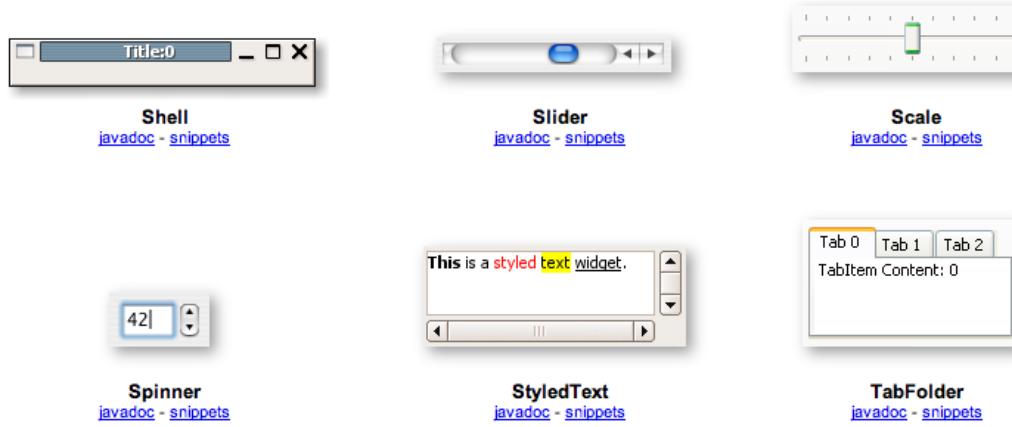


Image 69 SWT Widgets 4

Widgets SWT (5)

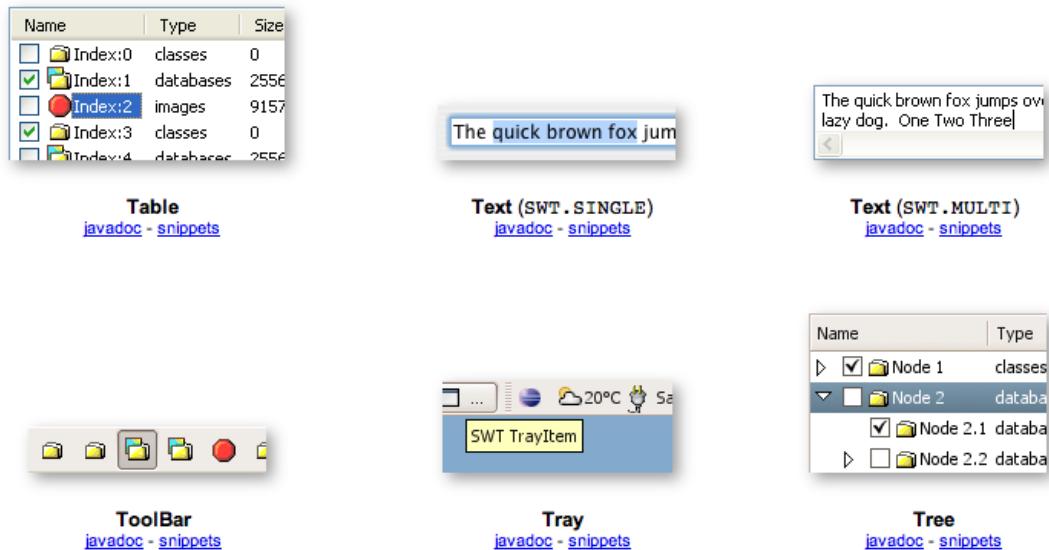


Image 70 SWT Widgets 5

Construction des widgets.

Tous les widgets s'utilisent de manière identiques :

- passage au constructeur du parent et du style
- impossibilité de changer le parent d'un widget
- interdit de dériver un widget sauf Composite et Canvas

Constructeur :

```
Label titre = new Label(parent, SWT.BORDER);
titre.setText("Nom : ");
```

Image 71 Label

Utilisation des styles :

- le style est une combinaison de valeurs binaires
- **SWT.BORDER | SWT.RADIO**
- Si le widget ne gère pas le style, il est ignoré
- Pour mettre aucun style, utiliser **SWT.NONE**
- La liste des styles par widget se trouve sur le wiki eclipse :
 http://wiki.eclipse.org/SWT_Widget_Style_Bits²⁵

25 - http://wiki.eclipse.org/SWT_Widget_Style_Bits

Exemple SWT

```

21@import org.eclipse.swt.SWT;
22 import org.eclipse.swt.graphics.Image;
23 import org.eclipse.swt.layout.GridLayout;
24 import org.eclipse.swt.widgets.Button;
25 import org.eclipse.swt.widgets.Display;
26 import org.eclipse.swt.widgets.Shell;
27
28 public class Snippet206
29 {
30
31@ public static void main(String[] args)
32 {
33     Display display = new Display();
34     Image image = display.getSystemImage(SWT.ICON_QUESTION);
35     Shell shell = new Shell(display);
36     shell.setLayout(new GridLayout());
37     Button button = new Button(shell, SWT.PUSH); ← Creation du bouton
38     button.setImage(image);
39     button.setText("Button");
40     shell.setSize(300, 300);
41     shell.open();
42     while (!shell.isDisposed())
43     {
44         if (!display.readAndDispatch())
45             display.sleep(); ← Boucle d'événements
46     }
47     display.dispose();
48 }
49 }
```



Création Display et Parent

Création du bouton

Boucle d'événements

Image 72 Snippet 206

Autres Exemples de résultat

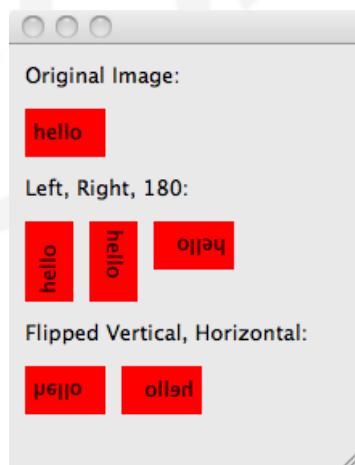


Image 73 Snippet 139

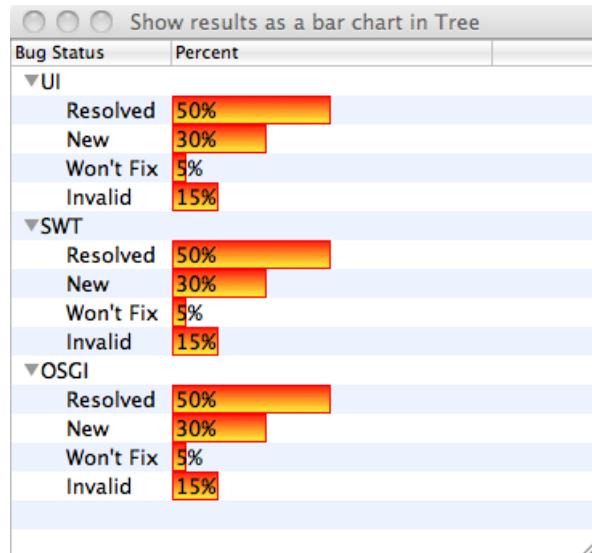


Image 74 Snippet 232

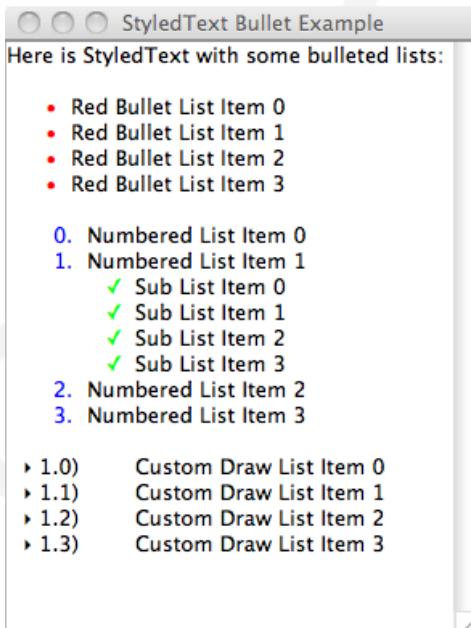


Image 75 Snippet 222



Complément : d'autres widgets évolutés

Le projet nebula (en incubation) dans Eclipse fournit aussi d'autres widgets évolutés :
<http://www.eclipse.org/nebula/>²⁶

26 - <http://www.eclipse.org/nebula/>

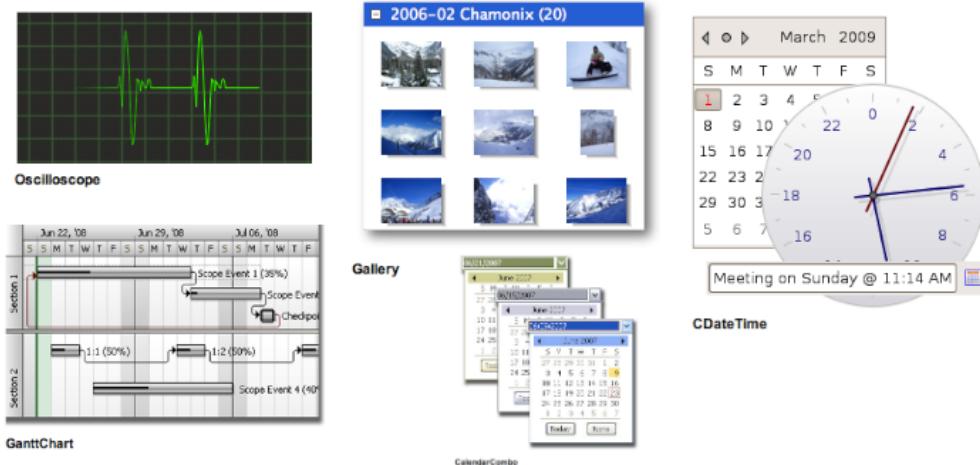


Image 76 Nebula widgets samples

Les Layouts dans SWT

Les layout permettent d'organiser les widgets sur la page.

Il existe différents algorithmes de placement

- **FillLayout** : widgets de taille identiques en ligne puis colonnes
- **RowLayout** : place les widgets en ligne(s) avec des options (spacing, wrap, fill)
- **GridLayout** : place les widgets dans un tableau
- **FormLayout** : place les widgets avec des attachements sur les cotés.

On utilise principalement le GridLayout en cascade...

Pour approfondir :

[http://www.eclipse.org/articles/article.php?file=Article-Understanding-Layouts/index.html²⁷](http://www.eclipse.org/articles/article.php?file=Article-Understanding-Layouts/index.html)

Affectation des layouts.

- Le layout se fixe sur un widget composite.
- On ne peut pas fixer un layout sur un widget terminal

```
Composite shell = new Shell();
// Set a layout of 2 columns with different sizes
shell.setLayout(new GridLayout(2, false));
```

Image 77 Layout

Paramétrage du layout

- Le layout se paramètre directement sur l'objet contenu
- On utilise une instance de `LayoutData` appropriée :
 - `RowData` -> `RowLayout`
 - `GridData` -> `GridLayout`
 - ...
- Il faut recréer une instance de `LayoutData` par objet à paramétrer.

27 - <http://www.eclipse.org/articles/article.php?file=Article-Understanding-Layouts/index.html>

```

Composite shell = new Shell();
// Set a layout of 2 columns with different sizes
shell.setLayout(new GridLayout());
// Create the button
Button button = new Button(shell, SWT.PUSH);
GridData gd = new GridData();
gd.horizontalAlignment = SWT.CENTER;
button.setLayoutData(gd);

```

Image 78 Grid Data



Attention: Gestion des ressources dans SWT

- Etant natives les ressources (Fontes, Images et Couleurs) nécessitent des précautions particulières.
- Une couleur peut être allouée dans la carte graphique par exemple.
- Il faut donc gérer l'allocation et la désallocation.
- Règle : dans la classe qui alloue, on trouve le code qui désalloue
- Pour désallouer une ressource on appelle la méthode 'dispose()'
- Le dispose() est appelé automatiquement pour les widgets.
- On obtient une exception : No more handles

JFace fournit des classes d'aide (Registries) pour gérer les ressources sans se soucier des problèmes de désallocation.

En cas de fuite on peut utiliser SLeak

(<http://eclipsesource.com/blogs/2009/04/17/finding-swt-leaks-with-sleak/>²⁸)



Attention: Thread et IHM

- Un thread ne peut pas modifier des objets de l'IHM directement.
- Il doit déléguer ce traitement dans un Runnable qui sera traité par le thread d'IHM.
- Si on le gère mal on obtient une : org.eclipse.swt.SWTException: Invalid thread access

Méthode :

```

Display.getDefault().asyncExec(new Runnable()
{
    public void run()
    {
        label.setText(text);
    }
});

```

Image 79 Display.asyncExec

Gestion du trackPad

- A partir de la version 3.7 (juin 2011), il est possible de gérer le trackpad
- Fonctionne sur mac et windows (certains pc). Pas sur Linux.
- Gère l'état du Touch (pour savoir où on met les doigts)
- Gère le mouvement (Gesture), pour savoir si on agrandit, déplace, etc...
- Voir la Snippet de code 353 dans SWT.

28 - <http://eclipsesource.com/blogs/2009/04/17/finding-swt-leaks-with-sleak/>

B. Crédit de Part

Définition des vues

- Dans le modèle d'application une vue est un Part qui se définit dans un POJO annoté
- La vue ne dérive d'aucune classe d'Eclipse
- On référence la classe du part dans le modèle d'application à l'endroit voulu :
 - sous windows/TrimmedWindow/Controls/....
 - créer les perspectives ou directement les parts

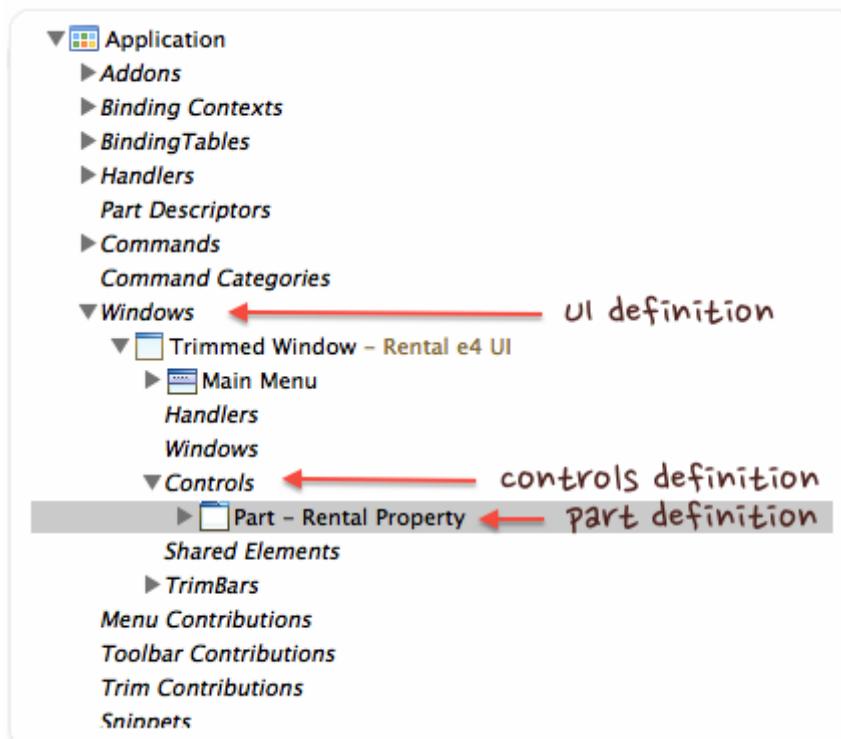


Image 80 Part in model

Puis on paramètre le Part en indiquant la classe contenant les annotations de Part

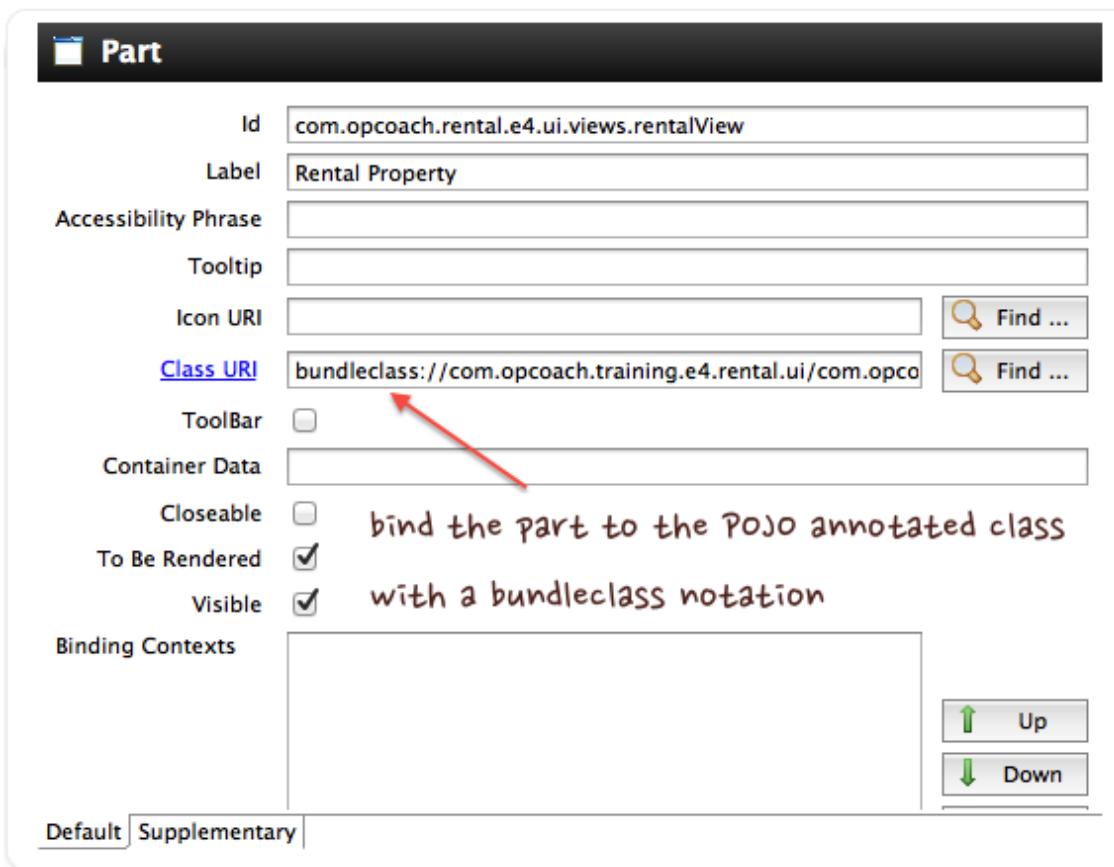


Image 81 part parameters

Classe du Part

- La classe décrivant le part est un simple POJO contenant des annotations
- Le contexte contient le composite parent où créer le Part
- On crée les widgets dans une méthode `createPartControl` annotée avec `@PostConstruct`
- Cette méthode sera appelée lors de la construction de la vue
- C'est à cet endroit que doivent être instanciés les widgets SWT composant la vue

```

29 public class RentalPropertyView
30 {
31     public static final String VIEW_ID = "com.opcoach.rental.e4.ui.views.rentalView"; //NON-NLS-1$
32
33     private Label rentedObjectLabel, customerNameLabel, startDateLabel, endDateLabel;
34
35     @PostConstruct
36     public void createContent(Composite parent, RentalAgency agency)
37     {
38         parent.setLayout(new GridLayout(1, false));           This Pojo is created by the UI
39
40         Group infoGroup = new Group(parent, SWT.NONE);      renderer, and this method is called
41         infoGroup.setText("Information");                  with injected parameters
42         infoGroup.setLayout(new GridLayout(2, false));
43
44         rentedObjectLabel = new Label(infoGroup, SWT.BORDER);
45         GridData gd = new GridData();
46         gd.horizontalSpan = 2;
47         gd.horizontalAlignment = SWT.FILL;
48         rentedObjectLabel.setLayoutData(gd);
49
    
```

Image 82 view code sample

Gestion du Focus

Il faut toujours prévoir une méthode avec `@Focus`

Attention il n'y a pas d'erreur indiquée si elle manque

```

145     @Focus
146     private void setFocus()
147     {
148         rentedObjectLabel.setFocus();
149     }
    
```

Focus

Utilisation du wizard de création de part

Eclipse 4 tooling propose un wizard pour créer la classe du part :

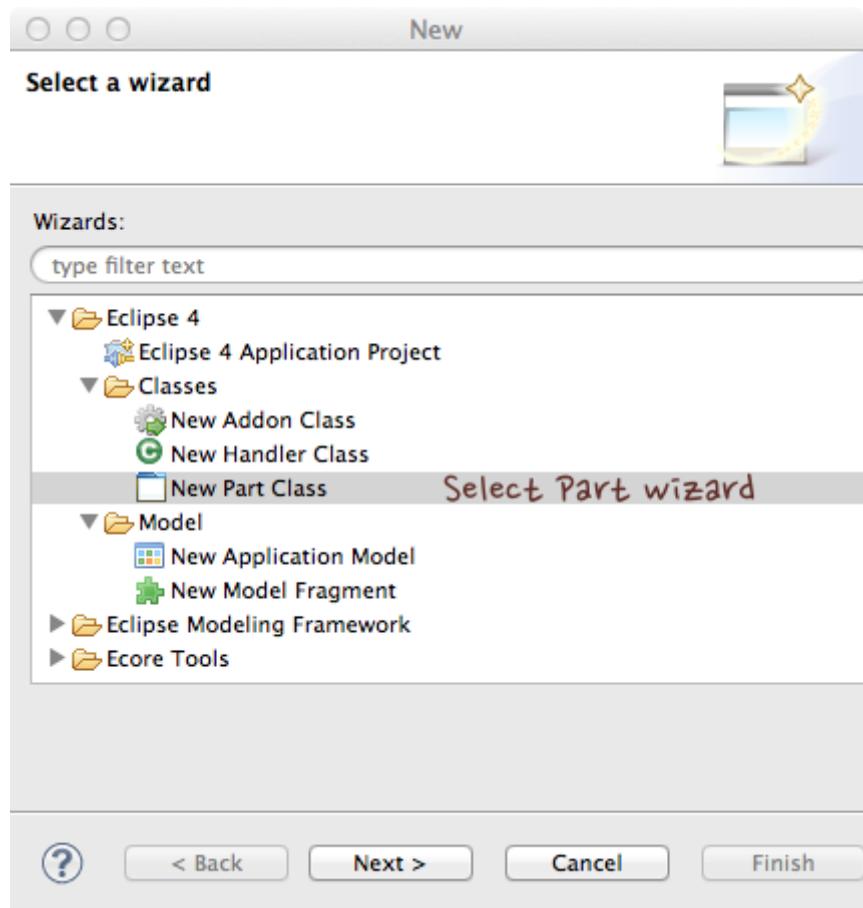


Image 83 New Part wizard

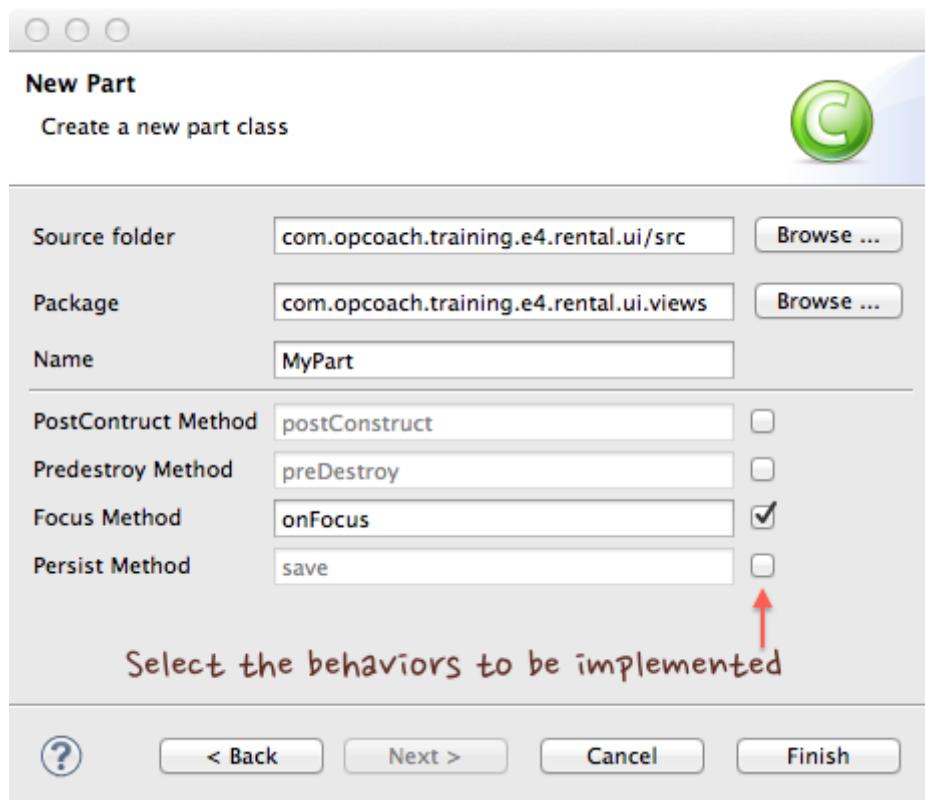


Image 84 New Part Wizard

Exercice EAP 030

Vue de propriété Rental

C. Gestion des événements SWT

Introduction

Le mécanisme d'événements permet de réagir

- à toute interaction de l'utilisateur
- à tout autre événement identifié
 - reseau
 - timer
 - ...

Le mécanisme met en oeuvre :

- des événements
- des listeners
- des adapters
- des classes cibles

Evenements

L'événement est une classe qui reflète les données associées

Exemple :

- MouseEvent
- KeyEvent

Tout événement référence la source

Listeners

Le listener décrit les comportements associés aux événements

- Que fait on si on presse une touche ?
- Que fait on si on reçoit un objet graphique ?
- Que fait on si les propriétés du logiciel changent ?

Le listener reçoit toujours l'événement associé en paramètre

Le listener s'enregistre sur l'objet à écouter

Le listener est implémenté par l'objet qui réagit

Le MouseListener

```
public interface MouseListener extends EventListener
{
    /** Invoked when the mouse button has been clicked (pressed and released) */
    public void mouseClicked(MouseEvent e);

    /** Invoked when a mouse button has been pressed on a component. */
    public void mousePressed(MouseEvent e);

    /** Invoked when a mouse button has been released on a component. */
    public void mouseReleased(MouseEvent e);

    /** Invoked when the mouse enters a component. */
    public void mouseEntered(MouseEvent e);

    /** Invoked when the mouse exits a component. */
    public void mouseExited(MouseEvent e);
}
```

Image 85 MouseListener

Adapters

- L'adapter fournit une implémentation vide d'un listener
- Il permet de n'implémenter que la méthode voulue

```
public class MyMouseAdapter extends MouseAdapter
{
    public void mouseClicked(MouseEvent e)
    {
        // To do when clicked
    }
}
```

Image 86 Adapter

Mise en oeuvre

```

30 // The listener to manage event
31 public interface XXListener
32 {
33     public void doSomething(XXEvent event);
34 }
35
36 // The source class that creates and sends events
37 public class Source
38 {
39     void addXXListener(XXListener x) { }
40 }
41
42 // The consumer class (reacts on events)
43 public class MyClass implements XXListener
44 {
45     Source source = new Source();
46
47 MyClass()
48 {
49     source.addXXListener(this);
50 }
51
52 public void doSomething(XXEvent event)
53 {
54     System.out.println("Do something with event");
55 }
56
57 }
```

Image 87 Listener design pattern



Attention : Fuites avec les listeners

Attention aux fuites lors de l'utilisation des listeners.

Toujours appeler le remove quand l'objet n'est plus utilisé !

D. Window Builder

Présentation

- Window Builder est un nouvel éditeur WYSIWYG, open source, donné par Google.
- Il permet d'éditer de manière bidirectionnelle (code/visu) des IHM :
 - SWING
 - SWT
 - GWT
- Il permet aussi de générer du code de data binding

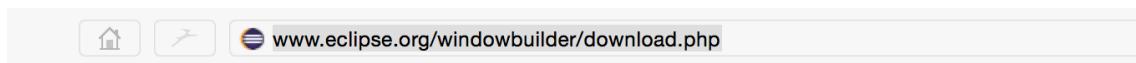
Installation Window Builder

Il faut utiliser l'update site correspondant à la version Eclipse courante.

Ces update sites sont rassemblés sur cette page :

[http://www.eclipse.org/windowbuilder/download.php²⁹](http://www.eclipse.org/windowbuilder/download.php)

29 - <http://www.eclipse.org/windowbuilder/download.php>



Update Sites

Eclipse Version	Release Version		Integration Version	
	Update Site	Zipped Update Site	Update Site	Zipped Update Site
4.5 (Mars)	link	link (MD5 Hash)	link	link (MD5 Hash)
4.4 (Luna)	link	link (MD5 Hash)	link	link (MD5 Hash)
4.3 (Kepler)	link	link (MD5 Hash)		
4.2 (Juno)	link	link (MD5 Hash)		
3.8 (Juno)	link	link (MD5 Hash)		

Image 88 Window builder download

Wizards

Window builder fournit de nombreux wizards :

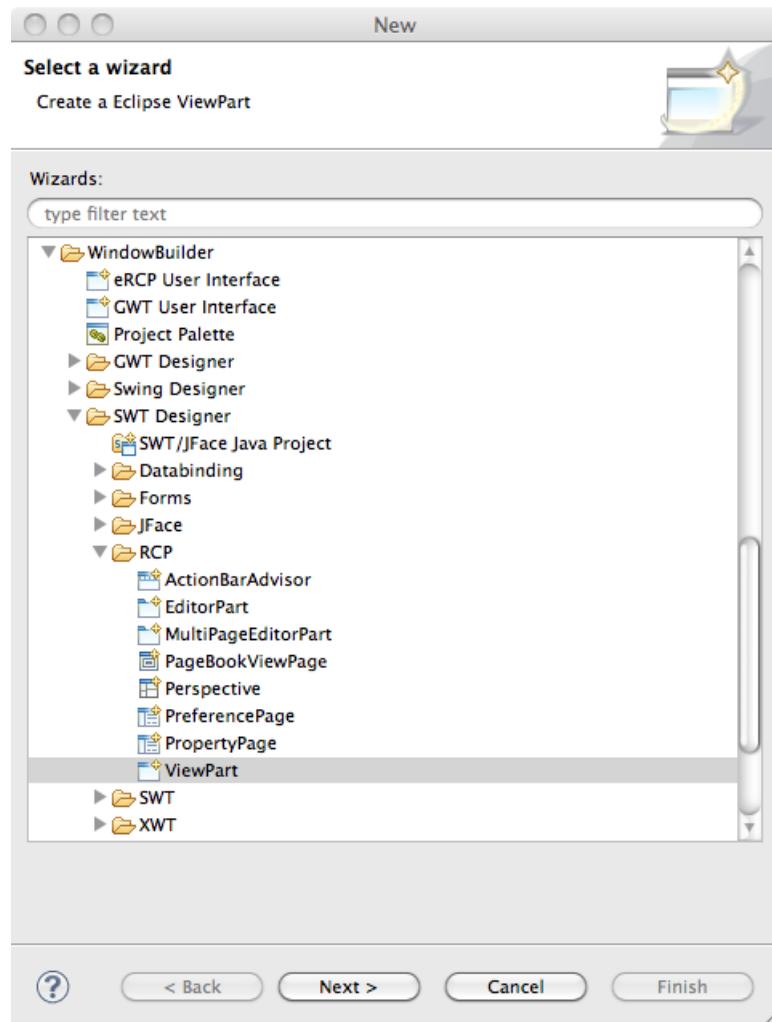


Image 89 Window Builder wizards

wizards RCP ou E4

- Pour Eclipse 3.X : génère une classe et les extensions associées
- Pour Eclipse 4 : génère une classe compatible avec Eclipse 4

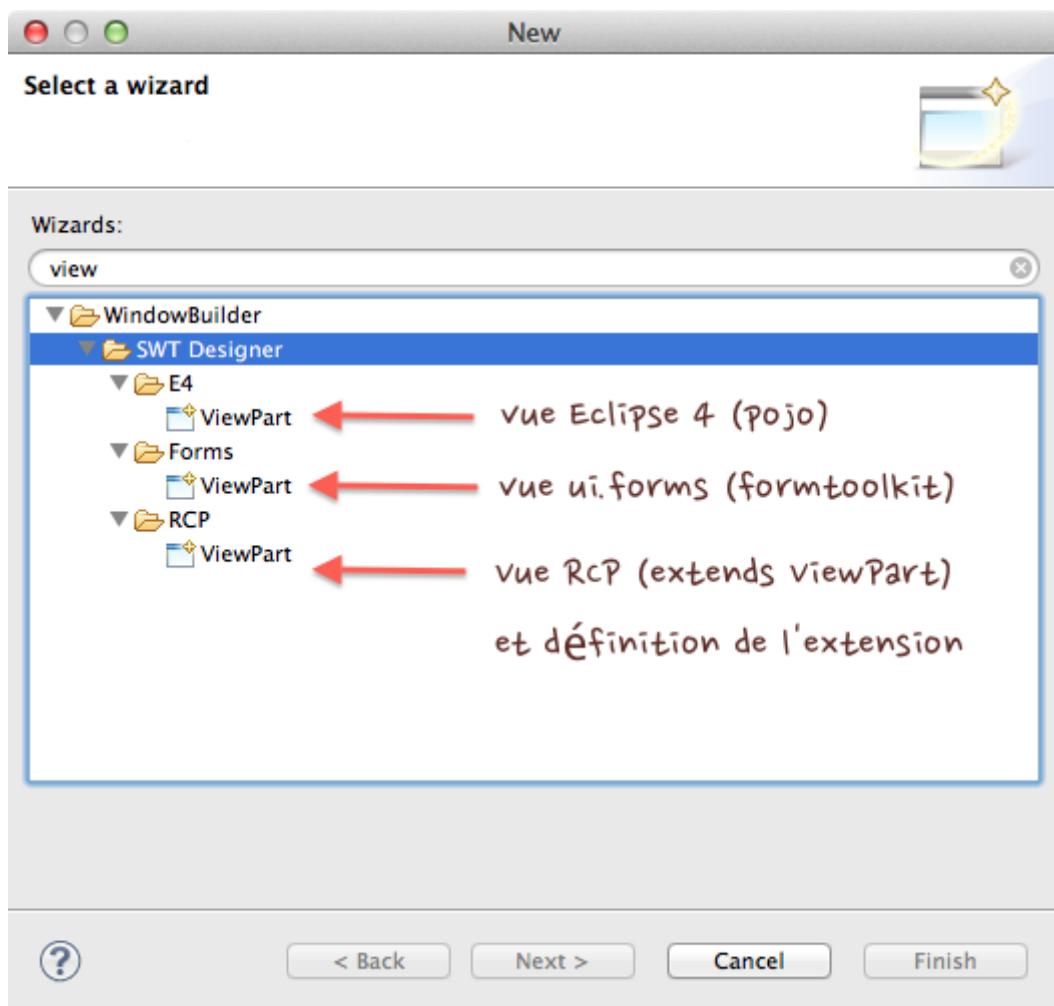


Image 90 View Wizards

Editeur

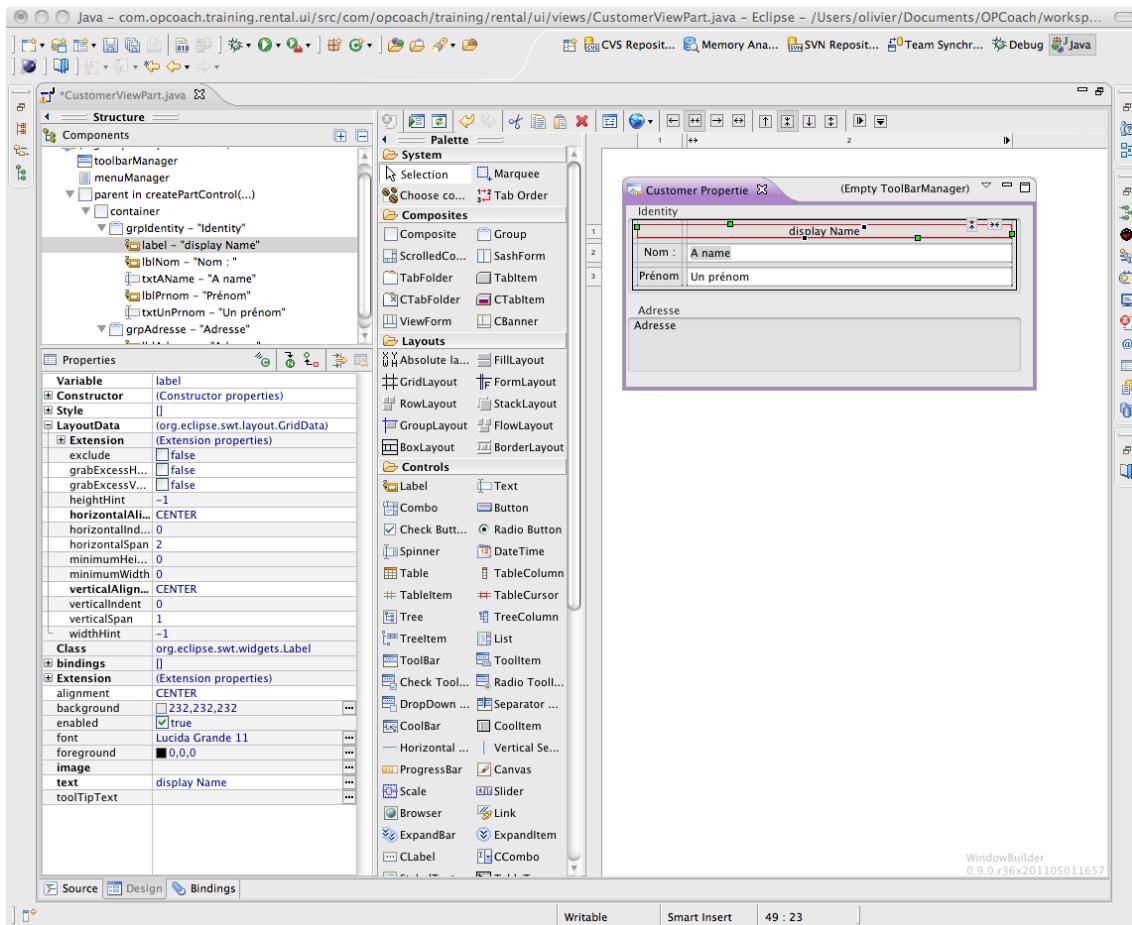


Image 91 Window Builder Editor

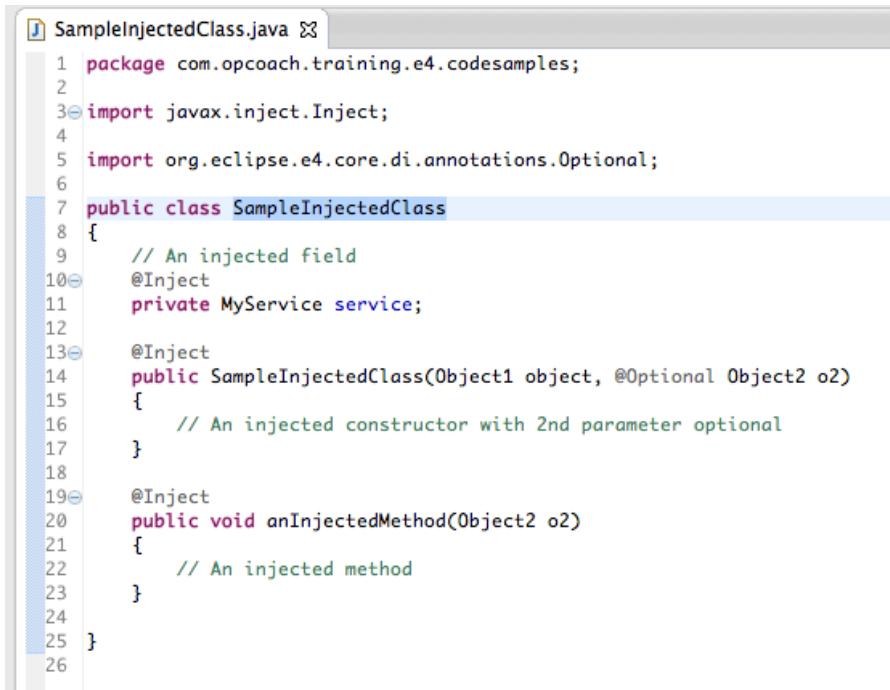
Exercice

Rééditer la vue Rental Property avec window builder

E. Injection et annotations

Introduction / Principe

- L'injection Eclipse 4 consiste à déléguer l'initialisation
 - de certaines instances d'objet
 - de certains champs d'un objet
 - de certains paramètres de méthodes ou de constructeurs.
- L'injection fonctionne en utilisant un contexte qui contient les valeurs injectables
- On utilise l'annotation **@Inject** (`javax.inject`), de nature Runtime, pour injecter les valeurs
- Elle peut s'appliquer sur un constructeur, une méthode ou un champ
- Le framework d'injection introspecte les classes pour retrouver les annotations **@Inject**



```

1 package com.opcoach.training.e4.codesamples;
2
3 import javax.inject.Inject;
4
5 import org.eclipse.e4.core.di.annotations.Optional;
6
7 public class SampleInjectedClass
8 {
9     // An injected field
10    @Inject
11    private MyService service;
12
13    @Inject
14    public SampleInjectedClass(Object1 object, @Optional Object2 o2)
15    {
16        // An injected constructor with 2nd parameter optional
17    }
18
19    @Inject
20    public void anInjectedMethod(Object2 o2)
21    {
22        // An injected method
23    }
24
25 }
26

```

Image 92 Sample injected class

La classe ContextInjectionFactory

- Une classe possédant des annotations d'injection :
- doit être instanciée par un appel à la **ContextInjectionFactory**
- **ne doit plus être instanciée avec un appel à new**
- **ContextInjectionFactory** instancie les objets en injectant le contexte.
- C'est une classe entièrement statique

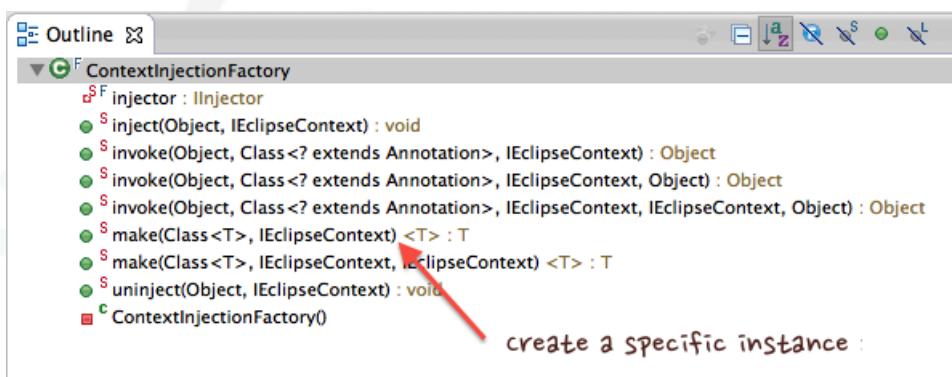


Image 93 ContextInjectionFactory

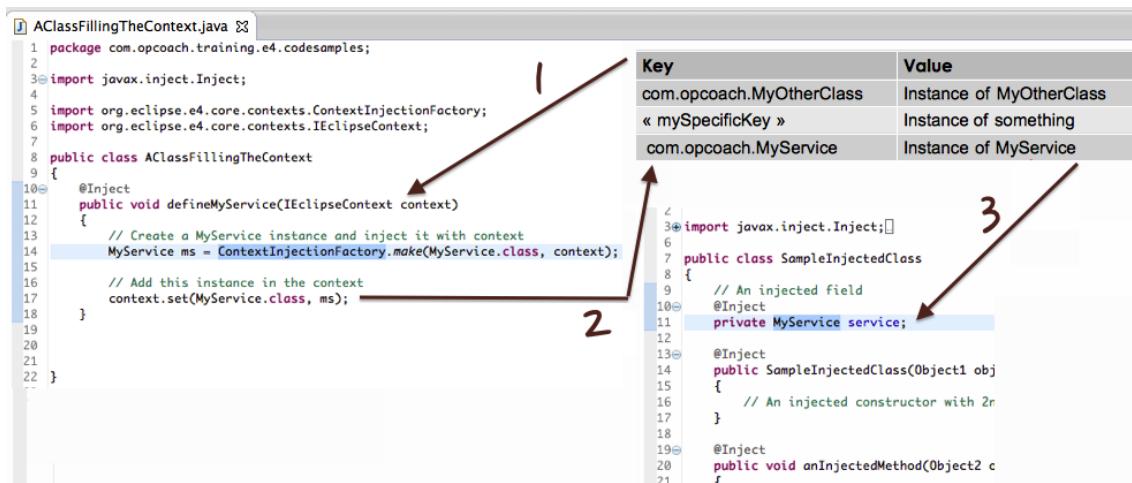
Le contexte de l'injecteur

- Le contexte permet de stocker les valeurs associées aux clés ou aux classes
- **Le contexte est hiérarchique**
- Les branches du contexte sont activées par le framework selon le moment (focus de l'UI, initialisations...)

