EDITORS

An editor is used in an Eclipse RCP application when you want to create or modify files, or other resources. Eclipse already provides some basic text and Java source file editors.

In your plug-in manifest editor, add in the `org.eclipse.ui.editors` extension point, and fill in the following details

Attribute	Required	Use
`id`	Yes	The unique id of this editor
`name`	Yes	A readable name for this editor
`icon`	No	The image to be displayed in the top left hand corner of the editor when it is open
`extensions`	No	A string of comma separated file extensions that are understood by the editor
`class`	No	The class that implements the `IEditorPart` interface
`command`	No	A command to run to launch and external editor
`launcher`	No	The name of a class that implements `IEditorLauncher` to an external editor
`contributorClass`	No	A class that implements `IEditorActionBarContributor` and adds new actions to the workbench menu and toolbar which reflect the features of the editor type
`default`	No	If true this editor will be used as the default for this file type. The default value is false
`filenames`	No	A list of filenames understood by the editor. More specific than the `extensions` attribute
`matchingStrategy`	No	An implementation of `IEditorMatchingStrategy` that allows an editor to determine whether a given editor input should be opened

Editors implement the `org.eclipse.ui.IEditorPart` interface, or subclass `org.eclipse.ui.parts.EditorPart`.

Like views, to facilitate lazy loading, a workbench page only holds `IEditorReference` objects, so that you can list out the editors without loading the plug-in that contains the editor definition.

PERSPECTIVES

Perspectives are a way of grouping you views and editors together in a way that makes sense to a particular context, such as debugging. By creating your own perspective, you can hook into the **Window>Open** Perspective dialog.

To create a perspective, you need to extend the `org.eclipse.ui.perspectives` extension point.

Attribute	Required	Use
`id`	Yes	The unique id of this perspective
`name`	Yes	A readable name for the perspective
`class`	Yes	The class that implements the `IPerspectiveFactory` interface
`icon`	No	The image to be displayed related to this perspective
`Fixed`	No	Whether this perspective can be closed or not. Default is false

The class driving the perspective implements `org.eclipse.ui.IPerspectiveFactory`. This class has one method `createInitialLayout()`, within which you can use the `IPageLayout.addView()` method to add views directly to the perspective. To group many views together in a tabbed fashion, rather than side by side, `IPageLayout.createFolder()` can be used.

Hot Tip

When running your application you need to ensure that you have all required plug-ins included. Do this by checking your Run Configurations. Go to the plug-ins tab and click Validate Plug-ins. If there are errors click on Add Required Plug-ins to fix the error.

PREFERENCES

Now that you have created a perspective and a view for your RCP application, you will probably want to provide some preference pages. Your contributed preference pages will appear in the **Window>Preferences** dialog.

To provide preference pages you will need to implement the `org.eclipse.ui.preferencePages` extension in the plug-in manifest editor.

Attribute	Required	Use
`id`	Yes	The unique id of this preference page
`name`	Yes	A readable name for the preference page
`class`	Yes	The class that implements the IWorkbenchPreferencePage interface
`category`	No	Path indicating the location of the page in the preferences tree. The path may be defined using the parent preference page id or a sequence of ids separated by "/". If no category is specified, the page will appear at the top level of the preferences tree.

While the preference page class will implement `org.eclipse.ui.IWorkbenchPreferencePage`, it is useful to extend `org.eclipse.jface.preference.FieldEditorPreferencePage` as it provides `createFieldEditors()` method which is all you need to implement, along with the `init()` method in order to display a standard preference page. A complete list of FieldEditors is provided in the `org.eclipse.jface.preference` package.

Loading and Storing Preferences

Preferences for a plug-in are stored in an `org.eclipse.jface.preference.IPreferenceStore` object. You can access a plug-ins preference through the Activator, which will typically extend `org.eclipse.ui.plugin.AbstractUIPlugin`. Each preference you add to the store has to be assigned a key. Preferences are stored as String based values, but methods are provided to access the values in number of formats such as double, int and Boolean.

PROPERTY SHEETS

While preferences are used to display the overall preferences for the plug-in, property sheets are used to display the properties for views, editors or other resources in the Eclipse environments. By hooking into the Properties API, the properties for you object will appear in the Properties view (usually displayed at the bottom of your Eclipse application).

The Properties view will check if the selected object in the workspace can supports the `org.eclipse.ui.views.properties.IPropertySource` interface, either through implementation or via the `getAdapter()` method of the object. Each property gets a descriptor and a value through the `IPropertySource` interface.

HELP

All good applications should provide some level of user assistance. To add help content to the standard **Help>Help Contents** window, you can use the `org.eclipse.help.toc` extension point. Add a number of toc items to this extension point – the only mandatory attribute for each toc entry is the file that contains the table of contents definition.

Hot Tip To see a quick example of what help content should look like, choose the Help Content item from the Extension Wizards tab when adding to the plug-ins manifest.

```
<toc label="Getting Started" link_to="toc.xml#gettingstarted">
      <topic label="Main Topic"  href="html/gettingstarted/
       maintopic.html">
            <topic label="Sub Topic" href="html/gettingstarted/
             subtopic.html" />
      </topic>
      <topic label="Main Topic 2">
            <topic label="Sub Topic 2" href="html/gettingstarted/
             subtopic2.html" />
      </topic>
</toc>
```

Each topic entry should have a link to a HTML file with the full content for that topic. The above XML extract from a table of contents file illustrates this. There is also the choice to use the definition editor for help content. This will open by default in Eclipse when choosing a *toc* file.

