

---

# Rich Client Platform Application 教程

---

共享版 1.0

**salever**

**2010-3-22**

# 目录

第 1 章	富客户端平台 .....	6
1.1	概述 .....	6
1.2	Eclipse RCP 建设风格——插件，扩展和扩展点 .....	6
第 2 章	创建你的第一个 RCP 程序 .....	7
2.1	创建一个 RCP 程序 .....	7
2.2	启动你的 RCP 程序 .....	10
2.3	应用程序 VS 产品 .....	11
2.4	应用程序的插件 ID .....	13
第 3 章	Actions 的用法（菜单和工具栏） .....	14
3.1	概述 .....	14
3.2	示例 .....	14
3.3	由“扩展”方式向程序添加菜单和工具栏 .....	17
第 4 章	添加组合键 .....	26
4.1	概述 .....	26
4.2	声明 actions 的组合键 .....	26
第 5 章	系统托盘 .....	32
第 6 章	外观 .....	38
6.1	向程序中添加视图模板 .....	38
6.2	向应用程序添加 VIEW .....	43
6.3	向 VIEW 里添加 action .....	48
第 7 章	和编辑器一起工作 .....	61
7.1	概述 .....	61
7.2	创建工程 .....	61
7.3	创建并准备 do main 模型 .....	61
7.4	在视口中使用 do main 模型 .....	67
7.5	加入编辑器 .....	68
7.6	调用编辑器 .....	76
7.7	向编辑器提供内容 .....	80
第 8 章	对话框 .....	81

8.1	概述 .....	81
8.2	预定义的对话框 .....	81
8.2.1	概述 .....	81
8.2.2	创建工程 .....	81
8.2.3	声明 action .....	81
8.2.4	调用对话框 .....	81
8.3	用户自定义对话框 .....	84
8.3.1	概述 .....	84
8.3.2	创建工程 .....	84
8.3.3	声明 action .....	84
8.3.4	声明 action .....	84
8.3.5	创建对话框 .....	85
第 9 章	向导 (wizard) .....	88
9.1	概述 .....	88
9.2	例子 .....	88
第 10 章	首选项 .....	96
10.1	首选项 .....	96
10.2	使用首选项 .....	96
10.3	首选项页 .....	101
第 11 章	添加状态条 .....	108
11.1	安装状态条 .....	108
11.2	共享状态条 .....	109
第 12 章	透视图 .....	115
12.1	向你的程序中添加透视图 .....	115
12.2	使透视图可选。 .....	119
12.2.1	使透视图可由一个 coolbar 可选 .....	119
12.2.2	使透视图可通过菜单选择 .....	121
第 13 章	进度报告 .....	125
第 14 章	将外部类包含进你的程序 .....	131
14.1	概述 .....	131
14.2	向构建路径中添加 jar .....	131

14.3	使 jar 在你的运行路径里有效 .....	132
第 15 章	提示和策略 .....	134
15.1	控制台日志 .....	134
15.2	保存用户的布局 .....	134
15.3	获得 display .....	136
15.4	装载模型 .....	140
15.5	向你的程序添加错误日志视口 .....	140
第 16 章	制造一个产品 .....	143
16.1	概述 .....	143
16.2	创建一个工程 .....	143
16.3	测试你的产品 .....	147
第 17 章	商标 .....	148
17.1	欢迎页面 .....	148
17.2	商标 .....	148
17.3	风格化 launcher .....	149
第 18 章	发布你的产品 .....	151
第 19 章	发布引入外部 jar 的产品 .....	155
19.1	整合外部 jar 和第三方库 .....	155
19.2	. 为你的 jar 创建一个插件工程 .....	155
第 20 章	向 RCP 应用程序中添加帮助 .....	158
20.1	创建一个新工程 .....	158
20.2	创建一个产品 .....	158
20.3	添加依赖性 .....	160
20.4	创建一个帮助插件工程////原著写的不好 .....	163
第 21 章	配置文件 .....	167
21.1	概述 .....	167
21.2	.project .....	168
21.3	Manifest.MF .....	169
第 22 章	使用接口技术 .....	170
22.1	SWT .....	170
22.2	Jface .....	170

## 声明

本教程由属于共享版，版权归其原著者所有。基本素材取自互联网，如内容涉及到您的利益，请联系本人。请勿用于商业用途。

Email:salver.cs@gmail.com

# 第1章 富客户端平台

## 1.1 概述

Eclipse 是一个重用框架的开发环境。

接下来将描述如何使用这个框架开发应用程序。

对 Eclipse 来说，整个 RCP 程序就是一个插件。一个 RCP 需要：

- 主程序：一个 RCP 程序继承了类 `org.eclipse.core.runtime.application`。它相当于主程序。
- 一个透视图：透视图是继承了 `org.eclipse.ui.perspective`。
- 工作空间顾问

工作空间顾问是个不可见的技术元件，它控制程序的外形（菜单、工具栏、透视图等等），对 RCP 来说外观是技术性的，而不是必需的。但是通常情况下，一个没有外观的应用程序很难给人留下什么感觉。

- 所有的插件必须提交一个 MANIFEST 名为 “`plugin.xml`”。
- 同时还需要另两个中心插件：`org.eclipse.core.runtime` 和 `org.eclipse.ui`。

## 1.2 Eclipse RCP 建设风格——插件，扩展和扩展点

插件(Plug-in)是 Eclipse 最小的可开发可安装元件。每一个插件可以定义 SO-CALLED 扩展点。(define possibilities for functionality contributions ( code and non-code ) by other plug-ins. Non-code functionality contributions are for example the provision of help content.)