- L'activation d'une branche donne accès aux valeurs disponibles dans la branche active
- Le contexte peut avoir accès aux valeurs de son parent (`getParent()`)
- Le contexte est fourni par une classe implémentant l'interface `IEclipseContext`

Exemple d'utilisation

Exemple de mise en oeuvre du contexte (dans la branche en cours) :



Key	Value
com.opcoach.MyOtherClass	Instance of MyOtherClass
« mySpecificKey »	Instance of something
com.opcoach.MyService	Instance of MyService

```

1 package com.opcoach.training.e4.codesamples;
2
3 import javax.inject.Inject;
4
5 import org.eclipse.e4.core.contexts.ContextInjectionFactory;
6 import org.eclipse.e4.core.contexts.IEclipseContext;
7
8 public class AClassFillingTheContext
9 {
10     @Inject
11     public void defineMyService(IEclipseContext context)
12     {
13         // Create a MyService instance and inject it with context
14         MyService ms = ContextInjectionFactory.make(MyService.class, context);
15
16         // Add this instance in the context
17         context.set(MyService.class, ms);
18     }
19
20
21 }
22 
```

```

3 import javax.inject.Inject;
4
5 public class SampleInjectedClass
6 {
7     // An injected field
8     @Inject
9     private MyService service;
10
11     // An injected constructor with 2r
12     @Inject
13     public SampleInjectedClass(Object obj
14     {
15         // An injected constructor with 2r
16     }
17
18     @Inject
19     public void anInjectedMethod(Object2 c
20     {
21     }
22 } 
```

Image 94 Injection Sample

Les annotations complémentaires de `@Inject`

Trois annotations complémentaires sont utilisées pour gérer l'initialisation :

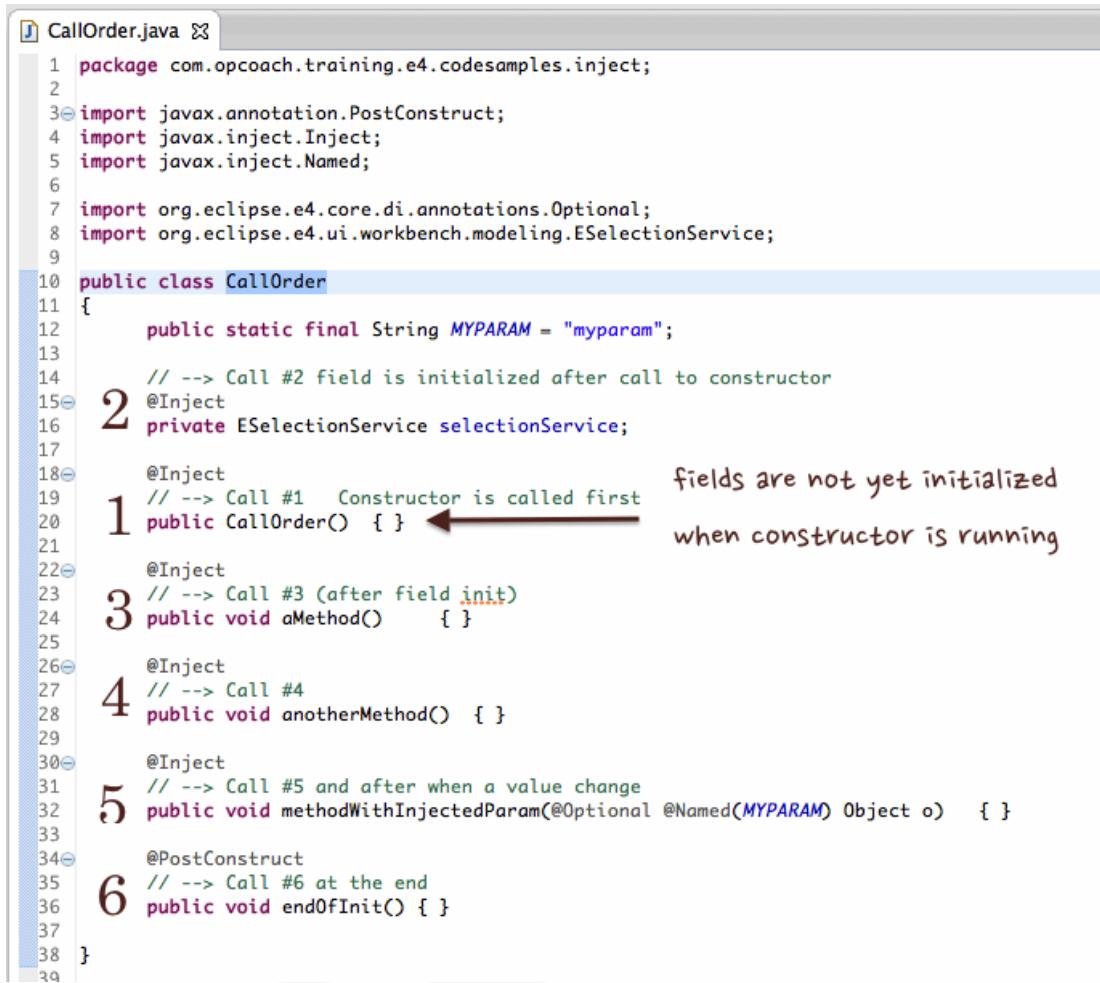
- **`@PostConstruct`** : permet d'annoter une méthode qui sera invoquée en fin de construction d'objet
- **`@Optional`** : annotation permettant d'indiquer qu'un champ, une méthode ou un paramètre sont optionnels (-> peut être null)
- **`@Named`** : permet d'accéder à un objet par son nom plutôt que par sa classe

Ordre d'appel

L'injection initialise une instance dans l'ordre suivant :

- Invocation d'un constructeur avec `@Inject`
- Initialisation des champs avec `@Inject`
- Invocation des méthodes avec `@Inject`
- Invocation de la méthode avec `@PostConstruct`

Exemple simple d'ordre d'appel :



```

1 package com.opcoach.training.e4.codesamples.inject;
2
3 import javax.annotation.PostConstruct;
4 import javax.inject.Inject;
5 import javax.inject.Named;
6
7 import org.eclipse.e4.core.di.annotations.Optional;
8 import org.eclipse.e4.ui.workbench.modeling.ESelectionService;
9
10 public class CallOrder {
11 {
12     public static final String MYPARAM = "myparam";
13
14     // --> Call #2 field is initialized after call to constructor
15     @Inject
16     private ESelectionService selectionService;
17
18     @Inject
19     // --> Call #1 Constructor is called first
20     public CallOrder() { } ←
21
22     @Inject
23     // --> Call #3 (after field init)
24     public void aMethod() { }
25
26     @Inject
27     // --> Call #4
28     public void anotherMethod() { }
29
30     @Inject
31     // --> Call #5 and after when a value change
32     public void methodWithInjectedParam(@Optional @Named(MYPARAM) Object o) { }
33
34     @PostConstruct
35     // --> Call #6 at the end
36     public void endOfInit() { }
37
38 }
39

```

Image 95 Call order

Invocation du constructeur (1)

- L'injecteur choisit le constructeur qui a une annotation `@Inject`
- Si plusieurs constructeurs sont annotés, il invoque celui qui a le maximum de paramètres injectables.
- Les champs annotés avec `@Inject` ont une valeur nulle dans le constructeur.
- Les autres champs peuvent être manipulés normalement
- Les constructeurs parents sont appelés soit par le `super()` implicite, soit par des `super(...)` explicites

Injection des champs (2)

- Chaque champs de la classe, annoté avec `@Inject` est initialisé, une fois le constructeur appelé.
- L'ordre d'initialisation est indéterminé
- Si un champs ne peut être injecté l'injecteur génère une exception
- Si la valeur change après l'injection, elle sera réinjectée automatiquement
- Si un champs peut ne pas avoir de valeur, on l'anneote avec `@Optional`

- Les champs hérités avec une annotation **@Inject** sont initialisés avant les autres

Invocation des méthodes @Inject (3)

- Les méthodes annotées avec **@Inject** sont invoquées:
 - après l'appel du constructeur et des champs injectés
 - **si un des paramètres injectés a changé de valeur**
- Si plusieurs méthodes sont annotées avec **@Inject**, l'ordre d'invocation est indéterminé
- Les méthodes héritées avec une annotation **@Inject** sont invoquées avant les autres
- Les méthodes héritées et surchargées, ayant une annotation **@Inject** ne sont pas invoquées
- Si la méthode est annotée avec **@Optional**, elle n'est pas invoquée si un des paramètres n'est pas injectable

Invocation de la méthode @PostConstruct (4)

- La méthode annotée **@PostConstruct** est invoquée:
 - après l'invocation des toutes les méthodes annotées avec **@Inject**
- La méthode héritée avec une annotation **@PostConstruct** est appelée avant les autres
- La méthode héritée et surchargée, ayant une annotation **@PostConstruct** n'est pas appelée
- Une méthode annotée avec **@PostConstruct** n'est jamais rappelée
- Il ne doit y avoir qu'une seule méthode annotée avec **@PostConstruct** par classe.

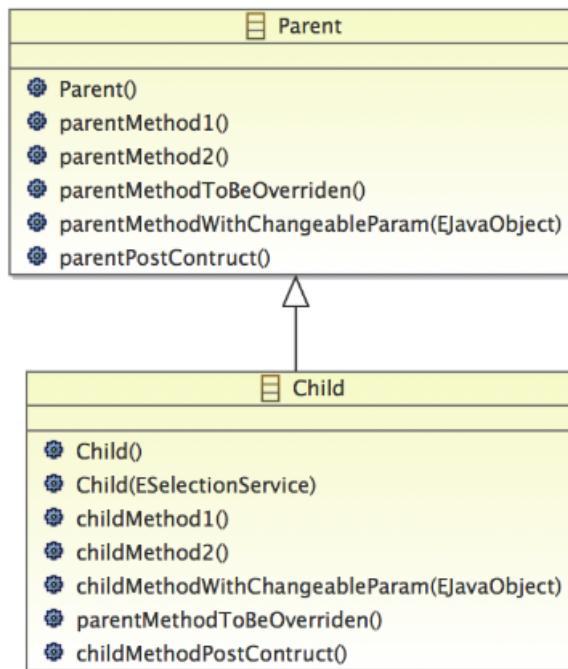
Fondamental : Règle de base de l'injection

Si une valeur précédemment injectée change de valeur dans le contexte :

- elle est automatiquement réinjectée dans les champs concernés
- les méthodes ayant reçu cette valeur en paramètre sont automatiquement ré-invoquées.

Exemple d'invocation de méthodes avec héritage

Soit le schéma suivant :



All methods and constructors have a @Inject
and *Postconstruct?() have a @Postconstruct

Parent and Child

Invocations effectuées lors d'une instanciation de child

```

public class Parent
{
    public static final String PARAM_IN_PARENT="parentParam";
    1 public Parent() {}

    @Inject public Parent(Adapter adapter) {}
    @Inject public Parent(EMenuService service) {}

    4 @Inject public void parentMethod1() {}
    5 @Inject public void parentMethod2() {}
    No Call@.Inject public void parentMethodToBeOverridden() {}
    6 @Inject public void parentMethodWithChangeableParam(@Optional

    11 @PostConstruct public void parentMethodPostContruct() {}

}

public class Child extends Parent
{
    public static final String PARAM_IN_CHILD ="myparam";
    3 @Inject private ESelectionService selectionService;

    public Child() {}
    2 @Inject public Child(EMenuService ss) {}

    7 @Inject public void childMethod1() {}
    8 @Inject public void childMethod2() {}
    9 @Inject public void childMethodWithChangeableParam(@Optional @N

    10 @Override @Inject public void parentMethodToBeOverridden() {}

    12 @PostConstruct public void childMethodPostContruct() {}

}

```

System.out.println("-----");
System.out.println("-- Step 1 : create the ChildCallOrder ---");
System.out.println("-----");
ContextInjectionFactory.make(Child.class, ctx);

-- Step 1 : create the ChildCallOrder ---
-----> Parent Call empty Constructor
--> Child Call : constructor (EMenuService)
----> Parent Call parentMethod1
----> Parent Call parentMethod2
----> Parent Call parentMethodWithChangeableParam with param : null
--> Child Call : childMethod1
--> Child Call : childMethod2
--> Child Call : childMethodWithChangeableParamnull
--> Child Call : parentMethodToBeOverridden
----> Parent Call : parentMethodPostContruct
--> Child Call : childMethodPostContruct

Call to make a child

Invocation effectuée lors de la modification du paramètre géré dans le parent

```

public class Parent
{
    public static final String PARAM_IN_PARENT="parentParam";
    public Parent() {}

    @Inject public Parent(Adapter adapter) {}
    @Inject public Parent(EMenuService service) {}

    @Inject public void parentMethod1() {}
    @Inject public void parentMethod2() {}
    @Inject public void parentMethodToBeOverridden() {}
    Call @Inject public void parentMethodWithChangeableParam(@Optional @N
    @PostConstruct public void parentMethodPostContract() {}

}

public class Child extends Parent
{
    public static final String PARAM_IN_CHILD ="myparam";
    @Inject private ESelectionService selectionService;

    public Child() {}
    @Inject public Child(EMenuService ss) {}

    @Inject public void childMethod1() {}
    @Inject public void childMethod2() {}
    @Inject public void childMethodWithChangeableParam(@Optional @N
    @Override @Inject public void parentMethodToBeOverridden() {}
    @PostConstruct public void childMethodPostContract () {}

}

```

System.out.println("-----
----- Step 2 : change the parameter used by parent -----

----- ctx.set(Parent.PARAM_IN_PARENT, "new value for parent parameter");

----- Step 2 : change the parameter used by parent -----
-----> Parent Call parentMethodWithChangeableParam with param :
new value for parent parameter

Image 96 Change parameters in parent

Invocation effectué lors de la modification du paramètre géré dans le fils

```

public class Parent
{
    public static final String PARAM_IN_PARENT="parentParam";
    public Parent() {}

    @Inject public Parent(Adapter adapter) {}
    @Inject public Parent(EMenuService service) {}

    @Inject public void parentMethod1() {}
    @Inject public void parentMethod2() {}
    @Inject public void parentMethodToBeOverridden() {}
    @Inject public void parentMethodWithChangeableParam(@Optional @N
    @PostConstruct public void parentMethodPostContract() {}

}

public class Child extends Parent
{
    public static final String PARAM_IN_CHILD ="myparam";
    @Inject private ESelectionService selectionService;

    public Child() {}
    @Inject public Child(EMenuService ss) {}

    @Inject public void childMethod1() {}
    @Inject public void childMethod2() {}
    Call @Inject public void childMethodWithChangeableParam(@Optional @N
    @Override @Inject public void parentMethodToBeOverridden() {}
    @PostConstruct public void childMethodPostContract () {}

}

```

System.out.println("-----
----- Step 3 : change the parameter used by child -----

----- ctx.set(Child.PARAM_IN_CHILD, "new value for child parameter");

----- Step 3 : change the parameter used by child -----
-----> Child Call : childMethodWithChangeableParam
new value for child parameter

Image 97 Change parameters in child

Les autres annotations

Les annotations normalisées par la JSR 330 :

- **@Named** : pour injecter un objet par son nom
- **@Singleton** : pour annoter une classe qui ne doit être instanciée qu'une fois
- **Provider<T>** : interface permettant de déléguer la construction d'une interface T à une classe qui créera l'implémentation voulue (via sa méthode get).

Les annotations E4 spécifiques pour l'injection

- **@Creatable** : permet d'annoter une classe pour provoquer son instantiation si elle est absente du contexte.
- **@PreDestroy** : permet d'annoter une méthode appelée avant la destruction de l'objet

Les annotations E4 qui calculent des valeurs :

- **@Preference** : permet d'injecter la valeur stockée dans les préférences
- **@UIEventTopic** : permet de recevoir un événement de l'Event Broker
- **@Active** : permet de récupérer une valeur dans la branche active (non utilisé)

Exemple avec @Named

```

116  --> @Inject
117  public void setSelection(@Optional @Named(IServiceConstants.ACTIVE_SELECTION) Object o,
118      Adapter adapter)
119  {
120      Rental r = adapter.adapt(o, Rental.class);
121      setRental(r);
122  }
123
124

```

Image 98 @Inject @Named @Optional

Objets nommés injectables

La classe **IServiceConstants** définit les objets nommés récupérables avec **@Named**

On trouve :

- ACTIVE_SELECTION
- ACTIVE_CONTEXTS
- ACTIVE_PART
- ACTIVE_SHELL

De la même manière **E4Workbench** définit d'autres objets nommés (paramètres de lancement)

Cf [## ***Attention: Gestion de la mémoire.***](http://wiki.eclipse.org/Eclipse4/RCP/EAS>List_of_All_Provided_Services#Services³⁰</p>
</div>
<div data-bbox=)

- Les POJOs instanciés et injectés par E4 sont désalloués par le framework.
- Par contre, si on instancie un objet avec make il faut bien penser à le désinjecter !



30 - http://wiki.eclipse.org/Eclipse4/RCP/EAS>List_of_All_Provided_Services#Services

```
// Object containing @Inject annotations, created with make
MyService serv = ContextInjectionFactory.make(MyService.class, ctx);

// Somewhere in the code, don't forget to uninject it !
ContextInjectionFactory.uninject(serv, ctx);
```

Uninject

Documentation Eclipse

La documentation complète sur l'injection est disponible sur le wiki Eclipse :

- https://wiki.eclipse.org/Eclipse4/RCP/Dependency_Injection³¹

Injection : avantages/inconvénients

Avantages :

- L'injecteur rassemble toutes les instances courantes de manière hiérarchique
- Les listeners et les initialisations sont simplifiés :
 - les méthodes annotées avec `@Inject` sont réinvoquées automatiquement si un paramètre change dans le contexte
 - les champs annotés avec `@Inject` sont réinitialisés automatiquement si la valeur a changé dans le contexte
- On a besoin d'un singleton, on l'injecte (inutile de savoir d'où il vient)
- Permet de rendre un framework totalement indépendant d'une librairie graphique (renderer par exemple)

Inconvénients :

- Lors du debug d'une méthode injectée, l'appelant est du code d'introspection
- Il faut connaître les objets pouvant être injectés et donc les contextes d'injection
- Il faut bien maîtriser le cycle de vie du framework :
 - see : <http://www.vogella.com/articles/Eclipse4RCP/article.html>³² : chapter 25 Behavior
- On ne constate qu'à l'exécution si l'injection fonctionne (pas de message à l'édition : 'object never initialized')

F. Contextes d'injection

Les contextes d'injection

Eclipse 4 permet de définir des contextes d'injection hiérarchiques

Par défaut le contexte Eclipse contient :

- l'application model
- les services OSGi
- tous les objets ajoutés explicitement au contexte

Un objet est recherché dans le contexte courant, puis dans le contexte parent si non trouvé

Contextes locaux

Les objets qui implémentent l'interface `MContext` possèdent un contexte local.

31 - https://wiki.eclipse.org/Eclipse4/RCP/Dependency_Injection

32 - <http://www.vogella.com/articles/Eclipse4RCP/article.html>

Ces objets sont reliés au contexte parent

On trouve notamment :

- **MApplication**
- **MWindow**
- **MPerspective**
- **MPart**
- **MPopupMenu**

Pour obtenir le contexte d'un **MContext** on appelle `getContext()`

Les contextes sont créés et affectés automatiquement

Contextes créés pour une application

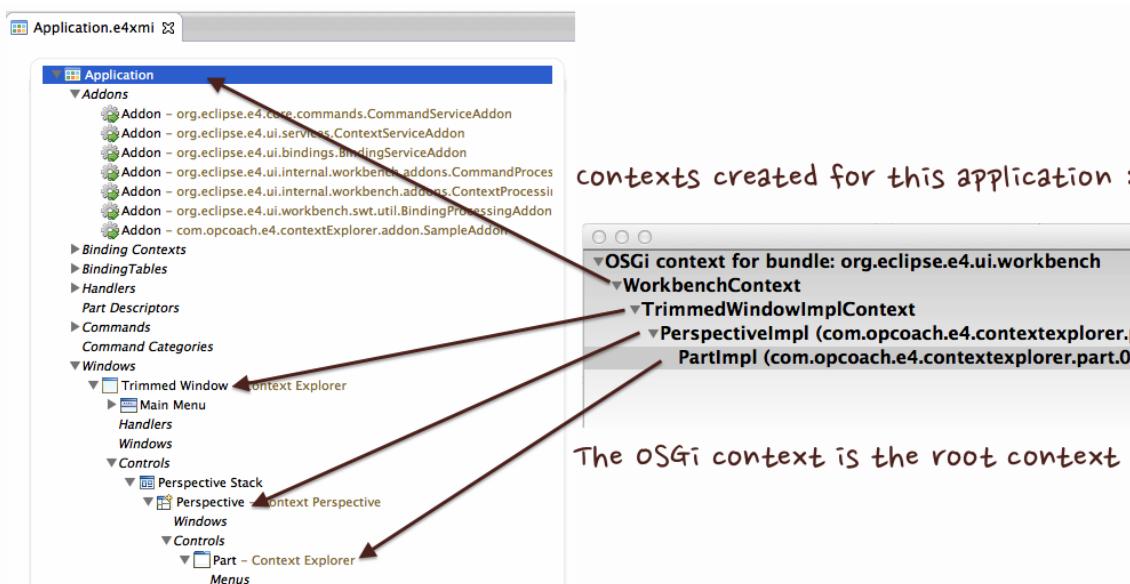


Image 99 Contexts in application

Algorithme de recherche

Les objets sont recherchés :

- dans le contexte local
- dans les contextes parents tant que ce n'est pas trouvé

Si un objet n'est pas trouvé dans le contexte :

- il peut encore être instancié si sa classe est annotée avec `@Creatable`
- pour les annotations `@Preference` et `@EventTopic` c'est l'`ExtendedObjectSupplier` associé qui calcule la valeur.
- si l'annotation `@Optional` est indiquée la valeur nulle est injectée
- si `@Optional` n'est pas présent et qu'aucune valeur n'est trouvée, une exception est lancée

Context Spy

- Livré dans e4 tools depuis janvier 2014
- Ajouter `org.eclipse.e4.tools.context.spy` dans la launch configuration
- Affichage ensuite par Alt Shift F10
- Il permet d'afficher tous les contextes courants et de voir leur contenu

Context Spy

Search data Ignore case Ignore WildCards

▼ OSGi context for bundle: org.eclipse.e4.ui.workbench

- ▼ WorkbenchContext
 - ▼ TrimmedWindowImplContext
 - PartImpl (org.eclipse.e4.tools.context.spy.example.contextview) Context
 - ▼ TrimmedWindowImplContext
 - PartImpl (org.eclipse.e4.tools.context.spy.example.contextspy) Context

Key	Value
Local values managed by this context	
activeChildContext	WorkbenchContext
debugString	OSGi context for bundle: org.eclipse.e4.ui.workbench
org.eclipse.core.runtime.IExtensionRegistry	org.eclipse.core.internal.registry.ExtensionRegistry@5eb9fde
org.eclipse.core.runtime.IProgressMonitor	org.eclipse.core.runtime.NullProgressMonitor@224577f9
org.eclipse.e4.core.internal.contexts.ContextObjectSupplier	org.eclipse.e4.core.internal.contexts.ContextObjectSupplier@3582c132
org.eclipse.e4.core.locale	fr_FR
org.eclipse.e4.core.services.adapter.Adapter	org.eclipse.e4.core.internal.services.EclipseAdapter@4b85c17
org.eclipse.e4.core.services.contributions.IContributionFactory	org.eclipse.e4.ui.internal.workbench.ReflectionContributionFactory@304caadb
org.eclipse.e4.core.services.events.EventBroker	No value could be yet computed (Exception : org.eclipse.e4.core.di.InjectionException)
org.eclipse.e4.core.services.log.ILoggerProvider	org.eclipse.e4.ui.internal.workbench.DefaultLoggerProvider@52639bf1
org.eclipse.e4.core.services.statusreporter.StatusReporter	No value could be yet computed (Exception : org.eclipse.e4.core.di.InjectionException)
org.eclipse.e4.core.services.translation.TranslationService	org.eclipse.e4.core.internal.services.BundleTranslationProvider@77bc1fb9
org.eclipse.e4.ui.model.application.MApplication	org.eclipse.e4.ui.model.application.impl.ApplicationImpl@67de0c09 (elementId: org.eclipse.e4.ui.internal.workbench.ExceptionHandler@604ee1f1)
org.eclipse.e4.ui.workbench.IExceptionHandler	org.eclipse.e4.ui.internal.workbench.ApplicationPartServiceImpl@2f26f304
org.eclipse.e4.ui.workbench.modeling.EPartService	null
org.eclipse.e4.ui.workbench.swt.modeling.EMenuService	org.eclipse.core.runtime.internal.adaptor.EclipseLogFactory\$1@796528a2
org.eclipse.osgi.framework.log.FrameworkLog	

E4 OSGI context

Le contexte du workbench

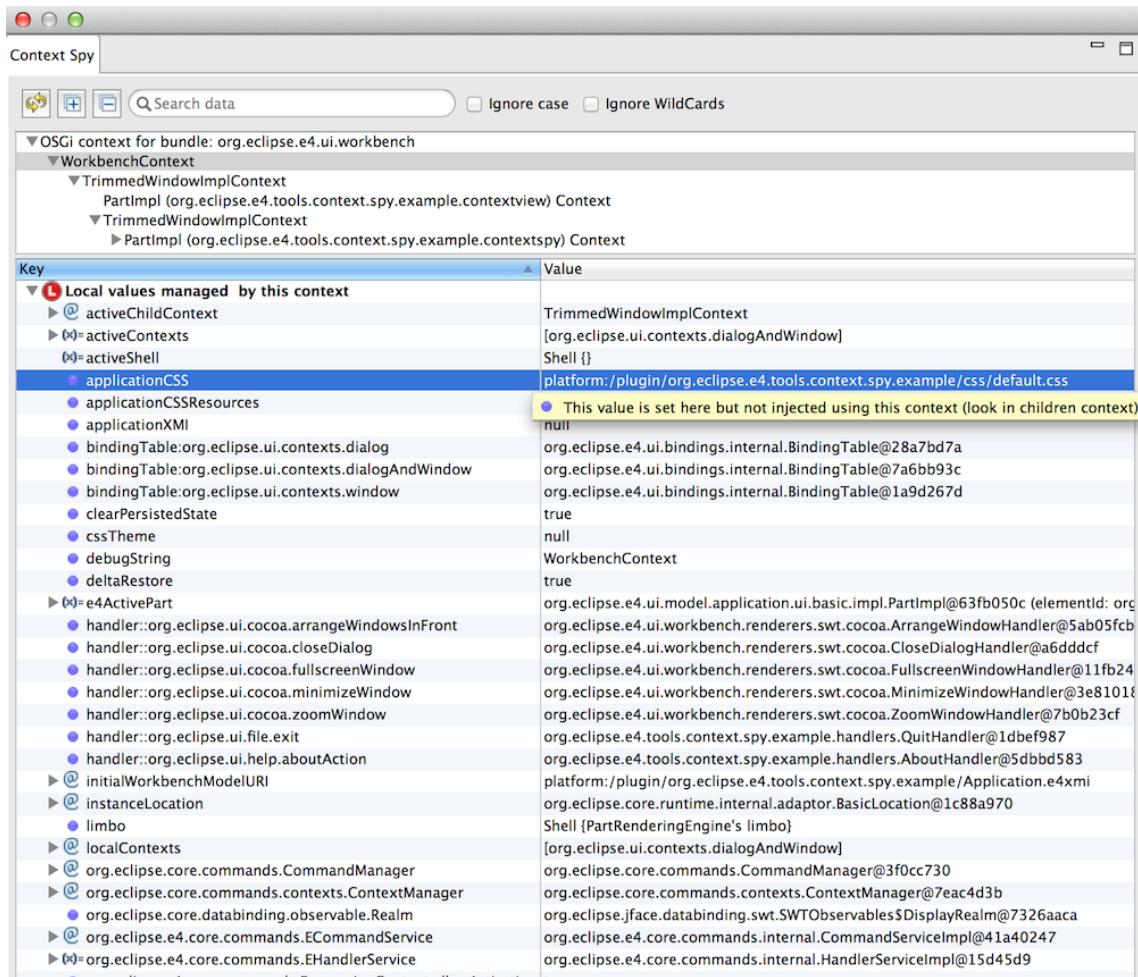


Image 100 Workbench Context

Le contexte de la trimmed window

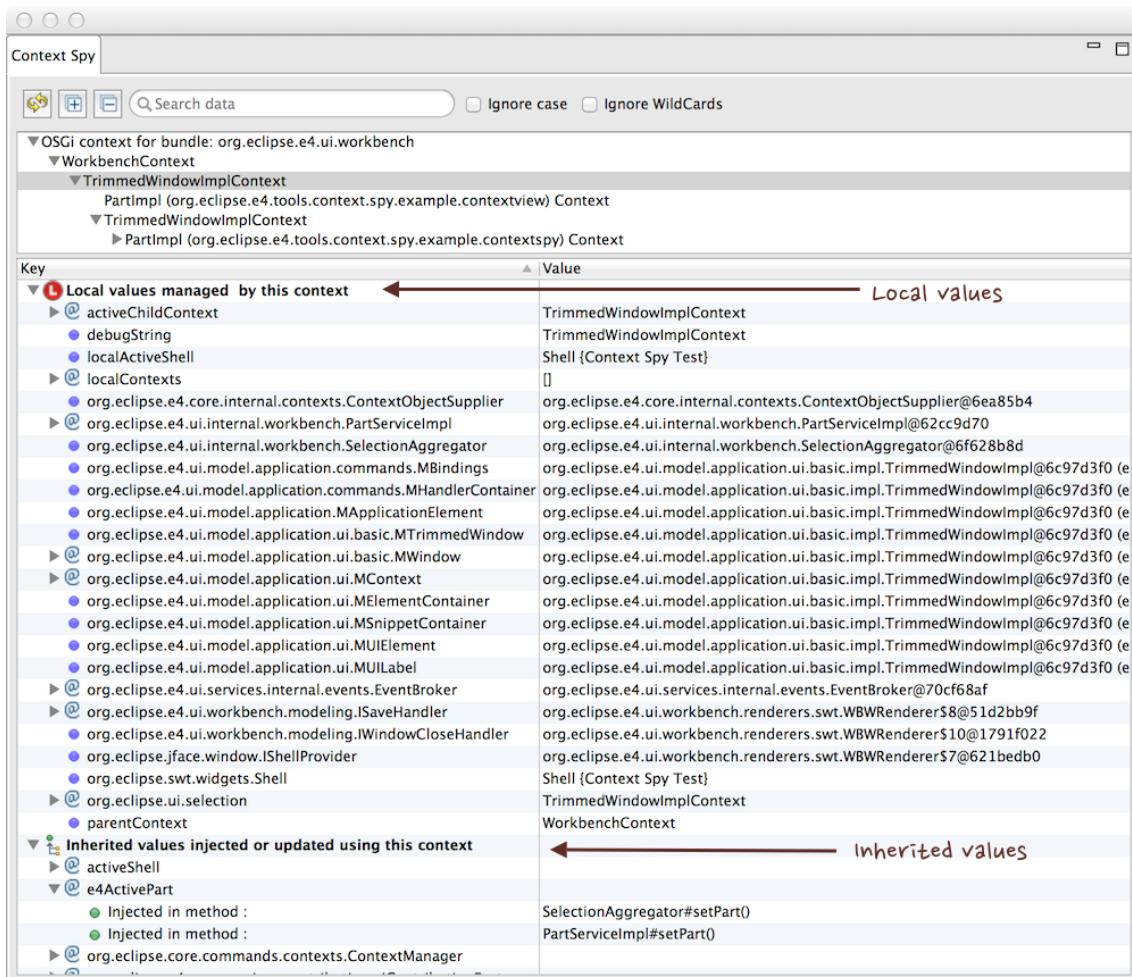
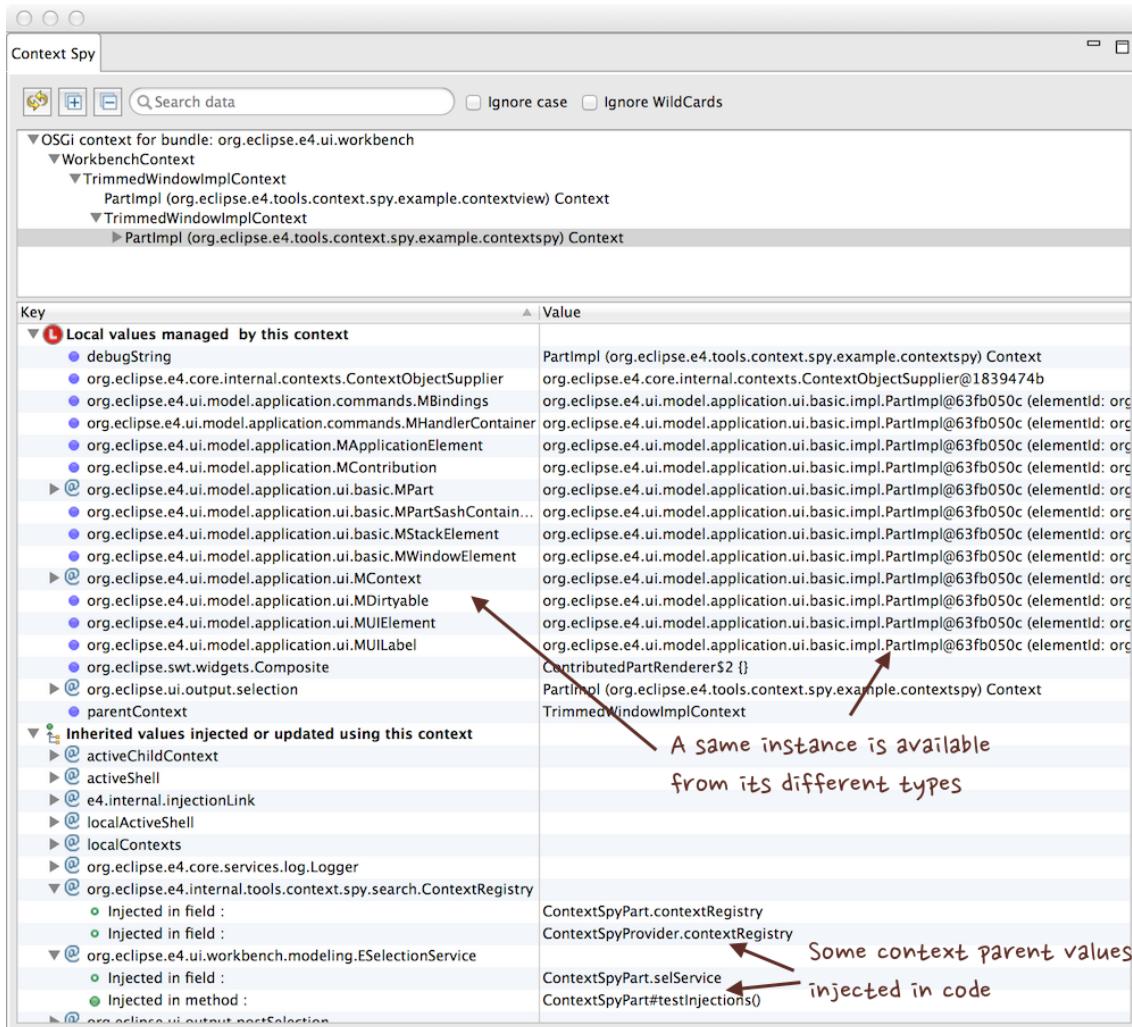


Image 101 Trimmed Window Context

Le contexte d'un part



The screenshot shows the Eclipse Context Spy tool interface. The tree view at the top shows the OSGI context for the bundle `org.eclipse.e4.ui.workbench`, specifically focusing on the `WorkbenchContext` and its nested `TrimmedWindowImplContext`. The table below lists local values managed by the context, including various Eclipse UI components like `MPart`, `MStackElement`, and `MWindowElement`.

Local values managed by this context:

- `debugString`
- `org.eclipse.e4.core.internal.contexts.ContextObjectSupplier`
- `org.eclipse.e4.ui.model.application.commands.MBindings`
- `org.eclipse.e4.ui.model.application.commands.MHandlerContainer`
- `org.eclipse.e4.ui.model.application.MApplicationElement`
- `org.eclipse.e4.ui.model.application.MContribution`
- `@ org.eclipse.e4.ui.model.application.ui.basic.MPart`
- `org.eclipse.e4.ui.model.application.ui.basic.MPartSashContainer...`
- `org.eclipse.e4.ui.model.application.ui.basic.MStackElement`
- `org.eclipse.e4.ui.model.application.ui.basic.MWindowElement`
- `@ org.eclipse.e4.ui.model.application.ui.MContext`
- `org.eclipse.e4.ui.model.application.ui.MDirtyable`
- `org.eclipse.e4.ui.model.application.ui.MUIElement`
- `org.eclipse.e4.ui.model.application.ui.MUILabel`
- `org.eclipse.swt.widgets.Composite`
- `@ org.eclipse.ui.output.selection`
- `parentContext`

Inherited values injected or updated using this context:

- `@ activeChildContext`
- `@ activeShell`
- `@ e4.internal.injectionLink`
- `@ localActiveShell`
- `@ localContexts`
- `@ org.eclipse.e4.core.services.log.Logger`
- `@ org.eclipse.e4.internal.tools.context.spy.search.ContextRegistry`
 - Injected in field :
 - Injected in field :
- `@ org.eclipse.e4.ui.workbench.modeling.ESelectionService`
 - Injected in field :
 - Injected in method :
- `@ org.eclipse.ui.output.selection`

Annotations and notes:

- A handwritten note states: "A same instance is available from its different types". It points to the row where `MPart` and `MWindowElement` both have the same value: `PartImpl (org.eclipse.e4.tools.context.spy.example.contextspy) Context`.
- Another handwritten note says: "Some context parent values injected in code". It points to the `ContextSpyPart` row in the table.

Image 102 Part context

Exercice EAP 032

Utilisation du context spy

Mise à jour du contexte

Le remplissage du contexte peut se faire de différentes manières :

- par l'utilisation du life Cycle URI
- par l'utilisation d'un Addon
- par l'injection d'un IEclipseContext ou de son objet qui le contient (MContext)
- par du code écrit en Eclipse 3 (dans le cas d'une migration E3->E4)

Mise à jour du contexte via le `lifeCycleURI`

On peut utiliser un 'lifeCycleURI' pour initialiser le contexte.

C'est une méthode utilisée pour les initialisations de haut niveau au démarrage de l'application

Pour remplir le contexte avec un lifeCycleURI :

- créer un POJO
- utiliser l'annotation `@PostContextCreate` sur une méthode qui reçoit le `IEclipseContext` par injection
- déclarer la propriété `lifeCycleURI` dans l'extension de produit

```

8  public class RentalUILifeCycle
9  {
10
11
12  @PostContextCreate
13  public void initContext(IEclipseContext ctx)
14  {
15      ctx.set(RentalAgency.class, RentalAgencyGenerator.createSampleAgency());
16  }
17 }
18
19

```

fill the context here

Image 103 `@PostContextCreate`

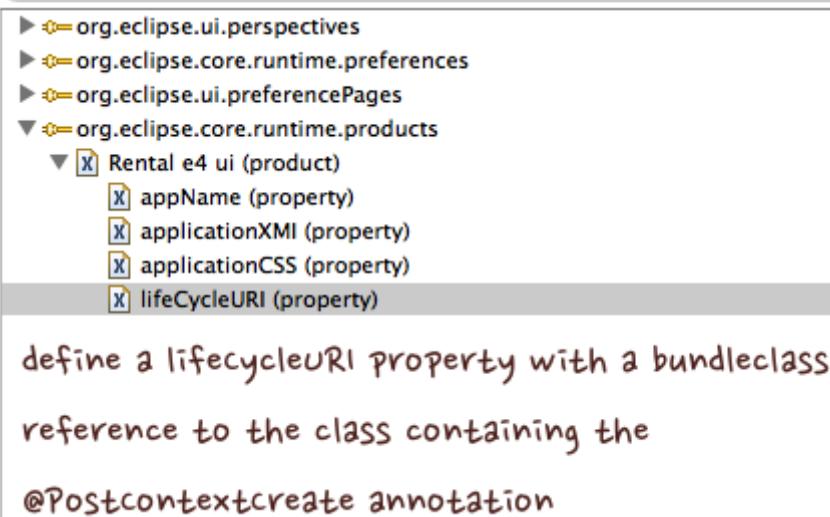


Image 104 `lifeCycleURI`

Référencer la classe avec un bundleclass ://

Extension Element Details

Set the properties of "property". Required fields are denoted by "*".

name*:

value*:

Image 105 `lifeCycleURI`

Attention : on ne peut avoir qu'une seule propriété `lifeCycleURI` par application

Mise à jour du contexte via les addon

Le modèle d'application permet d'ajouter des Addons

L'addon est utilisé pour remplir **le contexte du workbench** (reçu par injection dans la méthode `@PostConstruct`)

Il est utile pour modulariser les initialisations (chaque addon initialise sa partie)

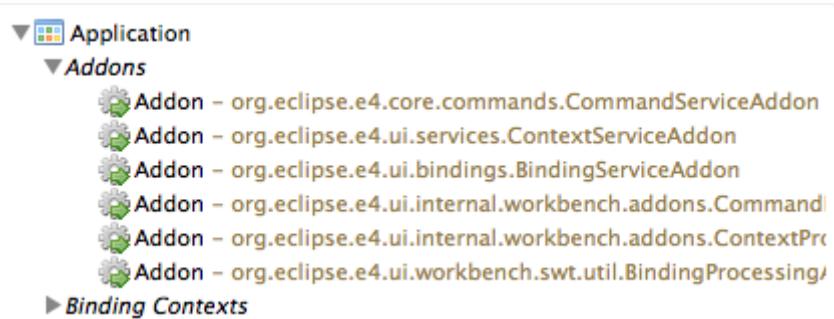


Image 106 Addons

Exemple de code d'addon :

```

25 */
26 public class CommandServiceAddon {
27     @PostConstruct
28     public void init(IEclipseContext context) {
29         // global command service. There can be only one ... per application :-
30         CommandManager manager = context.get(CommandManager.class);
31         if (manager == null) {
32             manager = new CommandManager();
33             setCommandFireEvents(manager, false);
34             context.set(CommandManager.class, manager);
35         }
36
37         CommandServiceImpl service = ContextInjectionFactory
38             .make(CommandServiceImpl.class, context);
39         context.set(ECommandService.class, service);
40
41         // handler service - a mediator service
42         context.set(EHandlerService.class.getName(), new HandlerServiceCreationFunction());
43     }
44

```

Image 107 Addon

Mise à jour du contexte par injection en paramètre

- Quand on veut initialiser un contexte de plus bas niveau que celui du Workbench
- Il suffit d'injecter le **IEclipseContext** à l'endroit voulu ou l'objet Mxxxx possédant un contexte
- Exemple 1 : dans une méthode injectée d'un Part, on reçoit le **IEclipseContext** du Part
 - On peut remonter dans les contextes parent en appelant context.**getParent()**
- Exemple 2 : dans une méthode injectée d'un Part, on reçoit la **MTrimmedWindow**
 - On peut accéder au contexte de la fenêtre en appelant : trimWindow.**getContext()**

Mise à jour du contexte dans du code E3

Lors d'une migration E3->E4 il peut être utile d'accéder au contexte par du code E3

Cela peut se faire à différents endroits :

- dans l'activateur d'un plugin (pour initialiser le contexte à partir de ce plugin)

- lors d'une auto injection : l'objet E3 est en cours de migration et doit injecter des valeurs
- lors de la création d'un nouveau contexte dans la hiérarchie existante

On peut récupérer le contexte de niveau OSGi en tant que service contexte :

```
// Get the OSGI global context :
Bundle e4Bundle = Platform.getBundle("org.eclipse.e4.ui.workbench");
if (e4Bundle != null)
{
    BundleContext e4BundleContext = e4Bundle.getBundleContext();
    IEclipseContext osgiCtx = EclipseContextFactory.getServiceContext(e4BundleContext);
    osgiCtx.set("myKeyInOsgi1", "value");
}
```

Image 108 get OSGi context

On peut aussi récupérer le contexte d'un objet en lui demandant le service IEclipseContext.
Par exemple sur l'application :

```
// Get the application context
IEclipseContext appliCtx = PlatformUI.getWorkbench().getService(IEclipseContext.class);
appliCtx.set("myKeyInAppli", "value");

// Can also get the OSGI context using parent access
appliCtx.getParent().set("myKeyInOSGi2", "value");
```

Image 109 get Application Context

Ou alors sur la fenêtre active :

```
// Get the workbench window context :
IWorkbench wb = PlatformUI.getWorkbench();
IEclipseContext windowCtx = wb.getActiveWorkbenchWindow().getService(IEclipseContext.class);
windowCtx.set("myKeyInWindow", "value");
```

Image 110 get window context

Exercice EAP 035

Définition de l'Addon pour Rental

G. API Annotation

Les annotations de comportement

- Pour compenser le manque de contrat d'API on utilise des annotations de comportements
- Ces annotations sont spécifiques à Eclipse 4
- Elles appliquent le mécanisme d'injection de paramètres sur la méthode
- Elles indiquent que le framework appellera la méthode à un certain moment
- Elles ne s'appliquent donc qu'à certains objets
 - Pour les parts : **@Focus, @Persist, @PersistState**
 - Pour les commandes : **@Execute, @CanExecute**

Les annotations réservées aux Parts

Ces annotations sont appelées par le moteur de rendu d'IHM :

- **@Focus** : appelée quand le Part prend le focus
- **@Persist** : appelée pour gérer la sauvegarde d'un éditeur
- **@PersistState**: appelée avant la destruction du Part pour sauver son état (position...). Utiliser la méthode **getPersistedState** sur le **MPart** injecté.

Ces annotations se trouvent dans le plugin [org.eclipse.e4.ui.di](#) (ajouter la dépendance)

Exemple d'utilisation dans un éditeur

```

12 public class SampleEditorPart
13 {
14     private static final String LAST_INPUT = "lastInput";
15     private Text inputText;
16
17     @PostConstruct  public void postConstruct(Composite parent, MPart part)
18     {
19         inputText = new Text(parent, SWT.NONE);
20
21         // Restore previous input text
22         String lastText = part.getPersistedState().get(LAST_INPUT);
23         if (lastText != null)
24             inputText.setText(lastText);
25
26         // Create here other widgets ...
27     }
28
29     @PersistState  public void saveTheState(MPart part)
30     {
31         // Save the states of some widgets to restore it later
32         part.getPersistedState().put(LAST_INPUT, inputText.getText());
33     }
34
35     @Focus  public void onFocus()  { inputText.setFocus();  }
36
37     @Persist  public void doSave() { System.out.println("Save editor data"); }
38
39 }
40 }
```

Inject MPart

Annotations usage

Les annotations pour les commandes

Ces annotations sont appelées par le moteur de rendu d'IHM quand il réagit aux commandes :

- **@Execute** : appelée lors de l'exécution d'un Handler
- **@CanExecute** : appelée pour contrôler si un handler peut s'exécuter

Les annotations de cycle de vie général

Ces annotations sont appelées par le framework application E4 :

- **@PostContextCreate** :
 - pour ajouter des objets au contexte
 - appelé une fois le contexte applicatif créé
- **@ProcessAdditions** :
 - pour ajouter des objets au modèle applicatif
 - appelé avant que le modèle soit affiché par le moteur de rendu

- **@ProcessRemovals :**
 - pour enlever des objets au modèle applicatif
 - appelé avant que le modèle soit affiché par le moteur de rendu
- **@PreSave :**
 - pour manipuler le modèle d'application avant sa sauvegarde

Attention : Toutes ces annotations ne sont appelées que si un lifeCycleURI a été défini sur le point d'extension de produit

Exemple final de code de 'Part'

```

26 public class ASampleView
27 {
28
29     @Inject
30     private ESelectionService selectionService;
31
32     @Inject
33     private EMenuService menuService; // This field is initialized by injector
34
35     @Inject
36     public void anotherMethod(@Optional EMenuService service)
37     {
38         // This method will be called
39     }
40
41     @PostConstruct
42     public void createContent(Composite parent, @Optional IStylingEngine styleEngine)
43     {
44         // Code to create the view
45     }
46
47     @PreDestroy
48     public void dispose()
49     {
50         // Call this method before deleting object (to remove listener for instance)
51     }
52
53     @Focus
54     public void onFocus()
55     {
56         // This method is called when part takes focus
57     }
58 }
59

```

Image 111 A viewPart with annotations

Annotation calculée, @Preference

- L'annotation **@Preference** récupère la valeur d'une préférence.
- C'est une annotation dont la valeur est calculée via un **ExtendedObjectSupplier**
- Elle peut s'utiliser pour un champ de classe ou pour un paramètre de méthode.

```

141
142
143 @Inject
144 public void refreshTree(@Preference(nodePath=PLUGIN_ID, value=CUSTOMER_KEY) String custCol,
145     @Preference(nodePath=PLUGIN_ID, value=RENTAL_KEY) String rk,
146     @Preference(nodePath=PLUGIN_ID, value=RENTAL_OBJECT_KEY) String rok)
147 {
148     if (agencyViewer != null)
149     {
150         labelProvider.initPalette();
151         agencyViewer.refresh();
152     }
153
154
155

```

Image 112 @Preference

Annotations de comportement : avantages/inconvénients

Avantages :

- Le code du framework devient totalement indépendant de la librairie d'IHM
- Le code se réduit
- Associé à l'injection, on peut centraliser les besoins d'une méthode dans son API :
 - ex: si le focus a besoin de la sélection: `@Focus void myFocus(@Named(ACTIVE_SELECTION) Object s)`

Inconvénients :

- Il faut connaître les annotations à utiliser et l'endroit où les appliquer
- Les classes annotées ne dérivent plus de classes de haut niveau (plus dur à conceptualiser)
 - Il faut organiser les classes dans des packages dédiés (parts, handlers, etc...)

Quelques liens

- http://wiki.eclipse.org/Eclipse4/RCP/Dependency_Injection³³
- <http://www.opcoach.com/category/blog/e4/>³⁴

H. JFace

Présentation

- JFace est une librairie pour faciliter la création d'IHM avec SWT
- Définit la notion de 'viewer' : modèle MVC
- Ajoute des facilités pour gérer du texte structuré (coloration, code assist...)
- fournit des 'helpers' pour aider à développer avec Eclipse : fieldPreference, registries, dialogues
- JFace fonctionne avec SWT (sans le cacher)
- JFace est indépendant de la plate-forme (pas de code natif)
- JFace est couplé avec 2 plugins d'Eclipse (core et osgi).

Les viewers

- Les viewers JFace permettent d'implémenter le design pattern MVC
- Ils définissent une méthode d'affichage des arbres, tables, arbres/tables, ...

