

Les bases graphiques : la librairie « JFace »

LES BASES GRAPHIQUES : LA LIBRAIRIE « JFACE »	1
CONTENU DE LA SECTION	2
GENERALITES (1)	3
<i>les objectifs</i>	3
<i>quelques informations pratiques</i>	3
LA GESTION DES PREFERENCES (1)	4
<i>les objectifs</i>	4
<i>les éléments</i>	4
LA GESTION DES PREFERENCES (2)	5
<i>la classe PreferenceStore</i>	5
<i>quelques méthodes de PreferenceStore</i>	5
LA GESTION DES PREFERENCES (3)	6
<i>le PreferenceDialog</i>	6
<i>le classe PreferenceManager</i>	6
LA GESTION DES PREFERENCES (4)	7
<i>la classe PreferencePage</i>	7
LA GESTION DES PREFERENCES (5)	8
<i>un exemple simple</i>	8
LA CLASSE FIELDPREFERENCEPAGE (1)	9
<i>les caractéristiques</i>	9
<i>les FieldEditor</i>	9
LA CLASSE FIELDPREFERENCEPAGE (2)	10
<i>un exemple simple (suite)</i>	10
LA CLASSE FIELDPREFERENCEPAGE (3)	11
<i>un exemple simple (suite)</i>	11
LA CLASSE FIELDPREFERENCEPAGE (4)	12
<i>un exemple simple</i>	12
LES WIZARDS (1)	13
<i>les objectifs et principes</i>	13
<i>la construction d'un Wizard</i>	13
LES WIZARDS (2)	14
<i>un exemple simple</i>	14

les objectifs

- librairie complémentaire à SWT
 - abstractions masquant les widgets SWT
 - abstractions MVC (Viewer)
 - widgets additionnels (Dialog JFace, Preference, ...)
 - meilleure utilisation des ressources (Action, Registry, ...)

quelques informations pratiques

- pas de distribution bien packagée : → récupérer les .jar
 - jface.jar** dans **org.eclipse.jface_3.0.0**
 - runtime.jar** dans **org.eclipse.core.runtime_3.0.0**
 - osgi.jar** dans **org.eclipse.osgi_3.0.0**
 - jfacetext.jar** dans **org.eclipse.jface.text_3.0.0**
 - text.jar** dans **org.eclipse.text_3.0.0**

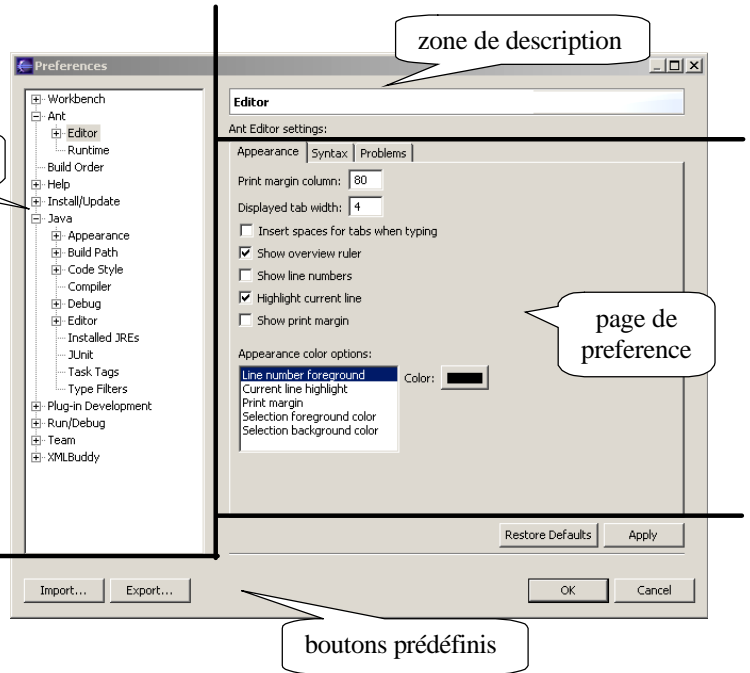
La gestion des Préférences (1)