一个插件可以使用扩展，例如，向扩展点提供方法，通常一个扩展点能够被用到数次（包括相同的插件或不同的插件）。建设风格的基础是基于 OSGI ALLIANCE 的 eclipse 运行时环境

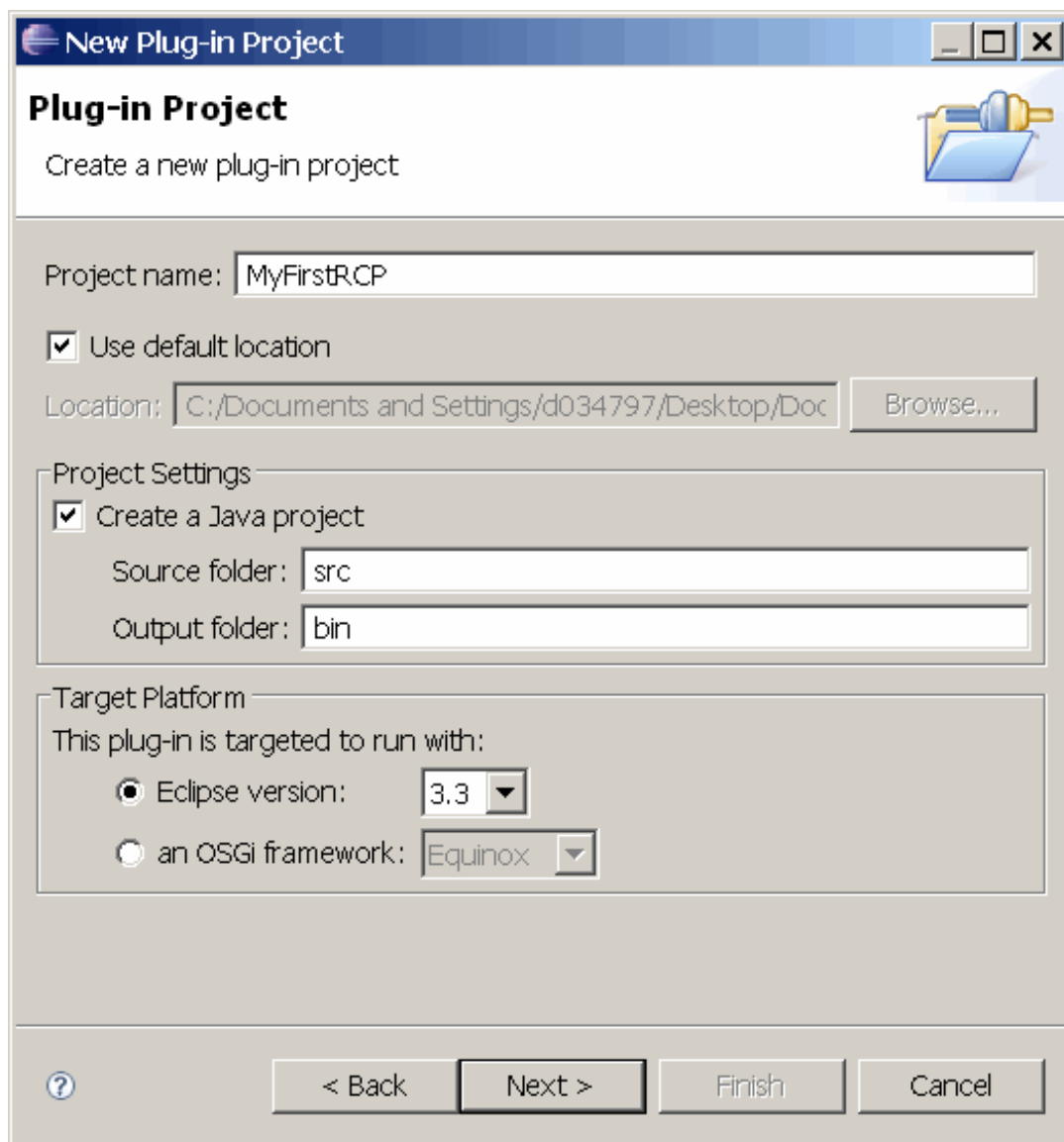
被用到的扩展和提供的扩展点都在 `plugin.xml` 里被描述。这个文件可以在 PDE(插件开发环境)里被很好的编辑 eclipse RCP 提供和甬道了与 ECLIPSE 工作区相同的框架,因此允许程序员提供程序方法到几个插件里，利用已存在的扩展点，且提供附加的扩展点