33 - http://wiki.eclipse.org/Eclipse4/RCP/Dependency_Injection

34 - <http://www.opcoach.com/category/blog/e4/>

- Ils fonctionnent directement avec les objets du modèle.
- Ils fournissent des fonctions de tri
- Ils produisent des sélections typées

Il faut considérer le viewer comme un algorithme d'affichage de données

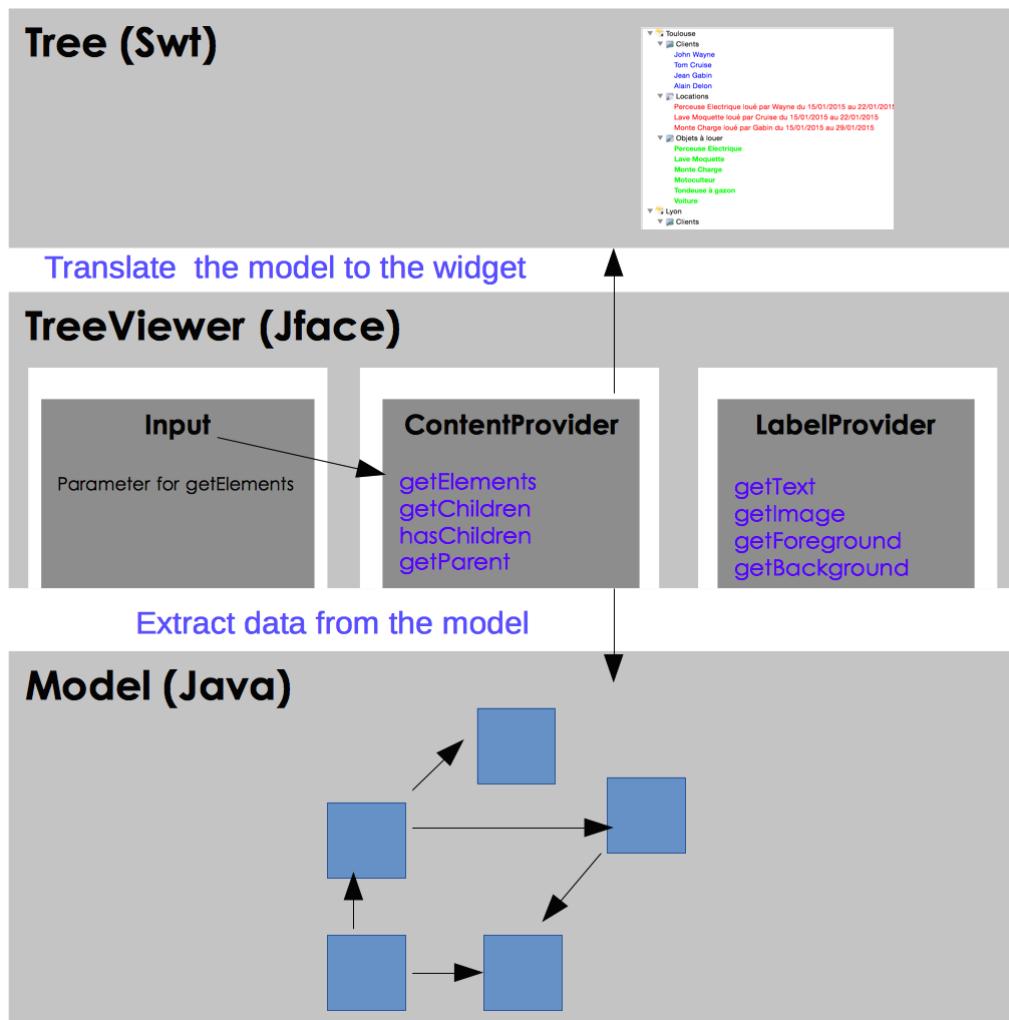
Caractéristiques des viewers

- Pour un arbre (**TreeViewer**), il faut indiquer comment naviguer dans les fils
- Pour une table (**TableViewer**) : il faut indiquer le contenu des colonnes

Définition d'un Treeviewer

Le TreeViewer est paramétré par 3 valeurs :

- L'input element : 'qui' ?
- Le content Provider : 'quoi' ?
- Le label Provider : 'comment' ?



TreeViewer overview

Input Element (racine)

- L'input est la donnée initiale d'un viewer
- Il n'est pas forcément affiché dans le viewer
- Il servira au contentProvider à trouver les données du modèle
- Il peut être de n'importe quel type : String, Collection<RentalAgency>, ...
- Exemples :
 - la clé dans une BDD
 - un path sur le disque
 - une collection des objets à afficher

ContentProvider

- Il s'agit d'une interface qui modélise la navigation dans les données.
- Il suffit d'implémenter l'interface correspondante à la structure visualisée.
 - **IStructuredContentProvider** : pour les listes, tables et combo viewers
 - **ITreeContentProvider** : pour les arbres
- La méthode **getElements** reçoit l'input en paramètre
- Les autres méthodes reçoivent l'objet sur lequel on navigue

```
public interface IContentProvider
{
    public void dispose();
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput);
}

public interface IStructuredContentProvider| extends IContentProvider
{
    // Get all elements to display from inputElement
    public Object[] getElements(Object inputElement);
}

public interface ITreeContentProvider extends IStructuredContentProvider {
    public Object[] getChildren(Object parentElement);
    public Object getParent(Object element);
    public boolean hasChildren(Object element);
}
```

Image 113 Content Providers

LabelProvider

- Il s'agit aussi d'une interface (**ILabelProvider**) donnant le texte et l'image pour un objet
- Il suffit de l'implémenter en dérivant de **LabelProvider**
- Attention aux images : c'est une ressource donc utilisation d'un **ImageRegistry**

```
public class MyProvider extends LabelProvider
{
    public Image getImage(Object element) { return null; }

    public String getText(Object element) { return super.getText(element); }
```

Image 114 Label Provider

ColorProvider, FontProvider

- Si le **LabelProvider** doit gérer les couleurs il peut aussi implémenter **IColorProvider**.
- Si le **LabelProvider** doit gérer les fontes il peut aussi implémenter **IFontProvider**.
- Attention ce sont des ressources ... Donc... utilisation d'un **ColorRegistry**, **FontRegistry** si les ressources sont dynamiques
- On peut utiliser les couleurs constantes: **Display.getCurrent().getSystemColor(SWT.COLOR_BLUE);**

```
public interface IColorProvider
{
    Color getForeground(Object element);

    Color getBackground(Object element);
}

public interface IFontProvider
{
    public Font getFont(Object element);
}
```

Image 115 Color, Font Providers

Algorithme du TreeViewer (1)

Appel du **getElements** (1er affichage)

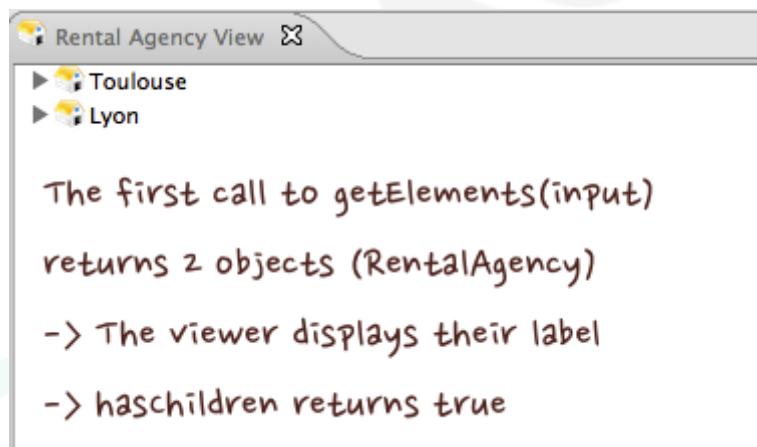


Image 116 **getElements**

Algorithme du viewer (2)

Navigation dans les données :

A click to expand a node calls `getChildren(Object e)`
and returns the array of children

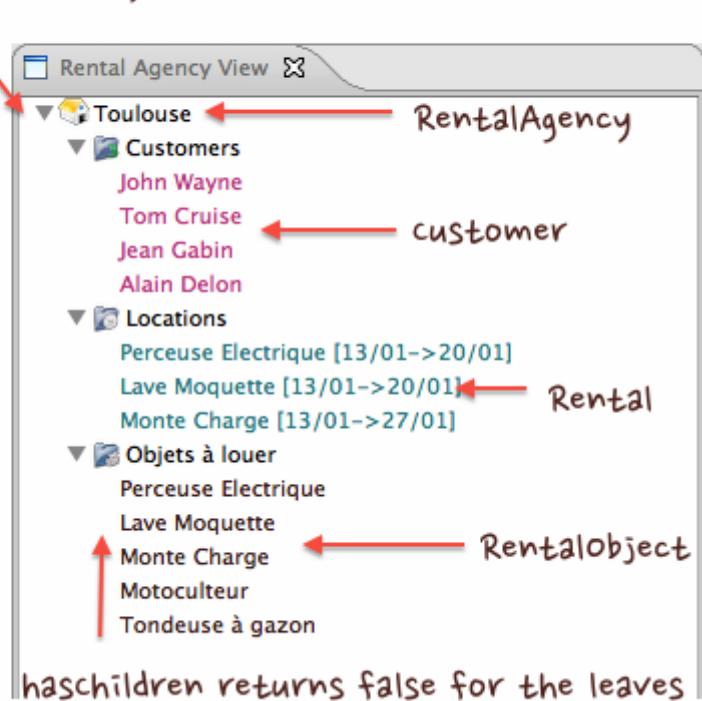


Image 117 `getChildren/hasChildren`

Exemple : TreeViewer pour un système de fichier

```
public void createPartControl(Composite parent)
{
    TreeViewer tv = new TreeViewer(parent);

    tv.setContentProvider(new FileTreeContentProvider());
    tv.setLabelProvider(new FileLabelProvider());
    tv.setInput(new File("C:\\\\"));
}
```

Image 118 Viewer create

Exemple : TreeContentProvider (1)

```

public class FileTreeContentProvider implements ITreeContentProvider
{
    private static final Object[] EMPTY_RESULT = new Object[0];

    // returns the first level element in the viewer
    public Object[] getElements(Object element)
    {
        Object[] result = null;

        if (element instanceof File)
        {
            result = ((File) element).listFiles();
        }

        return (result == null) ? EMPTY_RESULT : result;
    }
  
```

Image 119 TreeContentProvider

Exemple : TreeContentProvider (2)

Il faut implémenter en plus :

- `getChildren(Object elt)`
- `hasChildren(Object elt)`
- `getParent(Object elt)`

```

public Object[] getChildren(Object parentElement)
{
    if (parentElement instanceof File)
    {
        File parentFile = (File) parentElement;

        if (parentFile.isDirectory())
        {
            // list child files or directories
            return parentFile.listFiles();
        } else
        {
            // This is a file
            return EMPTY_RESULT;
        }
    }

    return EMPTY_RESULT;
}
  
```

Image 120

Exemple : TreeContentProvider (suite)

```
// hasChildren method
public boolean hasChildren(Object element)
{
    if (element instanceof File)
    {
        File file = (File) element;
        // return true if file is a drive or directory
        return (file.getParentFile() == null) || (file.isDirectory());
    }

    return false;
}

// getParent method
public Object getParent(Object element)
{
    if (element instanceof File)
    {
        return ((File) element).getParentFile();
    }
    return EMPTY_RESULT;
}
```

Image 121

Exemple : Label Provider

```
7 public class FileLabelProvider extends LabelProvider
8 {
9     // returns the text for a file element, or toString
10    public String getText(Object element)
11    {
12        if (element instanceof File)
13            return ((File)element).getName();
14        else
15            return super.getText(element); // toString
16    }
17 }
18 }
```

Image 122 Table label provider

Le TableViewer

On veut représenter les tables suivantes :

Agency Dashboard

Customers		Objects	
Firstname	Lastname	Name	
John	Wayne	Perceuse Electrique	
Tom	Cruise	Lave Moquette	
Jean	Gabin	Monte Charge	
Alain	Delon	Motoculteur	
		Tondeuse à gazon	

Object	Customer	Start Date	End Date
Perceuse Electrique	John Wayne	18/09/2012	25/09/2012
Lave Moquette	Tom Cruise	18/09/2012	25/09/2012
Monte Charge	Jean Gabin	18/09/2012	02/10/2012

Data displayed by Tableviewer are returned by getElements and managed by the Tableviewercolumn

Image 123 Table sample

TableViewer sample code

Pour la table customers par exemple, on écrira :

```

77    private TableViewer createCustomerTable(Composite top, RentalAgency currentAgency)
78    {
79        // Create the customer table with 2 columns: firstname and name
80        TableViewer viewer = new TableViewer(top);
81        final Table cTable = viewer.getTable();
82        cTable.setHeaderVisible(true);
83        cTable.setLinesVisible(true);
84        GridData gd_cTable = new GridData(SWT.FILL);
85        gd_cTable.setVerticalAlignment(SWT.TOP);
86        cTable.setLayoutData(gd_cTable);
87
88        // Create the first column for firstname
89        TableViewerColumn firstNameCol = new TableViewerColumn(viewer, SWT.NONE);
90        firstNameCol.getColumn().setWidth(200);
91        firstNameCol.getColumn().setText("Firstname");
92        firstNameCol.setLabelProvider(new ColumnLabelProvider() {@Override
93            public String getText(Object element)
94            {
95                return ((Customer)element).getFirstName();
96            }
97        });
98
99        // Create the second column for name
100       TableViewerColumn nameCol = new TableViewerColumn(viewer, SWT.NONE);
101       nameCol.getColumn().setWidth(200);
102       nameCol.getColumn().setText("LastName");
103       nameCol.setLabelProvider(new ColumnLabelProvider() {@Override
104           public String getText(Object element)
105           {
106               return ((Customer)element).getLastName();
107           }
108       });
109
110       // Set input data and content provider (default ArrayContentProvider)
111       viewer.setContentProvider(ArrayContentProvider.getInstance());
112       viewer.setInput(currentAgency.getCustomers());
113
114       return viewer;
115   }

```

Image 124 Table Viewer sample code

Exercice RCP 040

Treeviewer simple de l'agence

I. JFace Ressources

La gestion des ressources

- JFace facilite la gestion des ressources utilisées par SWT.
- JFace fournit des descripteurs de ressources :
 - **RGB** : composantes de couleur
 - **ImageDescriptor** :descripteur d'image
 - **FontDescriptor** : descripteur de fonte
- JFace fournit des classes de registry pour gérer les ressources :
 - **ColorRegistry**
 - **ImageRegistry**
 - **FontRegistry**

- On stocke et on récupère les ressources avec les descripteurs associés.

API de ColorRegistry

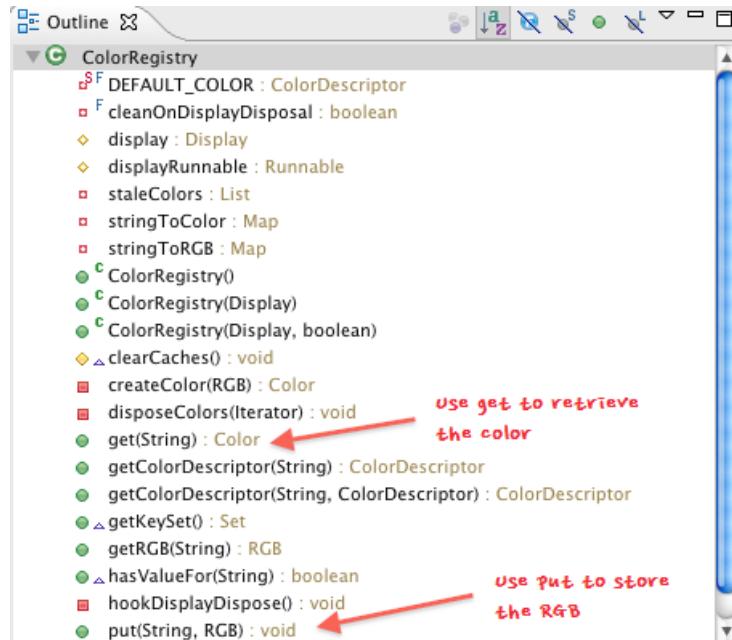


Image 125 API ColorRegistry

Exemple d'utilisation

```

68
69  /** A private methode to get a color in from a rgb key
70   * @param key the rgb value as String. For instance : "10,20,30" */
71  private Color getColor(String rgbKey)
72  {
73      // Use the global color registry (from jface)
74      ColorRegistry colorRegistry = JFaceResources.getColorRegistry();
75
76      // Test if a color exists for this key
77      Color col = colorRegistry.get(rgbKey);
78      if (col == null)
79      {
80          // If none, put a RGB for this key in registry
81          colorRegistry.put(rgbKey, StringConverter.asRGB(rgbKey));
82          // Get the created color by the registry
83          col = colorRegistry.get(rgbKey);
84      }
85      return col;
86  }
87

```

Image 126 ColorRegistry Usage

Initialisation de l'image registry d'un plugin.

- En Eclipse 4, il faut créer son propre ImageRegistry, l'initialiser et le mettre à disposition dans le contexte.
- Le mieux est de l'initialiser dans l'Addon spécifique du plugin
- Cet Addon sera référencé ensuite dans l'application model.
- Pour récupérer les images, on demande l'URL au bundle

```

67  /**
68  * A method to create and initialize an ImageRegistry
69  * @return a initialized ImageRegistry that can be put in the context
70  */
71 ImageRegistry getLocalImageRegistry()
72 {
73     // Get the bundle using the universal method to get it from the current class
74     Bundle b = FrameworkUtil.getBundle(getClass());
75
76     // Create a local image registry
77     ImageRegistry reg = new ImageRegistry();
78
79     // Then fill the values...
80     reg.put(IMG_CUSTOMER, ImageDescriptor.createFromURL(b.getEntry(IMG_CUSTOMER)));
81     reg.put(IMG_RENTAL, ImageDescriptor.createFromURL(b.getEntry(IMG_RENTAL)));
82     reg.put(IMG_RENTAL_OBJECT, ImageDescriptor.createFromURL(b.getEntry(IMG_RENTAL_OBJECT)));
83     reg.put(IMG_AGENCY, ImageDescriptor.createFromURL(b.getEntry(IMG_AGENCY)));
84
85     return reg;
86 }
87

```

Image 127 Setting ImageRegistry

On initialise les constantes directement avec les paths relatifs sur les images :

```

15
16     // Constants to manage object images in registry. Constant values are path to icons
17     public static final String IMG_AGENCY = "icons/Agency.png";
18     public static final String IMG_CUSTOMER = "icons/Customers.png";
19     public static final String IMG_RENTAL = "icons/Rentals.png";
20     public static final String IMG_RENTAL_OBJECT = "icons/RentalObjects.png";
21
22

```

Image 128 Constant values are file accessors

Publication de l'ImageRegistry dans l'Addon

On initialise simplement l'ImageRegistry dans le **@PostConstruct** de l'Addon

```

32 public class RentalAddon implements RentalUIConstants
33 {
34     @PostConstruct
35     void startRentalFeature(IEclipseContext ctx)
36     {
37         // Put objects in context
38         ctx.set(RentalAgency.class, RentalAgencyGenerator.createSampleAgency());
39         ctx.set(RENTAL_UI_IMG_REGISTRY, getLocalImageRegistry());
40     }
41
42
43

```

Put the ImageRegistry using an ID in context

Image 129 Publishing ImageRegistry

Utilisation de l'ImageRegistry

Pour l'utiliser on le récupère par injection et on l'interroge :

```

12
13  @Inject @Named(RENTAL_UI_IMG_REGISTRY)
14  private ImageRegistry registry;
15
16  @Override
17  public Image getImage(Object element)
18  {
19      if (element instanceof Customer)
20          return registry.get(IMG_CUSTOMER);
21      return null;
22 }

```

Image 130 Using ImageRegistry

Plug-in Image Browser

- Pour consulter les icônes ou images disponibles dans les différentes plugins
- Utiliser la vue Plug-in Image Browser

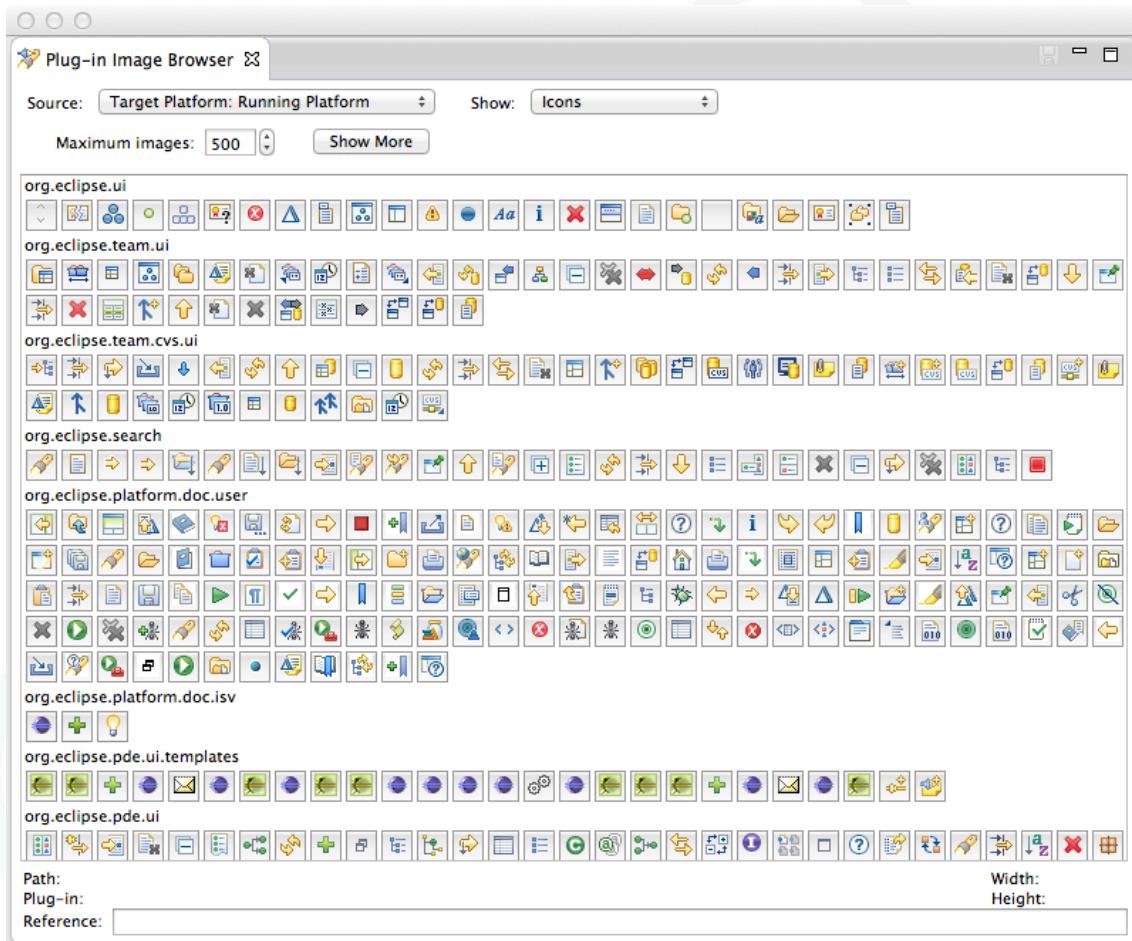


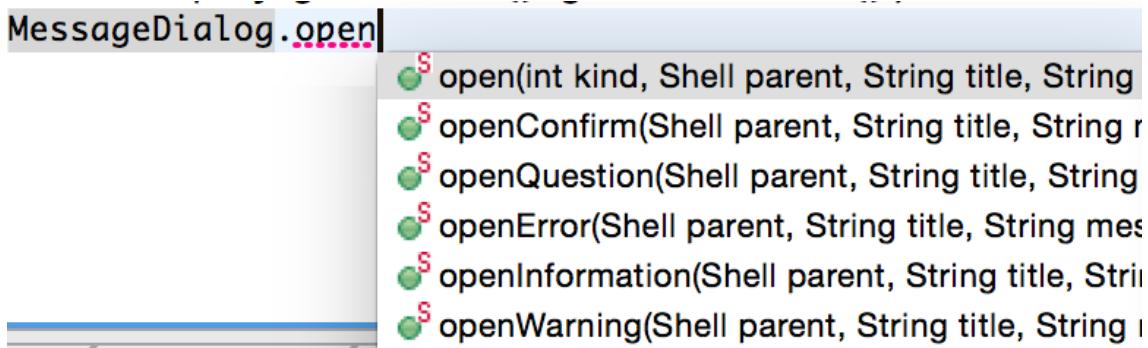
Image 131 Plug-in Image Browser

Exercice RCP 040

Treeviewer stylé de l'agence (couleurs, images)

Dialogues standards

JFace fournit également des dialogues standards via la classe **MessageDialog** :



Open methods

Exemple d'utilisation pour une confirmation :

```

11  public void displayConfirmationDialog()
12  {
13      Shell s = Display.getCurrent().getActiveShell();
14      if (MessageDialog.openConfirm(s, "Confirm your choice", "Do you confirm the delete ? "))
15      {
16          System.out.println("Do the delete");
17      }
18  }
19

```

MessageDialog

J. API pour l'IHM

La gestion de la sélection

- De part son indépendance avec le framework la sélection se gère par injection
- Elle s'injecte en utilisant l'objet nommé **IServiceConstants.ACTIVE_SELECTION**
- On récupère directement un 'Object' ou un 'Object[]' si sélection multiple
- Il n'y a pas de lien avec le framework autre que par le service de Sélection
- Une sélection dans l'IHM est propagée partout
- Il faut se déclarer envoyeur de sélection en envoyant les selections sur le **ESelectionService**
- Il faut se déclarer receveur de sélection par injection
- **Recevoir, par injection, la sélection courante génère l'équivalent du listener d'écoute**

Exemple de code

Branchemet de l'émetteur de sélection :

- On a la responsabilité de renseigner la selection du **ESelectionService**
- L'écoute et l'envoi se font au plus près des widgets

- On renvoie soit l'objet courant soit un tableau ou une liste des objets sélectionnés

```

105 @Inject
106 private ESelectionService selectionService; ← Get the SelectionService with injection
107
108 private void provideSelection()
109 {
110     // attach a selection listener to the jface viewer
111     agencyViewer.addSelectionChangedListener(new ISelectionChangedListener() {
112         public void selectionChanged(SelectionChangedEvent event)
113         {
114             // Get the selection in event
115             IStructuredSelection sel = (IStructuredSelection) event.getSelection();
116             // set the selection to the service
117             selectionService.setSelection(sel.size() == 1 ? sel.getFirstElement() : sel.toArray());
118         }
119     });
120 }
121 
```

do a specific code to set the selection

Image 132 selection provider

Ecoute de la sélection

- L'écoute se fait de manière automatique à partir du moment où on injecte la sélection
- Elle peut être nulle, donc utiliser l'annotation `@Optional` sur le paramètre ou sur la méthode
- Le nom de l'objet à injecter est : `IServiceConstants.ACTIVE_SELECTION`
- On peut caster directement l'objet sélectionné dans le type attendu

```

11
12 /**
13 * This method will be invoked only if current selection is a Rental instance */
14 @Inject @Optional
15 public void receiveSelection(@Named(IServiceConstants.ACTIVE_SELECTION) Rental r)
16 {
17     setRental(r);
18 }

```

Image 133 Get the selection

Ecoute de la sélection multiple

- Pour la sélection multiple on reçoit soit :
 - un `Object[]` si on a retourné `toArray()`
 - une `List<?>` si on a retourné `toList()`
- **Attention** : les types ne peuvent pas être contrôlés par java
- On ne peut donc pas recevoir un `Customer[]` ou une `List<Customer>`

```

105
106 @Inject @Optional
107 public void selectCustomers(@Named(IServiceConstants.ACTIVE_SELECTION) Object[] selection)
108 {
109     for (Object o : selection)
110         if (o instanceof Customer) ←
111             setCustomer((Customer) o); ← For multiple selection it is
112
113
114 } necessary to test the types

```

Image 134 Multiple Selection

Ecoute de la sélection (2)

- Il est aussi possible d'écouter le `ESelectionService` avec `addSelectionListener`

- C'est une méthode plus 'traditionnelle' qui nécessite d'appeler le `removeSelectionListener`

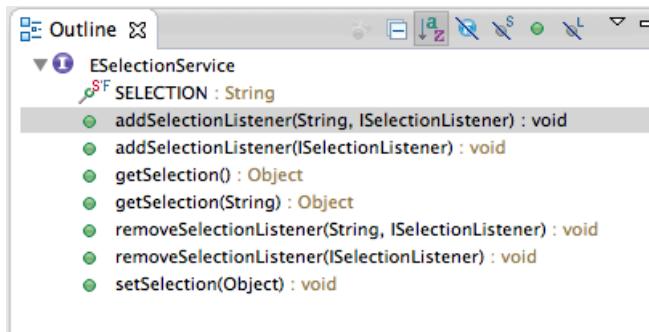


Image 135 ESelectionService

Eviter les boucles lors de l'injection de la sélection

Pour éviter que l'émetteur de sélection ne reçoive à nouveau la sélection il faut :

- injecter la sélection courante et le part actif
- tester si le part est actif est différent du part émetteur :

```

125
126     /** With the dashboard part, we can now receive selection from outside of this view */
127     @Inject
128     @Optional
129     public void selectionChanged(@Named(IServiceConstants.ACTIVE_SELECTION) Object selection,
130                                     @Named(IServiceConstants.ACTIVE_PART) MPart currentPart)
131     {
132         // Nothing to do if nothing available.
133         if (currentPart == null || selection == null || agencyViewer == null)
134             return;
135
136         // Is this selection coming from outside of this part ?
137         // In this case must set it on the viewer if created
138         if ( ! VIEW_ID.equals(currentPart.getElementId()) ) ← check if the
139         {
140             // Must recreate a structuredSelection !
141             IStructuredSelection ss = new StructuredSelection(selection);
142             agencyViewer.setSelection(ss, true);
143         }
144     }
145
146
    
```

selection is
coming from
outside

Image 136 Loop management

Outil : Plugin Spy

ALT+SHIFT+F1 : "Plug-in Spy"

- Retrouver la classe d'implémentation d'un éditeur, d'une vue, d'une page de préférences, ...
- **Connaître le type de la sélection courante**
- Retrouver quel plug-in contribue la vue ou l'éditeur actif
- etc...

Pour accéder au plugin spy dans son application il faut ajouter le plugin :

org.eclipse.pde.runtime



Le plugin spy ne fonctionne malheureusement pas encore entièrement en Eclipse 4.4 pur.



Exercice EAP 050

Gestion de la sélection

Drag and Drop - DragSource (org.eclipse.swt.dnd)

Principe :

- Définir une **DragSource**
- Définir les types de transferts envoyés (Local, Text...)

```

28     public void setLabelAsDragSource(final Label label)
29     {
30         DragSource source = new DragSource(label, DND.DROP_MOVE | DND.DROP_COPY);
31
32         // Define the transfer type (Local, Text...)
33         source.setTransfer(new Transfer[] { TextTransfer.getInstance() });
34
35         // Add a drag lister on source
36         source.addDragListener(new DragSourceAdapter()
37         {
38             public void dragSetData(DragSourceEvent event)
39             {
40                 if (TextTransfer.getInstance().isSupportedType(event.dataType))
41                 {
42                     event.data = label.getText();
43                 }
44             }
45         });
46     }
47

```

Image 137 DragSource sample

Drag and Drop - Drop Target

Principe :

- Définir le **DropTarget**
- Définir les types de transferts acceptés

```

18
19     public void setLabelAsDropTarget(final Label label)
20     {
21         DropTarget target = new DropTarget(label, DND.DROP_MOVE | DND.DROP_COPY);
22         target.setTransfer(new Transfer[] { TextTransfer.getInstance() });
23         target.addDropListener(new DropTargetAdapter()
24         {
25             public void drop(DropTargetEvent event)
26             {
27                 // Do your drop stuff here
28
29             }
30         });
31     }
32

```

Image 138 DropTarget

Exercice RCP 052

Drag and Drop

JFace Data Binding

Il s'agit d'un mécanisme très souple pour coupler des données

- approche déclarative
- implémentation simpliste
- peut être utilisé dans un contexte non-Eclipse
- c'est une interface : on utilise des implémentations dédiées (EMF, Bean, Swt...)
- Permet de spécifier des règles de validation ou de conversion
- Evite d'écrire des propertyChangeListeners plus compliqués

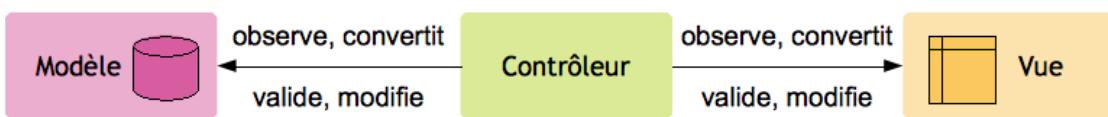


Image 139 Data Binding Mechanism

Implémentations DataBinding

Plusieurs sont disponibles :

- Javabeans : Beans et POJOs
- SWT : pour les contrôles (texte, selection, background, ...)
- JFace : pour les viewers (input, selection...).
- EMF : modifications du modèle

On peut aussi implémenter ses propres objets observables, en sous-classant

- **AbstractObservableValue**
- **AbstractObservableList**

Mécanismes de conversion et de validation.

- Ils se gèrent avec la classe **UpdateValueStrategy**
- Permet de travailler sur la valeur observée
- Conversion
 - **UpdateValueStrategy** : **:convert(Object)**
 - Renvoyer l'objet converti
- Validation
 - **UpdateValueStrategy** : **:validateAfterConvert(Object)**
(mais aussi validateBeforeGet, etc...)
 - Renvoyer un **IStatus** explicitant l'erreur (ou bien Status.OK_STATUS !)

Exemple d'utilisation sans conversion

On veut relier le champ 'Nom' d'une instance de Personne sur un widget Text

```
public void bindSimpleValue()
{
    DataBindingContext bc = new DataBindingContext();
    bc.bindValue(SWTObservables.observeText(name, SWT.Modify),
                 PojoObservables.observeValue(person, "name"),
                 null,
                 null);
}
```

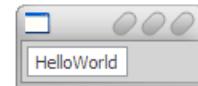


Image 140 Simple databinding

Exemple d'utilisation avec conversion

On ajoute une conversion sur le texte qui transite :

```
public void bindMyValues()
{
    DataBindingContext bc = new DataBindingContext();
    bc.bindValue(SWTObservables.observeText(name, SWT.Modify),
                 PojoObservables.observeValue(person, "name"),
                 null,
                 new UpdateValueStrategy()
                 {
                     public Object convert(Object value)
                     {
                         return ((String) value).toUpperCase();
                     }
                 });
}
```



Image 141 Databinding with strategy

Databinding et WindowBuilder

- WindowBuilder permet de générer facilement le code de databinding
- Il faut une instance de l'objet à binder définie en champ privé
- On peut utiliser les implémentations swt, bean et emf

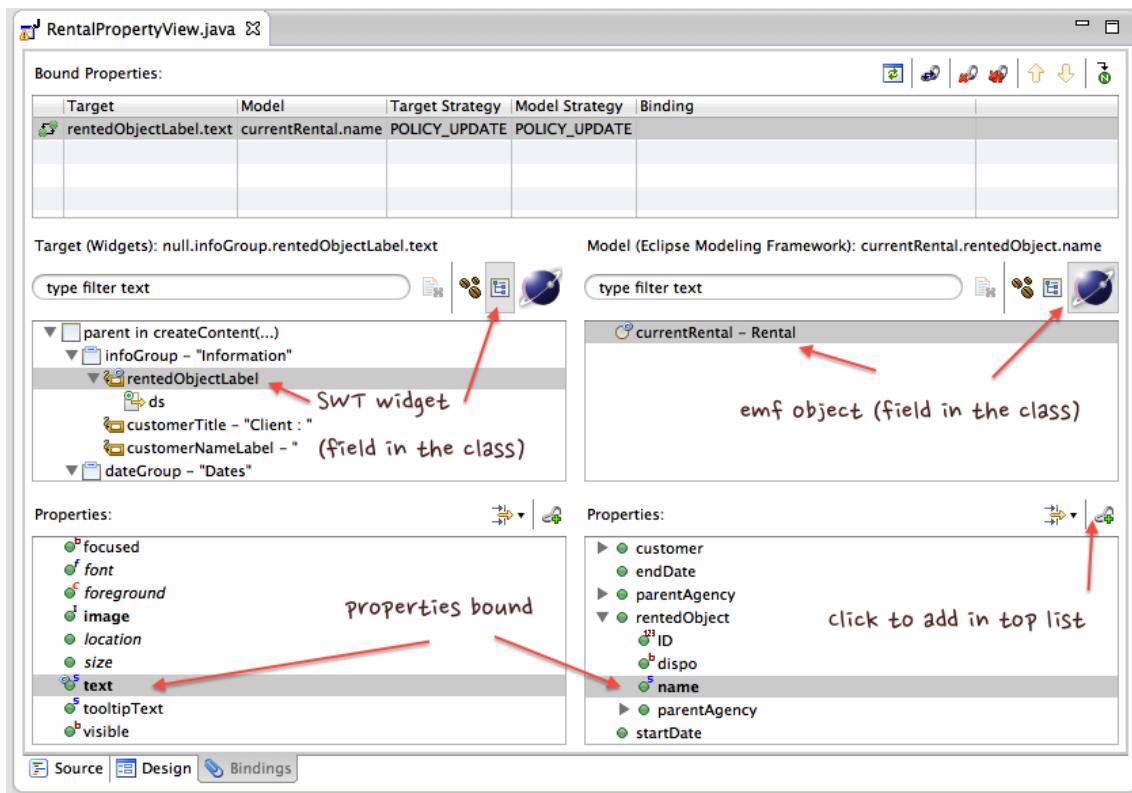


Image 142 Databinding

Exercice

Reprendre le setRental avec window builder pour faire le data binding

Jobs et Progress bar

Job

- Tache de fond
- Doit implémenter la méthode run(**IProgressMonitor** pm)
- Un Job est réutilisable (il peut se relancer plusieurs fois sans être détruit).
- Peut être lancé (schedule), suspendu (suspend), attendu (join)
- Peut être priorisé (setPriority ou setRule)
- Permet la réactivité des ihms (avec **UIJob**).
- L'avancée du Job peut se visualiser (progress monitor).

IProgressMonitor

- Barre de progression (**beginTask**(String, int), **worked**(int))
- Gère l'annulation du Job.

Définition d'un Job

```

10
11 public class SampleJob extends Job
12 {
13     public SampleJob()
14     {
15         super("Computation Job");
16     }
17
18     @Override
19     protected IStatus run(IProgressMonitor monitor)
20     {
21         // Compute 10000 steps
22         monitor.beginTask("Computing", 10000);
23
24         for (int i = 0; i < 10000; i++)
25         {
26             // Do Something...
27             monitor.setTaskName("Computing the " + i + " th block");
28             for (int j = 0; j < 1000; j++)
29             {
30                 double acos = Math.cos((double) j);
31             }
32
33             // One step done
34             monitor.worked(1);
35
36             // Stop the job if canceled in progress monitor.
37             if (monitor.isCanceled())
38             {
39                 return Status.CANCEL_STATUS;
40             }
41         }
42         return Status.OK_STATUS;
43     }
44 }
45

```

Image 143 Sample Job

Lancement d'un Job

- Pour lancer un job il suffit de le scheduler
- On peut aussi provoquer l'affichage de la Progress View

```

// Launch the Job
SampleJob sj = new SampleJob();
sj.schedule();

```

Image 144 Launching the job

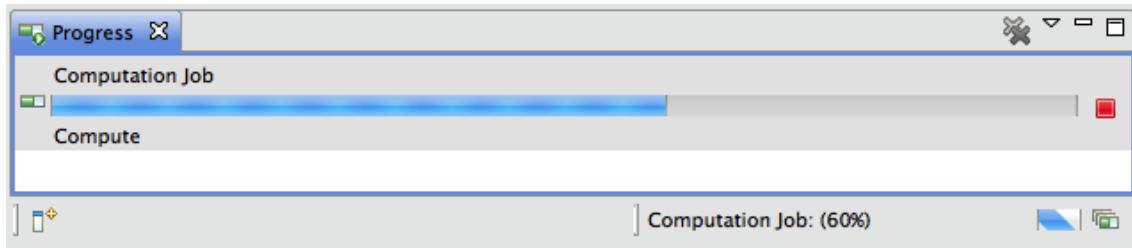


Image 145 Progress View

Jobs et IHM

- Un job ne peut pas modifier l'IHM gérée par le thread SWT (Invalid Thread Access Exception)
- Il faut utiliser Display.asyncExec(Runnable r)
- UIJob permet aussi de gérer les refresh trop longs
- Il faut implémenter la méthode **runInUIThread(IProgressMonitor pm)**
- **Utilisation : toujours dans un autre Job**

Jobs et IHM (exemple)

```
public void jobWithUIJob()
{
    Job j = new Job("Job Title Here")
    {
        protected IStatus run(IProgressMonitor progress)
        {
            // Your code to run in the background
            // ...
            UIJob updateUI = new UIJob("UIJob Title Here")
            {
                public IStatus runInUIThread(IProgressMonitor monitor)
                {
                    // Calls to SWT components
                    // ...
                    return Status.OK_STATUS;
                }
            };

            updateUI.schedule();
            return Status.OK_STATUS;
        }
    };
    j.schedule();
}
```

Image 146 UIJob

Conseil : Annulation des tâches en cours

Comment annuler plusieurs jobs lors d'une annulation ?

- Implémenter sur chaque Job concerné la méthode
`public boolean belongsTo(Object family)`
- Appeler ensuite : `Job.getJobManager().cancel(Object family)`

OPCOACH

Points d'extension et Model UI



Application Model UI	127
Eclipse 4 commands	131
Extension Expression definition	141
Gestions des Opérations	145
Gestion des wizards	146
Gestion des préférences	149

La construction d'une IHM Eclipse 4 se fait par :

- la création d'éléments UI dans le modèle d'application
- le codage des classes non encore gérées dans le modèle

Dans les deux cas du code SWT/JFace sera écrit

Window Builder permet l'édition graphique des éléments finaux

La documentation de tous les points d'extension est :

- Disponible dans l'aide en ligne :
 - Platform > Reference > Extension Points Reference
 - JDT > Reference > Extension Points Reference
- Très complète :
 - Description
 - Configuration
 - Exemple
 - API
- Générée à partir du XML Schema du point d'extension

A. Application Model UI

La partie UI de l'application model

Le modèle d'application permet de décrire la globalité d'une IHM

Cette partie permet :

- de définir la fenêtre principale
- les menus principaux
- les perspectives
- les parts
- les menu et commandes spécifiques à chaque part

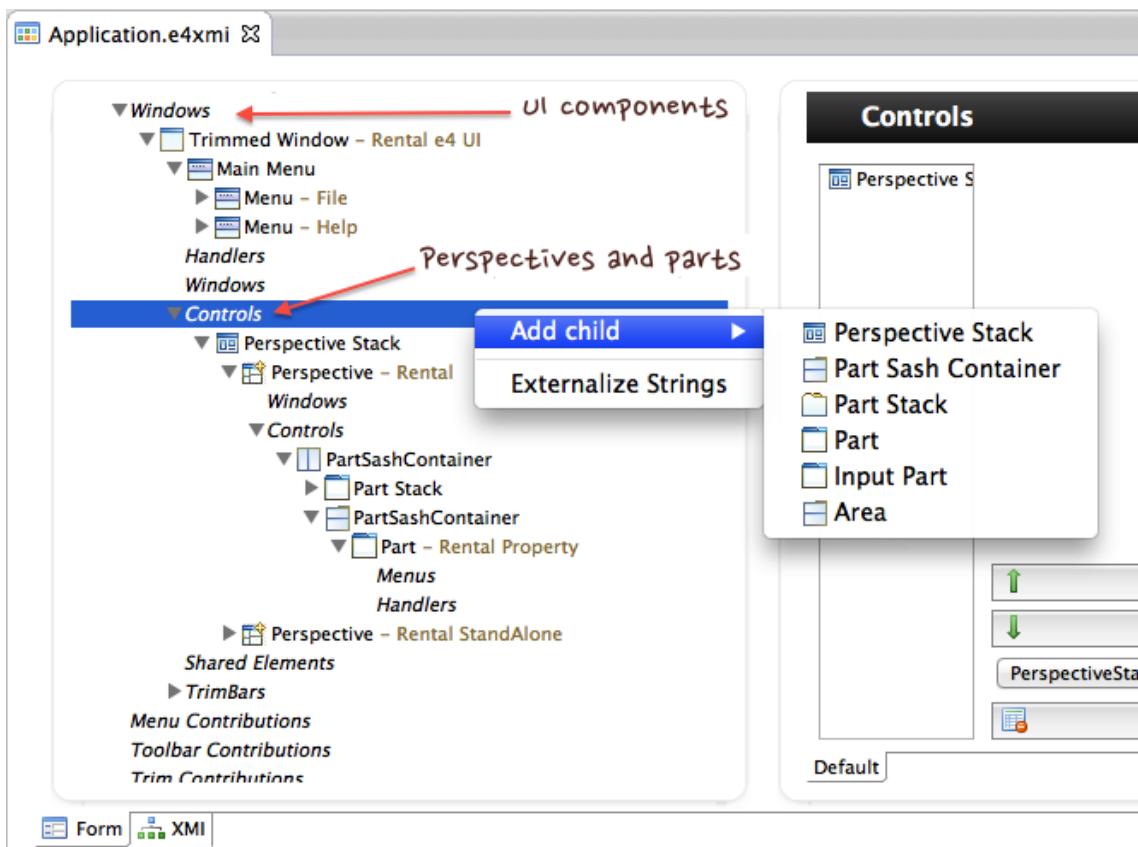


Image 147 UI management

Définition de la fenêtre principale

On définit la fenêtre principale dans une trimmed Window

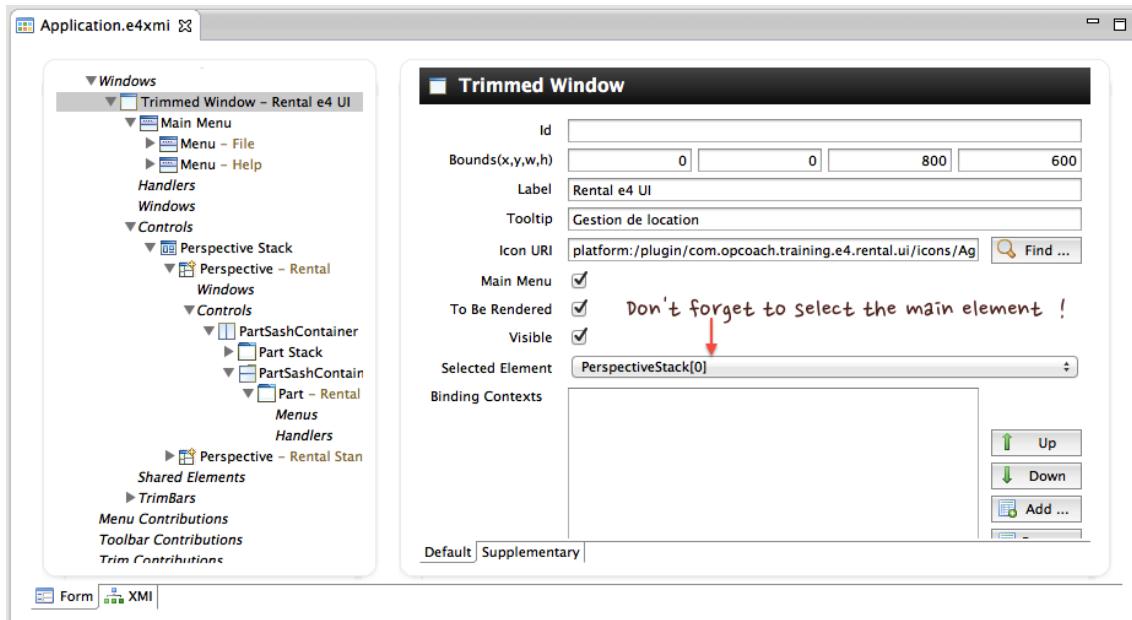


Image 148 Main window

Définition du contenu Perspective/Part

On peut créer les composants suivants :

- **PerspectiveStack** : facultatif. Pour stocker les perspectives
- **Perspective** : pour définir une perspective (facultatif)
- **Part Sash Container** : pour organiser en deux parties verticales ou horizontales
- **Part Stack** : pour voir les onglets des vues
- **Part** : pour mettre une vue (sans onglet) ou un éditeur
- **Area** : définit une zone pour y mettre des composants (editor area, ...)
- **PlaceHolder** : pour référencer un part partagé (partie 'shared elements')

Définition d'une perspective

La perspective se définit directement dans le modèle (pas de point d'extension)

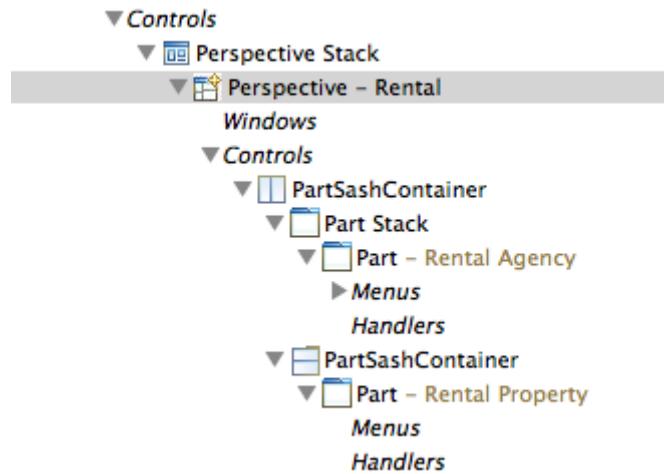


Image 149 Perspective

Paramètres de la perspective

Perspective

Id	com.opcoach.training.e4.rental.ui.rentalperspective				
Accessibility Phrase					
Selected Element	PartSashContainer[0]				
Label	Rental				
Tooltip	Rental Perspective				
Icon URI	platform:/plugin/com.opcoach.training.e4.r	Find ...			
To Be Rendered	<input checked="" type="checkbox"/>				
Visible	<input checked="" type="checkbox"/>				
Context Properties	<table border="1"> <thead> <tr> <th>Key</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table> Add ... Remove	Key	Value		
Key	Value				
Variables	<input type="text"/> Add				
Default	<input checked="" type="radio"/>				
Supplementary	<input type="radio"/>				

Image 150 Parameters

Exercice EAP 060

Création perspective Rental

B. Eclipse 4 commands

Système de commandes

La gestion des commandes dans Eclipse est une longue histoire...