les objectifs

- offrir un dépôt de préférences et une interface d'accès

les éléments

- le **PreferenceStore** persistant
 - table de Hash (clé, valeur)
- le **PreferenceDialog**
 - encapsule la gestion de
 - un arbre des pages dont chaque nœud (**PreferenceNode**) pointe sur une page de préférence
 - un ensemble de boutons pour la sauvegarde/restauration des valeurs, pour l'import/export ...
- le **PreferenceManager**
 - gère la correspondance entre les PreferenceNodes et les pages



La gestion des Preferences (2)

la classe PreferenceStore

- dépôt de 2 types de couples (cle, valeur) et (clé, valeur_par_défaut)
- dépôt associé à un fichier en cas de dépôt persistant
- mécanisme de notification des modifications (PropertyChangeListener)

quelques méthodes de PreferenceStore

- quelques méthodes
PreferenceStore()
PreferenceStore(String fName)

void load(InputStream in)
void save()

boolean getBoolean(String key), int getInt(String key), float getFloat(String key),, String getString(String key)
boolean getDefaultBoolean(String key), int getDefaultInt(String key),, String getDefaultString(String key)
void setValue(String key, boolean val), void setValue(String key, int val),, void setValue(String key, String val)
void setDefaultValue(String key, boolean val),, void setDefaultValue(String key, String val)

- gestion des listeners
void addPropertyChangeListener(IPropertyChangeListener l)
void removePropertyChangeListener(IPropertyChangeListener l)

```
interface IPropertyChangeListener {
    void propertyChange(PropertyChangeEvent evt)
}
```

méthodes de **PropertyChangeEvent**
Object getOldValue()
Object getNewValue()
String getProperty()

La gestion des Preferences (3)

le PreferenceDialog

- utilisation identique aux autres Dialog

```
PreferenceManager mger = new PreferenceManager();
PreferenceDialog dlg = new PreferenceDialog(sh, mger);
dlg.open();
.....
```

le classe PreferenceManager

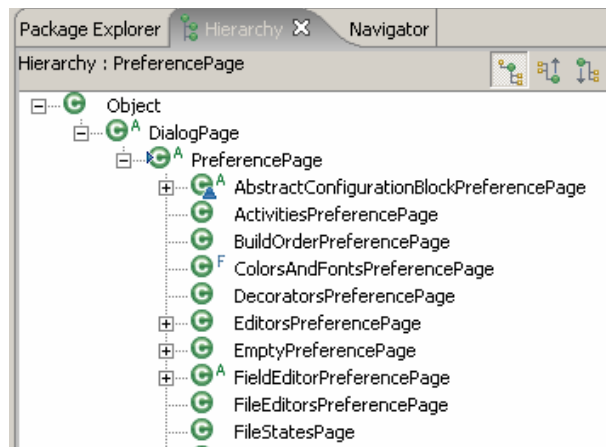
- gère l'arbre des **PreferenceNode**
- gère la correspondance entre les PreferenceNode et les PreferencePage
- désigne les noeuds via un path

le séparateur par défaut est ,

- quelques méthodes de la classe PreferenceManager
PreferenceManager(char separator), PreferenceManager()
void addToRoot(IPreferenceNode node)
boolean addTo(String path, IPreferenceNode node)
void remove(IPreferenceNode node)
List getElements(int order)

la classe PreferencePage

- implémentation de IPreferencePage
 - quelques méthodes de IPreferencePage
Control createContents(Composite parent)
boolean performOK(), boolean performCancel()
boolean okToLeave()
 - quelques méthodes additionnelles de PreferencePage
boolean performHelp()
boolean performDefaults()
Label createDescriptionLabel(Composite parent)
- + accesseurs**