## 第2章创建你的第一个 RCP 程序

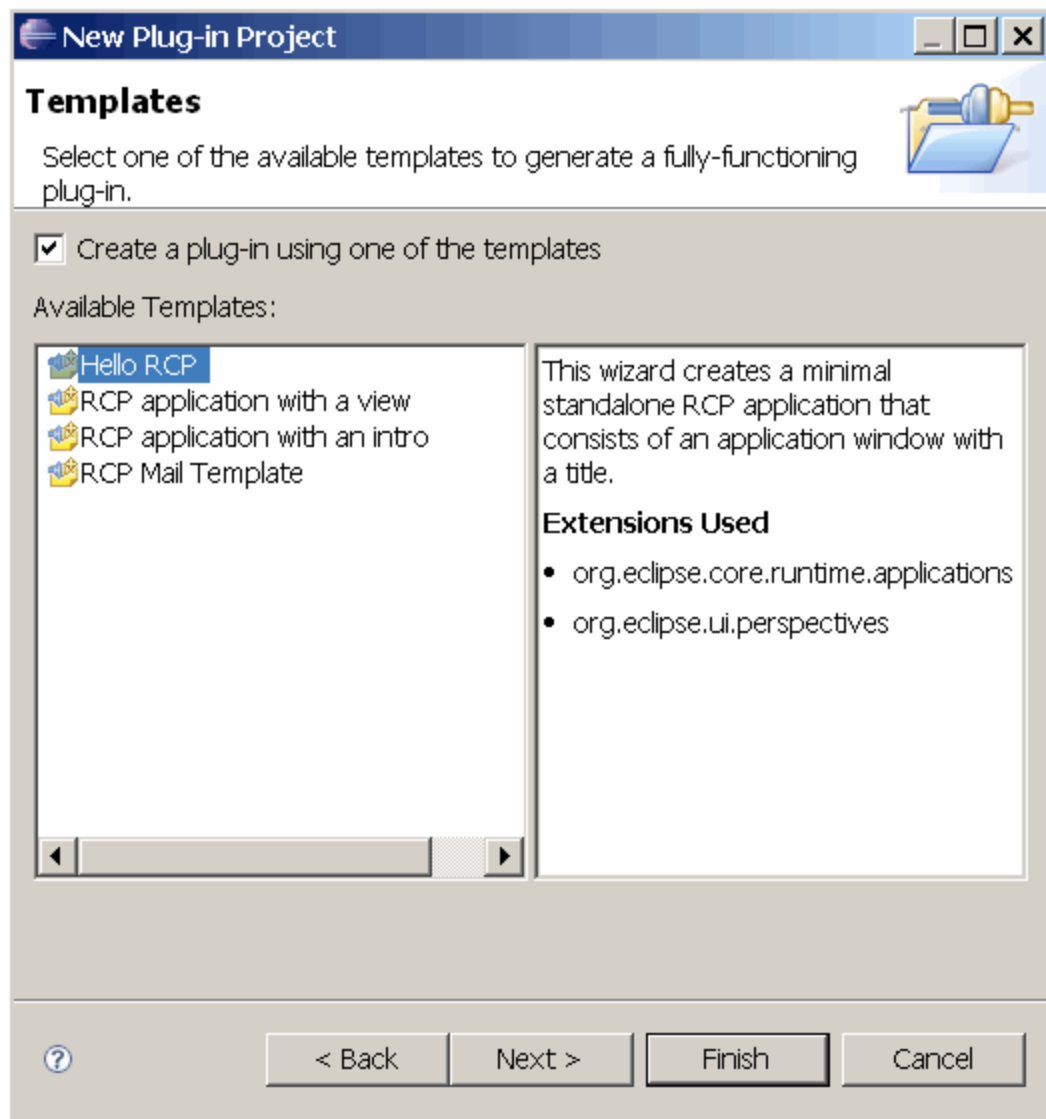
接下来给出快速指南帮助你创建一个简单的 RCP 程序

### 2.1 创建一个RCP程序

在 ECLIPSE 里 选择 File-> New Project，在列表中选择 PLUG-IN PROJECT  
接下来：给你的 RCP 插件命名，例如：“MyFirstRCP”