- Action JFace
- Framework de commandes Eclipse 3
- Commandes dans l'application modèle (Eclipse 4)

Les principes du framework de commande E3 sont conservés : command et handlers

Les points d'extension correspondants sont migrés dans le modèle d'application

- les commandes et handlers au niveau de l'application ou des parts
- les 'menuContribution' : à l'endroit voulu ou dans les contributions

Le point d'extension d'expression definition est conservé tel quel

Principe des commandes E4

On distingue :

- **command** : définition de la commande
- **handler** : son comportement
- **handledItem** : commande placée dans l'IHM
- **directItem** : handler appelé explicitement et placé dans l'IHM

Définition d'une commande

Une commande est définie par :

- un ID unique
- un nom
- une catégorie

Elle se définit dans l'application model :

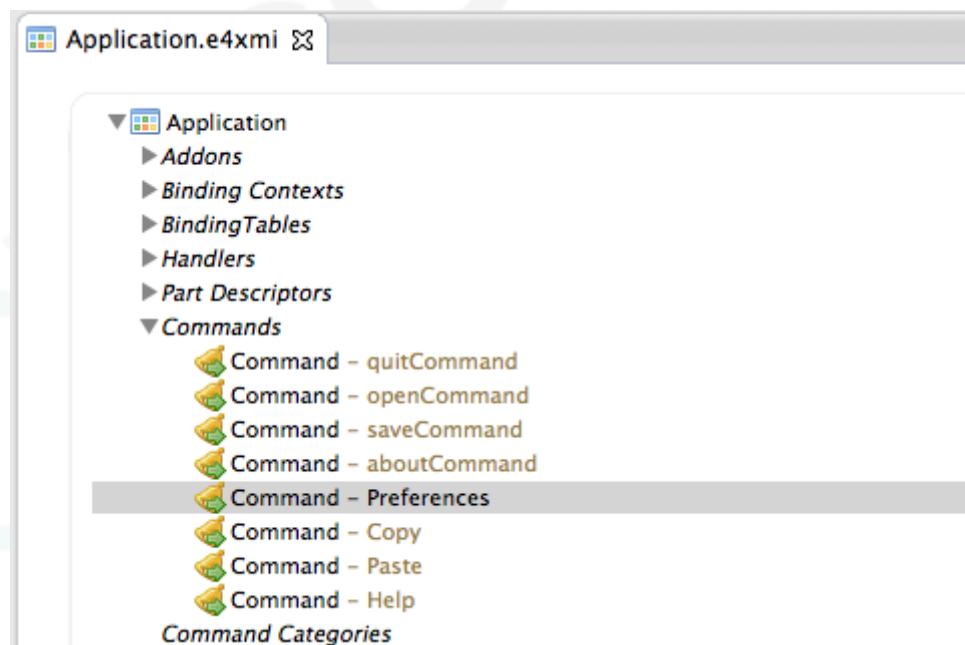


Image 151 Command in model

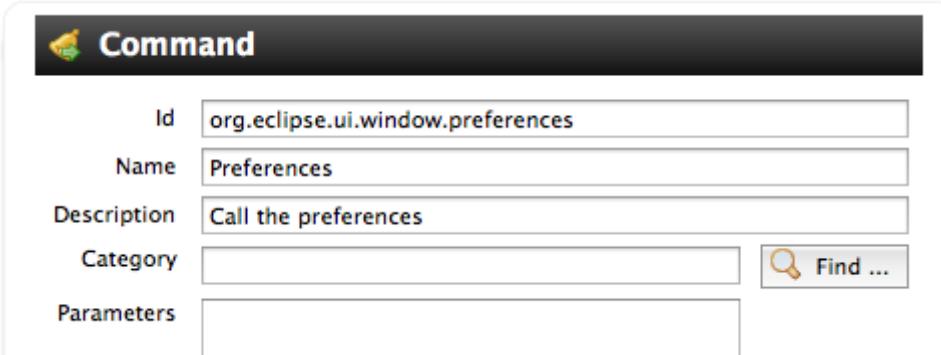


Image 152 Command parameter



Remarque : Commandes dans l'application model

- Toutes les commandes définies pour une application doivent être définies dans le modèle
- On ne récupère pas les définitions comme en Eclipse 3

ID des commandes

Les commandes standard d'Eclipse ont des ID normalisés

Il faut les utiliser si on mixe du code Eclipse 3 avec la couche de compatibilité

On les retrouve dans **IWorkbenchCommandConstants**

On peut les déduire par : 'org.eclipse.ui'+menu+commande

- Exemple : **org.eclipse.ui.edit.copy**

Autres exemples :

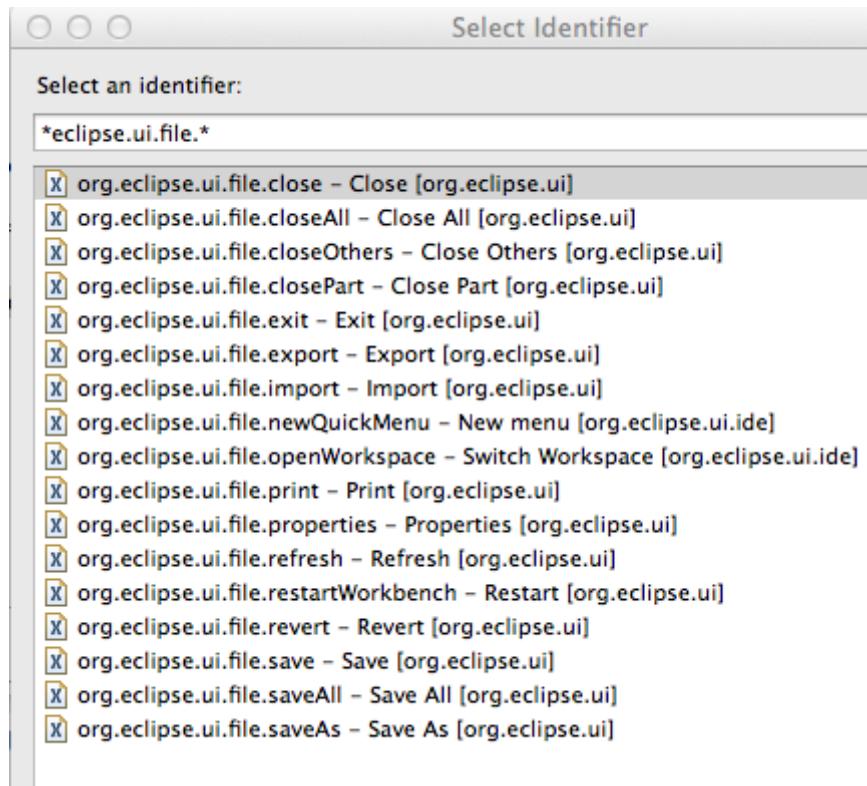


Image 153 Main IDs

Définition du Handler

- Le handler définit le comportement d'une commande
- Il se déclare dans l'application model
- Une commande ne sera active que si un handler est trouvé
- Ils peuvent être définis de manière globale :

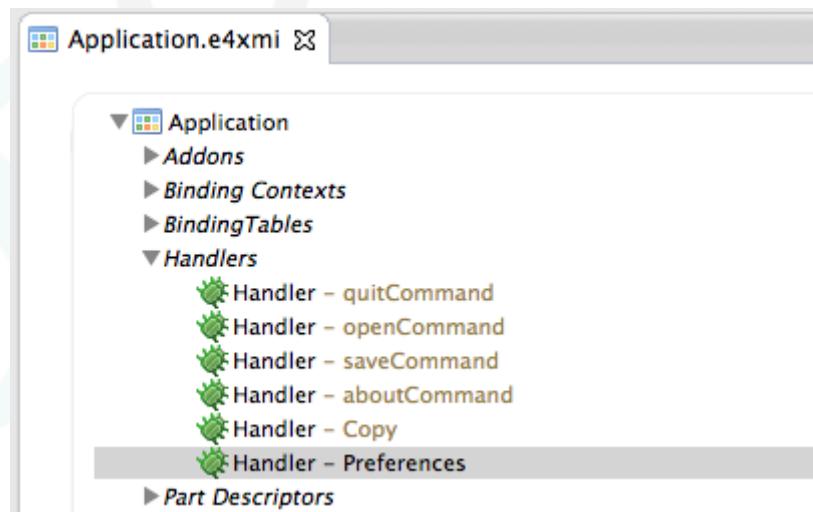


Image 154 Global Handler

- ou localement à l'endroit de son utilisation :

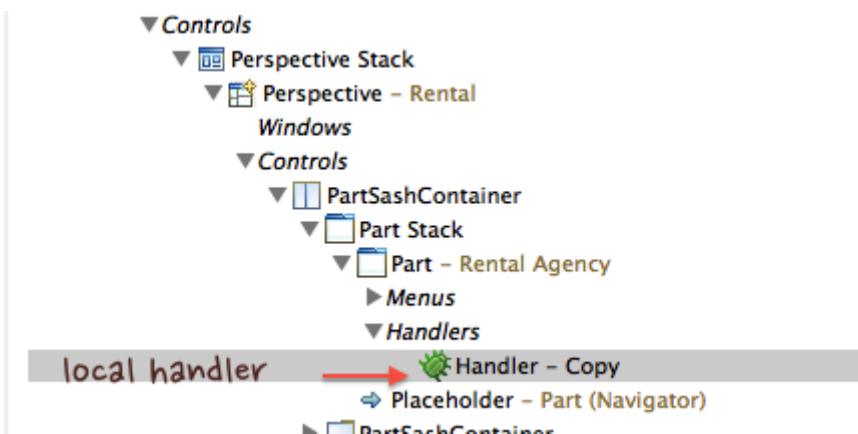
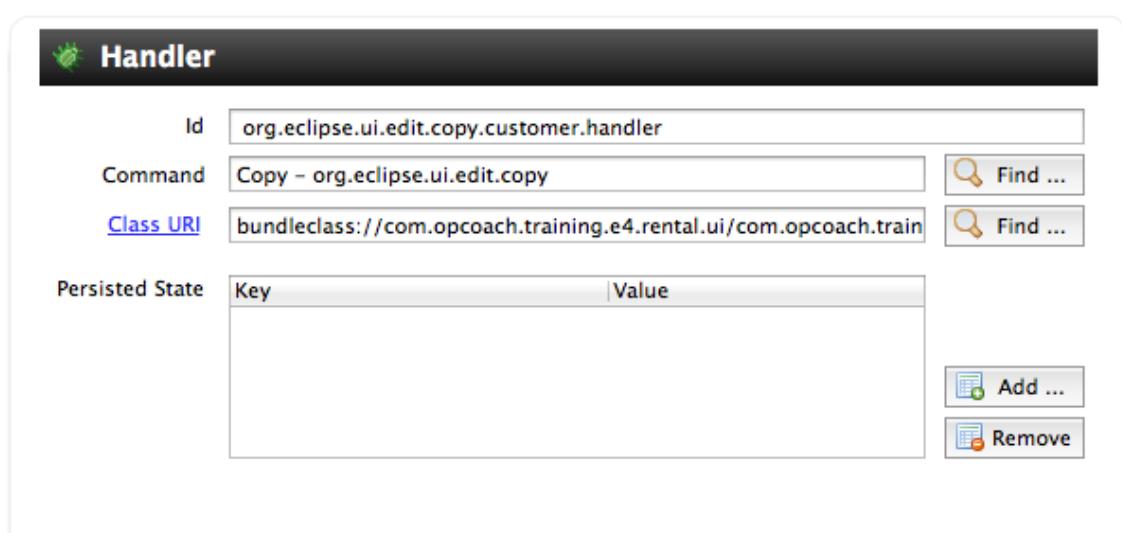


Image 155 Local Handler

Paramètres du handler

Le handler se définit par les paramètres suivants :



Handler					
Id	<input type="text" value="org.eclipse.ui.edit.copy.customer.handler"/>				
Command	<input type="text" value="Copy - org.eclipse.ui.edit.copy"/> Find ...				
Class URI	<input type="text" value="bundleclass://com.opcoach.training.e4.rental.ui/com.opcoach.train"/> Find ...				
Persisted State	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">Key</th> <th style="width: 90%;">Value</th> </tr> </thead> <tbody> <tr> <td colspan="2" style="height: 40px;"></td> </tr> </tbody> </table> Add ... Remove	Key	Value		
Key	Value				

Image 156 Handler definition

Code de handler

Le code du handler est un Pojo annoté utilisant les deux annotations :

- **@Execute** : contient le code d'exécution
- **@CanExecute** : retourne 'true' si le handler est actif

Ces annotations injectent les paramètres, on passe donc ce dont on a besoin

Exemple de canExecute

On injecte le **MPart** courant en tant que '**MDirtyable**' et on teste l'état :

```
-- 30 public class SaveHandler
31 {
32     @CanExecute
33     public boolean canExecute(@Named(IServiceConstants.ACTIVE_PART) MDirtyable dirtyable)
34     {
35         if (dirtyable == null)
36         {
37             return false;
38         }
39         return dirtyable.isDirty();
40     }
41 }
```

Injection of the current active part

Image 157 @CanExecute

Exemple de execute

On peut annoter plusieurs méthodes avec **@Execute** : celle qui aura le plus de paramètres renseignées sera appelée

On peut recevoir la sélection courante avec :

- `@Optional @Named(IServiceConstants.ACTIVE_SELECTION)`

```
-- 20 public class AboutHandler
21 {
22     @Execute
23     public void execute(@Named(IServiceConstants.ACTIVE_SELECTION) Shell shell)
24     {
25         MessageDialog.openInformation(shell, "About", "e4 Application example.");
26     }
27 }
28 
```

Active shell injection

Image 158 Execute sample



Attention: Réinjection des fields dans les handlers

Attention pour des raisons internes, les champs injectés dans une classe de handler ne sont pas réinjectés s'ils changent dans le contexte.

Cf : [http://wiki.eclipse.org/Eclipse4/RCP/FAQ#Why_aren.27t_my_handler_fields_being_re-injected.3F³⁵](http://wiki.eclipse.org/Eclipse4/RCP/FAQ#Why_aren.27t_my_handler_fields_being_re-injected.3F)

Mise en place dans les menus

La commande s'installe en utilisant soit un :

- **HandledMenuItem** : si on veut laisser le système retrouver le **Handler** à exécuter
- **DirectMenuItem** : si on veut explicitement appeler un **Handler** connu
- **DynamicMenuItem** : si on veut remplir le menu dynamiquement

35 - http://wiki.eclipse.org/Eclipse4/RCP/FAQ#Why_aren.27t_my_handler_fields_being_re-injected.3F

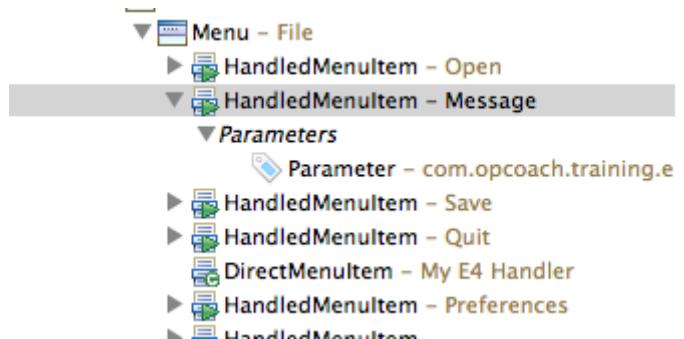


Image 159 HandledMenuItem

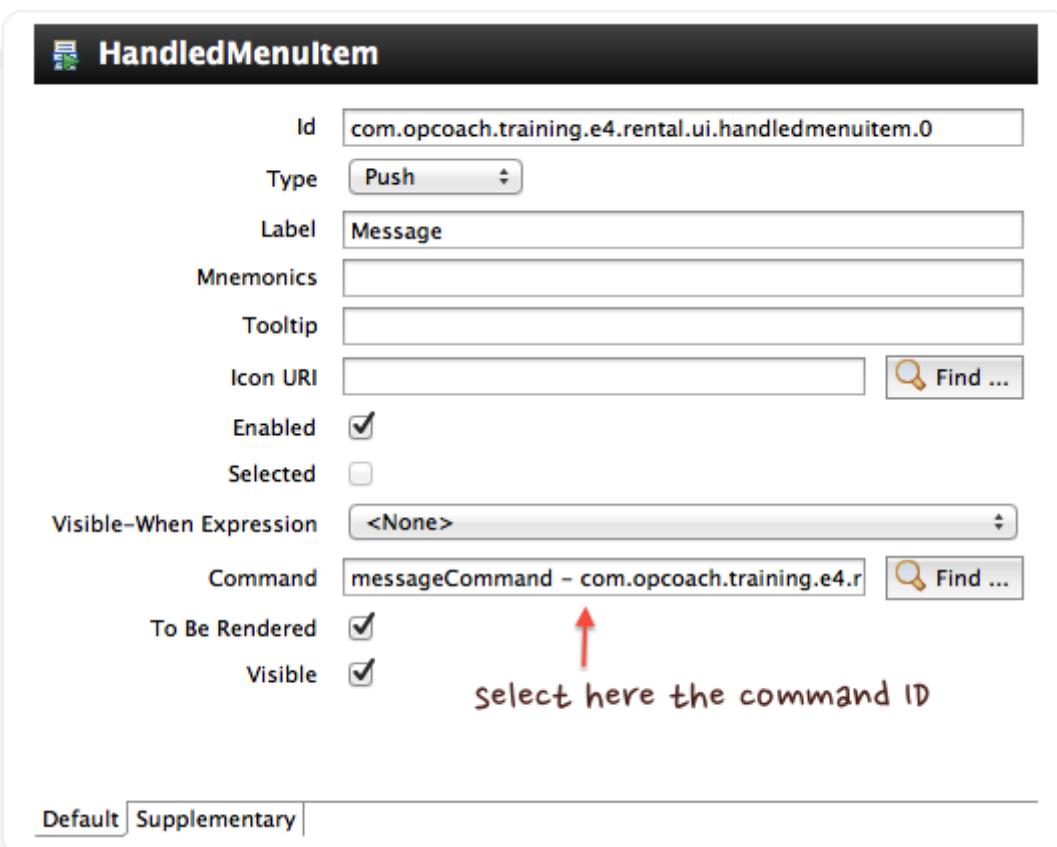


Image 160 HandledMenuItem Parameters

Paramètre de commande

La commande peut définir un paramètre qui sera passé à l'utilisation du handler



Image 161 Command Parameter

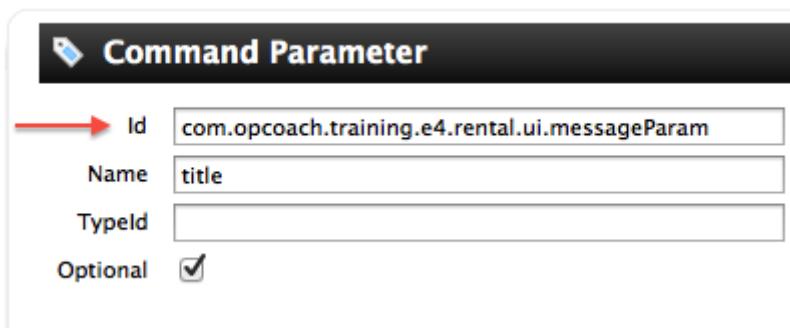


Image 162 Parameter Def

Utilisation du paramètre dans le handledItem

Le **HandledItem** ou le **DirectMenuItem** peut alors recevoir un paramètre :

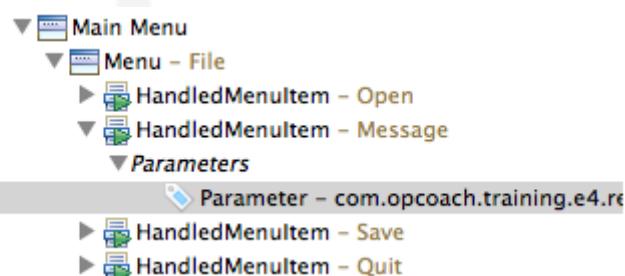


Image 163 Parameter

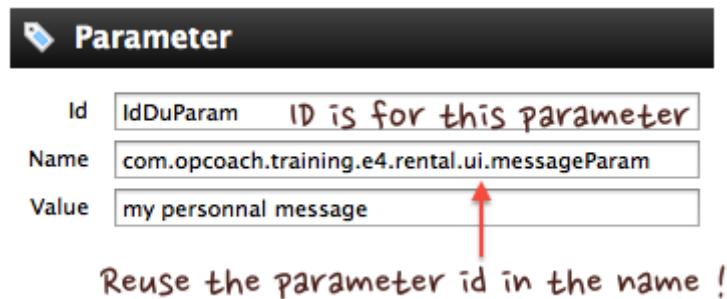


Image 164 HandledItem Parameter

Paramètre dans le handler

Le handler peut récupérer la valeur du paramètre par injection :

```

11
12 public class MessageHandler
13 {
14     @Execute
15     public Object execute(@Named("com.opcoach.training.e4.rental.ui.messageParam") String title,
16                           @Named(IServiceConstants.ACTIVE_SHELL) Shell shell) throws ExecutionException
17     {
18         MessageDialog.openInformation(shell, "Message", "Title : " + title);
19         return null;
20     }
21
22     @CanExecute
23     public boolean canExecute() { return true; }
24
25 }
26

```

Inject the parameter

Image 165 Handler using parameter

Ajout de menus, sous menus sur des parts existants

Il suffit tout simplement de les ajouter dans le modèle d'application



Image 166 popup menu

- Et ne pas oublier d'informer le menu service !
- On reçoit le menuService de type **EMenuService** en injection
- Pour utiliser le **EMenuService** il faut dépendre de **org.eclipse.e4.ui.services**

```

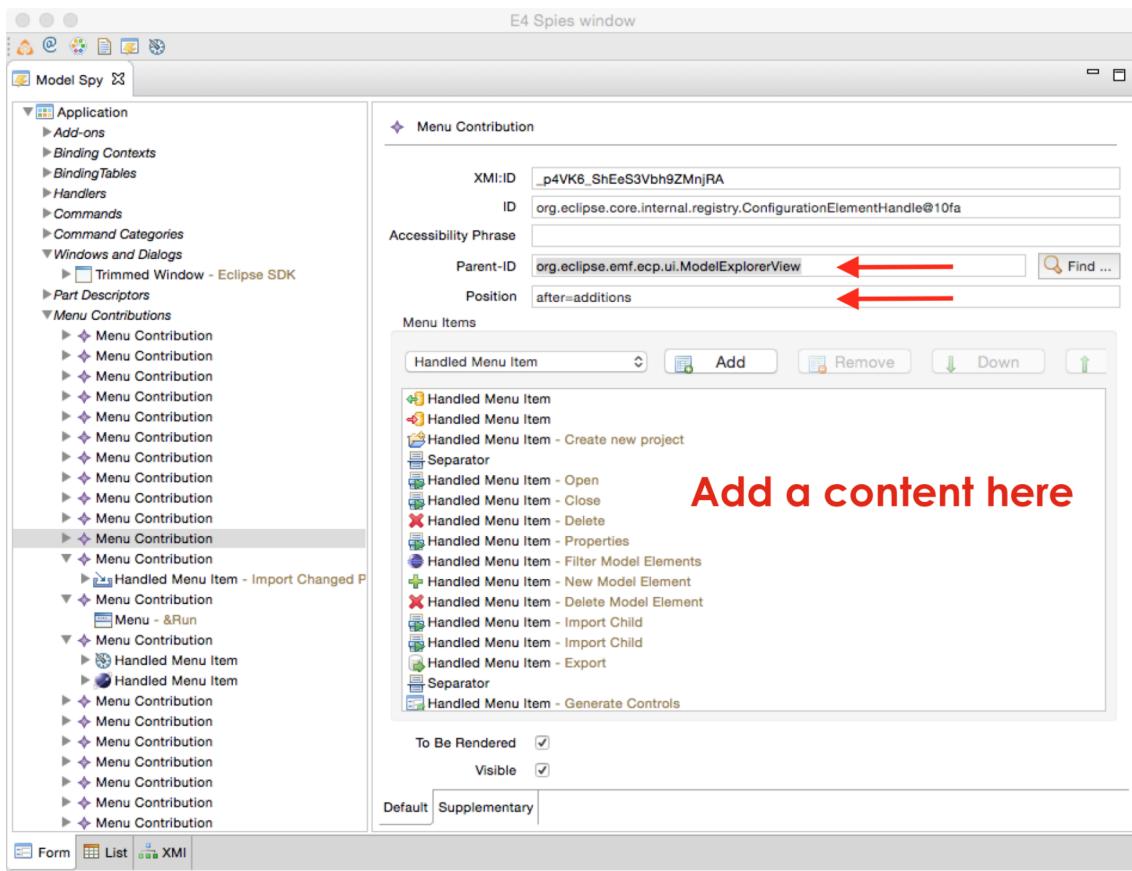
68
69 // Register a popup menu on viewer (MENU_ID is the id of popupmenu in
70 // application model)
71 menuService.registerContextMenu(agencyViewer.getControl(), MENU_ID);
72

```

Image 167 Menu Service Usage

Ajout de menu dans les contributions

- Il est également possible de rajouter un menu ou une commande de manière indirecte
- Il faut utiliser une menu/toolbar contribution
- Le lien se fait par l'ID du parent



Menu Contribution

MenuContribution / Paramètres

On peut utiliser comme parent ID :

- l'ID d'une vue existante (elle doit avoir informé le **EMenuService**)
- l'ID d'un menu existant
- **org.eclipse.ui.main.menu** : pour le menu principal
- **popup** : pour accéder à n'importe quelle menu contextuel
- **org.eclipse.ui.main.toolbar** : pour la toolbar principale dans une toolbar contribution

Pour la position :

- n'importe quel ID d'élément déjà présent (command, menu, etc...)
- **after=additions** : se place par défaut

On peut ouvrir le model Spy sur l'IDE Eclipse pour consulter les valeurs déjà utilisées.

Gérer le contenu d'un menu dynamiquement

- Le modèle d'application permet de définir les menu statiquement.
- Comment remplir un menu dynamiquement selon des conditions ?
- Il faut utiliser un **DynamicMenuContribution** :

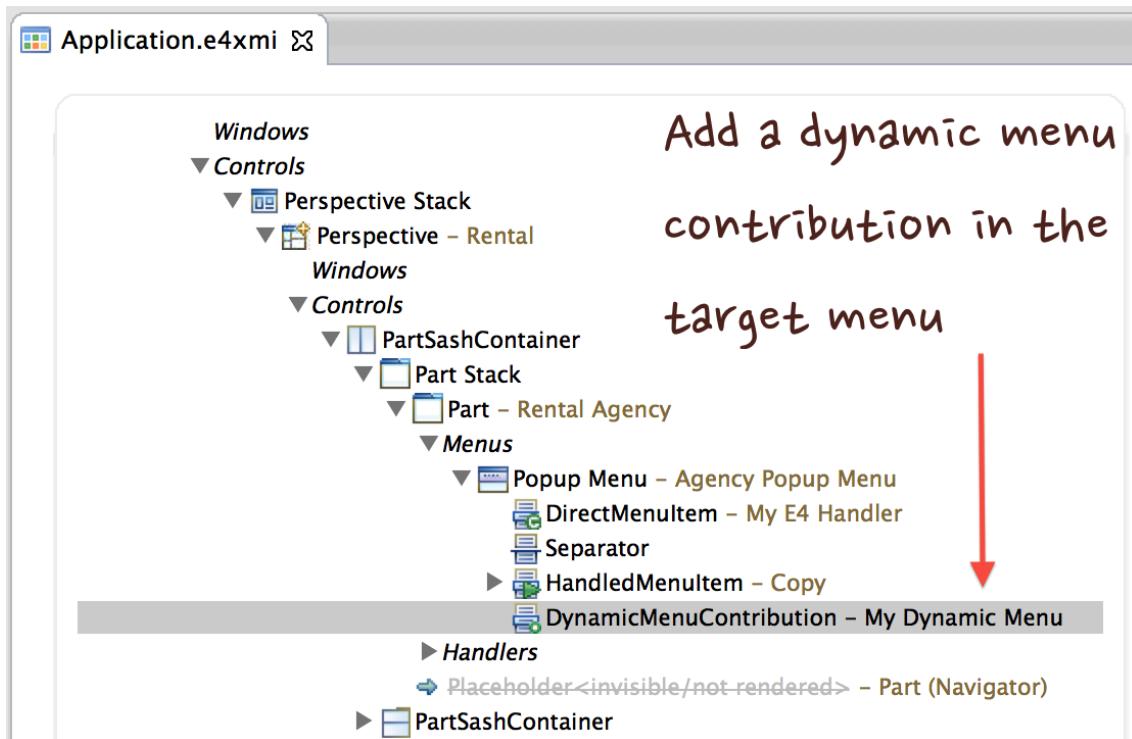


Image 168 Dynamic Item in Menu

- Avec en paramètre un POJO contenant une méthode annotée avec **@AboutToShow**
- Et optionnellement une méthode annotée avec **@AboutToHide**

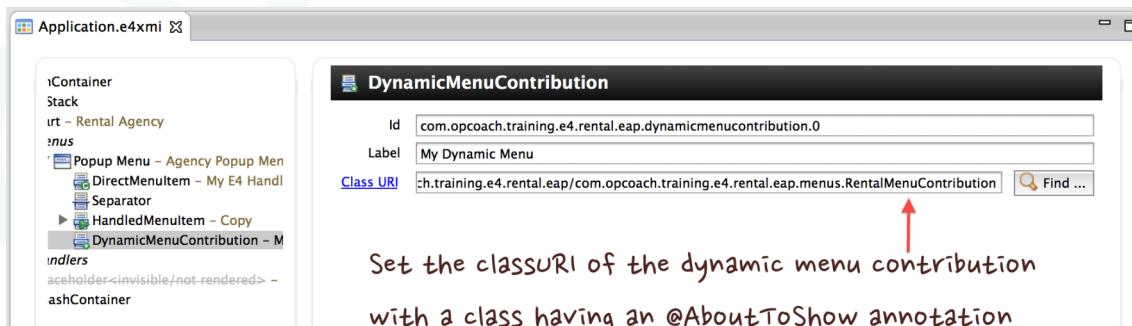


Image 169 Dynamic Menu Contribution parameters

- Le dynamicMenuContribution est remplacée par la liste d'items reçue par injection :

```

11 public class RentalMenuContribution
12 {
13     /**
14      * @param items an empty list of MenuElement to fill
15      * @param modelService the model service used to create elements
16      */
17     @AboutToShow ← Add an @AboutToShow
18     public void aboutToShow(List<MMenusElement> items, EModelService modelService)
19     {
20         String bc = "bundleclass://com.opcoach.training.e4.rental.eap/com.opcoach.training.e4.rental.eap.handlers.TestHandler";
21         String bundle = "platform:/plugin/com.opcoach.training.e4.rental.eap";
22
23         MDirectMenuItem directItem = modelService.createModelElement(MDirectMenuItem.class);
24         directItem.setLabel("Item added with aboutToShow");
25         directItem.setContributionURI(bc);
26         directItem.setContributorURI(bundle);
27
28         items.add(directItem);
29     }
30
31     /**
32      * An optional method called when the menu disappears
33      * @param items the list of MMELEMENT initialized by aboutToShow
34      */
35     @AboutToHide
36     public void aboutToHide(List<MMenusElement> items) ← Add an @AboutToHide method only if
37     {
38         // Nothing special to do with the items list. It will be destroyed automatically
39         // Use this method to dispose internal data computed in aboutToShow
40         // This method with this annotation is optional.
41     }
42 }

```

create DirectMenuItem(s) with valid URI(s)
and add them in the items list

you need to dispose internal data

Image 170 Class with @AboutToShow

Exercice EAP 070

Création de commandes

C. Extension Expression definition

Expressions

org.eclipse.core.expressions.definitions

- Pour gérer une condition de visibilité d'un menu ou d'une commande
- On peut déclarer des expressions réutilisables
 - large panel d'opérateurs (not, and, or, instanceof...)
 - de variables (activePart, activeEditor...)
 - accès aux propriétés des objets testés (noms fichiers, projet ouvert ? ...)
- Il faut dépendre de ***org.eclipse.core.expressions***
- ... extensible !
- voir aussi :
 - [http://wiki.eclipse.org/Command_Core_Expressions³⁶](http://wiki.eclipse.org/Command_Core_Expressions)

36 - http://wiki.eclipse.org/Command_Core_Expressions

Expression, choix d'opérations

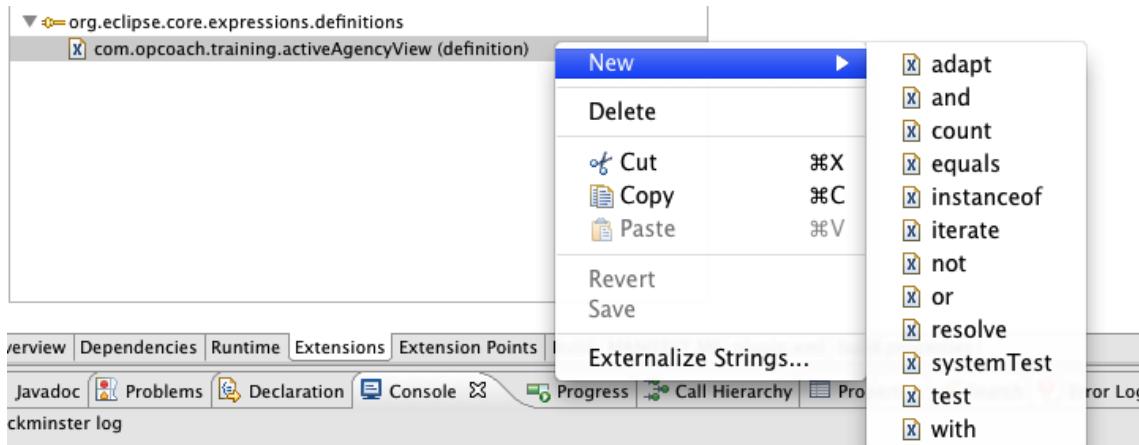


Image 171 Expression

Variables disponibles dans les expressions.

- Le framework définit des variables (activePartId,...) qui sont surtout valables pour Eclipse 3
- Elles n'ont plus de sens en Eclipse 4
- On utilisera alors uniquement la sélection courante, qui est implicite
- On peut aussi utiliser toute variable présente dans le contexte d'injection

Propriétés accessibles pour les tests

- La sélection courante ou certaines variables redéfinies peuvent être de n'importe quel type.
- On peut alors définir un 'propertyTester' qui permet d'extraire une valeur de l'instance courante
- Il faut étendre le point d'extension : [org.eclipse.core.expressions.propertyTesters](#)

Exemple, pour les ressources :

namespace (en gras) et type	propriété(s)
org.eclipse.core.runtime .Platform	product (id of current active product)
org.eclipse.core.resources .IResource	name, path, extension, projectNature, persistentProperty (name, expected value)...
org.eclipse.core.resources .IProject	open

Tableau 1 Principales propriétés et types d'application

```

29      <definition id="com.opcoach.training.e4.codesamples.txtFileSelected">
30          <adapt type="org.eclipse.core.resources.IFile">
31              <test property="org.eclipse.core.resources.extension" value="txt"/>
32          </adapt>
33      </definition>
34

```

propertyTester

Exemples

- Vérifier si la sélection courante (simple) est de type Customer :

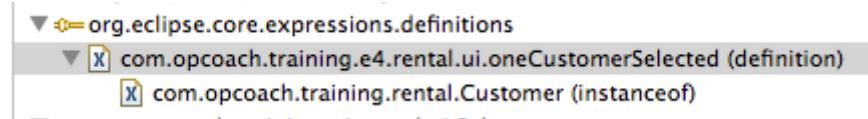


Image 172 Customer selected

- Pour la sélection multiple, il faut que le type de l'objet reçu implémente **List**
- Il faut donc retourner des List dans les sélections pour pouvoir itérer dans les expressions

Sélection multiple de fichiers

Tests sur les sélections de fichier

Attention : cet exemple n'est valable qu'en E3 à cause de la sélection multiple (ou faire un **Iterable**) :

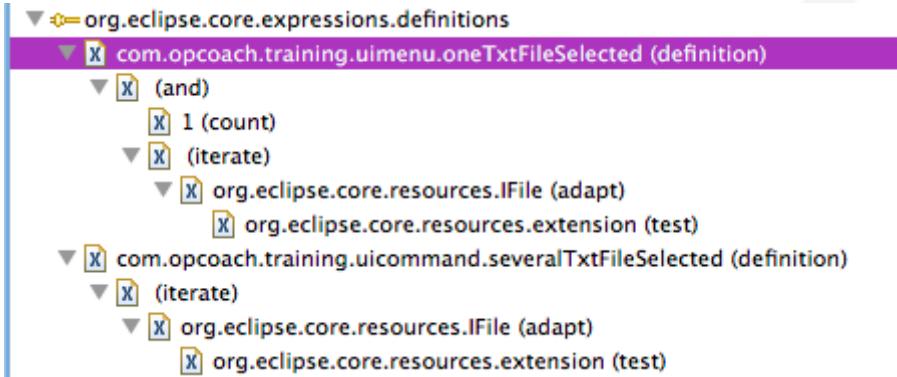


Image 173 Selection number test

Combinaison d'expression

On peut utiliser les opérateurs booléens pour combiner les expressions

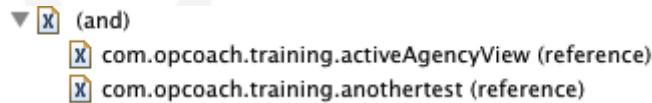
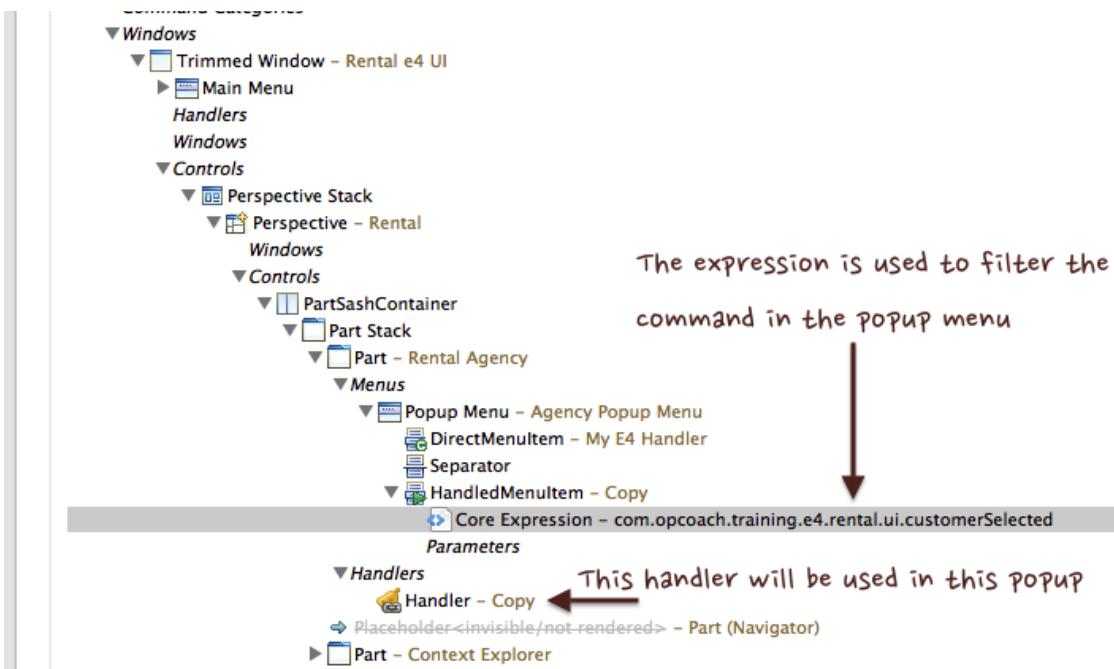


Image 174 Expression references

Utiliser l'expression definition dans le Model d'application (Eclipse 4)

On peut ajouter l'expression definition simplement sous un HandledItem :



Command filter

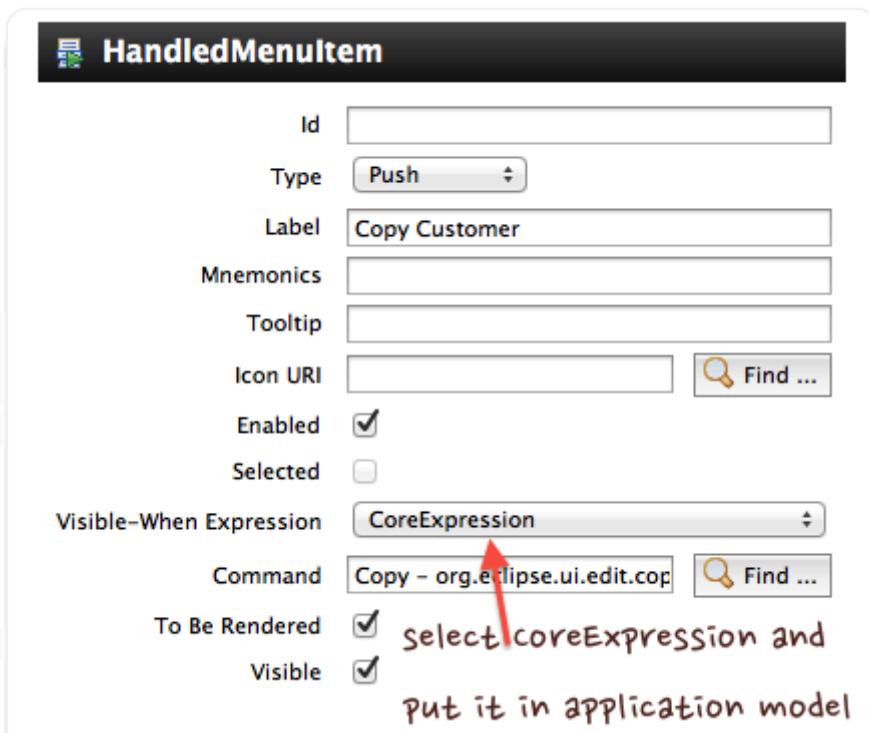


Image 175 Expression HandledItem



Conseil : : utilisation pour les commandes

- Donner des Id clairs aux commandes et aux expressions
- Séparer les tests dans des expressions
- Définir les choses communes dans des plugins partagés
- Créer des sous menus pour regrouper les commandes

Exercice EAP 071

Commandes conditionnelles

D. Gestions des Opérations

Présentation

- Les opérations sont utilisées pour les mécanismes undo/redo
- Une opération doit implémenter la méthode execute(...), redo(...), undo(...)
- Elle doit être ensuite enregistrée sur l'operation History qui l'exécutera

```

1 package com.opcoach.training;
2
3 import org.eclipse.core.commands.ExecutionException;
4
5 public class SepiaFilterOperation extends AbstractOperation
6 {
7     private ImageRegistry registry;
8     private Image source;
9     private Image filtered;
10    private String imageKey;
11
12    public SepiaFilterOperation(ImageRegistry reg, String imKey)
13    {
14        super("Sepia Filter Operation");
15        registry = reg;
16        imageKey = imKey;
17        source = registry.get(imageKey);
18    }
19
20    public IStatus execute(IProgressMonitor monitor, IAdaptable info) throws ExecutionException
21    {
22        SepiaFilter filter = new SepiaFilter();
23        filtered = filter.filter(source, monitor);
24        registry.put(imageKey, filtered);
25        return Status.OK_STATUS;
26    }
27
28    public IStatus redo(IProgressMonitor monitor, IAdaptable info) throws ExecutionException
29    {
30        // Restore the filtered image in registry
31        registry.put(imageKey, filtered);
32        return Status.OK_STATUS;
33    }
34
35    public IStatus undo(IProgressMonitor monitor, IAdaptable info) throws ExecutionException
36    {
37        // Restore the source image in registry
38        registry.put(imageKey, source);
39        return Status.OK_STATUS;
40    }
41
42}
43
44
45
46
47
48
49

```

Image 176 Operation Sample

Enregistrement sur l'historique

```

11 public class UndoableHandler
12 {
13     @Execute
14     public void execute(IOperationHistory opHist) throws ExecutionException
15     {
16         // Create an operation
17         IUndoableOperation op = new SepiaFilterOperation(null, null);
18         // Execute it using history
19         IProgressMonitor pm = new NullProgressMonitor();
20         opHist.execute(op, pm, null);
21     }
22
23     @CanExecute
24     public boolean canExecute()
25     {
26         return true;
27     }
28
29 }
30

```

Inject the operation History

Image 177 Undoable handler

Ce code est donné sous réserve (non vérifié avec E4)

E. Gestion des wizards

Wizards

- Les wizards sont une suite de pages qui aident à créer, exporter ou importer des objets.
- Ils sont gérés par JFace
- Chaque wizard calcule à tout moment son état de validation
- Chaque wizard possède une action finale

Fonctionnement du wizard

- Le fonctionnement est pris en charge par le **WizardDialog**
- On ne doit fournir que les instances de **IWizardPage** et le **IWizard**

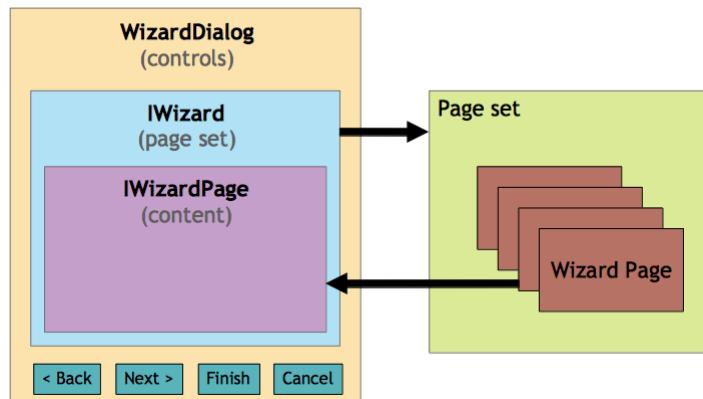
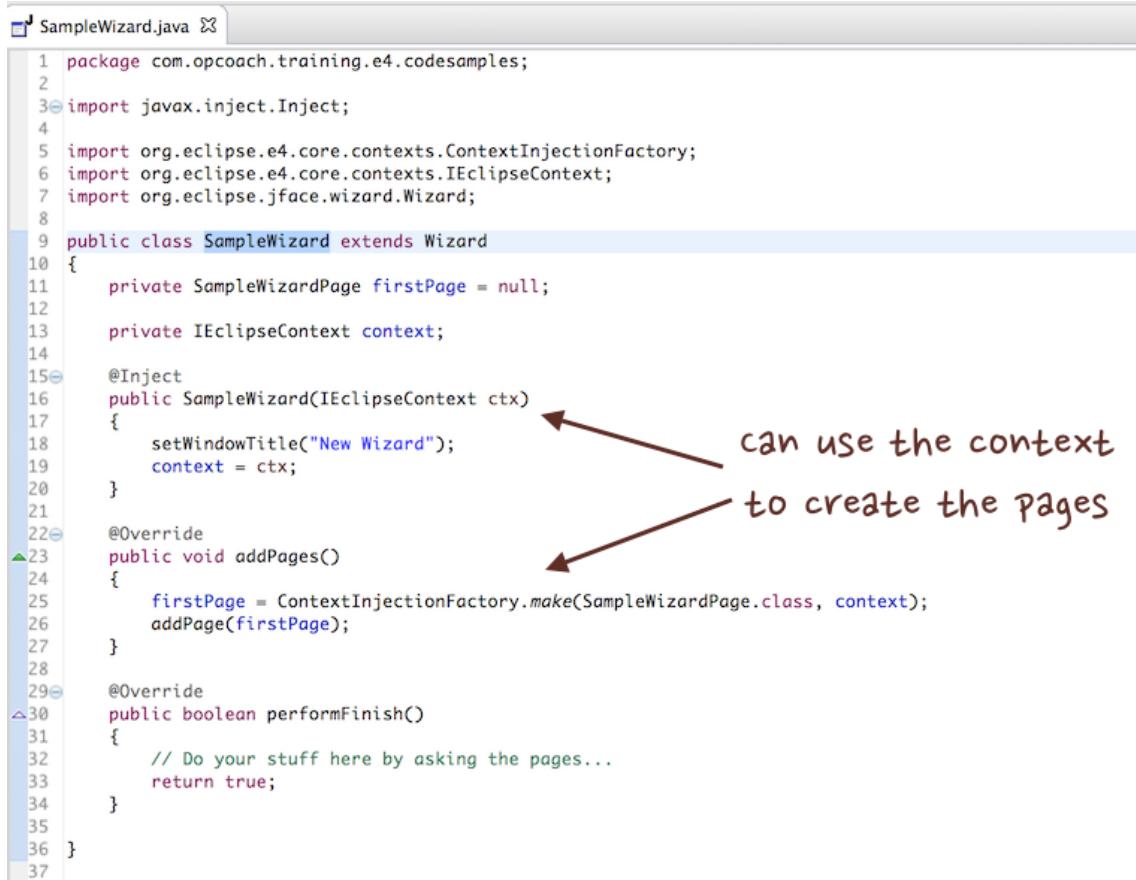


Image 178 Wizard operation

Implémentation d'un Wizard

- On dérive de `org.eclipse.jface.wizard.Wizard`
- On implémente également `addPages()` et `performFinish()`
- En Eclipse 4 il ne faut plus implémenter les interfaces `INewWizard`, `IImportWizard` ... (présentes dans `org.eclipse.ui`).