La gestion des Preferences (5)

un exemple simple

```

public class PreferenceExample0 {
    public void run() {
        Display display = new Display();
        Shell shell = new Shell(display);
        createPreferences(shell);
        shell.open();

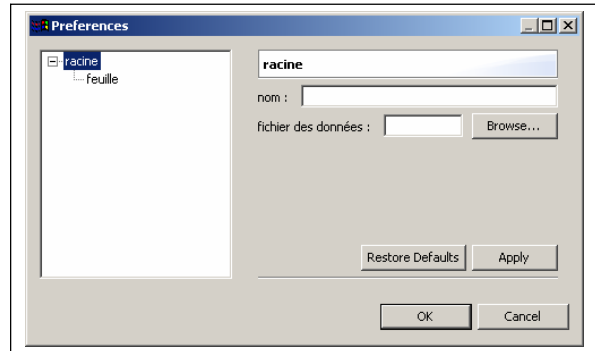
        while ( !shell.isDisposed() ) {
            if ( !display.readAndDispatch() ) display.sleep();
        }
        display.dispose();
    }

    private void createPreferences(Shell shell) {
        PreferenceManager mger = new PreferenceManager();
        PreferenceNode rNode = new PreferenceNode("r", "racine", null, new PreferencePage0());
        PreferenceNode fNode = new PreferenceNode("f1", "feuille", null, new PreferencePage1());
        mger.addToRoot(rNode);
        mger.addTo("r",fNode);

        PreferenceDialog dlg = new PreferenceDialog(null, mger);
        dlg.open();
    }

    public static void main(String[] args) { new PreferenceExample0().run(); }
}

```



label pour la description

identifiant interne

La classe FieldPreferencePage (1)

les caractéristiques

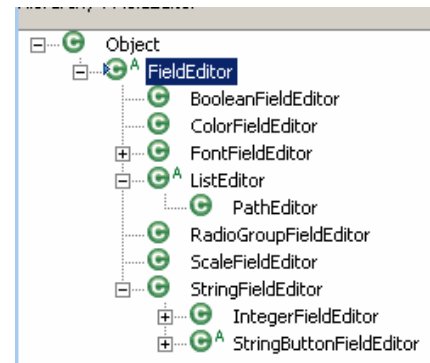
- classe abstraite
- repose sur l'utilisation des FieldEditor
- encapsule la synchronisation avec le PreferenceStore

- quelques méthodes de la classe
 - void createFieldEditors()**
 - boolean isValid()**
 - addField(FieldEditor editor)**

les FieldEditor

- éditeur spécialisé pour la saisie d'un type de données précis
- encapsule la synchronisation avec le PreferenceStore
 - les constructeurs possèdent tous :
 - un argument preferenceStoreKey
 - un argument label
 - un argument composite parent
 -

```
StringFieldEditor("org.fho.tools.Ex1.nom", "nom : ", getFieldEditorParent())
```



méthodes spécifiques aux types de données

La classe FieldPreferencePage (2)

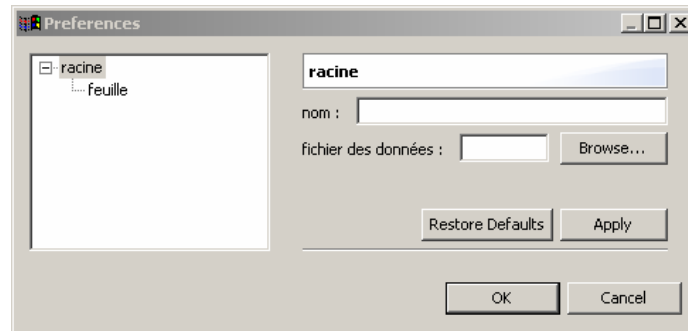
un exemple simple (suite)

```
public class PreferencePage0 extends FieldEditorPreferencePage {
    public PreferencePage0() { super(FLAT); }

    public boolean isValid() {return true; }

    protected void createFieldEditors() {
        StringFieldEditor sFE = new StringFieldEditor("org.fho.tools.Ex1.nom", "nom : ", getFieldEditorParent());
        FileFieldEditor fFE = new FileFieldEditor("org.fho.tools.Ex1.fichier", "fichier des données : ", getFieldEditorParent());
        addField(sFE);
        addField(fFE);
    }
}
```