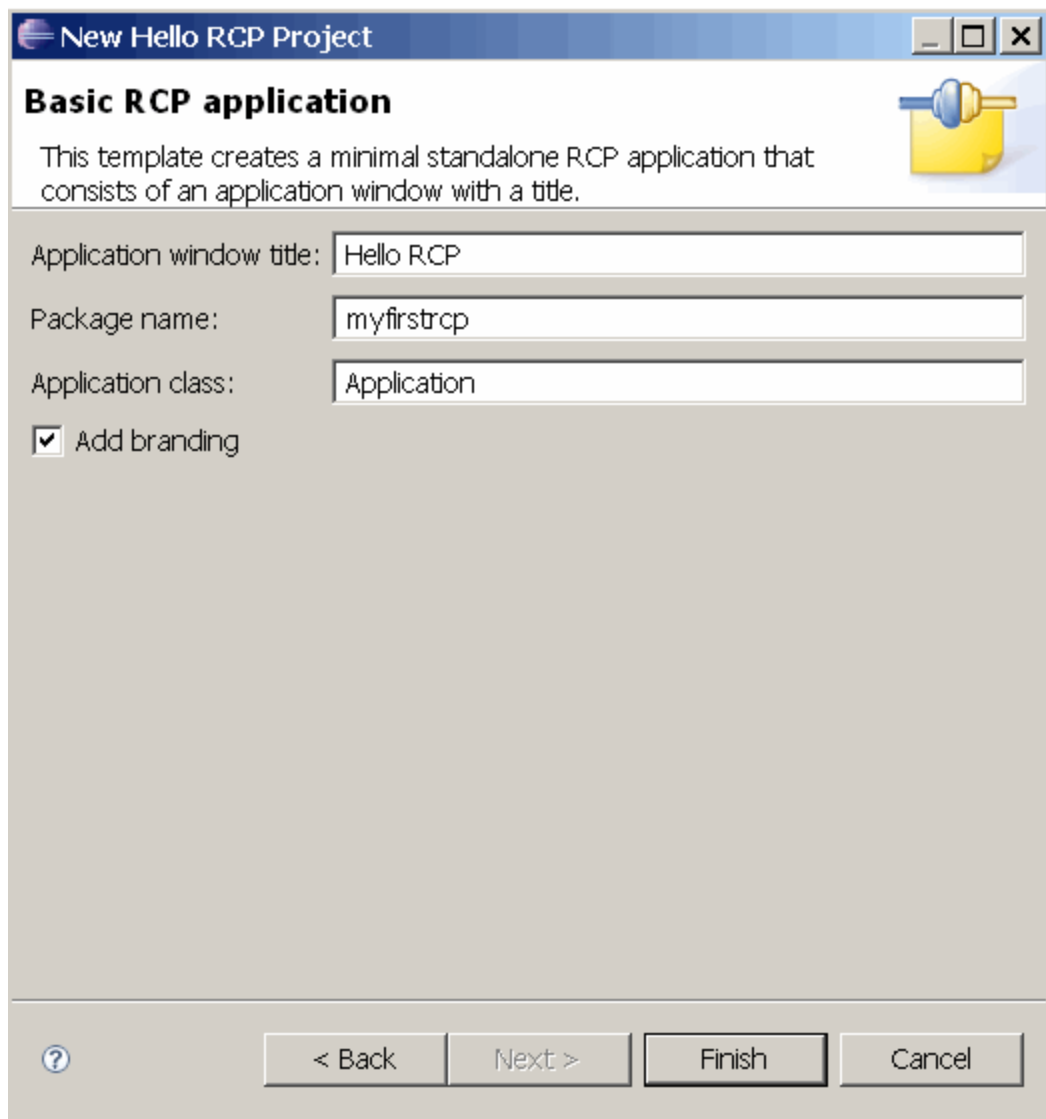


选择“Hello RCP”模版

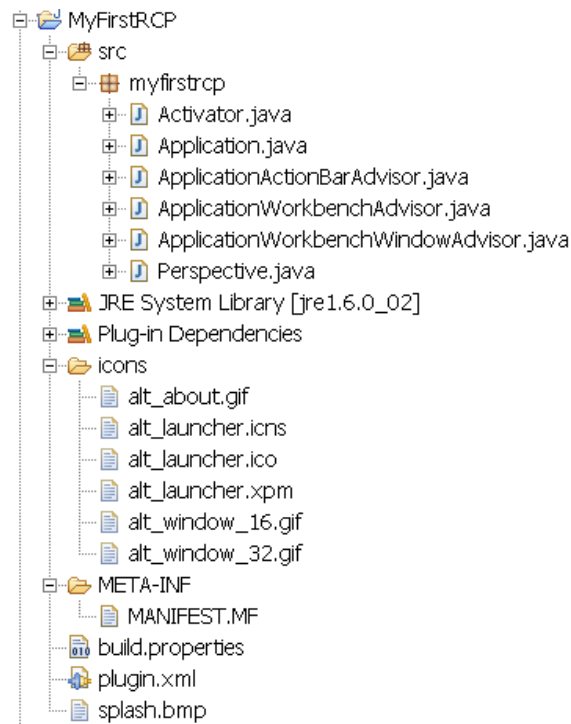


在下一屏选择“Add branding”，点击“Finish”





一个工程结构被创建好了。看一看不这些不同的 JAVA 文件，找一下对工程结构的第一感觉



## 2.2 启动你的RCP程序

找到 MANIFEST.MK，双击，转到编辑器，并且“OVERVIEW”被选中。选择“Testing”  
“launch an Eclipse Application”，或者右键点击“plugin.xml”文件。选择“run as”->“eclipse  
application”

**Overview**
?

**General Information**

This section describes general information about this plug-in.