Exemple :



```

1 package com.opcoach.training.e4.codesamples;
2
3 import javax.inject.Inject;
4
5 import org.eclipse.e4.core.contexts.ContextInjectionFactory;
6 import org.eclipse.e4.core.contexts.IEclipseContext;
7 import org.eclipse.jface.wizard.Wizard;
8
9 public class SampleWizard extends Wizard
10 {
11     private SampleWizardPage firstPage = null;
12
13     private IEclipseContext context;
14
15     @Inject
16     public SampleWizard(IEclipseContext ctx)
17     {
18         setWindowTitle("New Wizard");
19         context = ctx;
20     }
21
22     @Override
23     public void addPages()
24     {
25         firstPage = ContextInjectionFactory.make(SampleWizardPage.class, context);
26         addPage(firstPage);
27     }
28
29     @Override
30     public boolean performFinish()
31     {
32         // Do your stuff here by asking the pages...
33         return true;
34     }
35
36 }
37

```

Annotations on the code:

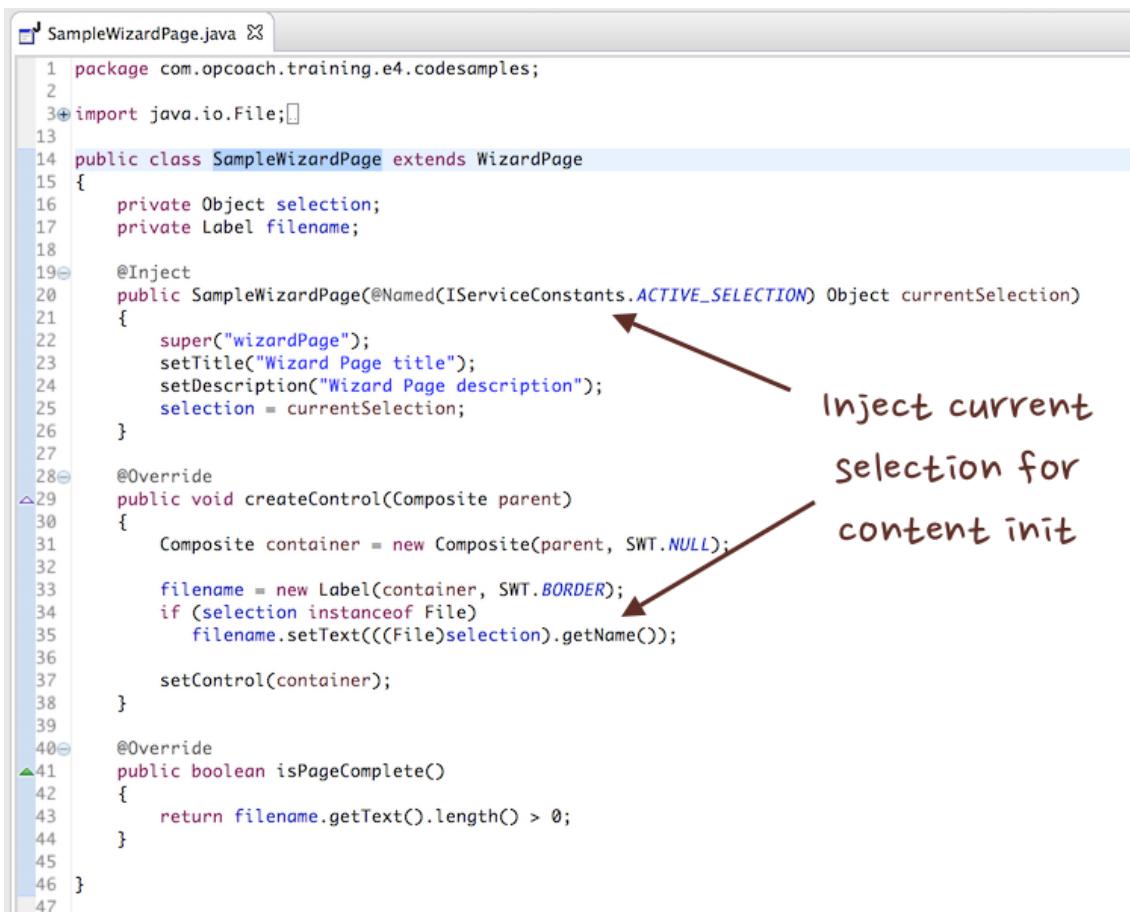
- An arrow points from the `context` field to the `setWindowTitle` call with the text "can use the context".
- An arrow points from the `context` field to the `make` call in `addPages` with the text "to create the pages".

Image 179 Sample wizard

Implémentation d'une page de wizard

On implémente l'interface `IWizardPage` avec :

- `createControl`(Composite parent)
- `handleEvent`(swt.Event), si implémentation de swt.Listener
- `isPageComplete`()...



```

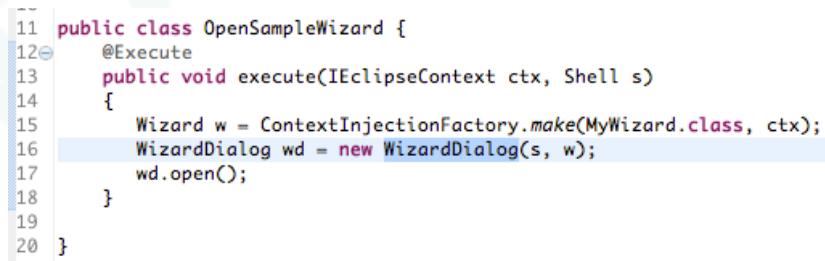
1 package com.opcoach.training.e4.codesamples;
2
3+ import java.io.File;
4
5 public class SampleWizardPage extends WizardPage
6 {
7     private Object selection;
8     private Label filename;
9
10    @Inject
11    public SampleWizardPage(@Named(IServiceConstants.ACTIVE_SELECTION) Object currentSelection)
12    {
13        super("wizardPage");
14        setTitle("Wizard Page title");
15        setDescription("Wizard Page description");
16        selection = currentSelection;
17    }
18
19    @Override
20    public void createControl(Composite parent)
21    {
22        Composite container = new Composite(parent, SWT.NULL);
23
24        filename = new Label(container, SWT.BORDER);
25        if (selection instanceof File)
26            filename.setText(((File)selection).getName());
27
28        setControl(container);
29    }
30
31    @Override
32    public boolean isPageComplete()
33    {
34        return filename.getText().length() > 0;
35    }
36
37 }
38
39
40
41 }
42
43
44
45
46 }
47

```

Image 180 Sample wizard page

Ouverture d'un wizard

- En Eclipse 4 on ne dispose pas des points d'extensions de wizards
- Il faut créer le wizard à l'endroit voulu puis l'ouvrir simplement
- On peut instancer le wizard en utilisant l'injection



```

11 public class OpenSampleWizard {
12     @Execute
13     public void execute(IEclipseContext ctx, Shell s)
14     {
15         Wizard w = ContextInjectionFactory.make(MyWizard.class, ctx);
16         WizardDialog wd = new WizardDialog(s, w);
17         wd.open();
18     }
19
20 }

```

Open wizard

F. Gestion des préférences

Remarque préliminaire

- La gestion des pages de préférence n'est pas finalisée à ce jour, mais les mécanismes internes fonctionnent
- Cette présentation montre une manière de récupérer les préférences E3 dans E4

Le mécanisme de gestion des préférences

- Les préférences permettent de stocker des configurations de paramètres pour une application
- Les valeurs sont stockées dans le répertoire de runtime :
 - .metadata/.plugins/org.eclipse.core.runtime/.settings/nom du plugin.prefs
 - sous la forme clé=valeur :
 - prefColor=238,80,42
 - on peut changer l'emplacement de stockage avec le paramètre -data
- On accède aux valeurs via les classes **IEclipsePreferences**, **IPreferenceNode**

Accéder aux préférences

- Pour accéder à une valeur il faut récupérer l'instance de **IEclipsePreferences** du plugin correspondant.
- Il faut ajouter la dépendance sur **org.eclipse.e4.core.di.extensions** pour avoir l'annotation **@Preference**
- On peut alors manipuler les préférences en get/set :

```

14
15
16 Don't forget the @Inject !
17 @Inject
18 public void updateValues(@Preference(nodePath="YOUR PLUGIN ID") IEclipsePreferences prefs)
19 {
20     // Getting values
21     prefs.getBoolean(PREF_OPTIM, false); // Get the optim preference, false by default
22     prefs.getInt(PREF_NB_ITER, 50); // Get the nb iter preference, 50 by default
23
24     // Setting values
25     prefs.putDouble(PREF_ANGLE, Math.PI/2); // set value for angle.
26
27 }
28

```

Image 181 IEclipsePreferences

Accéder à une valeur de préférence

- On peut aussi recevoir directement une valeur par injection
- La méthode est alors réinvoquée automatiquement si la préférence change
- Le paramètre 'nodePath' est optionnel et vaut par défaut l'id du plugin courant.

```
28  
29  
30 @Inject @Optional  
31 public void manageOptim(@Preference(nodePath="YOUR PLUGIN ID", value=PREF_OPTIM) boolean optim)  
32 {  
33     // preference value changed  
34     System.out.println("Optimization preference changed : " + optim);  
35 }  
36  
37
```

Image 182 Inject preference value

La gestion des pages de préférences

JFace propose un système centralisé de gestion des pages de préférences :

- IHM normalisée
 - Stockage automatique ([IPreferenceStore](#)) géré via chaque plugin
 - Classe [FieldEditorPreferencePage](#) simplifiant la création

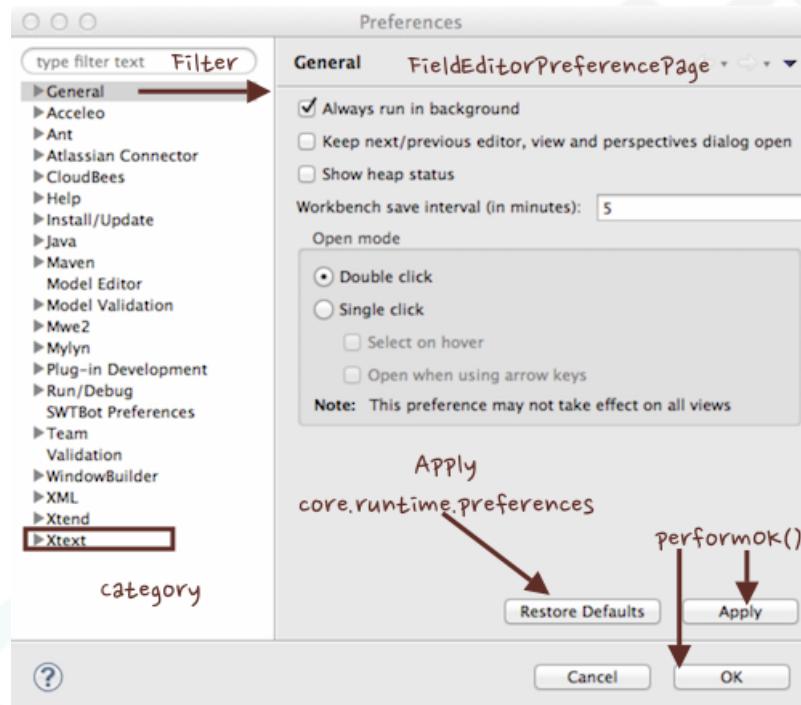


Image 183 Preference dialog

Gestion du dialogue.

- Le dialogue de Jface s'initialise avec un **PreferenceManager**
 - Le **PreferenceManager** contient la hierarchie de pages à afficher
 - En Eclipse 3 cet objet est initialisé avec le point d'extension **org.eclipse.ui.preferencePages**
 - En Eclipse 4 on ne peut plus utiliser **org.eclipse.ui**
 - Il faut gérer le **PreferenceManager** soi même !
 - Eclipse Luna devrait proposer une solution dans le cadre de la gestion des dialogues qui seront dans l'application model.

Le plugin OPCoach de gestion de préférences

- Le point d'extension de Eclipse 3 attend une instance de **IWorkbenchPreferencePage**
- Eclipse 4 ne fournit pas de moyen de décrire les pages de preferences
- Le plugin **com.opcoach.e4.preferences** permet de déclarer ses pages de préférences et de les afficher
- Il est sous licence EPL
- Il se télécharge à partir de github : <https://github.com/opcoach/e4Preferences>³⁷

Point d'extension com.opcoach.e4.preferences.e4PreferencePage

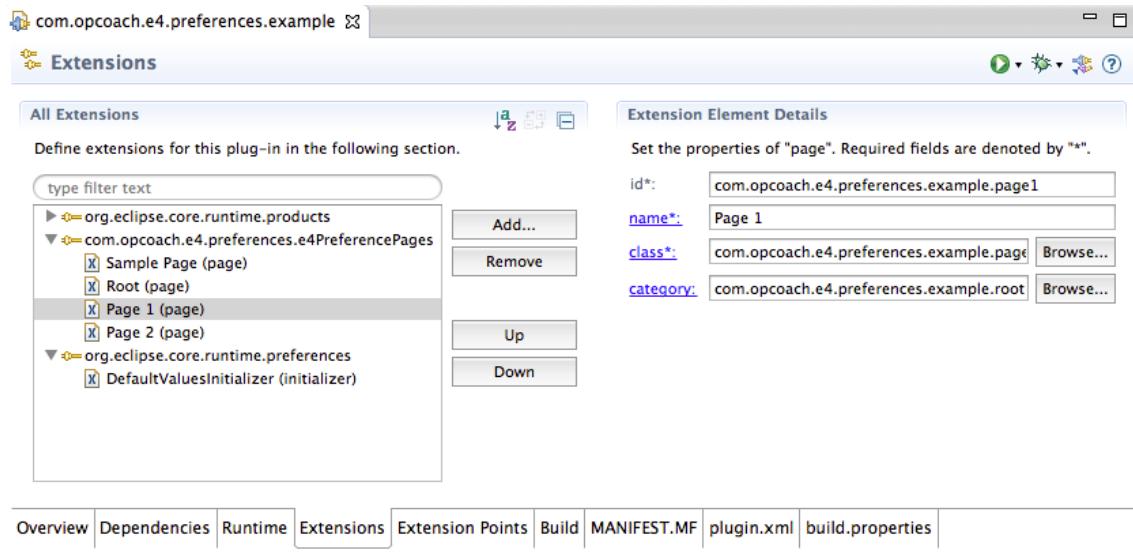


Image 184 E4 Preference page

La page de préférence indiquée dans la classe doit dériver de **FieldEditorPreferencePage**

37 - <https://github.com/opcoach/e4Preferences>

Exemple de code de page de préférence :

```

1 package com.opcoach.e4.preferences.example.pages;
2
3 import org.eclipse.jface.preference.BooleanFieldEditor;
4
5 /**
6  * A sample preference page to show how it works
7 */
8 public class SamplePage extends FieldEditorPreferencePage
9 {
10
11
12     public static final String PREF_STRING = "prefString";
13     public static final String PREF_COMBO = "prefCombo";
14     public static final String PREF_BOOLEAN = "prefBoolean";
15     public static final String PREF_COLOR = "prefColor";
16
17
18     public SamplePage()
19     {
20         super(GRID);    Set grid layout
21     }
22
23     @Override
24     protected void createFieldEditors()
25     {
26         String[][] keyAndValues = new String[][] { { "display1", "value1" }, { "display2", "value2" } };
27         addField(new ComboFieldEditor(PREF_COMBO, "A combo field", keyAndValues, getFieldEditorParent()));
28
29         addField(new ColorFieldEditor(PREF_COLOR, "Color for table items : ", getFieldEditorParent()));
30         addField(new BooleanFieldEditor(PREF_BOOLEAN, "A boolean : ", getFieldEditorParent()));
31         addField(new StringFieldEditor(PREF_STRING, "A string : ", getFieldEditorParent()));
32     }
33
34 }
35

```

extends this class (easier)

Define constants for each preference

The needed IPreferenceStore will be automatically set by the plugin

create your fields here

Image 185 E4 sample preference page

- Le stockage, le `performOK()`, le `reset()` sont gérés par le `FieldEditorPreferencePage`. Il suffit de créer des fields.
- La mise en relation avec le `IPreferenceStore` est faite par le handler du plugin

Les différents types de fields Editors

JFace fournit tout un ensemble de FieldEditors :

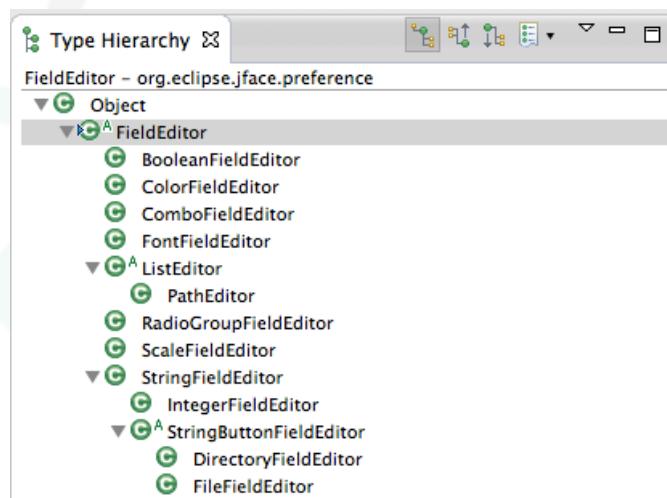


Image 186 Field Editors

Gestion du IPreferenceStore

- Le **IPreferenceStore** est initialisé par le handler d'ouverture du dialogue
- Il n'y a pas de code à produire pour l'initialiser
- Le preference Store permet d'accéder aux valeurs en gérant les valeurs par défaut de manière transparente
- C'est une interface complémentaire à **IEclipsePreferences** qui est utilisé dessous
- On peut récupérer le preference Store pour récupérer une valeur de la manière suivante :
- Le mieux est de le créer dans le code de l'addon de la feature et d'y donner accès

```

65
66
67
68      return new ScopedPreferenceStore(InstanceScope.INSTANCE, "MY PLUGIN ID");
69
70
71
72

```

Image 187 E4 Scoped Preference Store



Remarque : Origine du ScopedPreferenceStore

- La classe utilisée provient du plugin OPCoach
- C'est une copie de celle présente dans **org.eclipse.ui** que l'on ne peut pas utiliser en E4 pur !

Gestion des valeurs par défaut

Le point d'extension **org.eclipse.core.runtime.preferences** fonctionne toujours

- Permet de fixer les valeurs sans avoir ouvert la page
- Est rappelé lors du **setDefault**
- Peut être déclaré dans différents plugins selon l'OS.
- La classe fournie doit étendre **AbstractPreferenceInitializer**

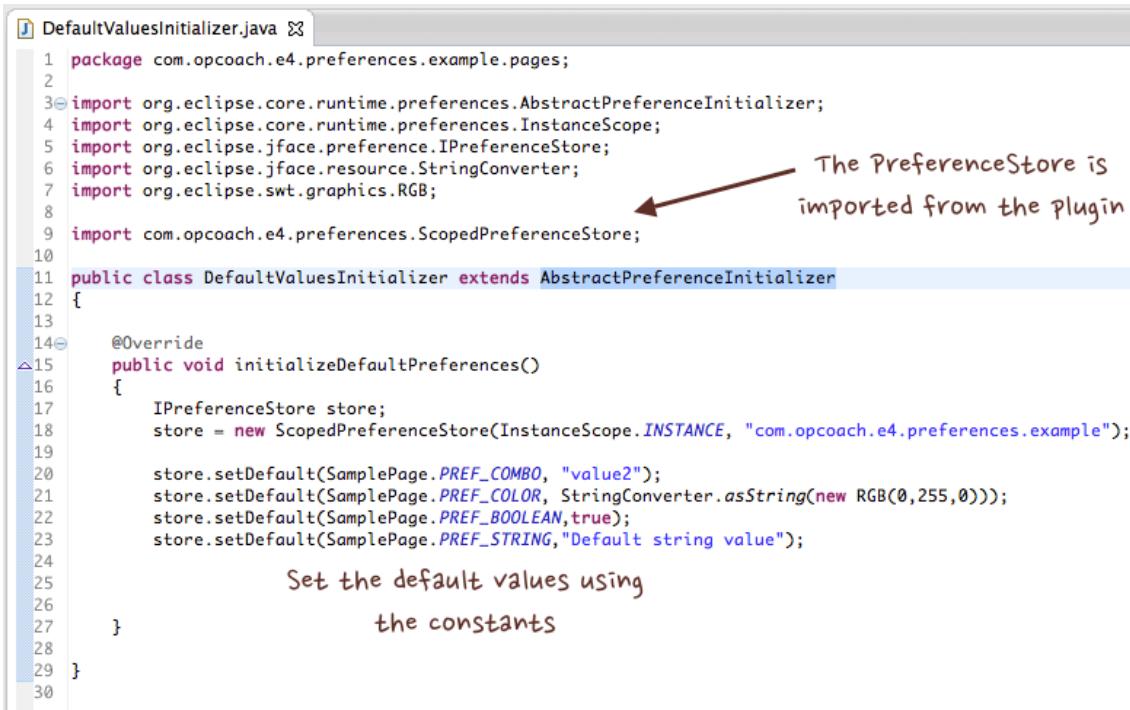
```

▼ org.eclipse.core.runtime.preferences
  └ com.opcoach.training.rental.ui.pref.RentalPrefInit (initializer)

```

Image 188 Runtime Preferences

Exemple de code pour les defaults préférences



```

1 package com.opcoach.e4.preferences.example.pages;
2
3 import org.eclipse.core.runtime.preferences.AbstractPreferenceInitializer;
4 import org.eclipse.core.runtime.preferences.InstanceScope;
5 import org.eclipse.jface.preference.IPreferenceStore;
6 import org.eclipse.jface.resource.StringConverter;
7 import org.eclipse.swt.graphics.RGB;
8
9 import com.opcoach.e4.preferences.ScopedPreferenceStore;
10
11 public class DefaultValuesInitializer extends AbstractPreferenceInitializer
12 {
13
14     @Override
15     public void initializeDefaultPreferences()
16     {
17         IPreferenceStore store;
18         store = new ScopedPreferenceStore(InstanceScope.INSTANCE, "com.opcoach.e4.preferences.example");
19
20         store.setDefault(SamplePage.PREF_COMBO, "value2");
21         store.setDefaults(SamplePage.PREF_COLOR, StringConverter.asString(new RGB(0,255,0)));
22         store.setDefault(SamplePage.PREF_BOOLEAN,true);
23         store.setDefault(SamplePage.PREF_STRING,"Default string value");
24
25             Set the default values using
26             the constants
27     }
28
29 }
30

```

Image 189 AbstractInitializer

Utilisation des préférences via le preference Store

- On interroge le preferenceStore du plugin sur chaque clé à utiliser :
- **prefStore.getString(PREF_KEY)**
- La valeur par défaut est gérée automatiquement

Attention :

- Les valeurs sont des Strings
- Il faut toujours les convertir dans le type voulu (String -> rgb par exemple...).
- La classe **StringConverter** aide à la conversion (couleurs, fontes, ...)

Intégration de la commande dans l'application

- Le plugin com.opcoach.e4.preferences fournit le handler **E4PreferencesHandler**
- Il faut l'associer à la commande ayant pour id : **org.eclipse.ui.window.preferences**

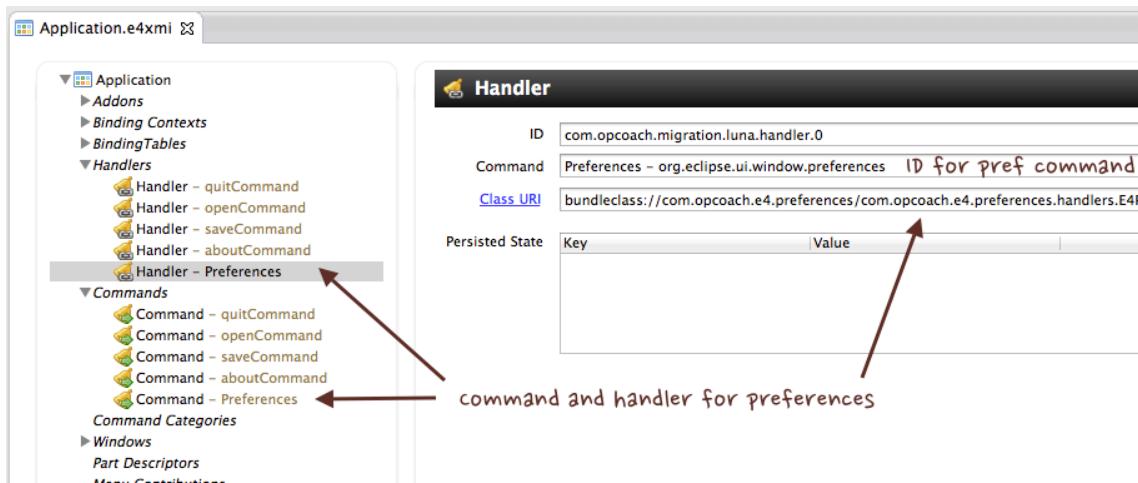


Image 190 Binding preferences in application

Exercice RCP 080

Préférences Rental

Il y a aussi beaucoup d'autres points d'extension au niveau de core
 Il ne faut pas utiliser les points d'extension [org.eclipse.ui](#) dans une application pure E4.
 On accède à leur documentation (triée par thèmes) :

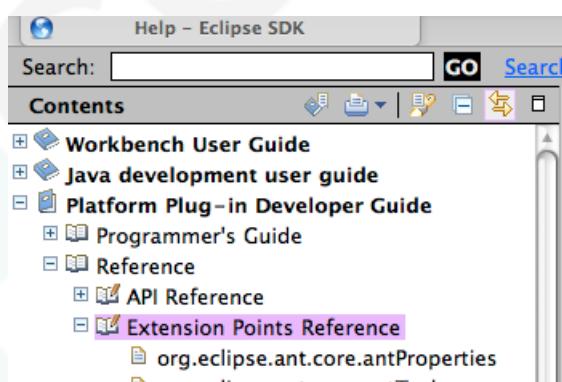


Image 191 Access to documentation

Par exemple :

- [org.eclipse.core.runtime.adapters](#) : ajout d'adapter factories.

Model Fragment et Processor



Introduction

Il est possible de contribuer à un modèle existant de deux manières :

- par un model fragment : méthode descriptive (connaitre l'ID des objets ou le XPath)
- par un processor : méthode par injection (on reçoit les objets à modifier)

Fragment de modèle

- Le fragment de modèle se comporte de manière similaire à un fragment de plugin
- Il complète un modèle d'application existant
- Une commande 'Extract Fragment' (dans le menu contextuel de l'éditeur de modèle), permet d'extraire un fragment
- Les liens se font par mise en correspondance d'ID :

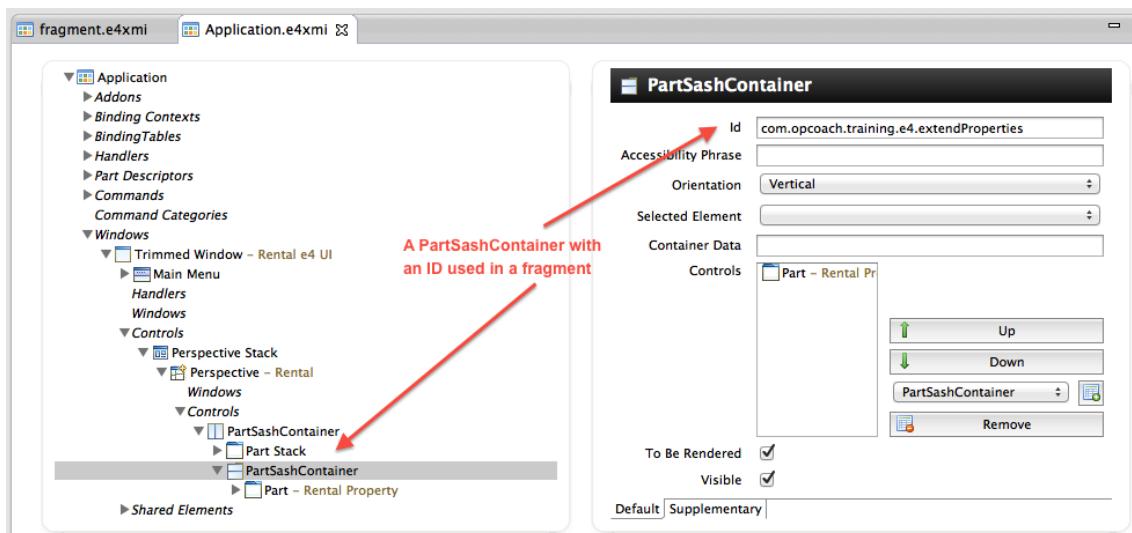


Image 192 Model sample

Fragment de modèle

Le modèle, sans le fragment, produit cette application :

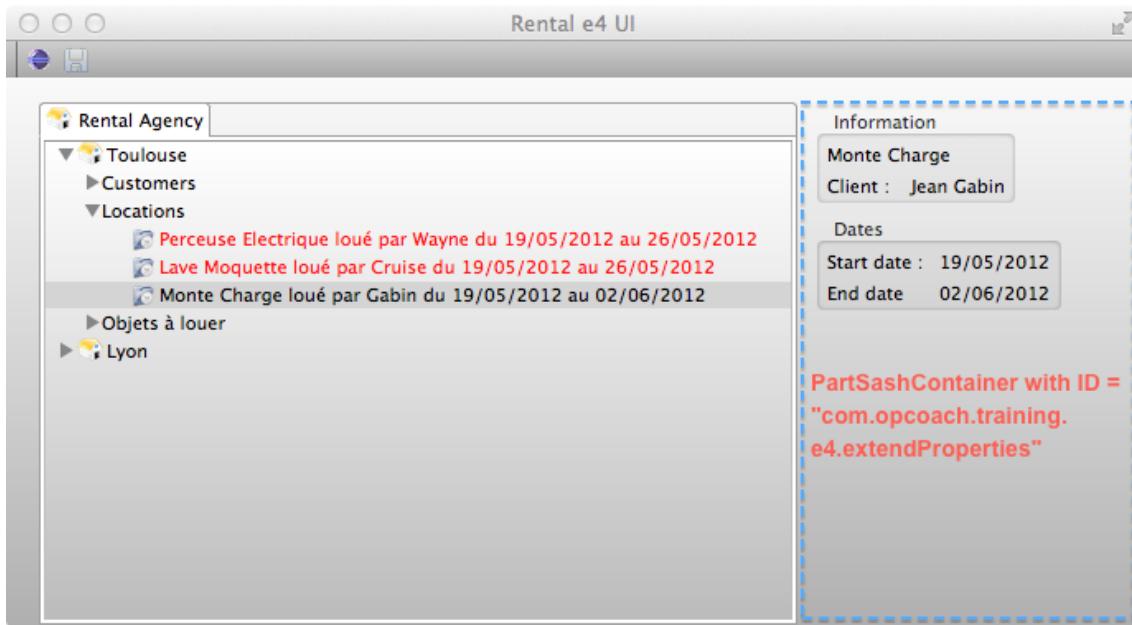


Image 193 UI without fragment

Fragment de modèle

On référence cet ID dans le fragment (créé dans un autre plugin) :

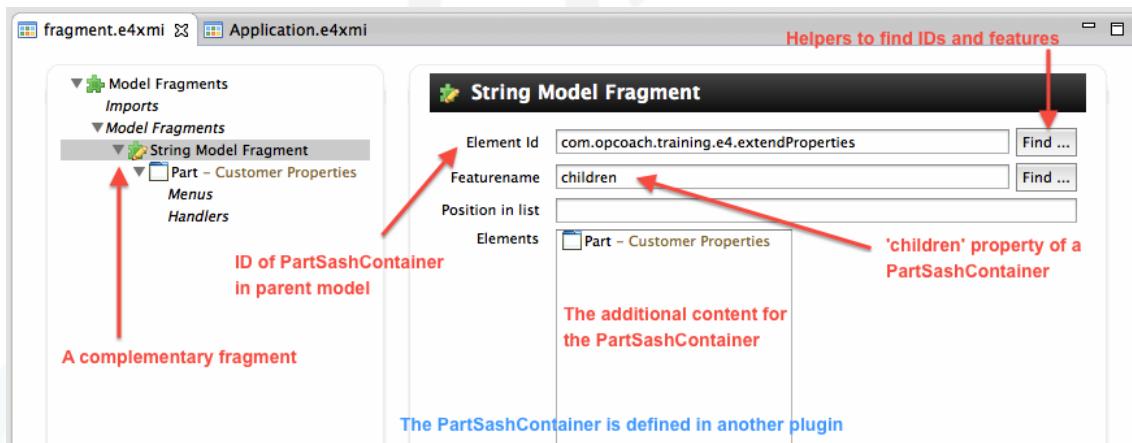


Image 194 Model Fragment

Fragment de modèle

On ajoute le fragment par point d'extension ([org.eclipse.e4.workbench.model](#))

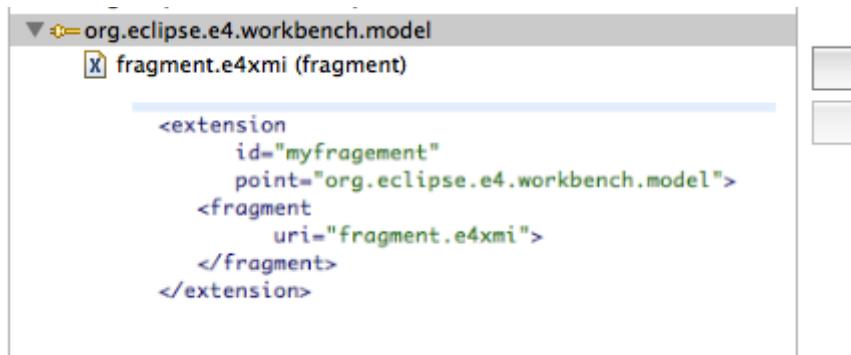


Image 195 Model Fragment

Fragment modèle résultat

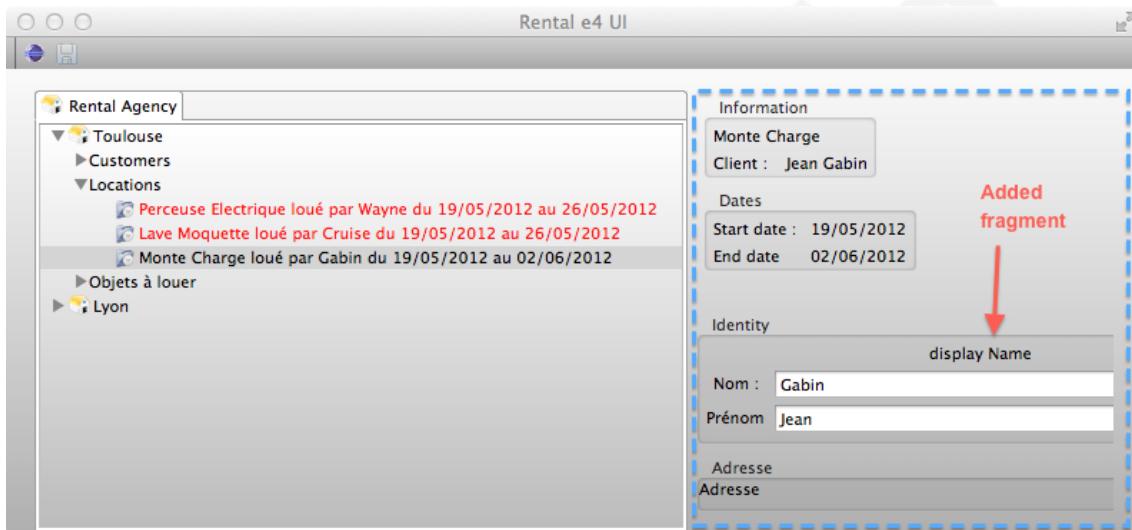


Image 196 UI with fragment

Addon dans un fragment

Il est aussi possible de définir un Addon dans un fragment

- mettre l'ID de l'application et choisir addons comme propriété.
- ou mettre l'ID de la legacy E4 application : [org.eclipse.e4.legacy.ide.application](#)

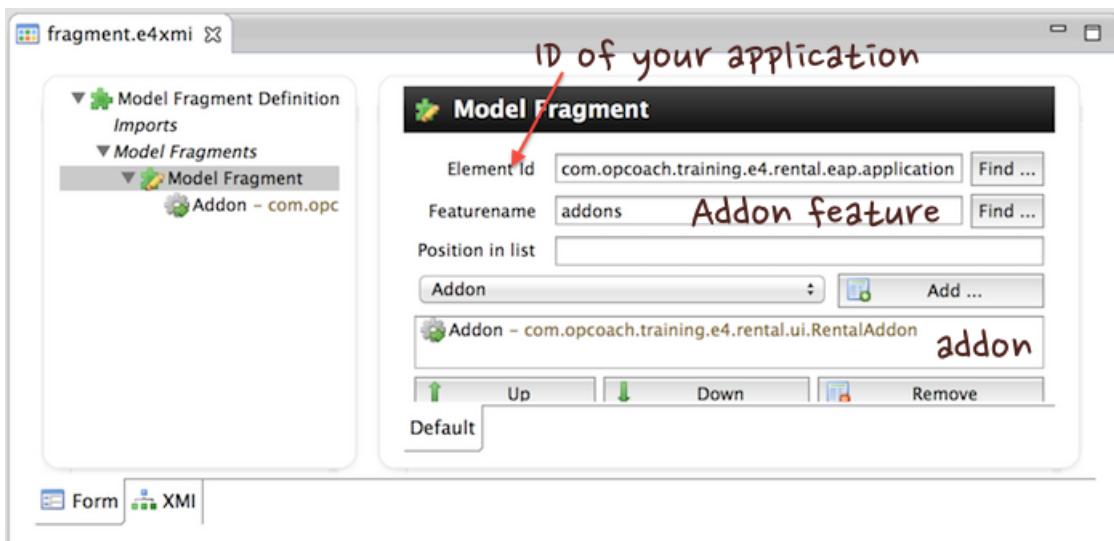


Image 197 Addon in fragment

Ajout d'un fragment avec un XPath

Il est possible d'indiquer qu'un fragment peut contribuer à un modèle en utilisant un Xpath :

Exemple :

- `xpath://*[@elementId='app.menu.primary'] | //*[@elementId='app.menu.secondary']`
- **xpath:/** : pour accéder à l'application (noeud racine)

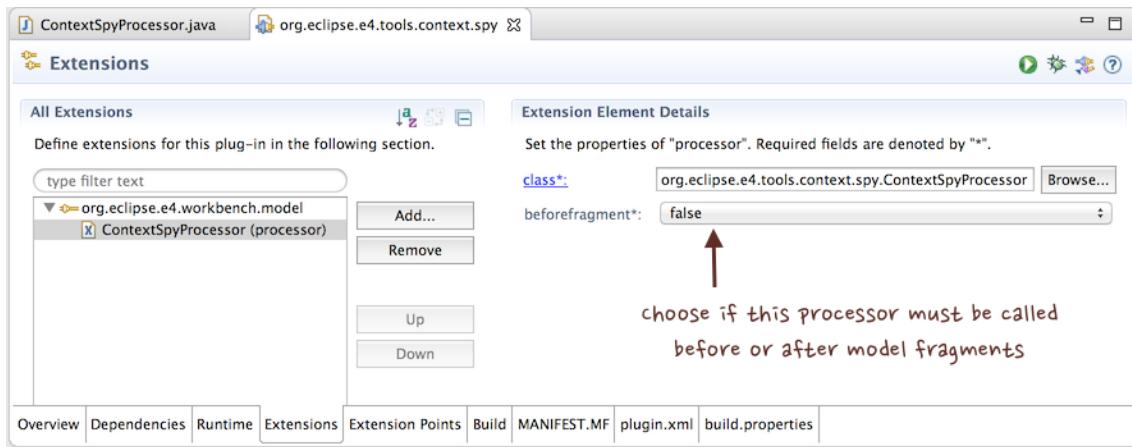
Ce mode de fonctionnement a été corrigé récemment et est décrit dans le bug #437958

- [https://bugs.eclipse.org/bugs/show_bug.cgi?id=437958³⁸](https://bugs.eclipse.org/bugs/show_bug.cgi?id=437958)

Déclaration d'un Processor

- Le processor est utilisé quand on ne connaît pas l'ID des objets à modifier
- On reçoit alors l'application par injection que l'on peut modifier
- Le processor s'ajoute par une extension de `org.eclipse.e4.workbench.model` de type processor :

38 - https://bugs.eclipse.org/bugs/show_bug.cgi?id=437958



Extension for a processor

Code d'un Processor

- Le code du processor est simplement un POJO avec un **@Execute**
- La méthode reçoit l'application et les services nécessaires pour modifier le modèle

```

37  /** A base class for all spies processors */
38  public class SpyProcessor
39  {
40      private static final String SPY_COMMAND = "org.eclipse.e4.tools.spy.command";
41      static final String SPY_COMMAND_PARAM = "org.eclipse.e4.tools.spy.command.partID";
42
43      private static final String SPY_HANDLER = "org.eclipse.e4.tools.spy.handler";
44      private static final String E4 SPIES_BINDING_TABLE = "org.eclipse.e4.tools.spy.bindi
45
46      @Inject MApplication application;
47      @Inject EModelService modelService; ← Inject what you need
48      @Inject Logger log;
49
50  @Execute
51  public void process(IExtensionRegistry extRegistry)
52  {
53      // This processor will read all spy extensions and automatically create
54      // the command, handler and binding
55      // to open this spy in the dedicated spy window.
56
57      // First of all, it creates the spyCommand having one parameter (Id of
58      // the part to display) and default handler for this command.
59      MCommand command = getOrCreateSpyCommand();
60
61 }
```

Code for a processor

Exercice EAP 085

Création du modèle de fragment pour l'addon

E4 Event Bus



Introduction

Pour être notifié d'un événement il faut utiliser des listeners

Il faut prévoir pour chaque cas une méthode de réaction

Exemple : si on veut écouter ce qui se passe sur les locations, on aurait :

- RentallListener avec rentalCreated(RentalEvent), rentalModified(RentalEvent),...
- ou il faut utiliser les adapters EMF.

Grâce à l'injection on peut envisager un mécanisme plus simple : **IEventBroker**

C'est le bus d'événements d'Eclipse 4

IEventBroker

L'eventBroker permet d'envoyer un événement identifié avec l'objet concerné

On peut ensuite :

- soit recevoir l'événement par injection automatiquement dès qu'il apparaît
- soit s'abonner et être notifié avec un **EventHandler** (java.beans)

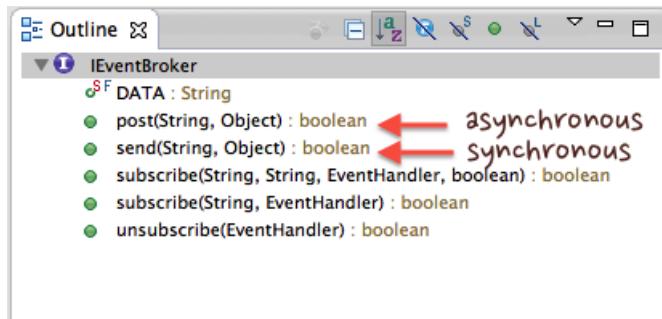


Image 198 Event Broker API

Envoi d'un événement

- L'envoi d'un événement se fait simplement en appelant post ou send
- L'IEventBroker est récupéré par injection

```

13
14 @Inject
15 IEventBroker broker;
16
17 @Execute
18 void execute()
19 {
20     // Create a new rental using factory
21     Rental r = RentalFactory.eINSTANCE.createRental();
22
23     // broadcast the rental creation in broker
24     broker.send("rental/new", r);
25 }
26
27

```

publish an event using a specific ID

Image 199 Event Broker send

Réception et traitement d'un événement

- On peut le faire dans une méthode injectée qui reçoit un paramètre `@UIEventTopic`
- La méthode étant injectée sera appelée à chaque envoi d'événement
- On peut utiliser des patterns pour recevoir les événements

```

public class RentalReact    receive here any event with ID starting
{
    @Inject
    @Optional
    void reactOnRentalEvent(@UIEventTopic("rental/*") Rental newRental)
    {
        // Use the rental to update it in a view for instance
    }
}

```

with rental

Image 200 EventBroker receive

UIEvents

La classe UIEvents déclare tous les événements du framework :

- les UILifeCycle (BRINGTOTOP, ACTIVATE...)
- les modifications sur l'application model (UIElement.VISIBLE...)

