

La gestion des ressources (suite)

LA GESTION DES RESSOURCES (SUITE) 1

CONTENU DE LA SECTION 2

LES PROPRIETES..... 3

le concept de Property..... 3

quelques méthodes de gestion de Property..... 3

un exemple simple 3

LE MARQUAGE DES RESSOURCES (1) 4

le concept de Marker..... 4

LE MARQUAGE DES RESSOURCES (2) 5

la création d'un Marker (et l'association à une ressource)..... 5

l'obtention d'un Marker 5

COMPLEMENTES SUR LE MARQUAGE DES RESSOURCES 6

la définition de nouveaux types de Marker..... 6

LA REACTION AUX MODIFICATIONS (1) 7

les principales interfaces et classes 7

LA REACTION AUX MODIFICATIONS (2) 8

La représentation des modifications de ressources..... 8

LA REACTION AUX MODIFICATIONS (3) 9

un exemple simple 9

le traitement récursif des ResourceDelta 9

Les Properties

le concept de Property

- objet nommé associé à une ressource
- Session Property vs Persistant Property
- possibilité d'associer une Property à n'importe quelle ressource
- Property désignée par un **QualifiedName** : couple formé d'un **identificateur global, nom local**

souvent l'id du plugin

quelques méthodes de gestion de Property

```
void setSessionProperty(QualifiedName key, Object value)
Object getSessionProperty(QualifiedName key)
void setPersistentProperty(QualifiedName key, Object value)
Object getPersistentProperty(QualifiedName key)
```

ATTENTION !!! méthodes de **IResource**

un exemple simple

```
.....
protected static final String TOOL_PLUGIN = "org.objectweb.examplePlugin";
QualifiedName qName = new QualifiedName(TOOL_PLUGIN, "property1");
try {
    Object propObject = ..... //obtention-création de propert
    resource. setSessionProperty(qName, propObject);
} catch(CoreException ex) { ..... }
```

Le marquage des ressources (1)

le concept de Marker

- conteneur passif d'attributs destiné à être associé à une ressource
 - types prédéfinis pour les attributs : **int**, **boolean**, **Object**
 - méthodes d'accès aux attributs
- existence de markers prédéfinis
 - Task marker (contient **priorité**, **message**, attribut "effectué")
 - Bookmark marker (contient **message**, attribut "localisation")
 - Problem marker (contient **sévérité**, **message**, attribut "effectué")
 - Text marker (contient **car début**, **car fin**, **numéro de ligne**)
 - existence de markers prédéfinis
- possibilité de construire ses propres markers
 - par composition de markers existants
 - par implémentation de nouvelles classes
- toute modification de marker est notifiée au **ResourceChangeService**



Le marquage des ressources (2)

la création d'un Marker (et l'association à une ressource)

- création selon le pattern :

```
IMarker marker = resource.createMarker("type-marker");
marker.setAttribute("attribute-name", ....);
```
- exemple

```
IMarker mark0 = resource.createMarker("IMarker.TASK");
marker.setAttribute("IMarker.PRIORITY", IMarker.PRIORITY_NORMAL);
marker.setAttribute("IMarker.MESSAGE", "ceci est message de marquage");
int mld0 = mark0.getId();
```

l'obtention d'un Marker

- création selon le pattern :

```
IMarker[] markers = null;
try {
    resource.findMarkers(IMarker.TASK, true, IResource.DEPTH_INFINITE);
} catch(CoreException ex) {
    .....
}
```

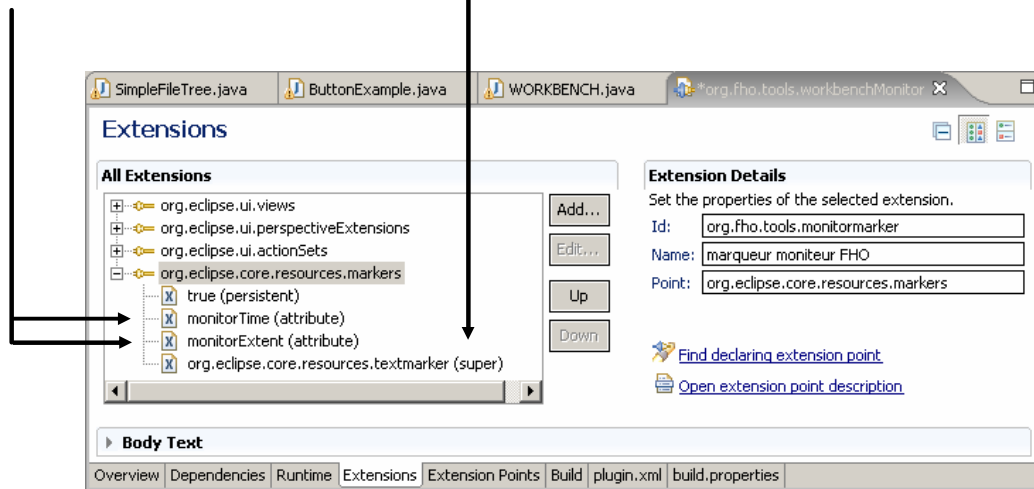
IResource.DEPTH_ZERO : dans la ressource uniquement
IResource.DEPTH_ONE : dans la ressource et ses descendants immédiats
IResource.DEPTH_INFINITE dans la ressource et tous ses descendants

