

Les Editeurs

LES EDITORS 1

CONTENU DE LA SECTION 2

LA DECLARATION D'UN EDITOR (1) 3

la marche à suivre 3

L'IMPLEMENTATION D'UN EDITOR (1) 4

l'interfaces à implémenter : l'interface IEditorPart 4

quelques méthodes à définir 4

la synchronisation des Editors et Viewers 4

L'IMPLEMENTATION D'UN EDITOR (2) 5

l'obtention des données : l'interface IEditorInput 5

le lancement d'un Editor 5

L'IMPLEMENTATION D'UN EDITOR (3) 6

l'activation du Save 6

la programmation de doSave() 6

L'IMPLEMENTATION D'UN EDITOR (5) 7

la programmation de doSaveAs() 7

L'IMPLEMENTATION D'UN EDITOR (6) 8

la validation des actions globales 8

COMPLEMENTS SUR L'IMPLEMENTATION D'UN EDITOR 9

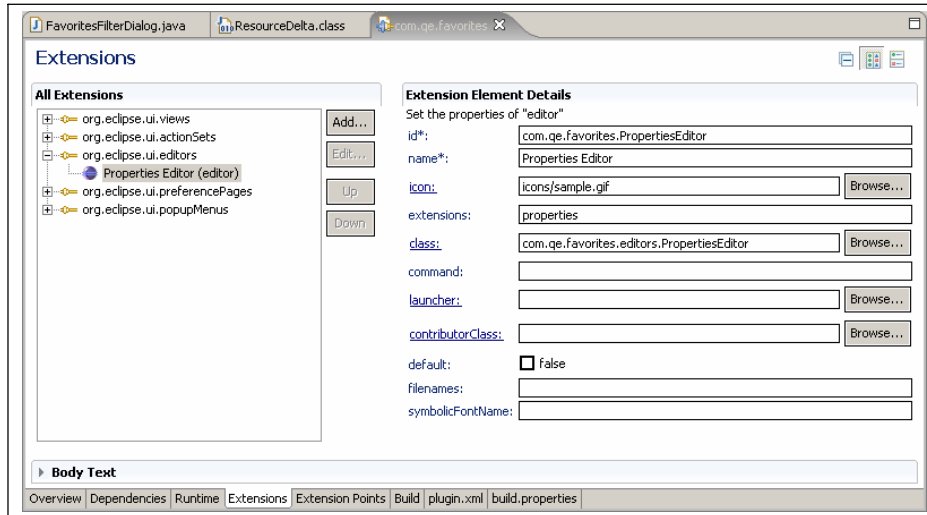
l'association de l'éditeur aux boutons Back et Forward 9

quelques éditeurs prédéfinis 9

La déclaration d'un Editeur (1)

la marche à suivre

- Sélectionner **Extensions**
- Cliquer sur **Add**
- Sélectionner **org.eclipse.ui.editors**
- Sélectionner **New > editors** et remplir le formulaire associé



L'implémentation d'un Editor (1)

l'interface à implémenter : l'interface IEditorPart

- un Editor implémente **IEditorPart** (qui étend **IWorkbenchPage**)
- un Editor étend généralement une sous classe de **EditorPart**

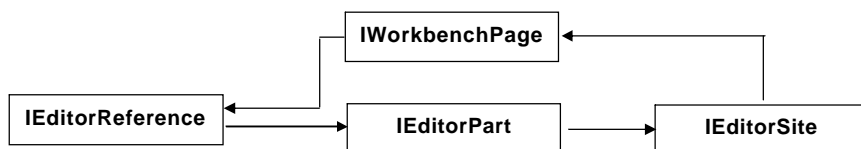
quelques méthodes à définir

```
void init(IEditorSite site, IEditorInput input)
void createPartControl(Composite parent)
boolean isDirty()
boolean isSaveAsAllowed()
void doSave(IprogressMonitor monitor)
void doSaveAs()
```

gère la partie graphique et installe généralement les listeners

la synchronisation des Editors et Views

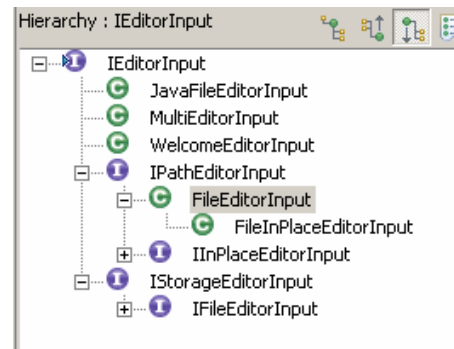
- par le biais du DP Observer : les Views sont des Listeners d'un modèle de données commun (cf View)