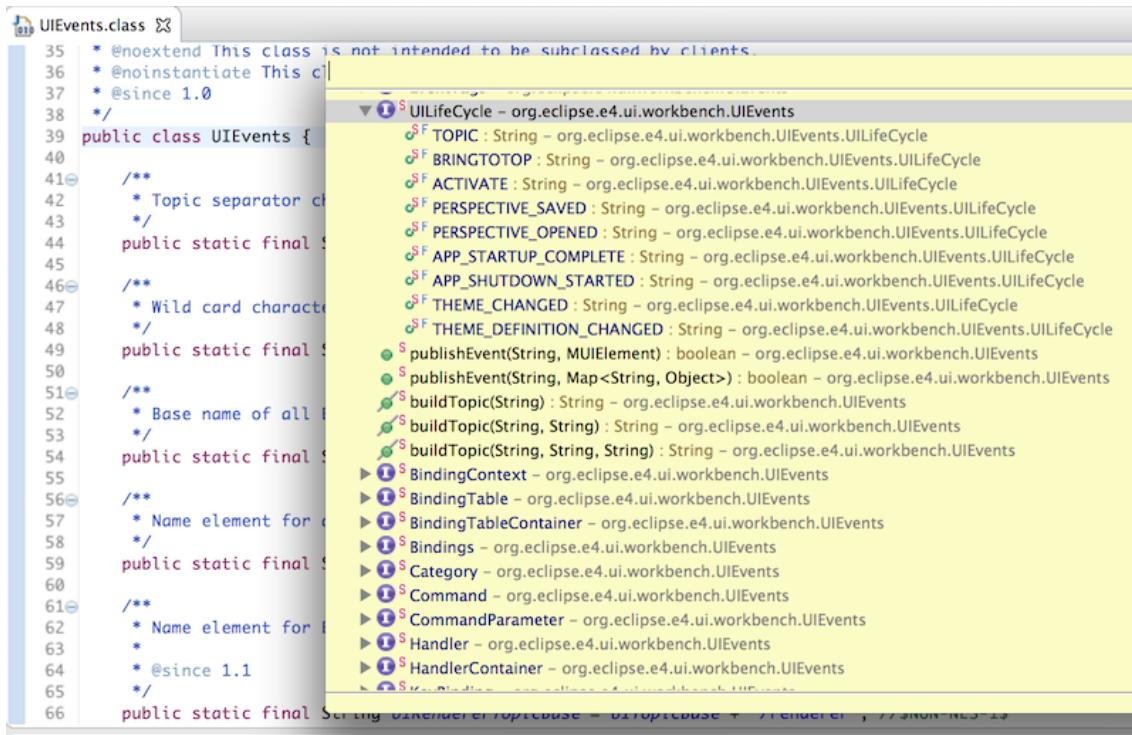


Image 201 UIEvents

Rafraîchissement de toute l'IHM avec le broker (état des commandes, etc...) :

```
broker.send(UIEvents.REQUEST_ENABLEMENT_UPDATE_TOPIC, UIEvents.ALL_ELEMENT_ID);
```

Refresh UI

Event Spy

- Il y a un 'spy' dédié à l'affichage des événements
- Il s'ouvre avec le raccourci **Alt Shift F8**
- Ou en utilisant le bouton event spy dans la fenêtre des espions e4
- Le plugin **org.eclipse.e4.tools.event.spy** doit être lancé pour le voir

E4 Spies window		
Topic	Event publisher	Changed element
▶ org/eclipse/e4/ui/LifeCycle/bringToTop	org.eclipse.e4.ui.internal.workbench.ModelServiceImpl	org.eclipse.e4.ui.model.app
▶ org/eclipse/e4/ui/LifeCycle/activate	org.eclipse.e4.ui.internal.workbench.PartServiceImpl	(a org.eclipse.e4.ui.model.app)
▶ org/eclipse/e4/ui/model/application/ApplicationElement/tags/REMOVE	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)
▶ org/eclipse/e4/ui/model/application/ApplicationElement/tags/REMOVE	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)
▼ org/eclipse/e4/ui/model/application/ApplicationElement/tags/ADD	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)
ChangedElement	org.eclipse.e4.ui.model.application.ui.basic.impl.Part	
Widget	CTabFolder {}	
AttName	tags	
NewValue	active	
EventType	ADD	
Position	1	
event.topics	org/eclipse/e4/ui/model/application/ApplicationElen	
▶ org/eclipse/e4/ui/model/application/ApplicationElement/tags/ADD	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)
▶ org/eclipse/e4/ui/model/ui/ElementContainer/selectedElement/SET	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)
▶ org/eclipse/e4/ui/model/ui/ElementContainer/selectedElement/SET	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)
▶ org/eclipse/e4/ui/LifeCycle/bringToTop	org.eclipse.e4.ui.internal.workbench.ModelServiceImpl	org.eclipse.e4.ui.model.app
▶ org/eclipse/e4/ui/LifeCycle/activate	org.eclipse.e4.ui.internal.workbench.PartServiceImpl	(a org.eclipse.e4.ui.model.app)
▶ org/eclipse/e4/ui/model/application/ApplicationElement/tags/REMOVE	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)
▶ org/eclipse/e4/ui/model/application/ApplicationElement/tags/REMOVE	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)
▶ org/eclipse/e4/ui/model/application/ApplicationElement/tags/ADD	org.eclipse.emf.common.notify.impl.BasicNotifierImpl	(org.eclipse.e4.ui.model.app)

Event Spy

Exercice EAP 087

EAP 087 : Gestion d'événement avec IEventBroker

API Interne Eclipse 4

XIII

API Eclipse (non IHM)

Services Eclipse 4

167

171

A. API Eclipse (non IHM)

BundleActivator

Activator

- Il peut dériver de **Plugin** ou implémenter **BundleActivator**
- C'est l'implémentation Java d'un Plugin et/ou d'un Bundle
- Il est sollicité lors du cycle de vie du Bundle
- Il référence le bundle OSGI
- Il peut définir des initialisations mais le mécanisme d'injection n'est pas disponible
 - Il n'est pas instancié avec la **ContextInjectionFactory**
- On préférera donc un Addon Eclipse 4 pour pouvoir utiliser l'injection

Logger Eclipse 4

- On utilise le service Eclipse 4 de Log (différent du ILog de Eclipse 3)

```
79  
80 @Inject  
81 private Logger logger; ← Inject a logger  
82  
83  
84 private void logAMessage()  
85 {  
86     logger.i ← log a message (error, debug...)  
87 }  
88  
89  
90  
91 public void  
92 {  
93     IExtensi  
94     System.o  
95     IExtensi  
96     IExtensi  
97     for (IE  
98     {  
99 }
```

Logger E4 sample

Ressources

Le workspace d'Eclipse est un arbre de **org.eclipse.core.resources.IResource**

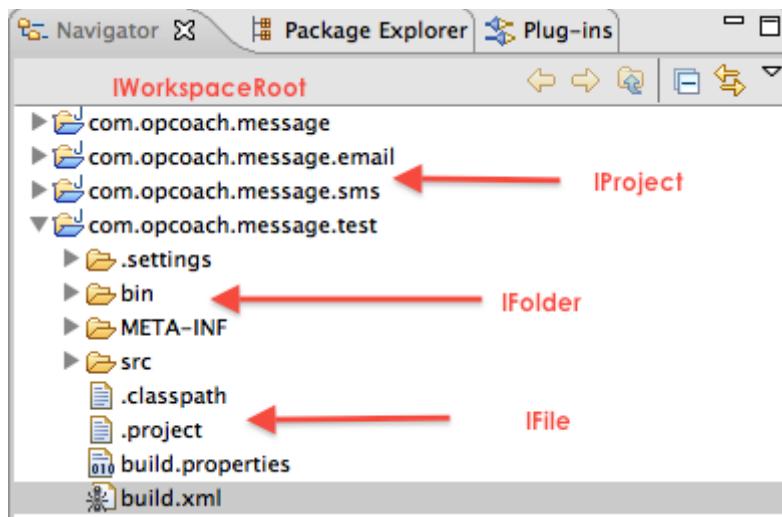


Image 202 Representation of Resources in the workspace

Pour utiliser les ressources du workspace, il faut dépendre du plugin [org.eclipse.core.resources](#)

Caractéristiques des ressources

Intérêt de IResource

- Markers (erreurs trouvées, bookmark, breakpoints...)
- Propriétés
- Ressources dérivées (encoding UTF-8,...)
- Historique local

Synchronisation

- S'il arrive d'utiliser [java.io.File](#) au lieu de [IFile](#)
- Perte de Synchronisation entre le Workspace et le système de fichiers
- Besoin de resynchronisation :
 - `resource.refreshLocal(IResource.DEPTH_INFINITE, new NullProgressMonitor());`

IPath

Le workspace est accédé en relatif par des IPath

- Pas de chemin par rapport au disque
- Chemin dans le workspace
- Format : Suite de segments
- Exemple :
 - Segment 1 = Nom du projet
 - Segment 2 = Nom du répertoire
 - Segment 3 = Nom du fichier
- Exemple : `new Path("/MonProjet/MonDirectory/MonFile");`

Création d'un fichier dans le workspace

Il faut récupérer le workspace root par injection et créer un fichier avec un path :

```
@Inject
public IFile createAFile(IWorkspaceRoot wsroot)
{
    Path path = new Path("myProject/folder1/folder2/filename.txt");
    return wsroot.getFile(path);
}
```

Image 203 IWorkspaceRoot

IAdaptable

Notion d'objet **adaptable**

- Cast avancé : transformer n'importe quel objet en n'importe quel autre
- « Polymorphisme par délégation »
- **org.eclipse.core.runtime.IAdaptable** : **getAdapter(Class c)** retourne une instance du type spécifié représentant l'objet courant ou null si aucun objet ne correspond

AdapterFactory

- Classe externe pouvant transformer un objet en un autre type
- Evite d'imposer un héritage (ex : Treeviewer sur File).
- S'enregistre par point d'extension sur l'application

Utilisation :

- Si on maîtrise l'objet on le fait implémenter **IAdaptable**
- Si on utilise un objet d'une librairie qu'on veut rendre adaptable, on crée une **IAdapterFactory**

Définition d'un adaptable

```
10 public class AdaptableIntoResource implements IAdaptable
11 {
12     private String name;
13
14     @Override
15     public Object getAdapter(Class adapter)
16     {
17         Object result = null;
18         if (IResource.class.equals(adapter))
19         {
20             IWorkspace ws = ResourcesPlugin.getWorkspace();
21             IWorkspaceRoot wsroot = ws.getRoot();
22             Path path = new Path("myProject/dossier" + name);
23             result = wsroot.getFile(path);
24         }
25         return result;
26     }
27 }
```

Image 204 IAdaptable

Definition d'une AdapterFactory

Code :

```

7  /** Transform a rental into a customer. Just extract customer from rental */
8  public class RentalToCustomerAdapterFactory implements IAdapterFactory
9  {
10     // Transform a Rental into a Customer object
11    public Object getAdapter(Object adaptableObject, Class<?> adapterType)
12    {
13        Customer result = null;
14        if ((adaptableObject instanceof Rental) && (adapterType == Customer.class))
15        {
16            result = ((Rental)adaptableObject).getCustomer();
17        }
18        return result;
19    }
20
21
22    public Class[] getAdapterList()
23    {
24        // Result returned types
25        return new Class[] { Customer.class }; Don't forget to return the result type(s)
26    }
27
28 }
```

Image 205 A sample adapter factory

Point d'extension :

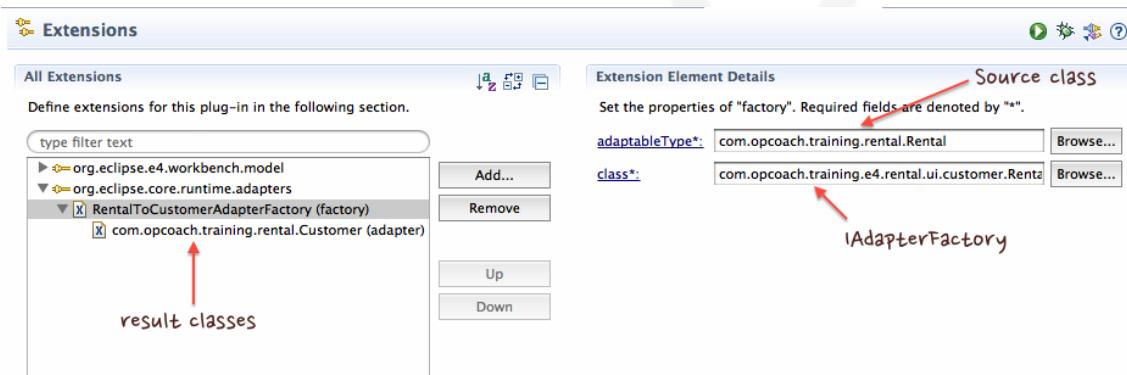


Image 206 core runtime adapters

Utilisation des adaptables

```

12
13    @Inject
14    public void useObjectAsCustomer(@Optional @Named(IServiceConstants.ACTIVE_SELECTION) Object o,
15                                    Adapter adapter) Inject selection and adapter
16    {
17        // Try to translate the selected Object o into a Customer instance
18        Customer cust = (Customer) adapter.adapt(o, Customer.class);
19
20        // If we get it, we use it
21        if (cust != null)
22        {
23            setCustomer(cust);
24        }
25    }
26 }
```

Image 207 Adapter usage

Bonnes pratiques

- Consulter l'aide Eclipse
- API Reference
- Consulter les exemples d'utilisation
 - dans le code des plugins
 - dans les forums
 - dans les snippets sur dev.eclipse.org
- Abuser de la navigation
 - **Ctrl+Shift+T** (navigation sur les types)
 - **Ctrl T** (quick type hierarchy)
 - **F4** (hiérarchie type)
 - **F3** (accès au source)
- Ne jamais utiliser une API dans un package org.eclipse.*internal*

Exercice EAP 055

Création d'une adapter factory

B. Services Eclipse 4

Les services

Eclipse 4 propose différents services que l'on peut utiliser via l'injection.

On trouve notamment :

- **EModelService** : pour accéder au modèle d'application
- **ESelectionService** : pour gérer la sélection
- **EPartService** : pour gérer les parts
- **EMenuService** : pour gérer les menu (ajout de menu contextuel par ex)
- **TranslationService** : pour gérer la traduction dynamiquement
- **IExtensionRegistry** : registre de tous les points d'extensions extensions
- **Adapter** : permet de convertir les objets en un autre (ex AdapterManager eclipse 3)
- **Logger** : pour logger des status
- **EContextService** : pour gérer les binding contextes (keybindings...)
- **IThemeEngine** : pour changer la CSS au runtime
- **ECommandService** et **EHandlerService** :pour la gestion des commandes.

EModelService

Il permet d'accéder aux composants du model d'application

Permet de :

- retrouver un élément
- manipuler le modèle : move, remove, insert
- de faire un reset sur une perspective
- de placer un élément devant les autres (**bringToTop**)

ESelectionService

Permet de mémoriser la sélection et de la transmettre

```
18 */
19 public interface ESelectionService {
20
21 /**
22 * Due to the possibly misleading nature of this field's name, it has been replaced with
23 * {@link IServiceConstants#ACTIVE_SELECTION}. All clients of this API should change their
24 * references to <code>IServiceConstants.ACTIVE_SELECTION</code>.
25 */
26 @Deprecated
27 public static final String SELECTION = IServiceConstants.ACTIVE_SELECTION; // "in.selection";
28
29 public void setSelection(Object selection);
30
31 public Object getSelection();
32
33 public Object getSelection(String partId);
34
35 public void addSelectionListener(ISelectionListener listener);
36
37 public void removeSelectionListener(ISelectionListener listener);
38
39 public void addSelectionListener(String partId, ISelectionListener listener);
40
41 public void removeSelectionListener(String partId, ISelectionListener listener);
42 }
43 }
```

Image 208 ESelectionService

EPartService

Permet de manipuler les parts :

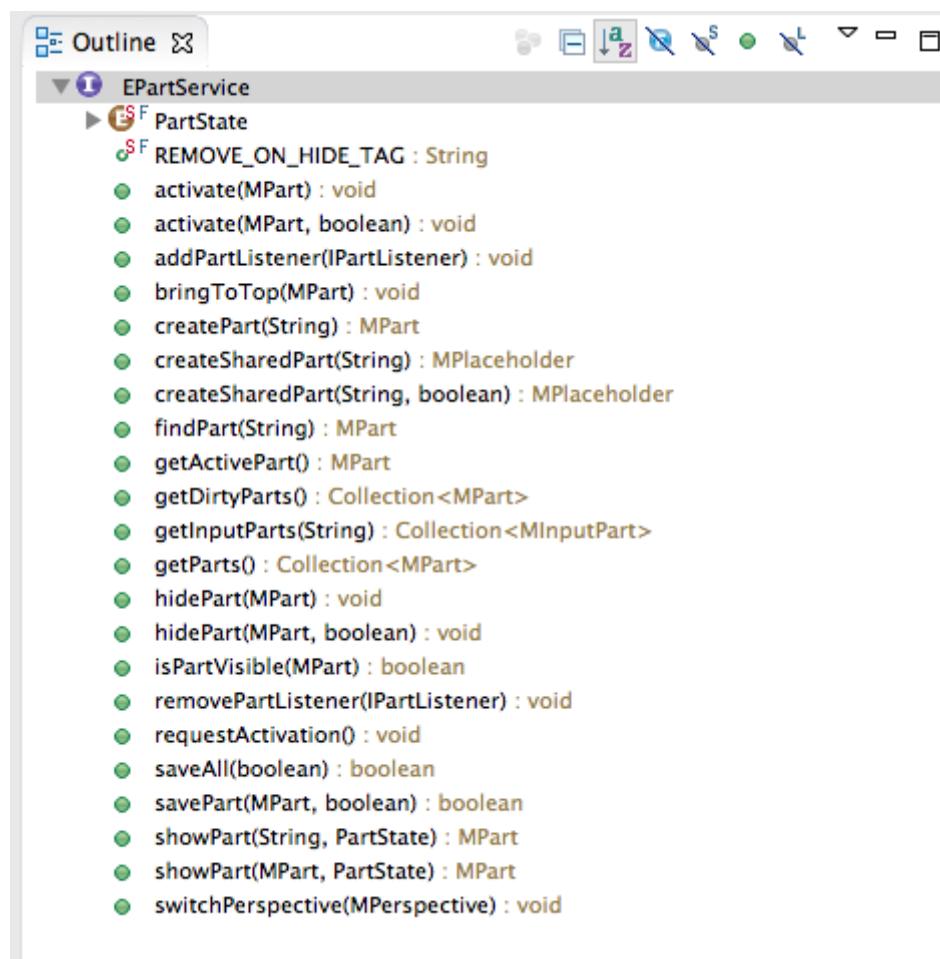


Image 209 EPartService

EMenuService

Il ne sert qu'à enregistrer un menu contextuel sur un widget :

```
-- 13 /**
14  * Provide for management of different menus.
15  *
16  * @noimplement This interface is not intended to be implemented by clients.
17  * @since 1.1
18  */
19 public interface EMenuService {
20
21 /**
22  * Create a menu for this control and hook it up with the MPopupMenu.
23  *
24  * @param parent
25  *          The parent for the context menu. A Control in SWT.
26  * @param menuId
27  *          the ID of the menu to use
28  * @return <code>true</code> if registration succeeded else <code>false</code>
29  */
30 boolean registerContextMenu(Object parent, String menuId);
31
32 }
33
```

Image 210 EMenuService

Usage :

```
-- 58
59 // Register a popup menu on viewer (MENU_ID is the id of popupmenu in
60 // application model)
61 menuService.registerContextMenu(agencyViewer.getControl(), MENU_ID);
62
63
```

Menu Service Usage

TranslationService

- Les mécanismes de traduction standards d'Eclipse (basés sur des chaînes ou des constantes) sont suffisants pour traduire mais ne sont pas configurables.
- Il faut redémarrer l'application pour changer la langue
- Les fichiers de properties sont stockés dans l'arborescence de sources
- Le Translation service permet :
 - de modifier la gestion d'accès aux messages (google traduction, base de données...)
 - de changer la langue sans redémarrer (géré par l'injection)
 - de placer les fichiers properties en dehors du code source (géré par OSGi)
- A utiliser pour une traduction dynamique sans redémarrage de l'application
- Lire les 4 articles :
 - <http://blog.vogella.com/2013/05/03/eclipse-internationalization-part-14-current-situation-by-dirk-fauth/>³⁹
 - <http://blog.vogella.com/2013/05/22/eclipse-internationalization-part-24-new-message-extension-by-dirk-fauth-and-tom-schindl/>⁴⁰
 - <http://blog.vogella.com/2013/06/25/eclipse-internationalization-part-34-migration-by-dirk-fauth/>⁴¹
 - <http://blog.vogella.com/2013/08/12/eclipse-internationalization-part-44-new-features->

39 - <http://blog.vogella.com/2013/05/03/eclipse-internationalization-part-14-current-situation-by-dirk-fauth/>

40 - <http://blog.vogella.com/2013/05/22/eclipse-internationalization-part-24-new-message-extension-by-dirk-fauth-and-tom-schindl/>

41 - <http://blog.vogella.com/2013/06/25/eclipse-internationalization-part-34-migration-by-dirk-fauth/>

*by-dirk-fauth/*⁴²



42 - <http://blog.vogella.com/2013/08/12/eclipse-internationalization-part-44-new-features-by-dirk-fauth/>



Créer des points d'extension

Rappel sur les points d'extension

- Eclipse est fourni avec des dizaines de points d'extension
- Un point d'extension est la modélisation d'un concept
- Le modèle des paramètres attendu est défini par un schéma
- Un point d'extension est traité par le plugin qui le définit
- Un point d'extension peut être étendu dès le plugin qui le définit

Définition d'un nouveau point d'extension

On utilise l'onglet 'Point Extension' du PDE :

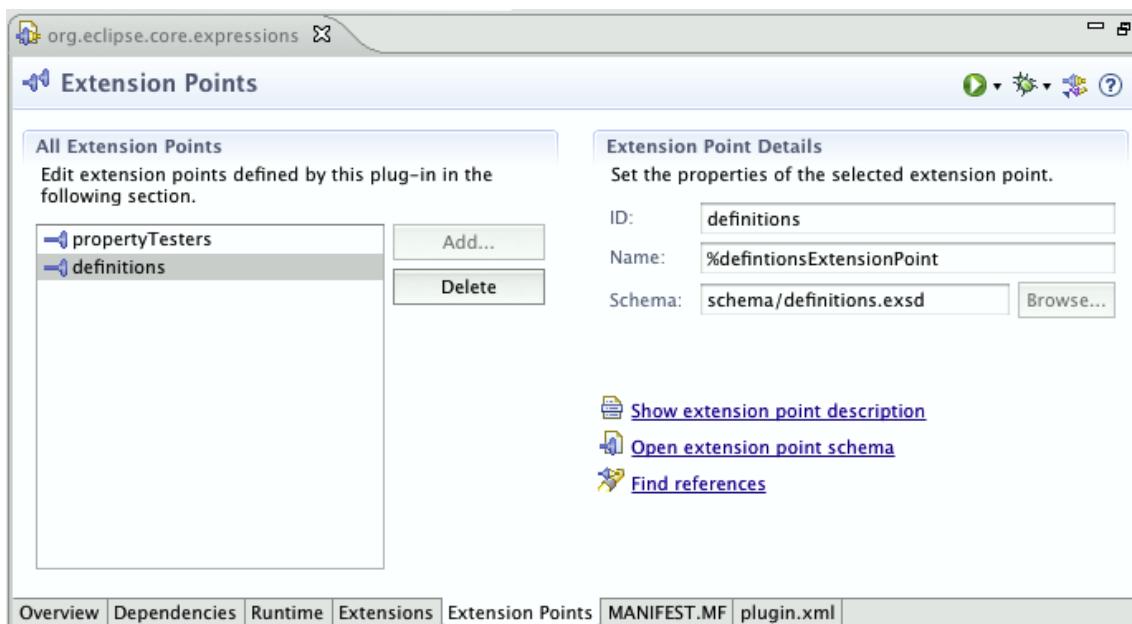


Image 211 Extension point tab

Définition du point d'extension

On déclare ID, name et schéma

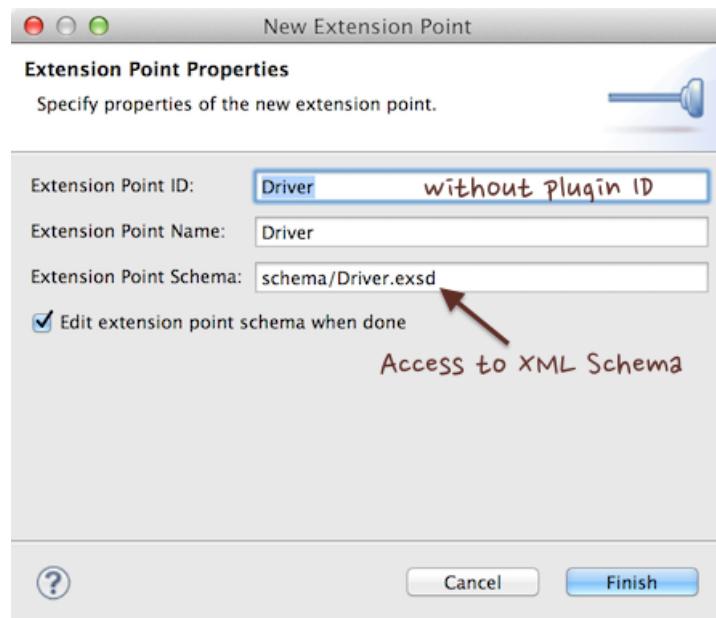


Image 212 Creation dialog

Schéma du point d'extension

- C'est le modèle du point d'extension
- Le XML Schéma est du XML pour décrire du XML
- Permet de générer la documentation
- S'édite à l'aide d'un éditeur dédié très pratique
- Est utilisé pour générer l'IHM de paramétrage de l'extension
- Permet de gérer les types simples et évolués : classes ou interfaces

Editeur de schéma

L'éditeur de schéma permet de définir :

- la documentation : overview, exemples, copyright, etc...
- les éléments XML de paramétrage
- les attributs de ces éléments
- les types de ces attributs (string, entier, classe, interface, ID...)
- l'ordonnancement de ces éléments entre eux
- les cardinalités des éléments

Edition de la documentation

- Onglet Overview

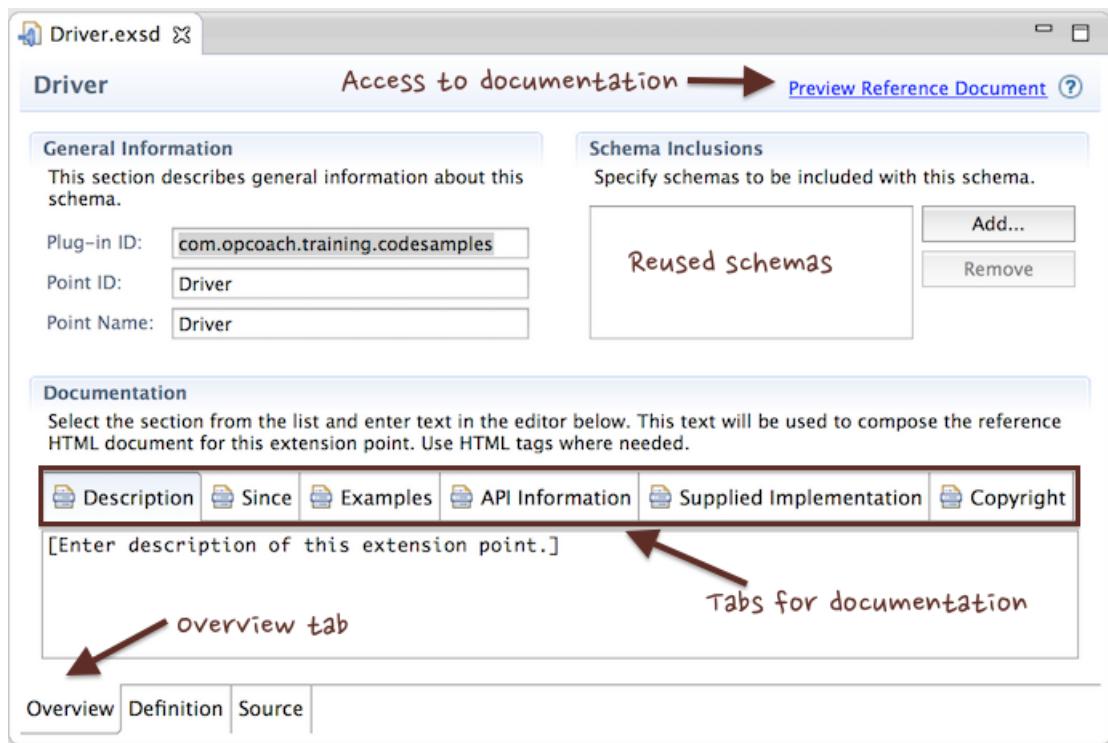


Image 213 Schema Editor Documentation

1. Edition du schéma / Crédation des Elements

- La première étape consiste à créer les éléments et les attributs :

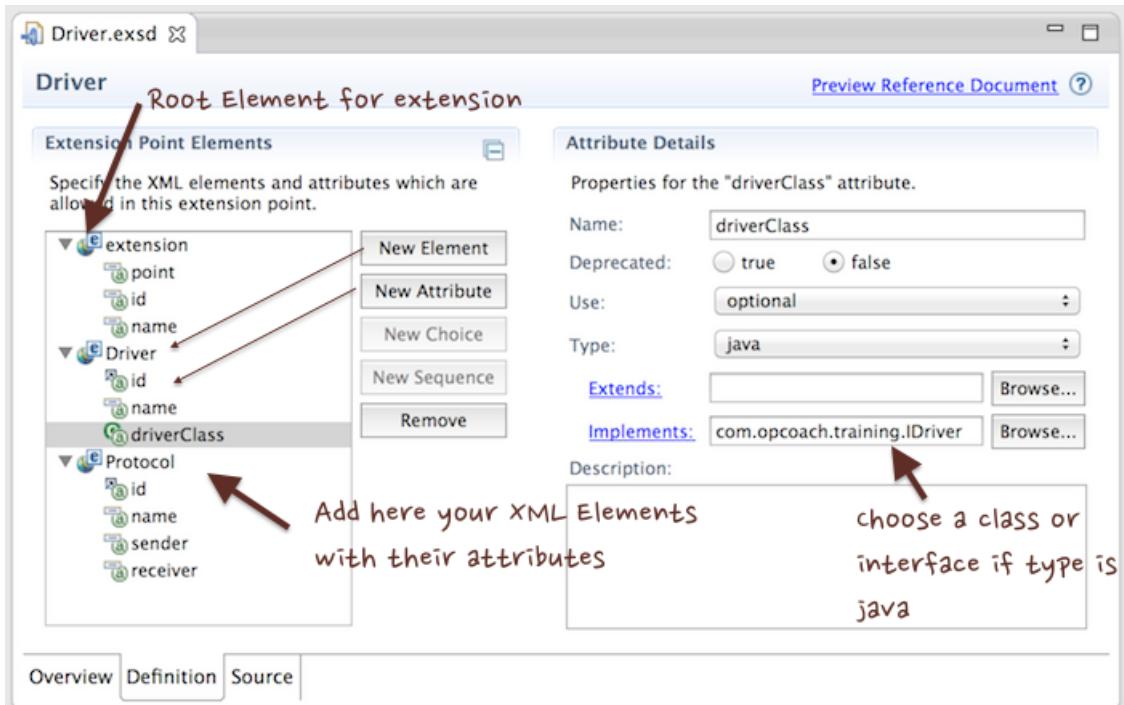


Image 214 Schema elements definition

- Un attribut peut être de type Boolean, String, Ressource, Java, Identifier (ou refID)
- Pour faire un énuméré, utiliser le type String avec des restrictions

Gestion des IDRef

Pour faire une référence sur un ID existant :

- indiquer le type ID
- indiquer la référence sous forme de xpath

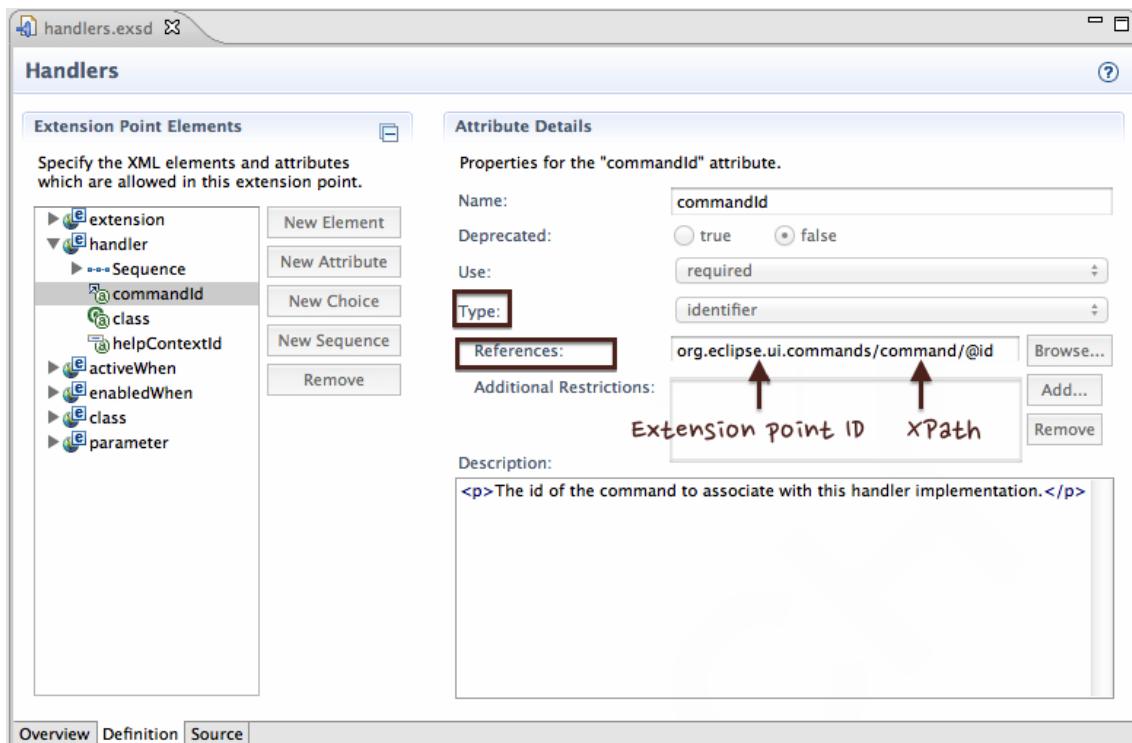


Image 215 ID Ref

2. Edition du schéma / Relation des éléments

- La seconde étape consiste à relier les éléments entre eux.

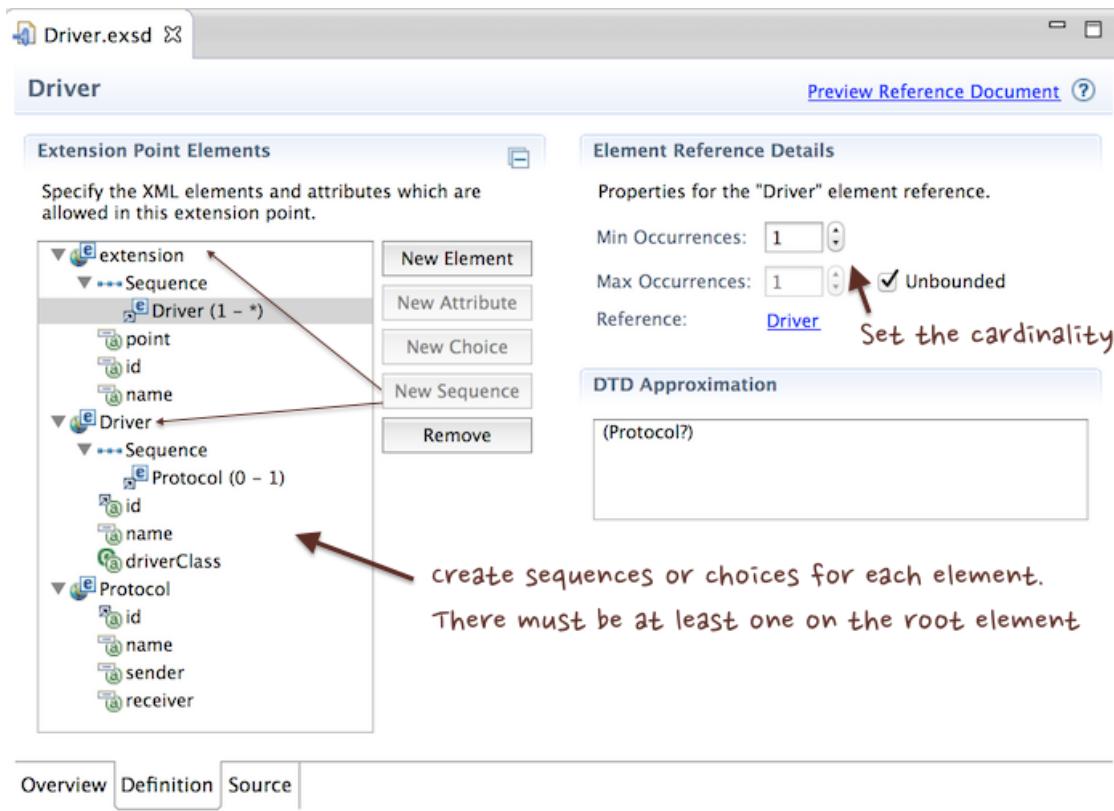


Image 216 Element sequence definition

Traitement du point d'extension

- Le plugin qui définit le point d'extension doit le traiter
- Il faut écrire du code pour initialiser les objets désignés par l'extension
- Ce code se met dans l'activator dans une méthode dédiée
- En général le code initialise un **XXXManager** spécifique à l'extension **XXX**
- Exemple : un **DriverManager** est rempli avec les extensions Driver.
- L'instanciation des classes et l'accès aux ressources est spécifique :
 - du au class loader spécifique de chaque plugin
 - nécessite l'emploi de méthodes dédiées
- On utilise des classes spécifiques pour aider à la lecture :
 - **IExtensionRegistry**, **IExtensionPoint**, **IExtension**
 - **ConfigurationElement**

Exemple de code de lecture de point d'extension.

```

131 @Inject
132 public void getExtensions(IExtensionRegistry reg) ← get ExtensionRegistry with injection
133 {
134     IExtensionPoint extp = reg.getExtensionPoint("pluginID.extensionID");
135     IExtension[] extensions = extp.getExtensions();
136
137     for (IExtension ext : extensions)
138     {
139         // Get all configuration element for current extension
140         for ( IConfigurationElement elt : ext.getConfigurationElements())
141         {
142             String attValue = elt.getAttribute("anAttributeName");
143             System.out.println(" Found this element : " + elt.getName() + " with attr=" + attValue);
144         }
145     }
146 }
147
  
```

Image 217 getExtensions

Configuration Elements

```

-- 
20
21 <extension point="org.eclipse.ui.preferencePages">
22   <page
23     category="com.opcoach.training.rental.preferences"
24     class="com.opcoach.training.e4.rental.ui.pref.RentalColorPreferences"
25     id="com.opcoach.rental.ui.colorprefpage"
26     name="Colors">
27   </page>
28   <page
29     class="com.opcoach.training.e4.rental.ui.pref.RentalPrefHomePage"
30     id="com.opcoach.training.rental.preferences"
31     name="Rental">
32   </page>
33 </extension>
34
35 <extension point="org.eclipse.core.expressionsdefinitions">
36   <definition id="com.opcoach.training.e4.rental.ui.customerSelected">
37     <iterate>
38       <instanceof
39         value="com.opcoach.training.rental.Customer">
40     </instanceof>
41   </iterate>
42 </definition>
43 </extension>
  
```

The definition contains an 'iterate' configuration element which contains an 'instanceof'

Image 218 Configuration Element

Exemple plus 'rapide'

```

119
120 @Inject
121 public void getExtensionsQuickAccess(IExtensionRegistry reg)
122 {
123     for (IConfigurationElement elt : reg.getConfigurationElementsFor("pluginID.extensionID"))
124     {
125         String attValue = elt.getAttribute("anAttributeName");
126         System.out.println(" Found this element : " + elt.getName() + " with attr=" + attValue);
127     }
128 }
129
130

```

Image 219 getExtension quick

Exercice EAP 090

Recherche d'extensions

Instanciation d'une classe désignée dans l'extension.

Il faut utiliser :

- la méthode **createExecutableExtension** définie sur le configurationElement
- faire attention de passer le nom de l'attribut et non sa valeur

```

110
111 @Inject
112 private void readDriverExtensions(IExtensionRegistry reg, Logger log, IEclipseContext ctx)
113 {
114     Collection<Driver> results = new ArrayList<Driver>();
115     for (IConfigurationElement e : reg.getConfigurationElementsFor("pID.Driver"))
116     {
117         try
118         {
119             results.add((Driver) e.createExecutableExtension("driverClass"));
120         }
121         catch (Exception ex)
122         {
123             log.error("Unable to create extension.", ex);
124         }
125     }
126     // Add the drive manager in the Eclipse Context
127     ctx.set(DRIVER_MANAGER, results);           Fill the context with the result
128 }
129

```

Inject here reg, log and context

Attribute name !

Image 220 Create instance

Instancie la classe en utilisant l'injection

```

127
130  @Inject
131  private void createDriverExtensionsWithInjection(IExtensionRegistry reg, Logger log, IEclipseContext ctx)
132  {
133      Collection<Driver> results = new ArrayList<Driver>();
134      for (IConfigurationElement e : reg.getConfigurationElementsFor("pID.Driver"))
135      {
136          try
137          {
138              // Instead of calling createExecutableExtension, use the following code to have injection
139              // results.add((Driver) e.createExecutableExtension("driverClass"));
140              Bundle b = Platform.getBundle(e.getNamespaceIdentifier());
141              Class<?> clazz = b.loadClass(e.getAttribute("driverClass"));
142              Driver instance = (Driver) ContextInjectionFactory.make(clazz, ctx);
143              results.add(instance);
144
145          } catch (Exception ex)
146          {
147              log.error("Unable to create extension", ex);
148          }
149      }
150      // Add the drive manager in the Eclipse Context
151      ctx.set(DRIVER_MANAGER, results);
152  }
153

```

Instanciate with injection

Accès à une ressource désignée dans l'extension.