ID:

Version:

Name:

Provider:

Platform Filter:

Activator:

☒ Activate this plug-in when one of its classes is loaded

**Execution Environments**

Specify the minimum execution environments required to run this plug-in.

[Configure JRE associations...](#)

[Update the classpath settings](#)

**Plug-in Content**

The content of the plug-in is made up of two sections:

- [Dependencies](#): lists all the plug-ins required on this plug-in's classpath to compile and run.
- [Runtime](#): lists the libraries that make up this plug-in's runtime.

**Extension / Extension Point Content**

This plug-in may define extensions and extension points:

- [Extensions](#): declares contributions this plug-in makes to the platform.
- [Extension Points](#): declares new function points this plug-in adds to the platform.

**Testing**

Test this plug-in by launching a separate Eclipse application:

- [Launch an Eclipse application](#)
- [Launch an Eclipse application in Debug mode](#)

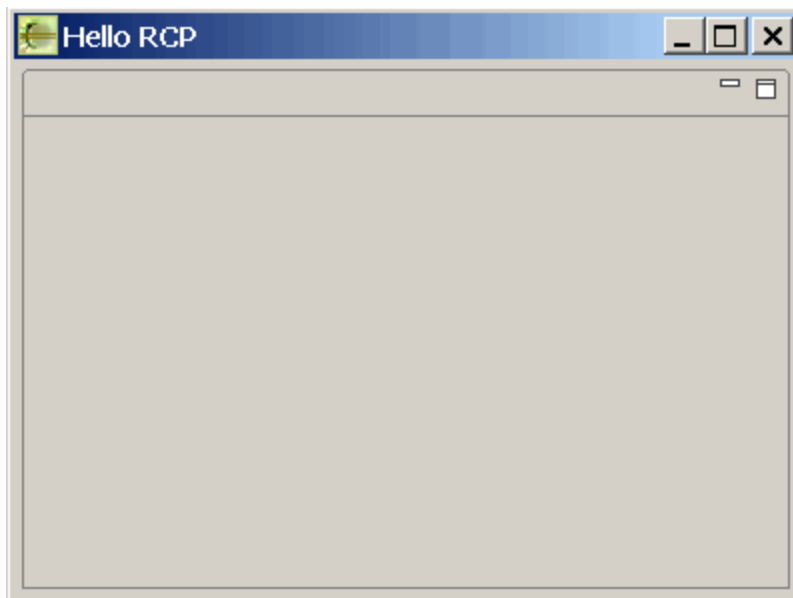
**Exporting**

To package and export the plug-in:

- Organize the plug-in using the [Organize Manifests Wizard](#)
- Specify what needs to be packaged in the deployable plug-in on the [Build Configuration](#) page
- Export the plug-in in a format suitable for deployment using the [Export Wizard](#)

Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties

可以看到如下结果:



恭喜你, 你已经创建了你的第一个 RCP 程序了

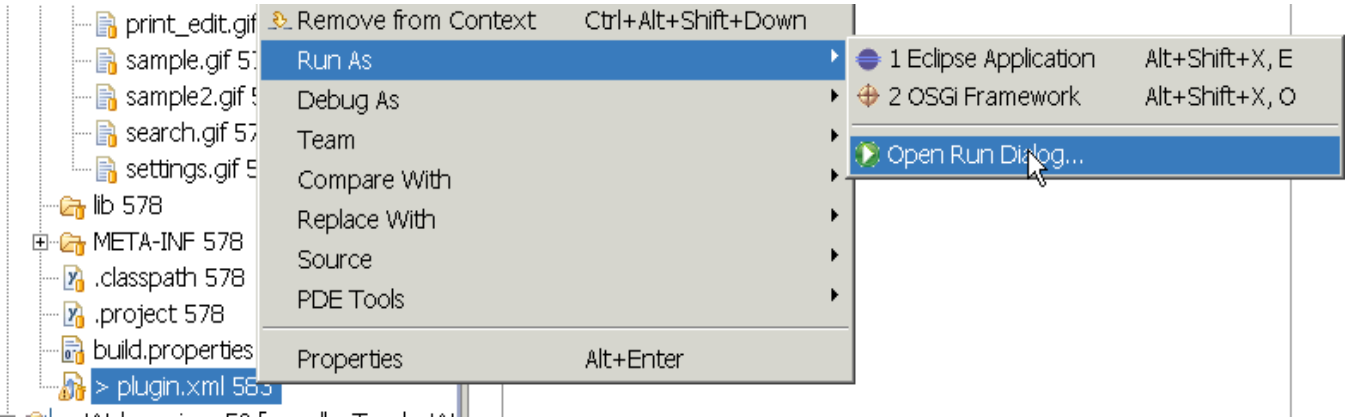
## 2.3 应用程序 VS 产品

运行一个ECLIPSE RCP程序,你必须定义一个APPLICATION,这个程序可以被视为带有main()函数的标准java程序。一旦应用程序被关闭,整个程序也将终结。