L'implémentation d'un Editor (2)

l'obtention des données : l'interface IEditorInput

- en pratique **FileEditorInput**
- quelques méthodes de **FileEditorInput** :
IPath getPath()
IFile getFile()
boolean exists()



le lancement d'un Editor

- par ouverture d'un fichier ayant une extension associée à l'Editor
- de façon programmatique :

```
IWorkbenchPage page = ... ;  
IFile file = ... ;  
IEditorDescriptor desc = PlatformUI.getWorkbench().getEditorRegistry().getDefaultEditor(file.getName());  
page.openEditor(new FileEditorInput(file), desc.getId());  
.....
```

IEditorRegistry.SYSTEM_EXTERNAL_EDITOR_ID appelle l'éditeur externe du système

L'implémentation d'un Editor (3)

l'activation du Save

- **dirty = true** → **Save** activé et * dans l'onglet de l'Editor
- méthode **setDirty()** par défaut invoque **firePropertyChange()** sur le PropertyChangeService

la programmation de doSave()

- les tâches à réaliser :
 - récupérer l'**EditorInput** (soit à partir d'un champ initialisé dans **init()**, soit directement)
 - récupérer l'**IFile** correspondant
 - connecter les données à sauvegarder sous la forme d'un **InputStream**
 - mettre l'attribut **dirty** à **false**

```
public void doSave(IProgressMonitor monitor) {  
    IFile file = ((FileEditorInput) getEditorInput()).getFile();  
    InputStream contents = ..... //obtention d'un InputStream sur les données à sauvegarder  
    try {  
        file.setContents(contents, IResource.KEEP_HISTORY, null);  
        setDirty(false);  
    } catch (CoreException e) { ..... }  
}
```

la programmation de doSaveAs()

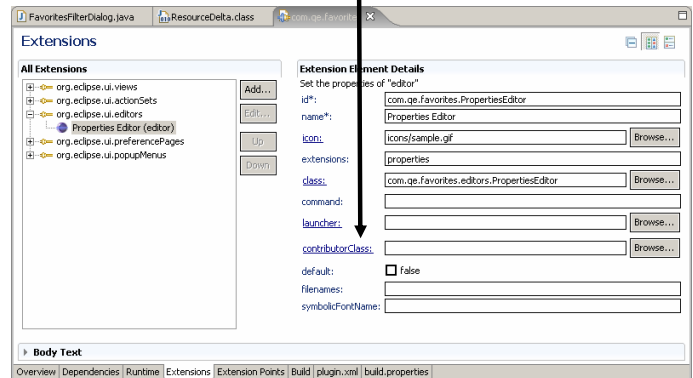
- les tâches à réaliser :
 - récupérer l'**IFile** correspondant
 - connecter les données à sauvegarder sous la forme d'un **InputStream**
 - setter l'**EditorInput**
 - mettre l'attribut **dirty** à **false**

```
public void doSaveAs() {  
    FileDialog fDialog = new FileDialog(getSite().getShell(), SWT.SAVE);  
    fDialog.setFilterExtensions(new String[] { "*.sgd", "*.SGD" });  
    fDialog.setFilterNames(new String[] { "*.sgd", "*.SGD" });  
    IPath fPath = ResourcesPlugin.getWorkspace().getRoot().getLocation();  
    fDialog.setFilterPath(fPath+"/fho-resources");  
    String fName = fDialog.open();  
    if (fName == null) return;  
    IFile file = .....  
    try {  
        file.setContents(contents, IResource.KEEP_HISTORY, null);  
        setEditorInput(new FileEditorInput(file));  
        setDirty(false);  
    } catch (CoreException e) {  
    }  
}
```

L'implémentation d'un Editor (6)

la validation des actions globales

- par le biais de **contributorClass** (implémentation de **IEditorActionBarContributor**)
- la méthode **void init(IActionBars bs, IWorkbenchPage pg)** invoque :
 - void contributeToMenu(IMenuManager mng)**
 - void contributeToToolBar(IToolBarManager mng)**
 - void contributeToCoolBar(ICoolBarManager mng)**
 - void contributeToStatusLine(IStatusLineManager mng)**
 - void setActiveEditor(IEditorPart prt)**
- définition de la méthode **setActiveEditor** associer l'action au menu



IAction action =

```
public void setActiveEditor(IEditorPart) {
    IActionBars bars = getActionBars() ;
    if (bars == null) return ;
    action.setEditor(part) ;
    bars.setGlobalActionHandler(<ActionFactory.<<CONSTANT>>.getId(), action);
    bars.updateActionBars() ;
}
```

<<CONSTANT>> vaut UNDO, REDO, SAVE,

Compléments sur l'implémentation d'un Editor

l'association de l'éditeur aux boutons Back et Forward

- chaque page maintient un historique des activations d'éditeur
- l'Editor doit implémenter l'interface **InavigationLocationProvider**
- création d'objets **NavigationLocation**
- invocation de la méthode **restoreLocation** sur un objet **NavigationLocation**

quelques éditeurs prédéfinis

- possibilité d'étendre MultiPageEditor, TextEditor