- Il faut passer par le bundle du plugin pour avoir le path absolu
- Le bundle propose la méthode getEntry()

```

/** Read the Driver extension point and initialize DriverManager in the context */
@Inject
private void readDrivers(IExtensionRegistry reg, Logger log, IEclipseContext ctx)
{
    // Get drivers and set their configuration files.
    Collection<Driver> result = new ArrayList<Driver>(); Inject reg, log and ctx
    for (IConfigurationElement e : reg.getConfigurationElementsFor("pID.Driver"))
    {
        // Must get the source plugin declaring this configuration element
        Bundle bdl = Platform.getBundle(e.getContributor().getName());
        try
        {
            // Create the driver
            Driver driver = (Driver) e.createExecutableExtension("driverClass");

            // get configuration file URL in attribute and set it on driver
            String configFile = e.getAttribute("configurationFile");
            URL url = bdl.getEntry(configFile); get attribute value and URL
            driver.setConfiguration(url);

            result.add(driver);
        } catch (Exception ex)
        {
            log.error("Unable to create Driver extension.", ex);
        }
    }
    // Add the drive manager in the Eclipse Context
    ctx.set(PALETTE_MANAGER, result);
}

```

Image 221 get URL

Exercice RCP 092

Création point extension



CSS Styling



Dans le code java

E4AP dispose de son propre moteur de rendu CSS

Il suffit de créer une css référençant les classes css renseignées dans le code :

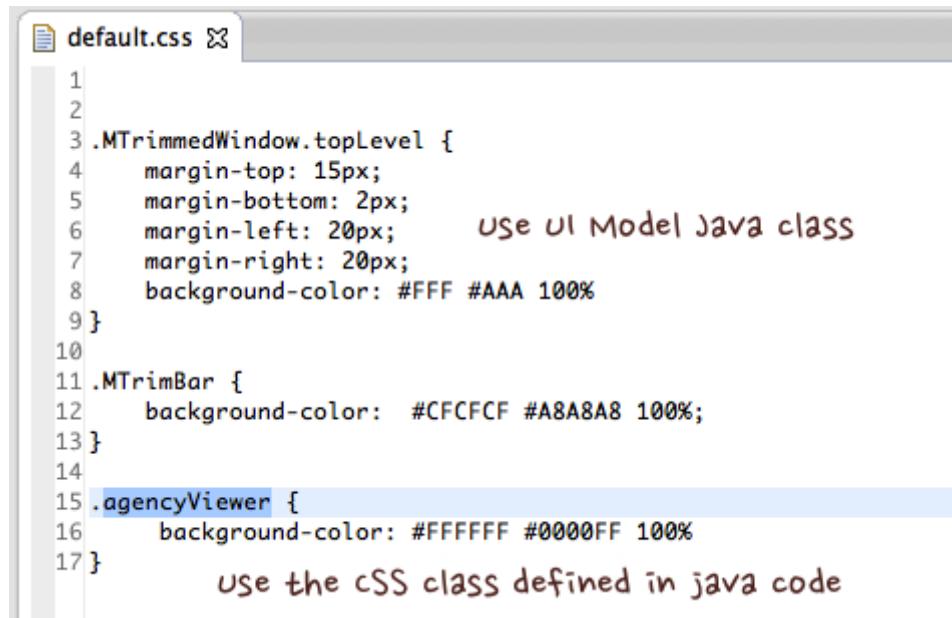
```
55
36 @PostConstruct
37 public void createContent(Composite parent, @Optional IStylingEngine styleEngine)
38 {
39     TreeViewer agencyViewer = new TreeViewer(parent);
40
41     // .... more code to fill the UI
42
43     // Add the e4 styling
44     if (styleEngine != null)
45     {
46         styleEngine.setClassname(agencyViewer.getControl(), "agencyViewer");
47     }
48
49
50 }
```

Image 222 Styling in Java

Contenu de la css

Il est possible de mettre à jour les styles de tout widget SWT, en utilisant le nom de la classe :

- Label : est la classe pour styler un widget
- Table : est la classe pour styler une Table



```

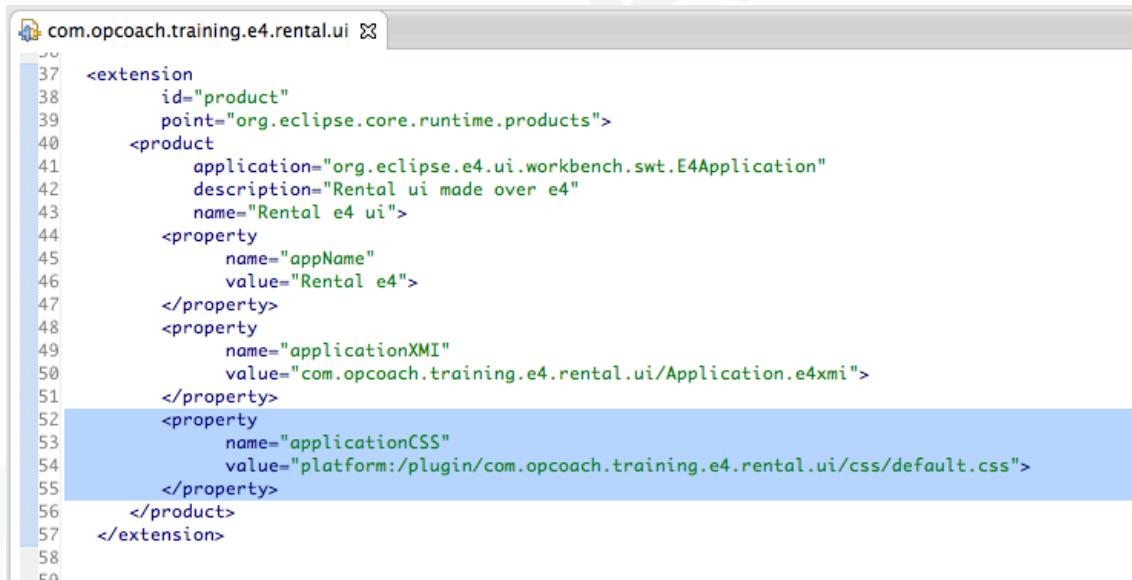
1
2
3 .MTrimmedWindow.topLevel {
4     margin-top: 15px;
5     margin-bottom: 2px;
6     margin-left: 20px;      USE UI Model Java class
7     margin-right: 20px;
8     background-color: #FFF #AAA 100%
9 }
10
11 .MTrimBar {
12     background-color: #CFCFCF #A8A8A8 100%;
13 }
14
15 .agencyViewer {
16     background-color: #FFFFFF #0000FF 100%
17 }      use the CSS class defined in java code

```

Image 223 Css

Référence à la CSS

La CSS est référencée dans le point d'extension de produit :



```

<extension
    id="product"
    point="org.eclipse.core.runtime.products">
<product
    application="org.eclipse.e4.ui.workbench.swt.E4Application"
    description="Rental ui made over e4"
    name="Rental e4 ui">
    <property
        name="appName"
        value="Rental e4">
    </property>
    <property
        name="applicationXMI"
        value="com.opcoach.training.e4.rental.ui/Application.e4xmi">
    </property>
    <property
        name="applicationCSS"
        value="platform:/plugin/com.opcoach.training.e4.rental.ui/css/default.css">
    </property>
</product>
</extension>

```

Image 224 product extension

Résultat de la CSS

Sans CSS :

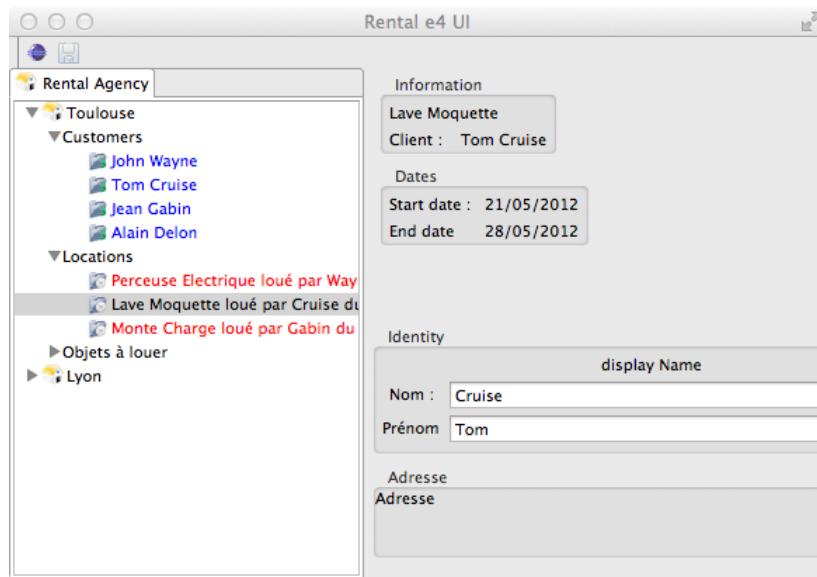


Image 225 without css

Avec CSS :

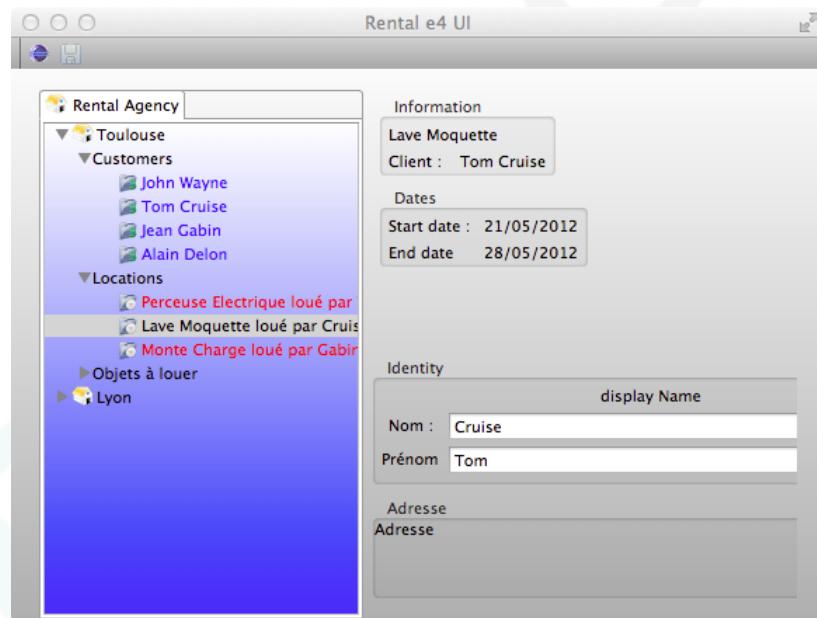


Image 226 with css

CSS Spy

On peut explorer les effets de la CSS avec le CSS Spy

Pour l'ouvrir utiliser le quick search en haut de l'écran ou **Ctrl Shift F5** (win32) ou **Alt Shift F5** (Mac)

L'élément concerné se visualise à l'écran

L'ID de la commande pour la mettre dans une application : **org.eclipse.e4.css.OpenSpy**

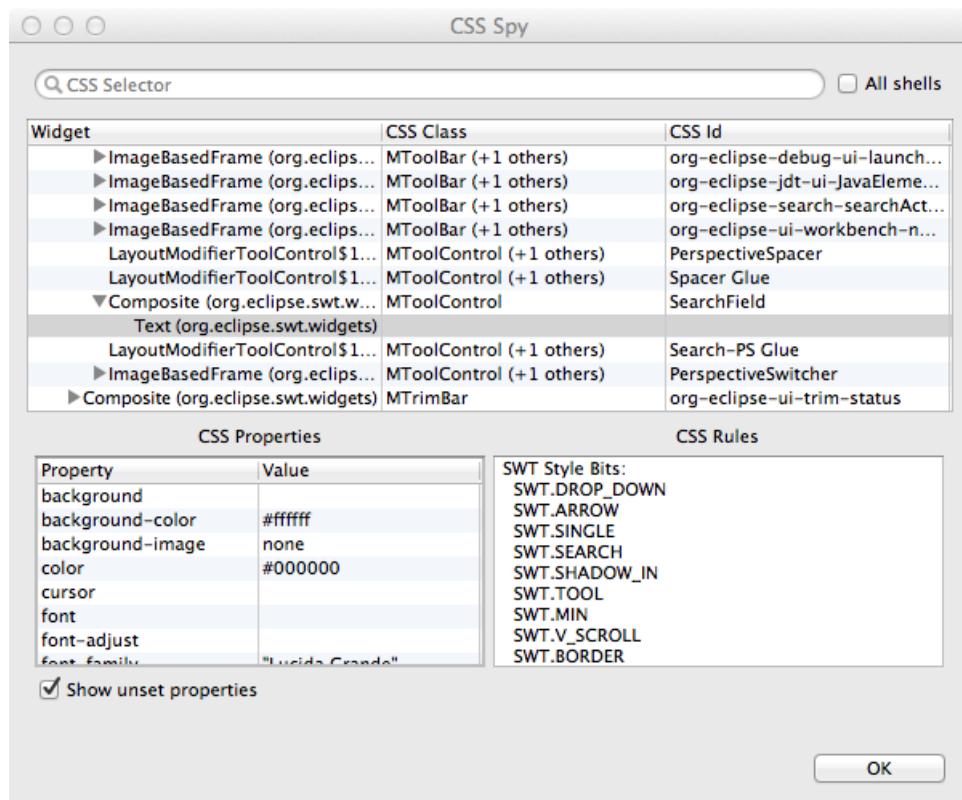


Image 227 Css Spy

Css selon les plateformes

On peut fixer une CSS selon une cible donnée

Il faut indiquer les css dans des extensions de : **org.eclipse.e4.ui.css.swt.theme**

```
<theme
    basestylesheeturi="css/e4_default_winxp_blu.css"
    id="org.eclipse.e4.ui.css.theme.e4_default"
    label="e4DefaultWinXPBlue"
    os="win32"
    ws="winxp">
</theme>
```

Image 228 css theme extension

Exercice EAP 100

Gestion de la CSS

Livraison Eclipse



Mécanismes.

Différentes livraisons sont possibles selon les besoins :

- plugins/features exportés
- update site
- exécutable stand alone

Le processus de livraison inclut une passe de compilation

- manuellement par Eclipse
- automatiquement via hudson et pde.build, buckminster ou maven tycho

Moyens de livraisons

Le PDE fournit tous les wizards pour aider à livrer :

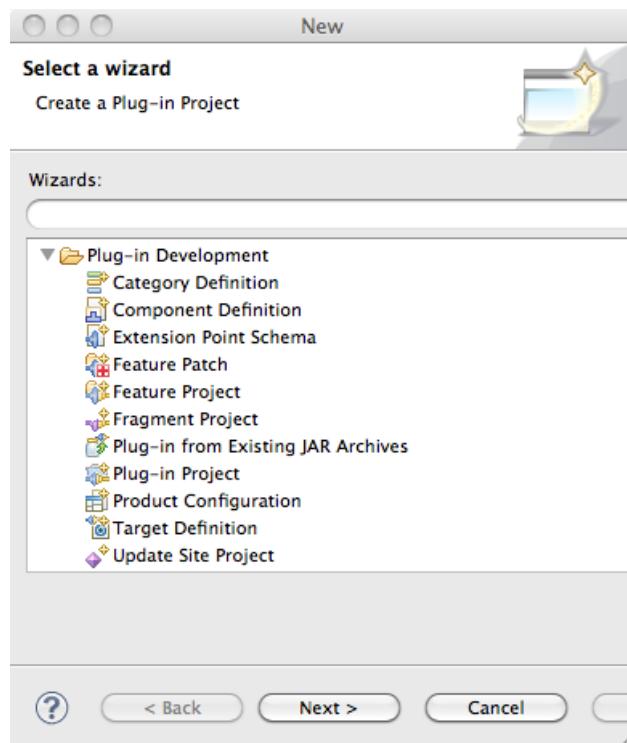


Image 229 PDE Wizards



Attention: Livraison de projets gérés sous git

- Les projets partagés sur git doivent être réimportés dans le workspace pour que les outils de livraison manuelle fonctionnent.
- Ceci est du au fait que le partage sous git déplace les projets du workspace vers le

\$HOME/git et Eclipse ne se remet pas à jour

La livraison sur update site

Elle concerne une livraison à installer dans un Eclipse existant.

Procédure :

- créer un projet feature pour chaque partie à livrer
- créer un projet update site, référençant les features
- lancer le build de l'update site (bouton build All dans le site.xml)
- copier le contenu du projet update site dans un endroit à partager

La livraison sous forme de produit

Un produit représente l'exécutable livrable indépendant d'eclipse.

Il se définit à l'aide d'un fichier '.product' qui contient :

- l'application à lancer
- le branding : icone, splash...
- les paramètres de lancement selon la plateforme
- etc...

On exporte ensuite le .product dans un répertoire

Livraison multi plateformes

- Comment livrer sur toutes les plateformes ?
- Le **RCP delta pack** répond au besoin
- **Téléchargement :**
 - aller sur : <http://download.eclipse.org/eclipse/downloads/>⁴³
 - choisir la version cible (3.7.2, 3.9, 4.3...)
 - dans l'onglet de gauche le delta pack est proposé
- **Installation :**
 - dans l'installation eclipse créer un répertoire 'deltapack'
 - dézipper le deltapack dedans
 - dans la target plateforme ajouter le répertoire delta pack
- **Utilisation :**
 - Le wizard d'export proposera le choix de la plateforme cible

Exercice RCP 110

Livraison Manuelle de l'application

43 - <http://download.eclipse.org/eclipse/downloads/>

Build Tycho Jenkins Eclipse



Maven Tycho

Jenkins

193

205

A. Maven Tycho

Présentation

Tycho permet la construction d'une application Eclipse en mode 'headless'

- <http://www.eclipse.org/tycho>
- ensemble de plugins Maven pour batir des applications OSGi
- basé sur maven 3
- interprète les informations des manifests OSGi
- maven reconnaît : bundles, plugins, test plugins, features, fragments, update site, rcp applications.
- gère la target platform : locale ou décrite dans un pom

Tycho vs Buckminster

- Modèle de build pour buckminster avec des éditeur dédiés
- Tycho orienté script maven
- S'intègrent tous les deux dans hudson⁴⁴
- Buckminster était utilisé chez Eclipse, tycho outsider (2010)
- Tycho est maintenant l'outil officiel de build Eclipse pour les nouveaux projets.

44 - <http://www.eclipse.org/tycho>

Principe Tycho

- On organise les projets de la manière suivante :

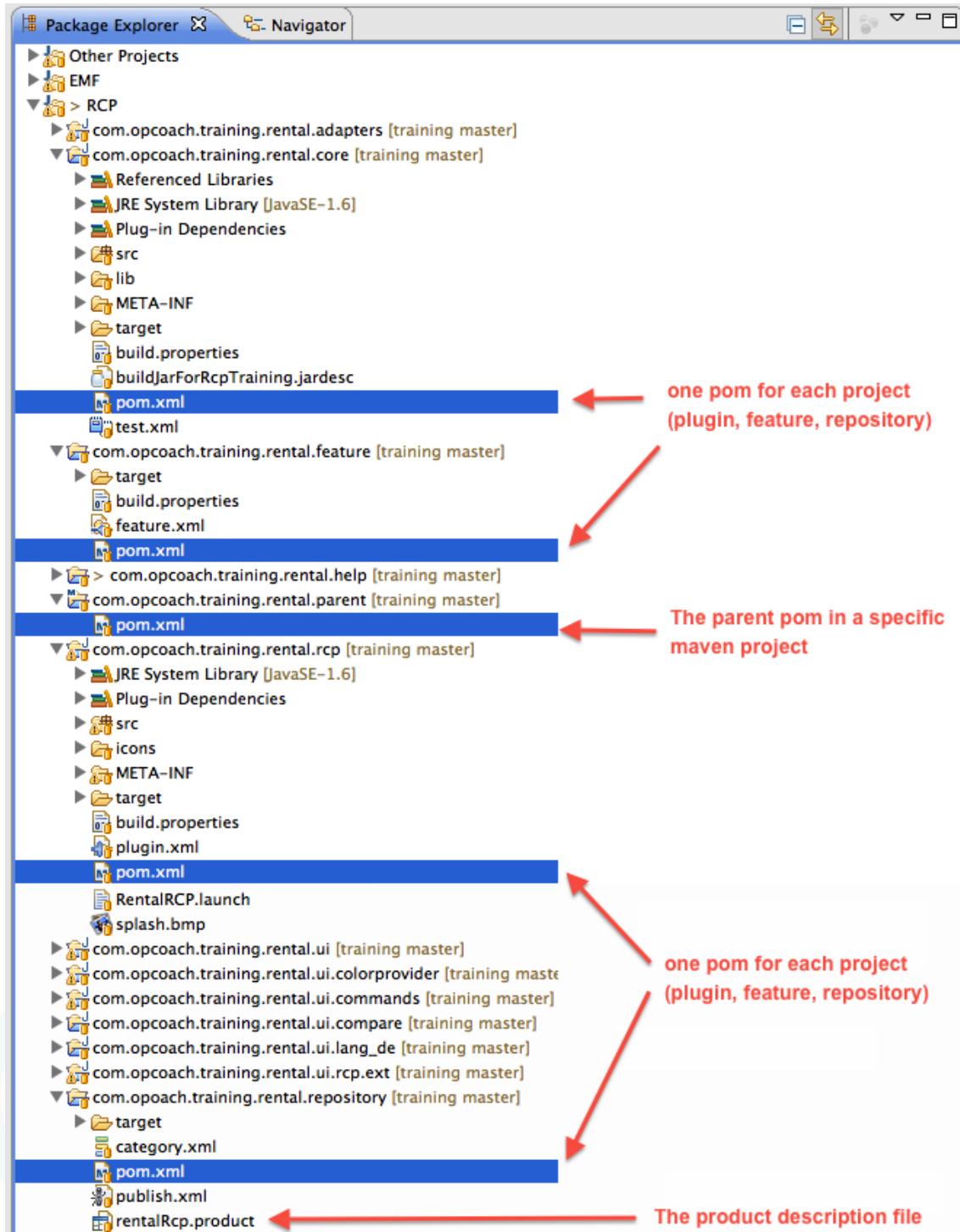


Image 230 Projects organization

Le pom projet

On distingue les différents types de projets :

- eclipse-plugin
- eclipse-test-plugin
- eclipse-repository
- eclipse-feature
- pom (pour le parent project)
- eclipse-target-definition
- eclipse-application et eclipse-update-site sont deprecated

Voir: http://wiki.eclipse.org/Tycho/Packaging_Types

Les types de projet s'indiquent dans la partie 'packaging' de l'overview (dans l'éditeur xml)

Contenu du pom projet (overview)

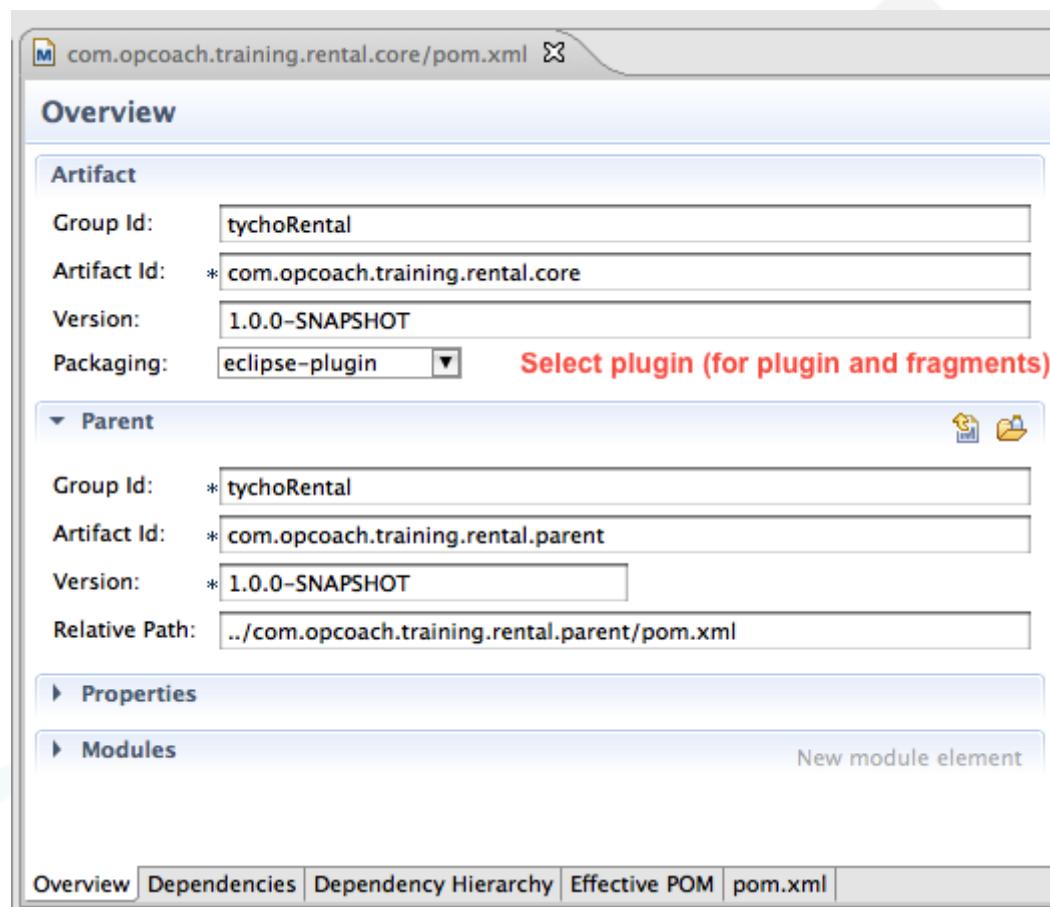
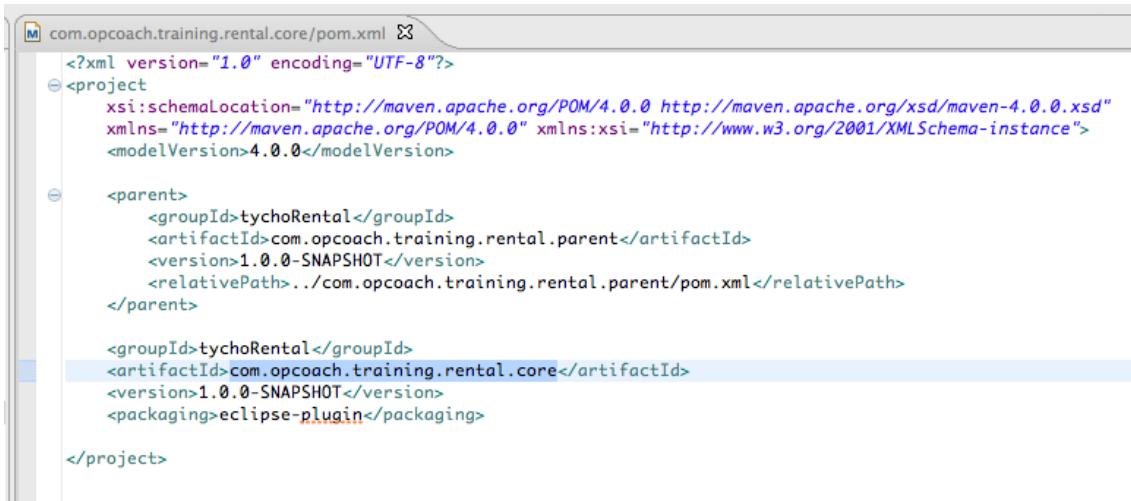


Image 231 Project pom overview

Contenu du pom projet (xml)



```

<?xml version="1.0" encoding="UTF-8"?>
<project
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
    xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>tychoRental</groupId>
        <artifactId>com.opcoach.training.rental.parent</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <relativePath>../com.opcoach.training.rental.parent/pom.xml</relativePath>
    </parent>

    <groupId>tychoRental</groupId>
    <artifactId>com.opcoach.training.rental.core</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>eclipse-plugin</packaging>
</project>

```

Image 232 Project pom XML contents

Le pom parent

- Le pom parent est référencé par tous les pom de projets
- Il contient la version de tycho, la target platform, les plateformes à supporter...
- Il référence les modules à compiler
- Il se place dans un projet maven dédié
- Son packaging est 'pom'

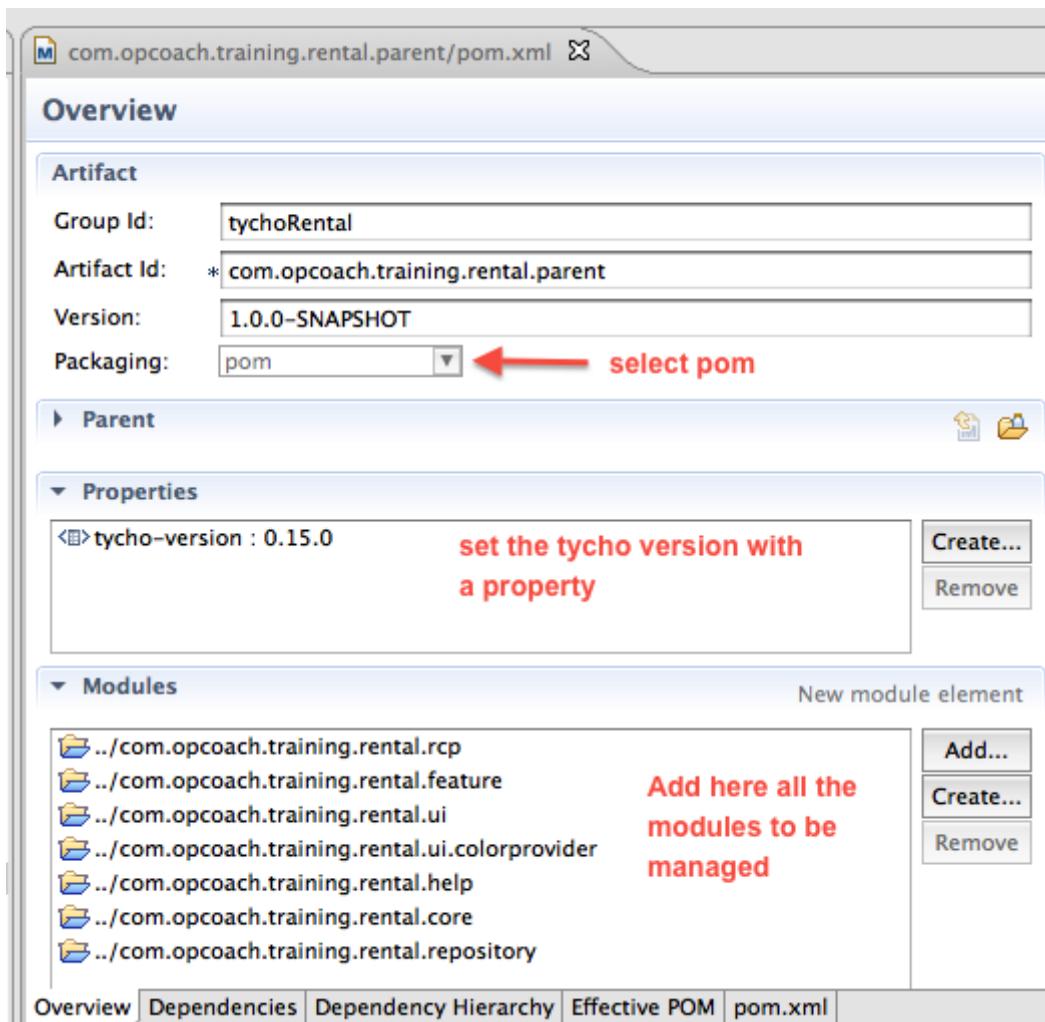


Image 233 The parent pom overview

Pom Parent entête (XML)

L'entête du fichier parent est classique, avec un packaging de type 'pom'



```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>tychoRental</groupId>
  <artifactId>com.opcoach.training.rental.parent</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>pom</packaging>

```

Image 234 Parent pom main part

Pom Parent / Repositories

On indique dans cette partie la target plateforme

```

13
14    <repositories>
15        <repository>
16            <id>luna</id>
17            <layout>p2</layout>
18            <url>http://download.eclipse.org/releases/luna</url>
19            <!-- Ou sous forme de fichier local : -->
20            <!-- <url>file:///Users/olivier/repo/luna</url> -->
21        </repository>
22    </repositories>
23

```

Image 235 Tycho repository

Pom Parent / Build part

On indique ici les plugins maven à utiliser (ie : tycho, gestion de target...)

```

-->
23    <build>
24        <plugins>
25            <plugin>
26                <!-- enable tycho build extension -->
27                <groupId>org.eclipse.tycho</groupId>
28                <artifactId>tycho-maven-plugin</artifactId>
29                <version>${tycho-version}</version>
30                <extensions>true</extensions>
31            </plugin>
32
33            <plugin>
34                <groupId>org.eclipse.tycho</groupId>
35                <artifactId>target-platform-configuration</artifactId>
36                <version>${tycho-version}</version>
37                <configuration>
38                    <environments>
39                        <environment>
40                            <os>win32</os> <ws>win32</ws> <arch>x86_64</arch>
41                        </environment>
42                        <environment>
43                            <os>macosx</os> <ws>cocoa</ws> <arch>x86_64</arch>
44                        </environment>
45                    </environments>
46                </configuration>
47            </plugin>
48        </plugins>
49    </build>

```

enter the tycho ID :
`org.eclipse.tycho`

**Additional
plugins added
to maven**

Parent Pom build part

Pom Parent / Modules

Indiquer dans cette partie tous les modules à compiler :
plugins, features, repositories, test-plugin, target project ...

```

51   <!-- the modules that should be built together -->
52<modules>
53   <module>../com.opcoach.training.rental.rcp</module>
54   <module>../com.opcoach.training.rental.feature</module>
55   <module>../com.opcoach.training.rental.ui</module>
56   <module>../com.opcoach.training.rental.ui.colorprovider</module>
57   <module>../com.opcoach.training.rental.help</module>
58   <module>../com.opcoach.training.rental.core</module>
59   <module>../com.opcoach.training.rental.repository</module>
60</modules>
  
```

Image 236 Parent pom module part

Target platform gérée avec un .target

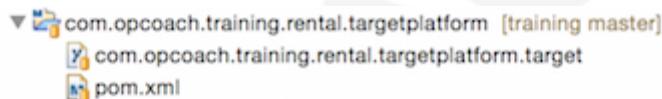
Il est possible aussi de référencer un fichier .target.

Il faut créer un projet spécifique contenant :

- Le fichier .target (même nom que l'artifact)
- Un pom spécifique pour la target

Puis il faut mettre à jour le pom parent :

- référencer ce projet dans la partie module
- indiquer la target dans la configuration du plugin tycho



Target project

Le pom du projet est de packaging **eclipse-target-definition** :

```

1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
3   <modelVersion>4.0.0</modelVersion>
4
5<parent>
6   <groupId>tychoRental</groupId>
7   <artifactId>com.opcoach.training.rental.parent</artifactId>
8   <version>1.0.0-SNAPSHOT</version>
9   <relativePath>../com.opcoach.training.rental.parent/pom.xml</relativePath>
10</parent>
11
12<artifactId>com.opcoach.training.rental.targetplatform</artifactId>
13<packaging>eclipse-target-definition</packaging>
14<version>3.7.2-SNAPSHOT</version>
15
16</project>
  
```

Pom for target project

Il faut également indiquer la target dans la configuration tycho

(cela remplace la partie <repositories> en début du pom parent) :

```

27
28      <plugin>
29          <groupId>org.eclipse.tycho</groupId>
30          <artifactId>target-platform-configuration</artifactId>
31          <version>${tycho-version}</version>
32          <configuration>
33              <dependency-resolution>
34                  <optionalDependencies>ignore</optionalDependencies>
35              </dependency-resolution>
36              <target>
37                  <artifact>
38                      <groupId>tychoRental</groupId>
39                      <artifactId>com.opcoach.training.rental.targetplatform</artifactId>
40                      <version>3.7.2-SNAPSHOT</version>
41                  </artifact>
42              </target>
43              <environments>
44                  <environment><os>macosx</os><ws>cocoa</ws><arch>x86_64</arch></environment>
45                  <environment><os>win32</os><ws>win32</ws><arch>x86_64</arch></environment>
46              </environments>
47          </configuration>
48      </plugin>
49

```

Parent config with target project

Recopier un repository p2

Dans certains cas (pas d'accès internet), il peut être intéressant d'avoir une copie d'un repository P2. On peut alors le référencer dans le pom parent.

Il faut lancer deux commandes Eclipse :

- Récupérer les metadata:
 - `eclipse.exe -nosplash -verbose -application org.eclipse.equinox.p2.metadata.repository.mirrorApplication -source http://download.eclipse.org/releases/luna -destination file:///C:/p2mirrors/luna`
- Récupérer les artifacts:
 - `eclipse.exe -nosplash -verbose -application org.eclipse.equinox.p2.artifact.repository.mirrorApplication -source http://download.eclipse.org/releases/luna -destination file:///C:/p2mirrors/luna`

Cf : <https://www.bsi-software.com/scout/local-p2-mirrors-to-work-offline>

Projet du repository

- Il faut créer un projet spécifique pour le repository
- C'est un projet de nature 'update-site'
- On le nomme 'xxx.repository'
- Ce projet contient aussi le fichier .product à livrer
- **Pour le créer :**
 - créer un projet de nature 'update site project'
 - **renommer le fichier 'site.xml' en 'category.xml'**

Pom du repository

- Le pom du repository permet de créer le repository p2 et/ou les produits
- Il se stocke à la racine du projet repository

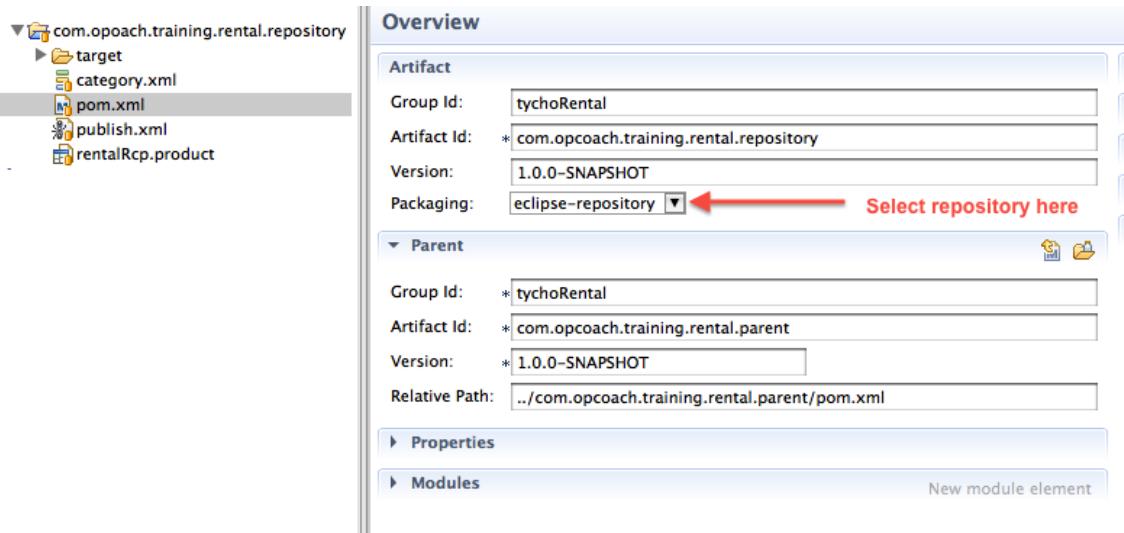


Image 237 Pom for repository

Pom du repository (xml)

Pour construire un produit :

- stocker le fichier .product à la racine du projet
- ajouter la configuration produit dans la partie build

```

19
20<build>
21    <plugins>
22        <plugin>
23            <groupId>org.eclipse.tycho</groupId>
24            <artifactId>tycho-p2-director-plugin</artifactId>
25            <version>${tycho-version}</version>
26            <executions>
27                <execution>
28                    <id>materialize-products</id>
29                    <goals>
30                        <goal>materialize-products</goal>
31                    </goals>
32                </execution>
33                <execution>
34                    <id>archive-products</id>
35                    <goals>
36                        <goal>archive-products</goal>
37                    </goals>
38                </execution>
39            </executions>
40        </plugin>
41    </plugins>
42</build>

```

Pom repository product build

***Attention: Fichier product***

Attention le fichier .product mis dans le repository doit être défini à partir de features :

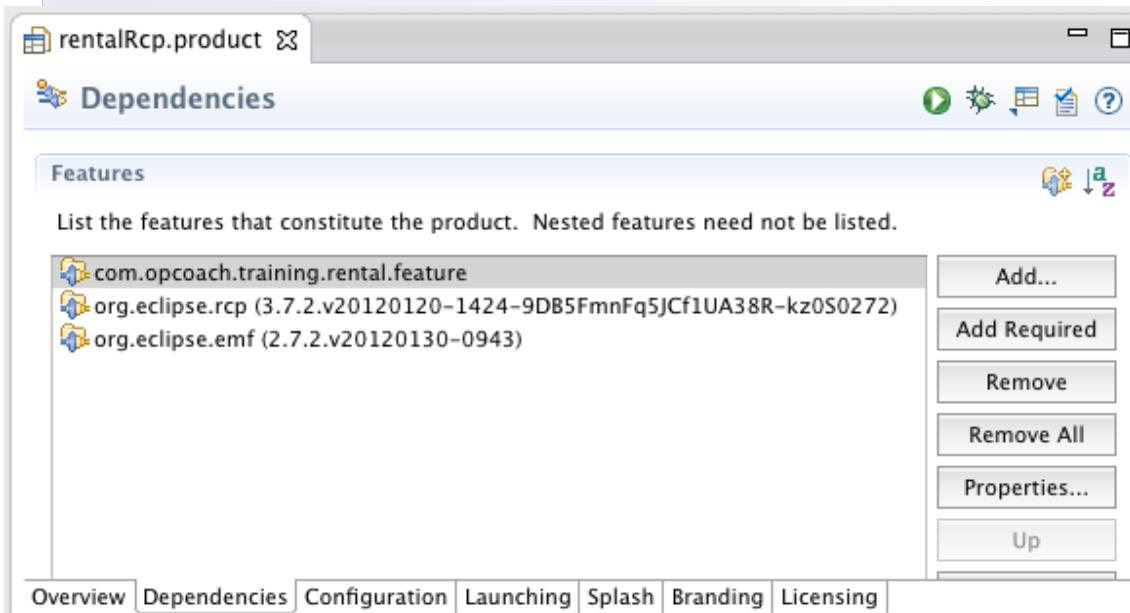
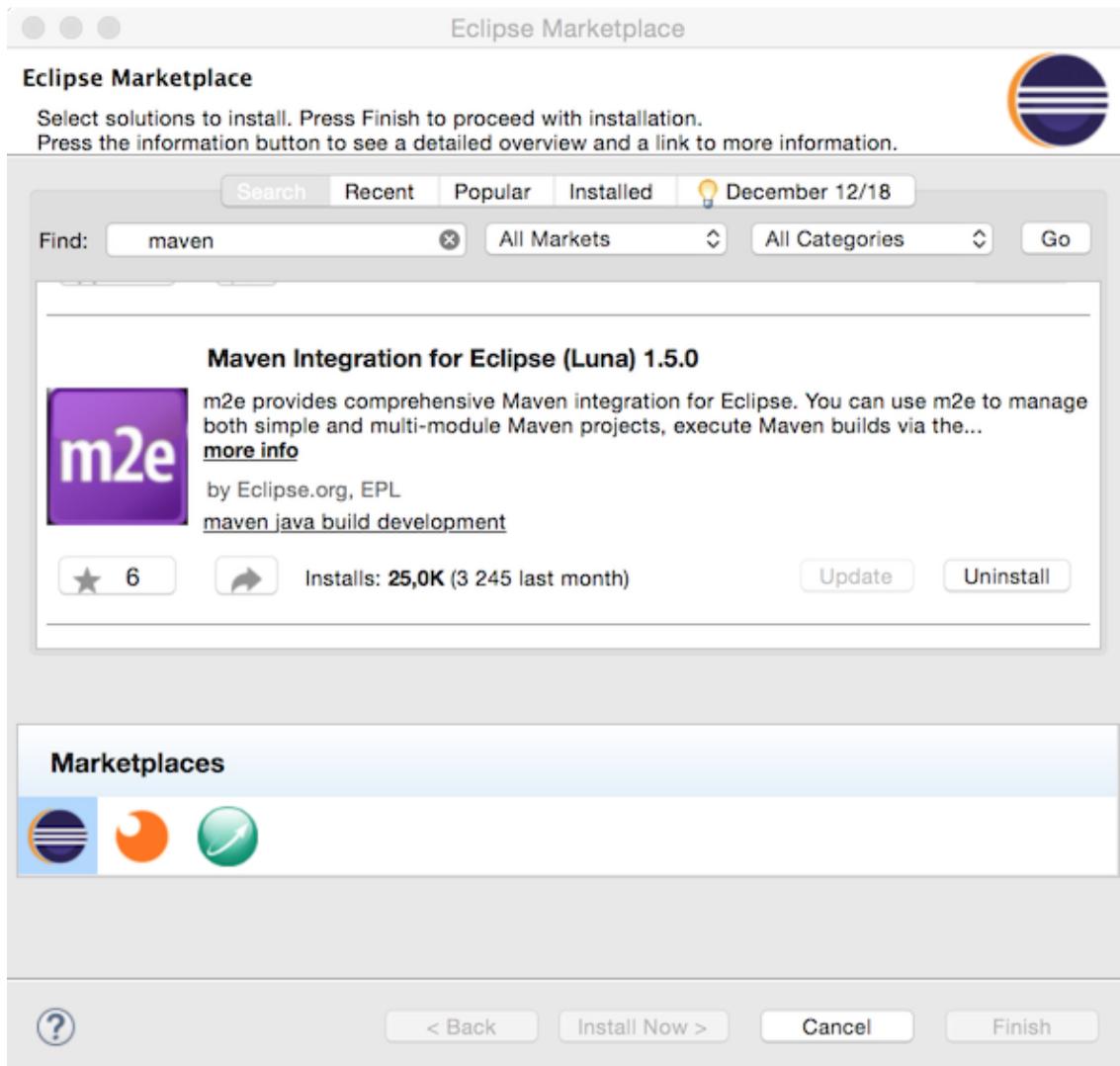


Image 238 Product file based on features

Installer maven tycho

Tycho s'installe dans le pom parent (référence au plugin maven)

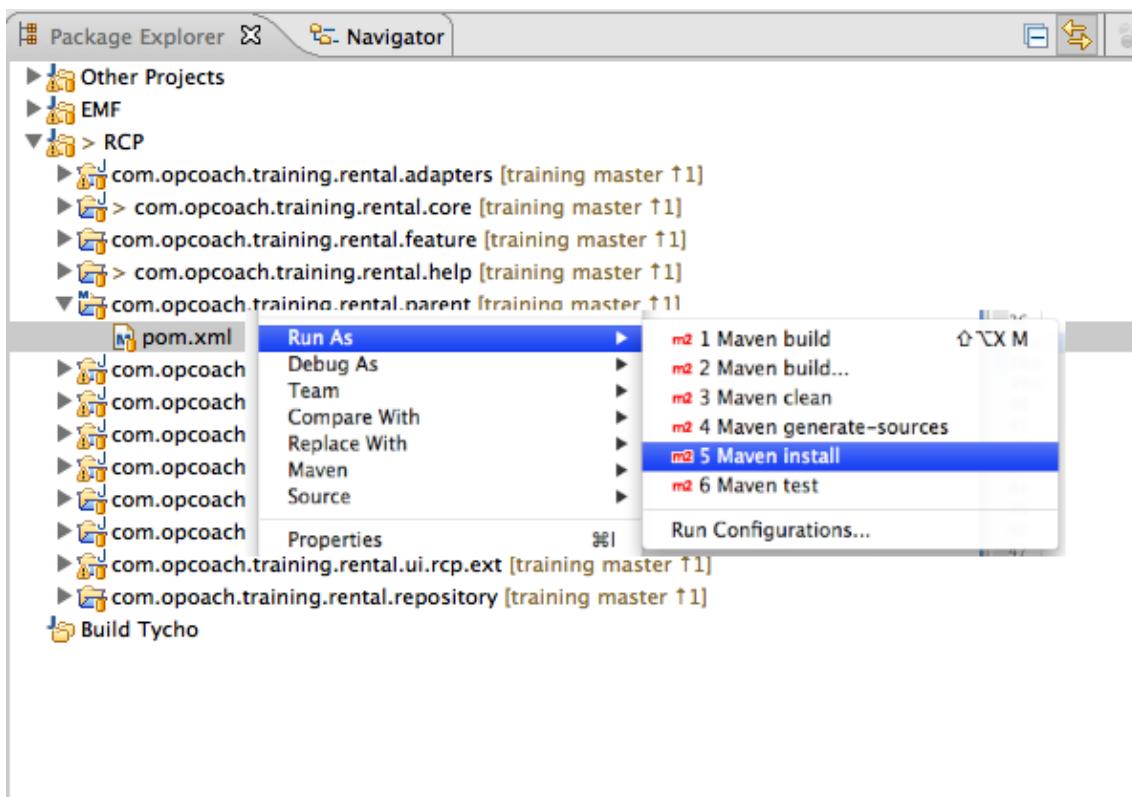
Installer maven 3 dans Eclipse en utilisant le market place :



Maven Market Place

Lancement de maven

Pour lancer les build, selectionner le pom (projet ou parent) puis Run as



Running maven

Debugger des tests lancés avec maven tycho

Les tests se définissent dans un plugin fragment avec un pom de type : eclipse-test-plugin

Pour débugger les tests :

- lancer mvn avec l'option de debug : `mvn -DdebugPort=8000 install`
- dans Eclipse se connecter en remote debugger sur le port 8000

Générer les pom automatiquement

- Il est possible d'obtenir des pom par défaut en les générant avec :

`mvn org.eclipse.tycho:tycho-pomgenerator-plugin:generate-poms`

`-DgroupId=com.opcoach.mygroup -DtestSuffix=.test`
- Le groupId est restitué dans chaque pom
- Le testSuffix permet de désigner le suffix pour les fragment de test (ne pas oublier le '.')
- Le fichier pom parent est généré à la racine du répertoire (il faut le mettre à jour)
- Les pom des features, plugins et plugins de tests référencent le parent

Cette commande est aussi expliquée sur cette page :

- http://wiki.eclipse.org/Tycho/Reference_Card#Generating_POM_files

Références

- Eclipse maven tycho web site : <http://www.eclipse.org/tycho>
- Le tutorial avec toutes les étapes :

<http://codeandme.blogspot.ch/2012/12/tycho-build-1-building-plug-ins.html>

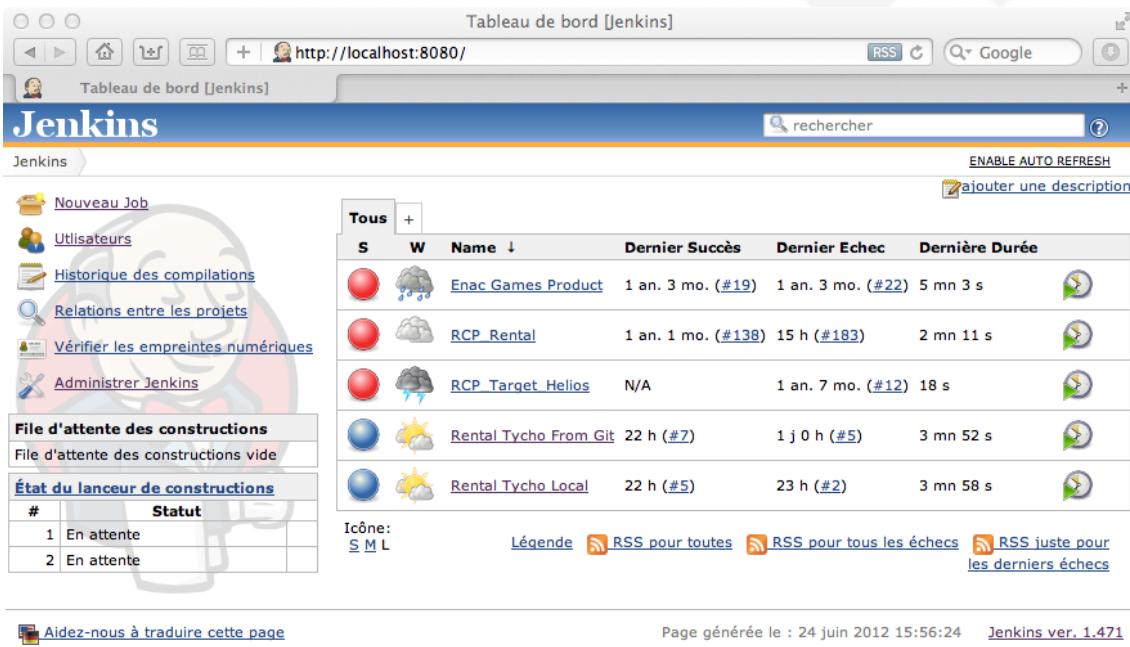
Exercice BLD 030

Livraison headless avec Tycho

B. Jenkins

Présentation

- Jenkins est un outil open source d'intégration continue
- Jenkins est la souche open source (ancien Hudson récupéré par Oracle)
- Il se télécharge à l'adresse :⁴⁵ <http://jenkins-ci.org>⁴⁶
- C'est une application web qui intègre son propre serveur
- Jenkins peut aussi fonctionner en daemon
- Jenkins est très configurable : plusieurs centaines de plugins
- Il se lance simplement : java -jar jenkins.war



The screenshot shows the Jenkins home page at <http://localhost:8080/>. The dashboard displays five active projects: Enac Games Product, RCP_Rental, RCP_Target_Helios, Rental_Tycho_From_Git, and Rental_Tycho_Local. Each project entry includes a status icon, name, last success date, last failure date, and duration. Below the dashboard, there are sections for 'File d'attente des constructions' (empty) and 'État du lanceur de constructions' (two entries: 'En attente'). A sidebar on the left provides links for 'Nouveau Job', 'Utilisateurs', 'Historique des compilations', 'Relations entre les projets', 'Vérifier les empreintes numériques', and 'Administre Jenkins'. At the bottom, there are links for 'Aidez-nous à traduire cette page', 'Page générée le : 24 juin 2012 15:56:24', and 'Jenkins ver. 1.471'.

Image 239 Home page Jenkins

Projet Jenkins

- Jenkins permet de définir des projets
- Un projet permettant de réaliser un 'build'

45 - <http://hudson-ci.org/>

46 - <http://jenkins-ci.org>

The screenshot shows the Jenkins project 'Rental Tycho From Git' interface. At the top, there's a navigation bar with links like 'Retour au tableau de bord', 'État', 'Modifications', 'Espace de travail', 'Lancer un build', 'Supprimer ce Projet', 'Configurer', 'Supprimer tous les modules désactivés', 'Modules', and 'GitHub'. Below this is a sidebar with a cartoon character icon and sections for 'Historique des builds' (listing #7, #6, #5) and RSS feeds. The main content area is titled 'Projet Rental Tycho From Git' and contains sections for 'Espace de travail' and 'Changements récents'. A 'Liens permanents' section lists recent builds. At the bottom, there are links for 'Aidez-nous à traduire cette page', 'Page générée le : 24 juin 2012 15:53:47', and 'Jenkins ver. 1.471'.