Eclipse的产品指包括其他所有东西的你的程序,例如:图标,欢迎界面,外部JARS,其他一些插件等等。

2. 4. 维护你的 launch 设置

选择你的 plugin.xml->Run as->Open Run Dialog

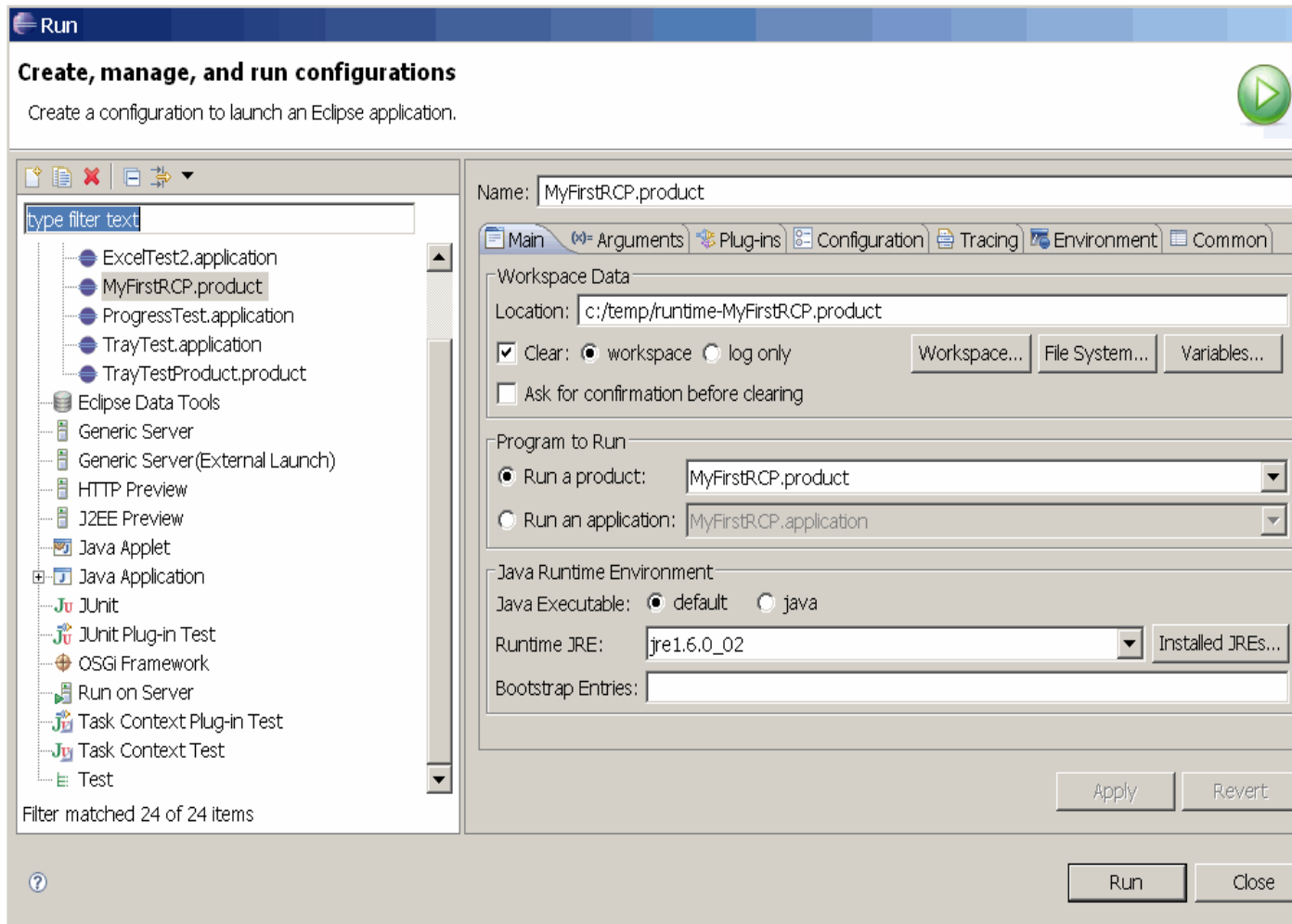


这个例子里,我们将运行产品。试着也运行应用程序。

这里有两项重要设置:

首先工作空间数据加载应该被设置。这里PDE将创建目录和所有需要的文件以运行你的RCP应用程序。我总是使用在我工作空间之外的临时目录。

第二,设置“Clear Flag”否则工作空间的所有改变都不能影响到你的新应用程序。推荐关闭“Ask for confirmation” flag



## 2.4 应用程序的插件ID

插件 ID 通常被用在几个地方，所以最好在应用程序类里把它声明为静态。这个 ID 必须与定义在 plugin.xml 里的 ID 一样。ID 是一个传感器

例如如果你的 RCP 应用程序调用 MyFirstRCP，可以在 Application.java 里添加以下声明

```
public static final String PLUGIN_ID = "MyFirstRCP";
```

## 第3章 Actions 的用法（菜单和工具栏）

### 3.1 概述

在 ECLIPSE 里，是由 actions 来描述菜单及工具栏的

你有两种方法向你的应用程序里添加菜单和工具栏

编写代码

扩展（Extensions）

如果是第一种方法，利用 `ApplicationActionBarAdvisor` 类的 `makeActions()` 声名 actions。你可以利用方法 `fillMenuBar()` 或者 `fillCoolBar()` 向你的程序添加菜单或者 coolbar。