Attention : ctor public pour osgi



La classe FieldPreferencePage (3)

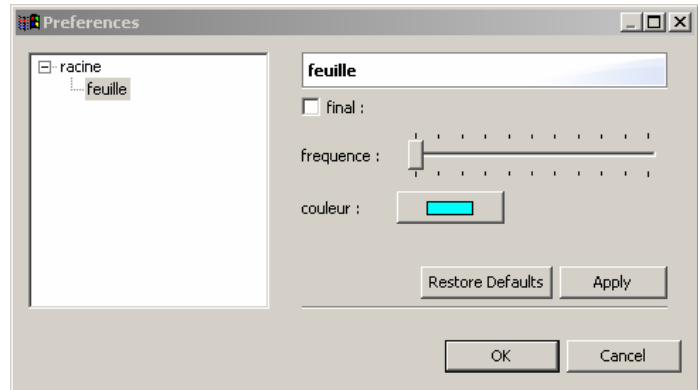
un exemple simple (suite)

```
public class PreferencePage1 extends FieldEditorPreferencePage {
    public PreferencePage1() { super("page 1", GRID); }

    public boolean isValid() {return true; }

    protected void createFieldEditors() {
        BooleanFieldEditor bFE = new BooleanFieldEditor("org.fho.tools.Ex1.final", "final : ", getFieldEditorParent());
        ScaleFieldEditor sFE = new ScaleFieldEditor("org.fho.tools.Ex1.freq", "frequence : ", getFieldEditorParent(), 0, 100, 1, 10);
        ColorFieldEditor cFE = new ColorFieldEditor("org.fho.tools.Ex1.color", "couleur : ", getFieldEditorParent());
        addField(bFE);
        addField(sFE);
        addField(cFE);
    }
}
```

Attention : ctor public pour osgi



La classe FieldPreferencePage (4)

un exemple simple

```

public class PreferenceExample0 {
    public void run() {
        Display display = new Display();
        Shell shell = new Shell(display);
        createPreferences(shell);
        shell.open();

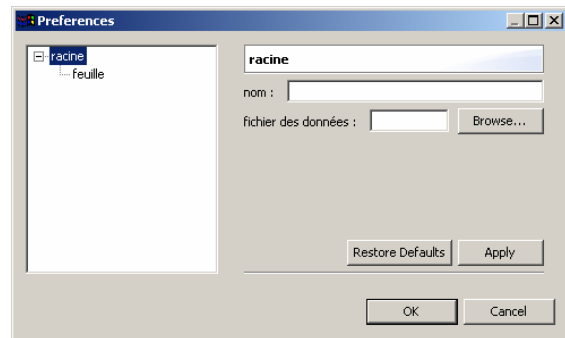
        while ( !shell.isDisposed() ) {
            if ( !display.readAndDispatch() ) display.sleep();
        }
        display.dispose();
    }

    private void createPreferences(Shell shell) {
        PreferenceManager mger = new PreferenceManager();
        PreferenceNode rNode = new PreferenceNode("r", "racine", null, PreferencePage0.class.getName());
        PreferenceNode fNode = new PreferenceNode("f1", "feuille", null, PreferencePage1.class.getName());
        mger.addToRoot(rNode);
        mger.addTo("r", fNode);

        PreferenceDialog dlg = new PreferenceDialog(null, mger);
        dlg.open();
    }

    public static void main(String[] args) { new PreferenceExample0().run(); }
}

