können für Toolbars, Top-level menus, Context menus, Status lines, Buttons und viele mehr eingesetzt werden.

Actions können deklarativ wie auch programmatisch erzeugt werden.

Action programmatisch erzeugen

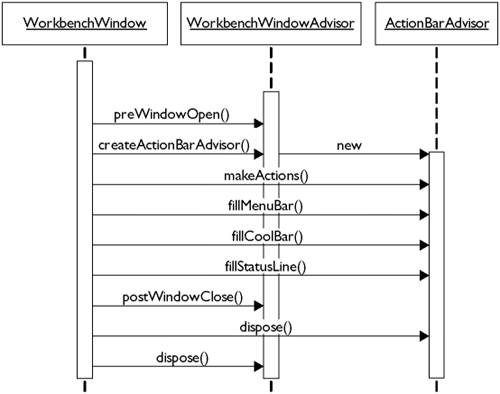


Abbildung ActionBarAdvisor

In der preOpen() Methode des WorkbenchWindowAdvisor‘s kann mittels des Methodenaufrufs getWindowConfigurer() eine Instanz des IWorkbenchWindowConfigurer geholt werden. Auf diesem kann definiert werden welche Elemente angezeigt werden sollen. Das passiert über Methodenaufrufe wie zum Beispiel setShowMenuBar(true) oder setShowCoolBar(true).

Im ActionBarAdvisor kann man nun Actions erzeugen und registrieren. Ein Beispiel dafür findet sich im folgenden Snippet:

….

@Override

protected void makeActions(IWorkbenchWindow window) {

register(ActionFactory.SAVE.create(window));

}

….

Abbildung ActionBarAdvisor makeActions

Die Registrierung der Action garantiert, dass sie gelöscht wird wenn das entsprechende Workbench Window geschlossen wird. Die Registrierung ermöglicht auch das Key Binding.

Die registrierte Action kann nun in der gewünschten Methode eingesetzt werden. Im folgenden Beispiel wird die Action benötigt um einen Menu-Eintrag in der MenuBar zu erzeugen:

….

@Override

protected void fillMenuBar(IMenuManager mm){

MenuManager saveMenu = new MenuManager(„&Speichern“, „Speichern“);

IWorkbenchAction saveAction = getAction(ActionFactory.SAVE.getId());

saveMenu.add(saveAction);

mm.add(saveMenu);

}

….

Abbildung ActionBarAdvisor fillMenuBar

Analog zum obenstehenden Beispiel kann

* + dem IStatusLineManager in fillStatusLine();
  + dem ICoolBarManager in fillCoolBar();

eine Action hinzugefügt werden. Zusätzlich zu den Actions können den Manager‘s auch IContributionItems hinzugefügt werden.

In den Beispielen oben kamen vom Framework zur Verfügung gestellte Actions zum Einsatz. Es ist jedoch auch mögliche eigene Actions zu definieren. Dazu erweitert man die Klasse org.eclipse.jface.action.Action die ihrerseits das Interface org.eclipse.jface.action.IAction implementiert. Die Klasse Action stellt bereits einige nützliche Standard-Implementationen zur Verfügung, einzig die run()-Methode sollte überschrieben werden um überhaupt eine Aktion zu erzeugen.

Actions und Context Menus programmatisch erstellen

In einer View kann mit folgendem Beispiel-Code ein PopUp-Menu hinzugefügt werden:

….

MenuManager mm = new MenuManager(„&Beispiel“, „Beispiel“);  
 mm.add(new ExampleAction());

mm.add(new Separator(IWorkbenchActionConstants.MB\_ADDITIONS))  
 Menu menu = mm.createContextMenu(viewer.getControl());  
 treeViewer.getControl().setMenu(menu);  
 getSite().registerContextMenu(mm , treeViewer);

….

Abbildung Context Menu programmatisch hinzufügen

Action deklarativ erstellen

Bei der deklarativen Erzeugung von Actions sind die folgenden Extension Points wichtig:

|  |  |
| --- | --- |
| **Extension point** | **Bedeutung** |
| org.eclipse.ui.actionSets | Diese Erweiterung beschreibt ein Set von Menus und Actions die zum Top-Level-Menu und Toolbar hinzugefügt werden. Action Sets werden als Gruppe aktiviert oder deaktiviert. |
| org.eclipse.ui.popupMenus | Dieser Extension Point wird benötigt um den in der Workbench registrierten Context/Popup Menus Actions hinzuzufügen |
| org.eclipse.ui.editorActions | Diese Erweiterung wird benützt um Actions dem Top-Level-Menu und der Toolbar hinzuzufügen wenn ein bestimmter Editor aktiviert wird. |
| org.eclipse.ui.viewActions | Diese Erweiterung wird benützt um Actions dem lokalen Menu und der Toolbar auf Views hinzuzufügen. |

Die deklarative Definition von Actions hat die folgenden Vorteile:

* + Deklarative Actions werden im UI angezeigt ohne das Plugin zu dem sie gehören zu laden. In grossen Applikation mit vielen Plugins hat dies positive Auswirkungen auf die Performance.
  + Mit der Verbindung von ActionSets mit Perspective‘s wird die dynamische Rekonfiguration von Top-Level-Menus und Toolbars basierend auf der aktiven Perspective möglich.
  + Benutzer können Top-Level-Menus und Toolbars über den Perspective Customization Dialog konfigurieren

Die Nachteile sind:

* + Die Reihenfolge der Actions innerhalb eines Menus wird von der Workbench gesteuert
  + Die Eingabe der Toolbar und Menu-Pfade ist fehleranfällig. Wenn ein Eintrag falsch ist so erscheint das Menu nicht.

Jetzt stellt sich noch die Frage wie man Actions deklariert. Dies geschieht im plugin.xml File und zwar folgendermassen:

<extension point="org.eclipse.ui.actionSets">  
 <actionSet id="myplugin.actionSet" label="Beispiel ActionSet" visible=“true“>  
 <menu id="myplugin.menus.beispiel"  
 icon="icons/beispiel.gif"  
 label="&amp;Beispiel Menu"  
 path="additions">  
 <groupMarker name="group1" />  
 </menu>  
 <action id="myplugin.actions.beispielAction"   
 icon="icons/beispiel.gif"  
 class="myplugin.actions.BeispielAction"  
 label="Beispiel Action"  
 menubarPath="myplugin.menus.beispiel/group1"  
 toolbarPath ="myplugin.toolbar.beispiel "  
 style ="push"  
 />  
 <action id="myplugin.actions.beispielAction2"   
 …………….  
 />  
 </actionSet>  
</extension>

Abbildung Deklaration Action und ActionSet

In diesem Beispiel wird ein ActionSet mit zwei Actions definiert.

Actions und Context Menus deklarativ erzeugen

Context Menus können folgendermassen im plugin.xml deklariert werden:

<extension point="org.eclipse.ui.popupMenus">  
 <action id="myplugin.actions.beispielAction"   
 icon="icons/beispiel.gif"  
 class="myplugin.actions.BeispielAction"  
 label="Beispiel Action"  
 />  
</extension>

Abbildung Deklaration Action und PopupMenu

Standard Workbench Actions

Standard Workbench Actions können über die ActionFactory geholt werden. Beim „makeActions“ Beispiel haben wir gesehen wie das funktioniert: ActionFactory.SAVE.create(window). Die folgenden Standard Actions stehen zur Verfügung: ABOUT, CLOSE, EXPORT, QUIT, SAVE und viele mehr, siehe dazu die ActionFactory-Klasse.