如果你用第二种方法，将使用 ECLIPSE 向导以扩展点形式创建 ACTIONS。

### 3.2 示例

创建新工程“MenuTest”使用“Hello RCP”模板

打开 `ApplicationActionBarAdvisor.java` 做如下更改

```
package menutest;
```

```
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.Separator;
import org.eclipse.ui.IWorkbenchActionConstants;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;

public class ApplicationActionBarAdvisor extends ActionBarAdvisor {
    private IWorkbenchAction iExitAction;
    private IWorkbenchAction iAboutAction;
```

```
private IWorkbenchAction iNewWindowAction;

private IWorkbenchAction iSaveAction;


public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
    super(configurer);
}


protected void makeActions(IWorkbenchWindow window) {
    iExitAction = ActionFactory.QUIT.create(window);
    register(iExitAction);

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

    iAboutAction = ActionFactory.ABOUT.create(window);
    register(iAboutAction);

    iNewWindowAction = ActionFactory.OPEN_NEW_WINDOW.create(window);
    register(iNewWindowAction);
}


protected void fillMenuBar(IMenuManager menuBar) {
    MenuManager fileMenu = new MenuManager("&File",
        IWorkbenchActionConstants.M_FILE);

    MenuManager helpMenu = new MenuManager("&Help",
        IWorkbenchActionConstants.M_HELP);

    menuBar.add(fileMenu);
    menuBar.add(helpMenu);


    // File Menu

    fileMenu.add(iNewWindowAction);
    fileMenu.add(iSaveAction);
    fileMenu.add(new Separator());
```

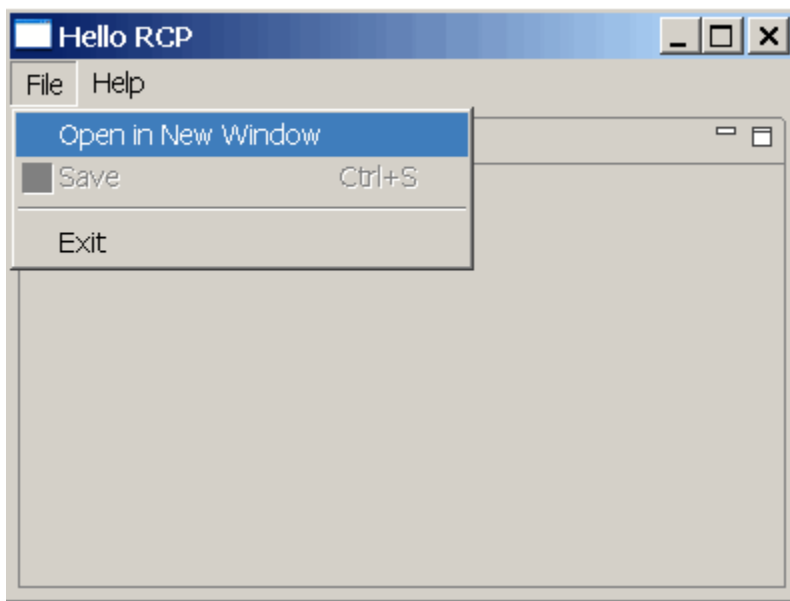
```

        fileMenu.add(iExitAction);

        // Help Menu
        helpMenu.add(iAboutAction);
    }
}

```

运行程序，结果如图



现在，通过方法 `fillCoolBar()` 方法，向 `ApplicationActionBarAdvisor.java` 添加工具栏，代码如下：

这里要用到 `Jface` 先将 `swt` 包导入。

```
@Override
```

```

protected void fillCoolBar(ICoolBarManager coolBar){
    // This will add a new toolbar to the application
    IToolBarManager toolbar = new ToolBarManager(SWT.FLAT | SWT.RIGHT);
    coolBar.add(new ToolBarContributionItem(toolbar, "main"));
    // Add the entry to the toolbar
    toolbar.add(iSaveAction);
    toolbar.add(iExitAction);
}

```

你必须也得在 `ApplicationWorkbenchWindowAdvisor.java` 里转变 `coolbar` 属性

```
public void preWindowOpen() {
```