Cheat Sheets

Another user assistance mechanism used in Eclipse is a cheat sheet, which guides the user through a series of steps to achieve a task. To create your initial cheat sheet content, use the **New>Other...>User Assistance>Cheat Sheet**. This presents you with an editor to add an Intro and a series of items, with the option to hook in commands to automate the execution of the task.

To add this cheat sheet to your plug-in manifest, the cheat sheet editor has a *Register this cheat sheet* link on the top right hand corner. When registering the cheat sheet you will need to provide it with a category and a description.

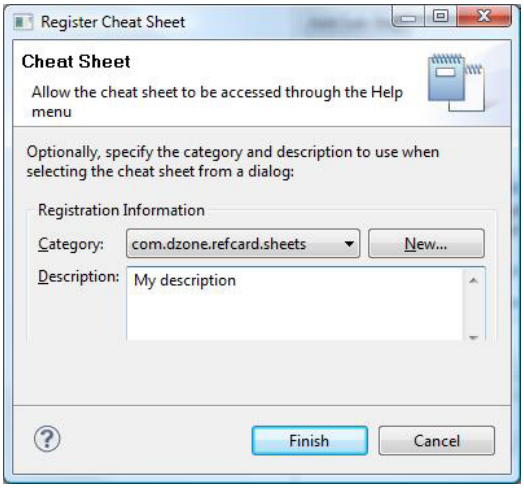


Figure 5: Cheat sheet registration dialog

Clicking finish on this dialog will add the `org.eclipse.ui.cheatsheets.cheatSheetContent` extension point to your manifest. You can modify the details of the cheat sheet from here if necessary.

FEATURES

You can help the user to load up your plug-in(s) as a single part, by combining them into one feature. Eclipse provides a wizard to create your feature through the **New Project> Plug-in Development >Feature Project** wizard.

This wizard generated a **feature.xml** file which has an editor, similar to the plug-in manifest editor, where you can change the details of your feature.

The most important section is the **Plug-ins** tab, which lists the plug-ins required for your feature. The **Included Features** tab allows you to specify sub-features to include as part of your feature. On the **Dependencies** tab, you can get all the plug-ins or features that you are dependent on by clicking on the **Compute** button.

A simple feature.xml may look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<feature
      id="my.feature"
      label="Feature"
      version="1.0.0.qualifier"
      provider-name="James">

   <description url="http://www.example.com/description">
      [Enter Feature Description here.]
   </description>

   <copyright url="http://www.example.com/copyright">
      [Enter Copyright Description here.]
   </copyright>

   <license url="http://www.example.com/license">
      [Enter License Description here.]
   </license>

   <requires>
      <import plugin="org.eclipse.ui"/>
      <import plugin="org.eclipse.core.runtime"/>
   </requires>

   <plugin
         id="com.dzone.refcard.rcpapp"
         download-size="0"
         install-size="0"
         version="0.0.0"
         unpack="false"/>

</feature>
```

BRANDING

The feature also provides a single location where you can define all the branding for your application. In the **Overview** tab, you can assign a Branding Plug-in to the feature.

The branding plug-in needs to contain the following artefacts:

Item	Purpose
`about.html`	A HTML file that will be displayed in the **Plug-in Details>More** Info dialog
`about.ini`	This file contains most of the branding information for the feature Described below
`about.properties`	Used for localisation of the strings from the **about.ini** file. The values are referenced using the %key notation

about.ini

Property	Purpose
`aboutText`	Multiline description containing name, version number and copyright information. Will appear in the **About>Feature Details>About Features** dialog.
`featureImage`	A 32x32 pixel icon representation of the feature to be used across the relevant About dialogs

All of the icons and files referenced by the about.ini file should be placed in this plug-in also.

Product Branding

A product is an entire distribution of an RCP application, rather than a feature intended to be part of an existing distribution. As such, products have additional branding requirements. To specify these extra parameters, a contribution to the `org.eclipse.core.runtime.products` extension point is required.

The **product** must be assigned the application to run, the name of the product (for the title bar) and a description. Further properties are added as name/value pairs underneath the product.

Hot Tip An application can be provided by using the org.eclipse.core.runtime.applications extension point

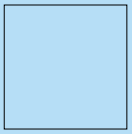
Property	Purpose
`windowImages`	The image used for this application, in windows and dialogs. This should be in the order of the 16x16 pixel image, followed by the 32x32
`aboutImage`	Larger image to be placed in the About dialog
`aboutText`	Multiline description containing name, version number and copyright information. Will appear in the **About>Feature Details>About Features** dialog.

You can also provide most of these details in the **Branding** tab of the generated **.product** file.

Splash Screen

The **.product** file that is generated while creating your product includes a **Splash** tab. Here you can specify the plug-in that contains the splash.bmp file for your Splash screen. Typically, this should reside in your branding plug-in. The splash screen can also be customized with templates, and can include a progress bar with messages.

ABOUT THE AUTHOR



James Sugrue is a software architect at Pilz Ireland, a company using many Eclipse technologies. James is also editor at both EclipseZone and JavaLobby. Currently he is working on TweetHub, a Twitter client based on RCP and ECF. James has also written a Refcard on EMF and has another Refcard on the way covering Eclipse Plug-ins.

RECOMMENDED BOOKS

Building on two internationally best-selling previous editions, **Eclipse Plug-ins**, Third Edition, has been fully revised to reflect the powerful new capabilities of Eclipse 3.4.

In **Eclipse Rich Client Platform**, two leaders of the Eclipse RCP project show exactly how to leverage Eclipse for rapid, efficient, cross-platform desktop development.

BUY NOW

books.dzone.com/books/eclipse-plug-ins

books.dzone.com/books/eclipse-rcp

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-75-2
ISBN-10: 1-934238-75-9
50795
9 781934 238752
$7.95