includeSubType

Compléments sur le marquage des ressources

la définition de nouveaux types de Marker

- définition par extension de types existants
- introduction de nouveaux attributs

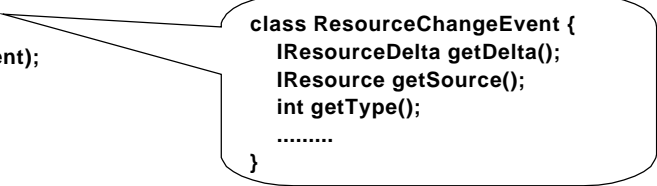


La réaction aux modifications (1)

les principales interfaces et classes

- notification de toutes les modifications de ressources à un service intégré au Workspace
 - ajout, suppression, modification, ouverture/fermeture d'un projet,, ajout/retrait de markers, démarrage/arrêt de builders)
- association d'un listener **IResourceChangeListener** au **Workspace**

```
public interface IResourceChangeListener() {
    public void resourceChanged(IResourceChangeEvent event);
}
```



```
class ResourceChangeEvent {
    IResourceDelta getDelta();
    IResource getSource();
    int getType();
    .....
}
```

valeur de Type	description	A un ResourceDelta
PRE_CLOSE	fermeture d'un projet	non
PRE_DELETE	destruction d'un projet	non
PRE_AUTOBUILD	lancement d'exécution d'un build automatique	oui
POST_AUTOBUILD	fin d'exécution d'un build automatique	oui
POST_CHANGE	modification d'une ressource	oui

La réaction aux modifications (2)

la représentation des modifications de ressources

- modifications exprimées par des **IResourceDelta**
- structure arborescente groupant les modifications (pour permettre le traitement groupé)

IResourceDelta[] getAffectedChildren(int kindMask, int memberFlags)

IMarkerDelta[] getMarkerDeltas (

IResource getResource()

int getType()

.....

valeur de Type	applicables aux ressources	commentaires
REPLACED	IFile, IFolder,IProject	
CONTENT	IFile, IFolder	le modification timestamp de la ressource a changé
MOVED_FROM	IFile, IFolder,IProject	
MOVED_TO	IFile, IFolder,IProject	
OPEN	IProject	lorsque le projet est ouvert ou fermé
TYPE	IFile, IFolder	un fichier devient un répertoire ou inversement
MARKERS	toutes les ressources	
DESCRIPTION	IProject	
SYNC	toutes les ressources	

La réaction aux modifications (3)

un exemple simple

```
public void resourceChanged(IResourceDelta aRDelta) {
    switch(aEvt.getType()) {
        case IResourceChangeEvent.PRE_CLOSE :
            System.out.println("Fermeture du projet : " + aEvt.getResource().getFullPath()); break;
        case IResourceChangeEvent.PRE_DELETE :
            System.out.println("Suppression du projet : " + aEvt.getResource().getFullPath()); break;
        case IResourceChangeEvent.PRE_AUTOBUILD :
            System.out.println("lancement d'un autobuild"); break;
        case IResourceChangeEvent.PRE_AUTOBUILD :
            System.out.println("autobuild terminé");
            .....
            break;
        case IResourceChangeEvent.CHANGE :
            System.out.println("resource modifiée");
            .....
            break;
    }
}
```

Attention si accès à des objets SWT :
Display.getCurrent().asyncExec(.....)

le traitement récursif des ResourceDelta

```
private void applyChange(IResourceDelta aDelta, ..... ) {
    //traitement de la ressource : aDelta.getResource();
    //.....
    IResourceDelta[] children = delta.getAffectedChildren();
    for (int i = 0; i < children.length; i++) applyChange(children[i], .....);
}
}
```