Image 240 Project Definition

Build Jenkins

- Un build est un ensemble de commandes à réaliser (build steps)
- Chaque build est planifiable
- Les résultats du build sont archivés

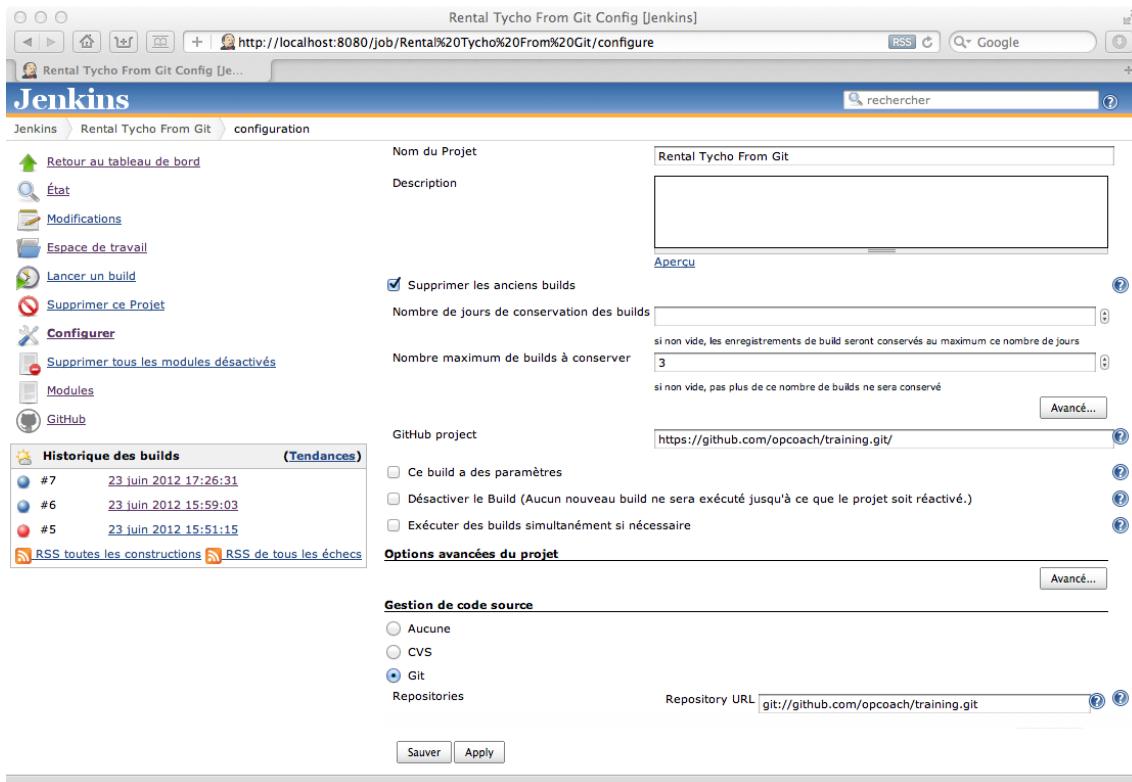


Image 241 Build configuration

Build maven

Pour créer un projet de build basé sur maven, sélectionner maven dès la création :

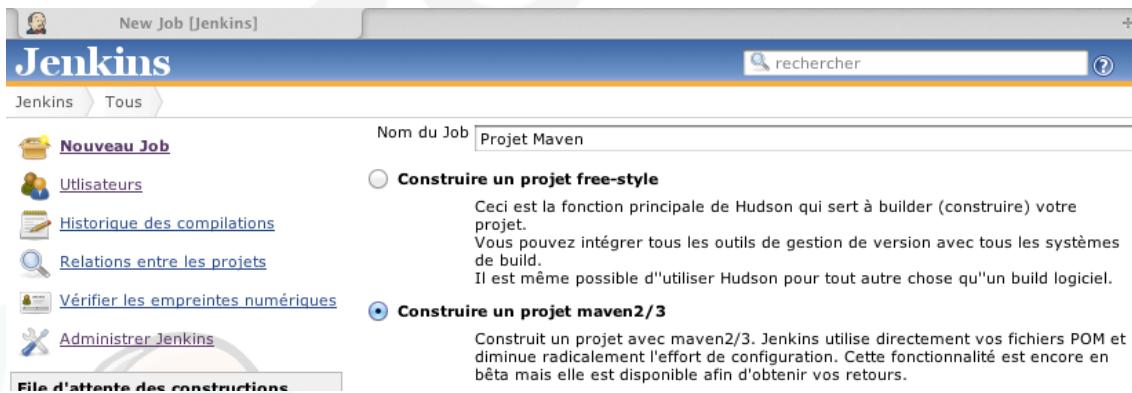


Image 242 Maven project

Configuration du projet maven :

Il suffit simplement de référencer le fichier pom dans la partie build !



Image 243 Pom link

Traces de build

Après ou pendant un build, les traces sont accessibles dans la console :

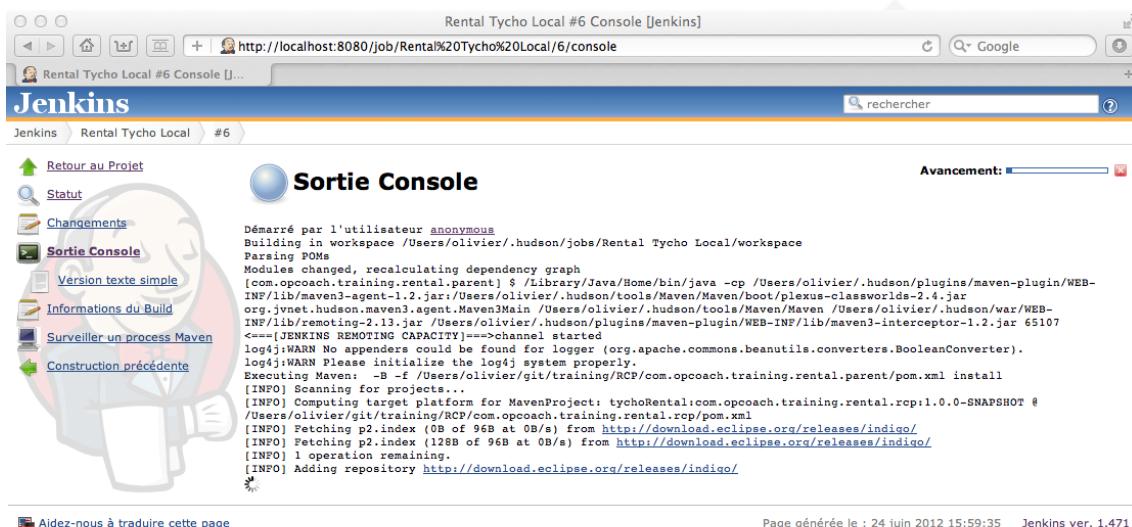


Image 244 Build Console

Exercice

BLD 100: Installer Jenkins et brancher un fichier pom

Cahier d'exercices

Eclipse 4



EAP 010 : Installation Eclipse 4	209
EAP 015 : Création d'une application Eclipse 4	210
EAP 30 : Création d'un Part dans Eclipse 4	211
EAP 032 : Utilisation du context Spy	213
EAP 035 : Création d'un Addon pour Rental	213
RCP 40 : TreeViewer	214
EAP 50 : Gestion de la sélection	214
RCP 052 : Drag And Drop	215
EAP 055 : Gestion des IAdaptables.	215
EAP 60 : Création d'une perspective	215
EAP 70 : Ajout d'une commande	216
EAP 71 : Ajout de commande conditionnelle	216
EAP 80 : Ajout de pages de préférences	217
EAP 085 : Création de fragments de modèle	218
EAP 087 : Gestion d'événements avec EventBroker	219
EAP 090 : Parcours des extensions	219
RCP 92 : Création d'un point d'extension	219
EAP 100 : Mise en place de la css	221
RCP 110 : Livraison Manuelle de l'application	221
BLD 030 : Livraison headless avec Tycho	222
BLD 100 : Jenkins, installation et tycho	223

A. EAP 010 : Installation Eclipse 4

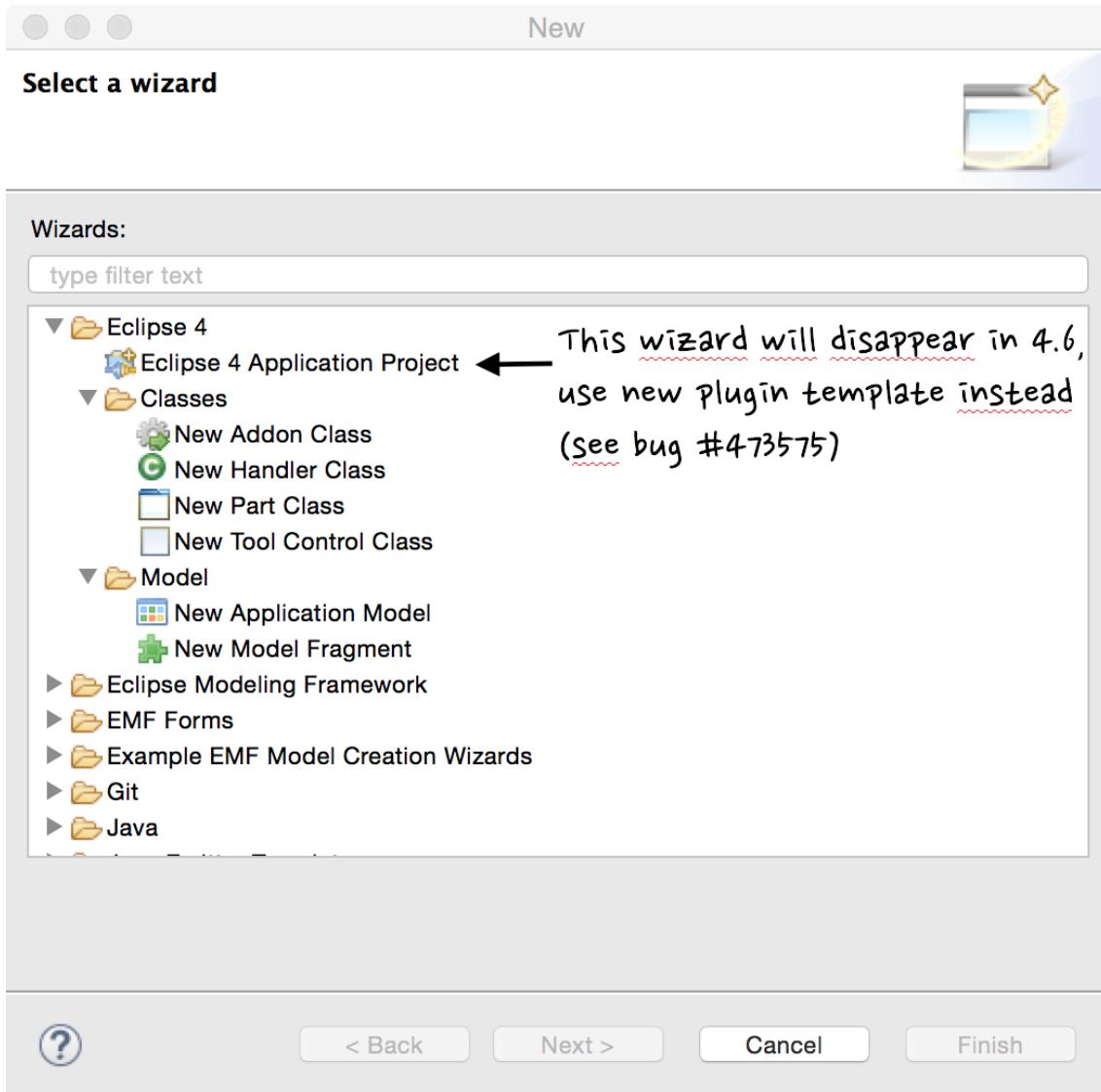
Mise en place des outils

- Installer Eclipse Mars (version Modeling ou "for RCP developper")
- Récupérer le zip des E4 spies sur le site d'OPCoach ou sur le site Eclipse
- Installer les spies
- Redémarrer Eclipse
- Constater la présence des spies avec le raccourci : **Alt Shift F9**

B. EAP 015 : Création d'une application Eclipse 4

Créer une application E4

- Créer un projet E4 (nommer le projet correctement)



e4 wizards

- Créer la launch configuration

- Lancer l'application par défaut :

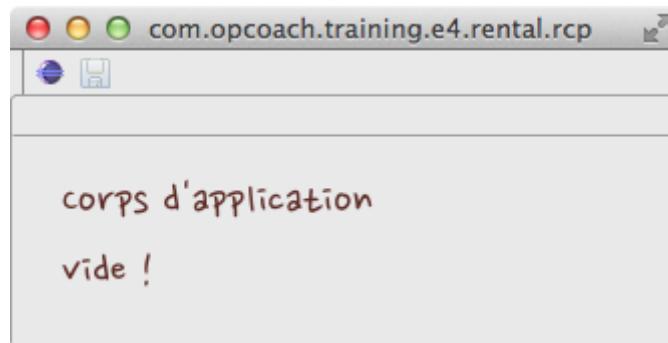


Image 245 Default application

Affichage du modèle d'application au runtime

- Ajouter le plugin **org.eclipse.e4.tools.emf.liveeditor** à la launch configuration
- Afficher le modèle d'application avec **Alt Shift F9**
- Parcourir le contenu.
- Changer la commande 'Open' en 'Ouvrir'
- Remarque : en 4.3_ tout n'est pas encore restitué en live (menu principaux par ex)

C. EAP 30 : Création d'un Part dans Eclipse 4

Description du modèle

Soit une agence de location générique.

Une agence de location (**RentalAgency**) loue des objets (**RentalObject**) à des clients (**Customer**). Une location (**Rental**) consiste à associer, pour une période donnée, un client (**Customer**) à un objet loué (**RentalObject**) dans une agence. Client (**Customer**) et Agence de location (**RentalAgency**), sont tous localisés à une adresse (**Address**).

Les différents objets sont définis par les caractéristiques suivantes :

- **RentalAgency** : nom, ensemble d'objets loués, ensemble de clients, adresse.
- **Rental** : date début et de fin, objet loué, client loueur, agence de location
- **RentalObject** : ID, désignation, disponibilité
- **Customer** : adresse, liste de locations.

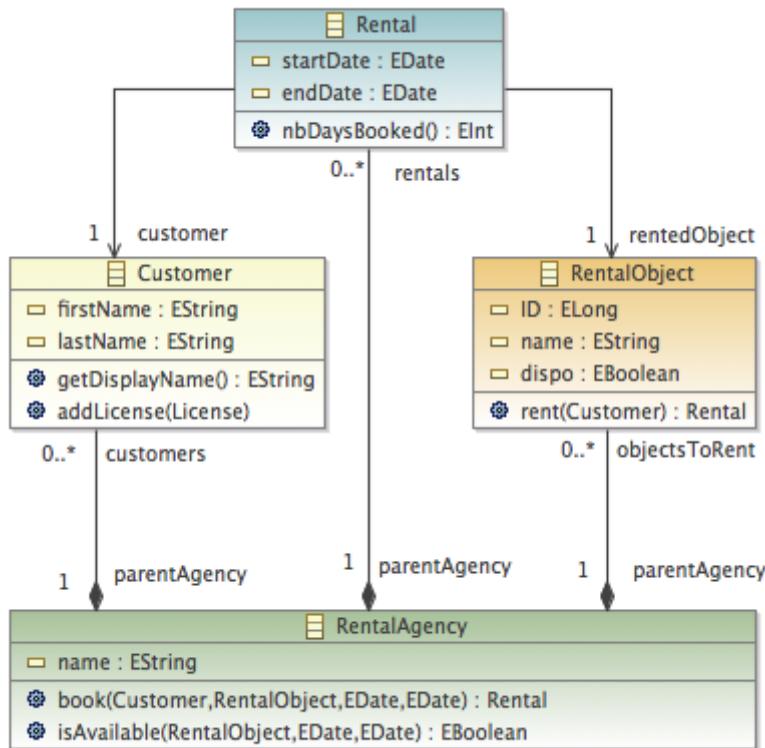


Image 246 Rental Agency Model

Mise en place de l'architecture métier

- Créer un plugin projet **xxx.rental.core** contenant le fichier rental.jar
- Le faire dépendre de **org.eclipse.emf.ecore**
- Exporter le rental.jar dans le classpath
- Exporter les packages nécessaires (rental et helpers)

Création du projet rental.ui

- Créer un plugin Project **xxx.rental.ui**
- Dépendre de rental.core et de org.eclipse.jface
- Créer un package **xxx.rental.ui.views**
- Séparer la création du contenu de l'affichage des valeurs
 - écrire la méthode **createContent**(Composite parent) qui crée les widgets (ne faire que le groupe information)
 - mettre les valeurs en dur pour l'instant

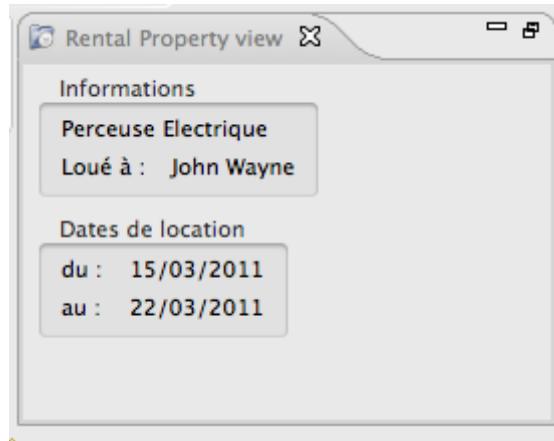


Image 247 Property View

Création du projet xxx.rental.eap

- Créer un E4 application project vide **xxx.rental.eap**
- Ajouter la dépendance sur **xxx.rental.ui**
- Créer l'ossature de l'IHM et référencer la vue créée
- Dans **RentalCoreActivator** créer un champs privé static d'agence :
 - **private static RentalAgency a = RentalAgencyGenerator.createSampleAgency();**
 - Mettre une méthode statique qui retourne l'agence
- Dans le part **RentalPropertyPart** :
 - écrire une méthode **setRental(Rental r)** qui value les widgets
 - tester en appelant **setRental** à la fin du **createContent**
 - Récupérer l'agence par le singleton et la 1ere location :
 - **setRental(RentalCoreActivator.getAgency().getRentals().get(0));**

D. EAP 032 : Utilisation du context Spy

Context Spy

- Ajouter le plugin **org.eclipse.e4.tools.context.spy** dans la launch configuration
- Afficher le context spy avec le raccourci : Alt Shift F10
- Naviguer dans les contextes

E. EAP 035 : Crédation d'un Addon pour Rental

Création de la classe d'Addon

- Créer un Addon dans le package principal de rental ui (utiliser le wizard)
- Supprimer la méthode **getAgency** de **RentalCoreActivator**
- Définir l'agence dans le contexte
- Recevoir l'agence par injection dans la vue de property
- Contrôler la présence de l'agence dans le context avec le context Explorer

F. RCP 40 : TreeViewer

Création d'une vue arborescente de l'agence

- Dans le plugin rental.ui, ajouter une vue AgencyView
- Créer un treeviewer qui affiche les informations arborescentes d'une agence de location
 - Pour l'agence : afficher son nom
 - Pour le client : afficher leur prénom, nom
 - Pour les objets à louer: afficher leur nom
 - Pour les locations courantes : afficher le nom de la personne + l'objet loué

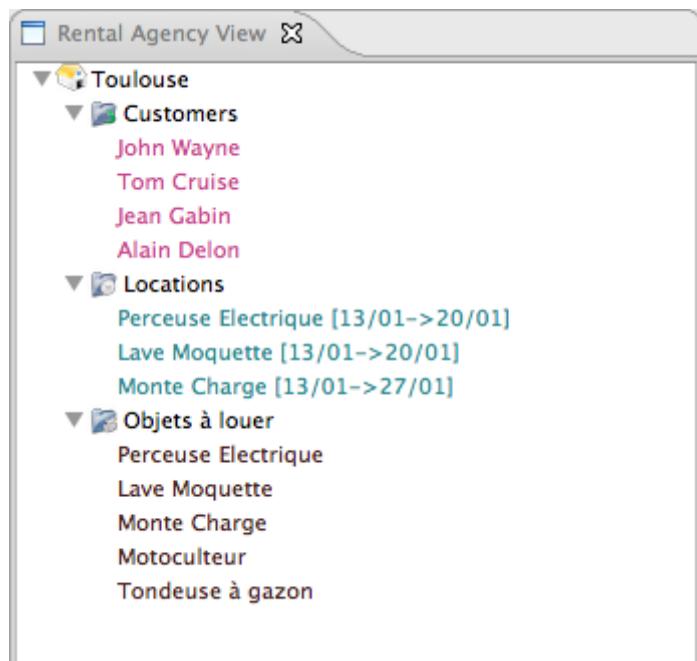


Image 248 Agency treeviewer

Gestion des couleurs et des images

En utilisant les colorRegistry, appliquer une couleur différente pour chaque type d'objet

Utiliser comme clé dans le colorRegistry, la valeur String du RGB : ex : 0,0,255 pour le RGB(0,0,255)

En utilisant un imageRegistry, positionner une image pour chaque élément charnière

G. EAP 50 : Gestion de la sélection

Relier la vue agence à la vue propriété de location.

- Déclarer le treeviewer d'agence comme étant fournisseur de sélection
- Injecter la selection dans la vue de propriété
- Gérer le refresh et la réactivité entre les deux fenêtres

H. RCP 052 : Drag And Drop

Implémenter une drag source sur un label

- Dans la fenêtre de property view, créer une dragsource sur le label de l'objet loué
- Ecrire le dragListener pour récupérer le texte
- Déplacer ensuite ce label dans un text editor et constater le transfert
- Gérer les types de transfert Text et RTF.
- Pour voir un exemple de transfert, consulter le javadoc de la méthode setContents de Clipboard

Implémenter une drag source sur un arbre

On veut pouvoir déplacer un Customer de l'arbre sous format texte, dans un éditeur de texte.

Cela doit afficher par exemple : "Customer : Nom Prenom"

Méthode :

- Dans la fenêtre de l'agence, créer une dragsource sur l'arbre
- Ecrire le dragListener pour récupérer un noeud Customer
 - Le dragListener doit référencer un SelectionProvider (ie : le treeviewer)
 - Le setData doit récupérer la sélection courante et la transférer si c'est un Customer
 - Le setData doit encoder le texte du customer en : Text, RTF et URL
 - Pour le format URL, générer une recherche google avec le nom du client
- Déplacer ensuite ce noeud dans un text editor et constater le transfert
- Déplacer le noeud dans un navigateur et constater que la recherche s'effectue
- Pour changer l'image de déplacement consulter la classe **DragSourceEffect**

I. EAP 055 : Gestion des IAdaptables.

Gérer l'adaptabilité de la fenêtre de gestion de propriété de Customer

Contexte

- La fenêtre de propriétés de l'objet Customer sait afficher un objet de nature Customer lorsqu'il est sélectionné.
- On voudrait que si on sélectionne un Rental le client correspondant soit aussi rafraîchi

Consignes

- Creer une AdapterFactory qui sait transformer un objet 'Rental' en objet Customer
- Gérer l'adaptabilité dans la gestion de la sélection de la fenêtre de propriété Customer

J. EAP 60 : Crédation d'une perspective

Perspective de location.

- Assembler les vues Agency et Rental dans une perspective dans le modèle d'application

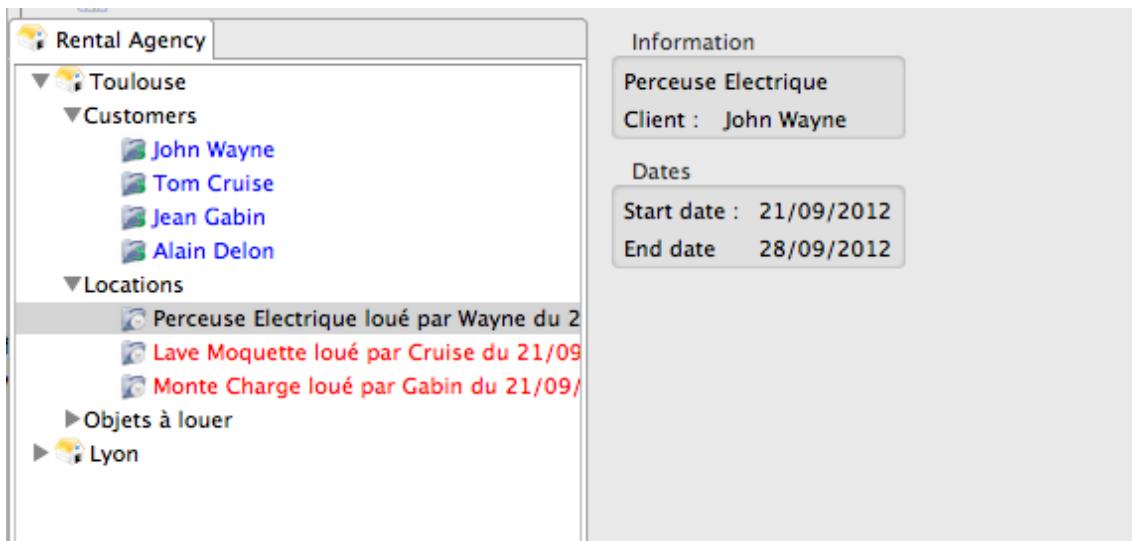


Image 249 Rental Perspective

K. EAP 70 : Ajout d'une commande

Branchemet d'une commande simple

- Créer une nouvelle commande pour afficher un message
- Créer le handler pour afficher un message dialog
- Ajouter la commande dans le menu principal
- Ajouter la commande dans le menu de la vue des agences
- Ajouter la commande dans le popup (menu contextuel) de la vue des agences
- Ajouter la commande dans toutes les vues en utilisant une menu contribution

L. EAP 71 : Ajout de commande conditionnelle

Ajout de commandes conditionnelles sur Objet

On veut ajouter une commande Copy Customer

- qui doit surcharger la commande Copy générique
- qui doit ne s'afficher que si un Customer est sélectionné
- implémenter le code de copy (doit permettre de coller le nom du client dans un text edit externe)

Consignes :

- créer une expression de sélection d'un customer
- utiliser l'ID de la commande générique : `org.eclipse.ui.edit.copy`
- Pour le code de copy, consulter le javadoc de la méthode `setContents` de Clipboard

M. EAP 80 : Ajout de pages de préférences

Préférences de couleurs

- Créer une page de préférences permettant de choisir les couleurs pour chaque objet
- Utiliser ces préférences pour l'affichage
- Gérer le rafraîchissement de l'arbre

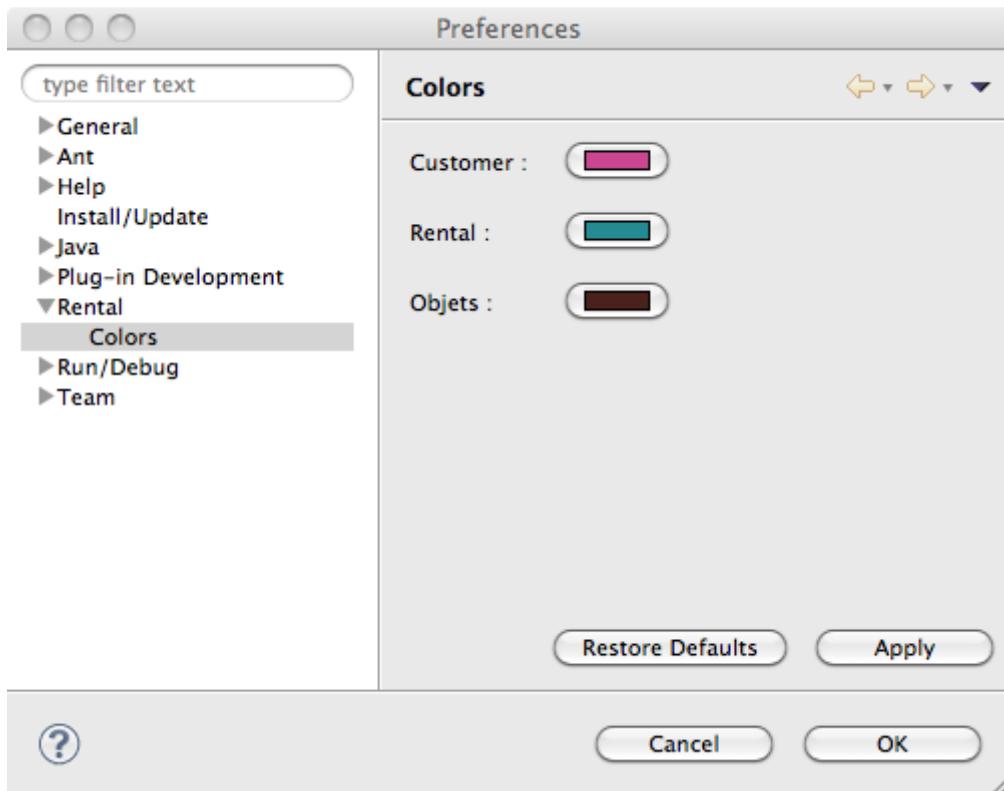


Image 250 Color preferences

Etapes à suivre pour la création de la page de préférence :

- Copier le projet de e4 preferences du github : <https://github.com/opcoach/e4Preferences.git>⁴⁷
- Faire dépendre xxx.rental.ui de ce projet
- Créer la page de préférence
 - créer une extension de e4 préférence Page dans le plugin xxx.rental.ui
 - coder la page
- Définir le préférence Store
 - ajouter la définition du preferenceStore dans l'Addon (plugin rental.ui) et le mettre dans le contexte (pas obligatoire)
- Gérer la commande d'affichage de préférences
 - créer la commande de préférence (org.eclipse.ui.window.preferences) dans le plugin rental.eap
 - la relier au handler fournit par le plugin e4Preferences
 - ajouter la commande de préférences dans le menu principal de l'application
- Tester la page

47 - <https://github.com/opcoach/e4Preferences.git>

- tester l'affichage et le bon fonctionnement des préférences (sauvegarde notamment)
- constater la présence du PreferenceStore dans le contextExplorer
- Prendre en compte les valeurs de préférences :
 - utiliser la méthode getColor indiquée dans la formation (partie JFace/ressources)
 - dans le labelProvider de l'arbre mettre à jour le getForeground
- Gérer les valeurs par défaut des préférences
 - ajouter une extension de org.eclipse.core.runtime.preferences dans le fragment et mettre des valeurs par défaut
- Rafraîchir l'arbre
 - dans le part de l'arbre d'agences recevoir par injection les valeurs de préférence
 - faire un refresh de l'arbre
 - ajouter les méthodes equals et hashCode sur le Node de l'arbre pour éviter le repliement

N. EAP 085 : Création de fragments de modèle

1. Création d'un addon d'initialisation

- Créer un addon pour publier les objets métiers
- Créer le fragment et le brancher sur l'application
- Relancer l'application et constater la prise en compte.

2. Création de la vue Customer

- Créer un nouveau plugin : xxx.rental.ui.customer
- Créer le Part Customer Properties en utilisant Window Builder
- Faire la mise à jour de l'écran avec le Data binding
- Faire réagir la fenêtre à la sélection courante

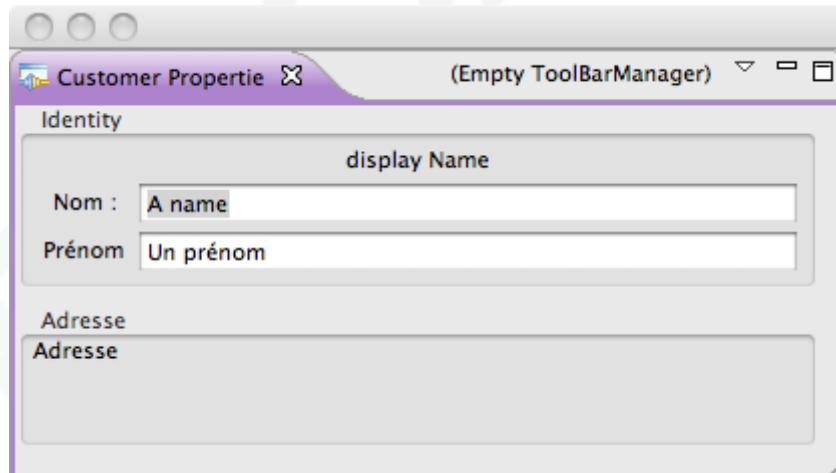


Image 251 Customer Properties

Insertion de la vue dans l'application

Créer le fragment de modèle référençant la vue et se connectant au sash container de l'appli
Lancer l'application avec le plugin customer (faire une autre launch config)

O. EAP 087 : Gestion d'événements avec EventBroker

Notifier la copie d'un client

- Envoyer un événement relatif à la copie d'un customer
- Recevoir l'événement et afficher le nom du client copié
- Dans l'event spy, filtrer pour visualiser l'événement

P. EAP 090 : Parcours des extensions

Retrouver toutes les extensions d'adapter Factory

- Au démarrage de RentalUI, faire une méthode d'affichage de toutes les extensions du points d'extension 'org.eclipse.core.runtime.adapters'
 - Afficher le résultat sous la forme :
- ```
pluginID, nom de la classe de l'adapter
```

|                                                     |                                                                                     |
|-----------------------------------------------------|-------------------------------------------------------------------------------------|
| Plugin : org.eclipse.team.core                      | Adapter : org.eclipse.team.internal.core.AdapterFactory                             |
| Plugin : org.eclipse.e4.ui.workbench.swt            | Adapter : org.eclipse.e4.ui.internal.workbench.swt.SelectionAdapterFactory          |
| Plugin : org.eclipse.ui                             | Adapter : org.eclipse.ui.internal.SelectionAdapterFactory                           |
| Plugin : org.eclipse.ui                             | Adapter : org.eclipse.ui.internal.testing.PluginContributionAdapterFactory          |
| Plugin : org.eclipse.ui                             | Adapter : org.eclipse.ui.internal.testing.PluginContributionAdapterFactory          |
| Plugin : com.opcoach.training.e4.rental.ui.customer | Adapter : com.opcoach.training.e4.rental.ui.customer.RentalToCustomerAdapterFactory |
| Plugin : org.eclipse.e4.ui.workbench                | Adapter : org.eclipse.e4.ui.internal.workbench.ConfigurationElementAdapter          |
| Plugin : org.eclipse.e4.ui.workbench                | Adapter : org.eclipse.e4.ui.workbench.modeling.EObjModelHandler                     |
| Plugin : org.eclipse.core.resources                 | Adapter : org.eclipse.core.internal.resources.mapping.ResourceAdapterFactory        |

Image 252 List Adapter Extensions

## Q. RCP 92 : Crédation d'un point d'extension

Définition du point d'extension

Le TreeViewer affiche les objets selon un code de couleur géré dans les préférences

On veut maintenant pouvoir choisir une nouvelle palette selon les cas.

Le comportement initial est isolé dans la palette défaut.

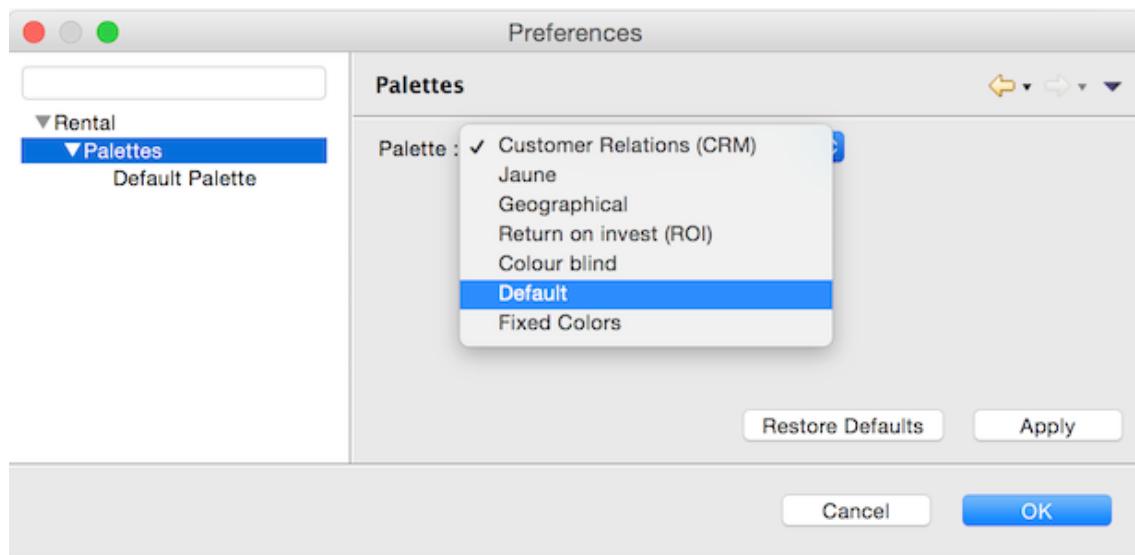
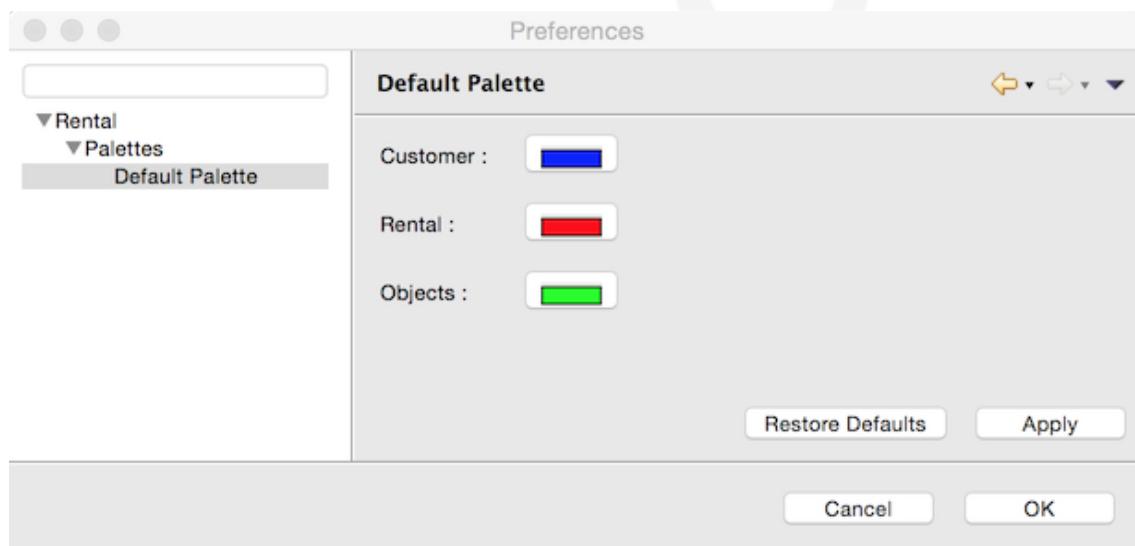


Image 253 Palette preferences

La page de préférence précédente devient la page de paramétrage de la palette défaut :



### Default Palette Preferences values

On pourra créer différents types de palettes :

- **daltonien** : pour gérer le daltonisme de l'utilisateur (couleurs fixes)
- **géographique** : pour afficher les clients proches de l'agence en vert, les autres en noir
- **rentabilité** : pour afficher en rouge les clients/rental/rentableObject non rentables, en vert sinon
- **crm** : pour afficher en vert vif les clients réguliers, en bleu les nouveaux, etc...
- **stock** : pour distinguer les objets loués des objets en stocks
- **pastel** : pour reprendre les couleurs des préférences et les pasteliser
- **fixe** : pour tout afficher en noir

### Implémentation de la structure

A partir du besoin exprimé :

- Définir la structure et les besoins de ce point d'extension.
- définir le plugin de support de ce point d'extension
- implémenter le schéma du point d'extension
- tester le point d'extension en l'étendant

### Implémentation du code de lecture du point d'extension

A partir de la structure choisie :

- Définir l'endroit où le code de lecture doit être implémenté
- Ecrire le code de lecture du point d'extension (gérer un PaletteManager).
- Proposer les méthodes permettant d'exploiter ces informations

### Mettre à jour le code métier

A partir du code ajouté, et des méthodes de gestion des palettes

- Afficher une nouvelle préférence selon les palettes présentes
- Mettre à jour le color provider pour prendre en compte la palette choisie

## R. EAP 100 : Mise en place de la css

### Création d'une css

- Dans le plugin rental ui
- Ajouter la css
- Modifier le code de la classe
- Ajouter le paramètre de css dans le point d'extension de produit

### Modification dynamique de la CSS

Créer une seconde CSS

Créer une commande qui switche de css

## S. RCP 110 : Livraison Manuelle de l'application

### Livraison manuelle sur Update Site

- Définir la feature
- Définir l'update site
- Effectuer la livraison
- Tester l'update site

### Livraison manuelle sous forme de produit

- Définir un produit, à partir de la launch configuration rcp (qui fonctionne)
- Indiquer le splash screen, les icônes, le nom du lanceur...
- Exporter le product et constater le fonctionnement

### Livraison manuelle sous forme de produit pour d'autres plateformes

- Dézipper le deltapack correspondant à la version
- Ajouter le delta pack dans la target plateform
- Reexporter le product, et constater que l'on peut choisir la plateforme
- Livrer pour windows et mac

## T. BLD 030 : Livraison headless avec Tycho

### Installation maven

- Installer maven dans l'eclipse de développement via le market place :

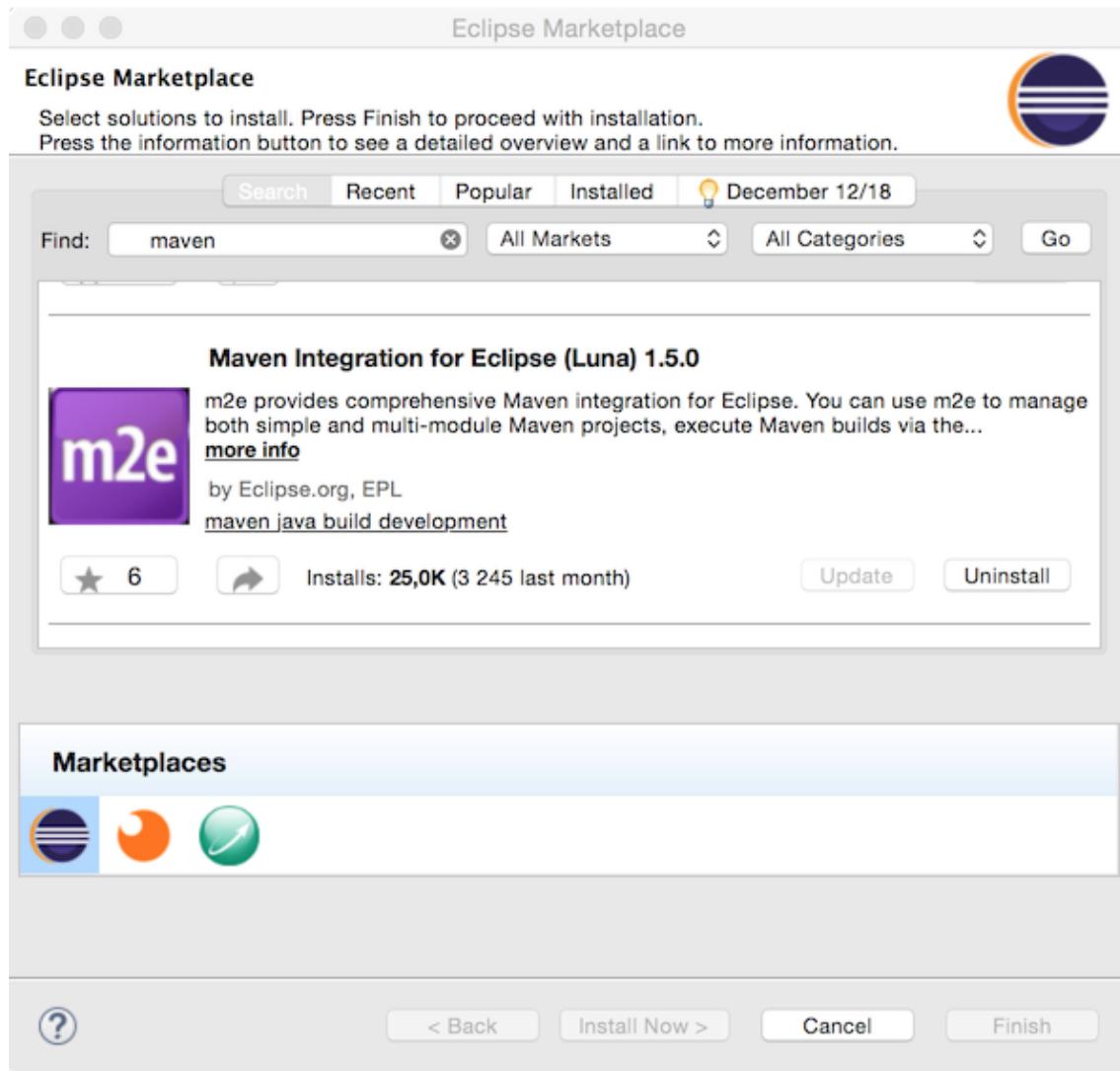


Image 254 Maven Market Place

### Installation du pom parent

**Remarque** : vous pouvez récupérer des exemples de fichiers pom dans les projets de training sur github opcoach :

[https://github.com/opcoach/training/tree/master/RCP<sup>48</sup>](https://github.com/opcoach/training/tree/master/RCP)

Créer un projet maven de nom 'xxx.rental.parent'

Créer un pom correspondant au paramétrage du parent :

- indiquer l'entête (groupId, artifactId, packaging=pom)
- fixer la properties tycho-version
- indiquer un repository

48 - <https://github.com/opcoach/training/tree/master/RCP>

- ajouter les plugin tycho (tycho-maven-plugin et target-platform-configuration)
- ajouter les modules en référençant les projets à compiler (path relatif)

#### Installation des pom projets

- Créer un fichier pom.xml à la racine de chaque projet (plugin, fragment, feature...)
- Fixer le packaging selon sa nature (feature, eclipse-plugin, eclipse-test-plugin)
- Référencer le projet maven parent
- Tester en lançant maven sur chaque projet

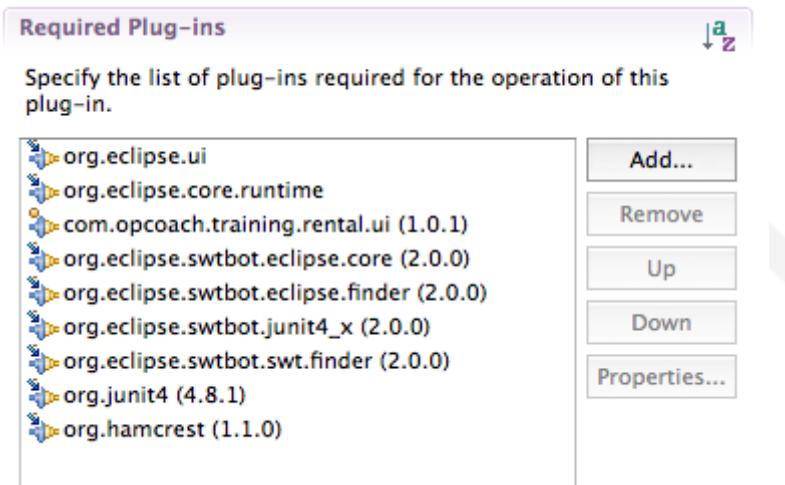


Image 255 SWTBot dependencies

#### Installation du pom repository

- Créer un fichier pom dans le projet repository (packaging eclipse-repository)
- Ajouter un fichier .product
- Ajouter le projet repository dans les modules du projet parent

## U. BLD 100 : Jenkins, installation et tycho

#### Téléchargement et lancement

- Télécharger jenkins à partir de l'adresse <http://jenkins-ci.org/><sup>49</sup>
- Installer le fichier war dans un répertoire tools/jenkins
- Lancer le war avec la commande : java -jar jenkins.war
- Lancer un navigateur internet et entrer l'url : http://localhost :8080 :

#### Configuration

Dans la rubrique Administration, effectuer la configuration système (path java, ant, svn, git...)

#### Création du projet de build maven

Ajouter un nouveau projet de type maven

Configurer un nouveau build en référençant le pom parent

49 - <http://hudson-ci.org/>