```



label pour la description

nom de la classe
(de la page)

identifiant interne

Les Wizards (1)

les objectifs et principes

- suite de pages destinée à guider l'utilisateur
- **Wizard** : un moteur d'enchaînement pour des pages
- **WizardDialog** encapsule un moteur d'enchaînement pour des pages
- pages formées de 3 zones :
 - une zone de titre (comportant un titre + image + message)
 - une page (comportant une content area + progress area)
 - une bande contenant les boutons **Back**, **Next**, **Finish**, **Cancel**

myWizard regroupe les pages à enchaîner

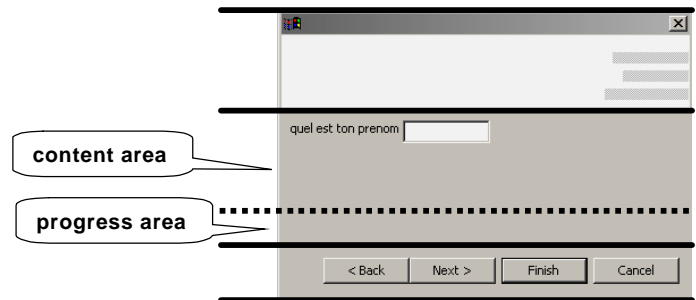
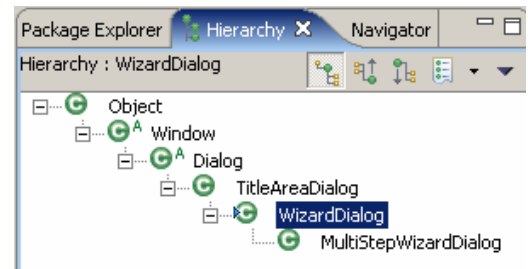
```
WizardDialog dlg = new WizardDialog(shell, myWizard) ;
int result = dlg.open() ;
```

la construction d'un Wizard

- définition par extension de la classe **Wizard**

```
void addPage(IWizardPage page)
```
- définition chaque page par extension de la classe **WizardPage**

```
String getName()
IWizardPage getNextPage(), void setNextPage(IWizardPage page),
IWizardPage getPreviousPage(), void setPreviousPage(IWizardPage page),
boolean isPageComplete(), ....
```



Les Wizards (2)

un exemple simple

```

public class JFaceDialogExample1 {

public class WizardExample0 {

    static class WPage0 extends WizardPage { ..... }

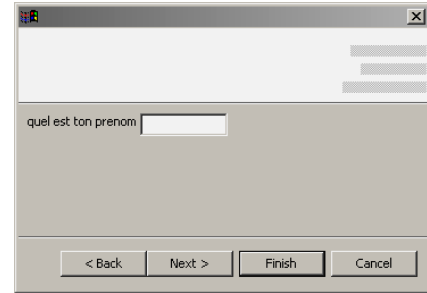
    static class WPage1 extends WizardPage {
        public WPage1() { super("seconde page"); }
        public void createControl(Composite parent) {
            Composite compo = new Composite(parent, SWT.NONE);
            compo.setLayout(new RowLayout());
            Label label = new Label(compo, SWT.NONE);
            label.setText("quel est ton prenom");
            Text txt = new Text(compo, SWT.BORDER);
            setControl(compo);
        }
    }

    static class WPage2 extends WizardPage { .....}

    static class SimpleWizard extends Wizard {
        public boolean performFinish() {
            if (.....) return false;
            return true;
        }
    }
}

```

NECESSAIRE dans chaque page



```

.....
SimpleWizard wizard = new SimpleWizard();
wizard.addPage(new WPage0());
wizard.addPage(new WPage1());
wizard.addPage(new WPage2());
final WizardDialog dlg = new WizardDialog(shell, wizard);

Button button = new Button(shell, SWT.PUSH);
button.setText("afficher Wizard");
button.addSelectionListener(new SelectionListener() {
    public void widgetSelected(SelectionEvent e) {
        int res = dlg.open();
    }
    public void widgetDefaultSelected(SelectionEvent e) {}
});
.....

```