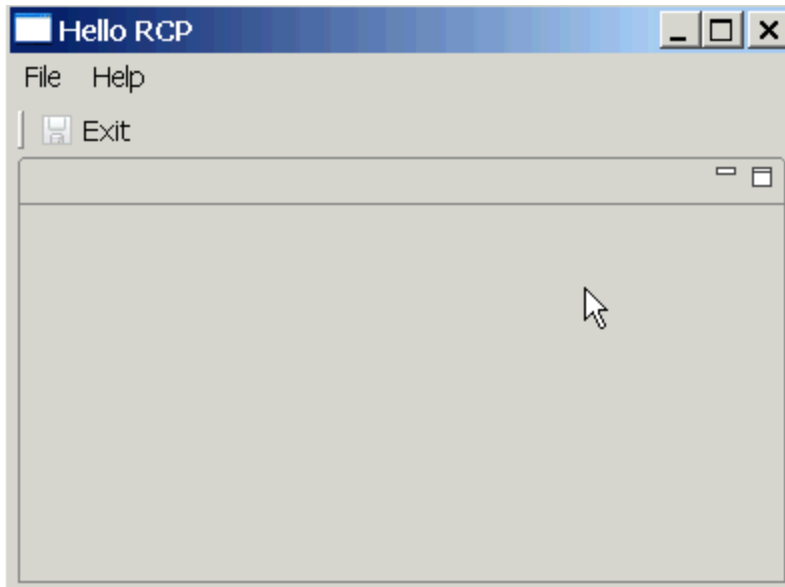


```

IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
configurer.setInitialSize(new Point(400, 300));
configurer.setShowCoolBar(true); // Changed to true
configurer.setShowStatusLine(false);
configurer.setTitle("Hello RCP");
}

```

保存后运行，结果如下



### 3.3 由“扩展”方式向程序添加菜单和工具栏

创建一个新工程，命名为“ExtensionMenuTest”，使用“Hello RCP”模板。

双击 `ApplicationWorkbenchWindowAdvisor` 类，设置 `preWindowOpen()` 方法里的 `configurer.setShowCoolBar(true)`，以激活工具栏

```
package extensionmenutest;
```

```

import org.eclipse.swt.graphics.Point;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;

```

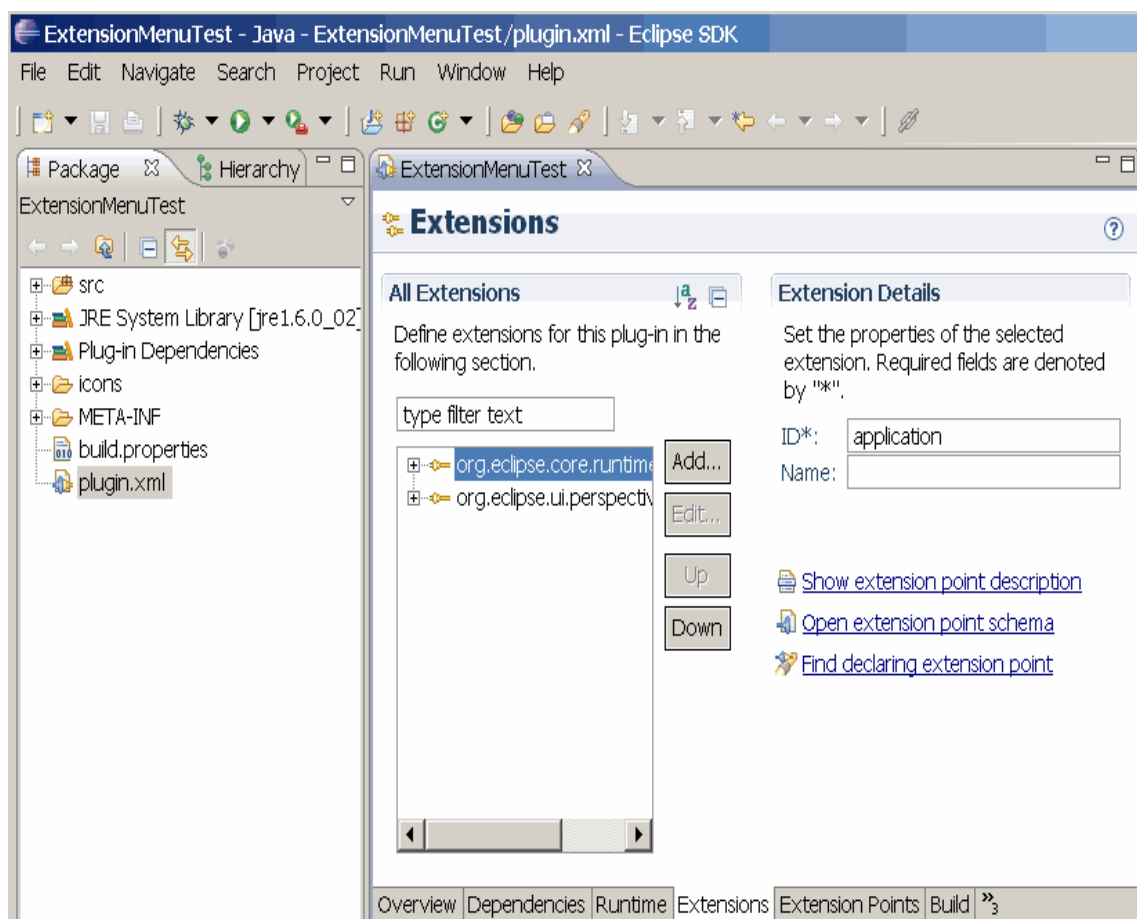
```
public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    public ApplicationWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurer) {
        super(configurer);
    }

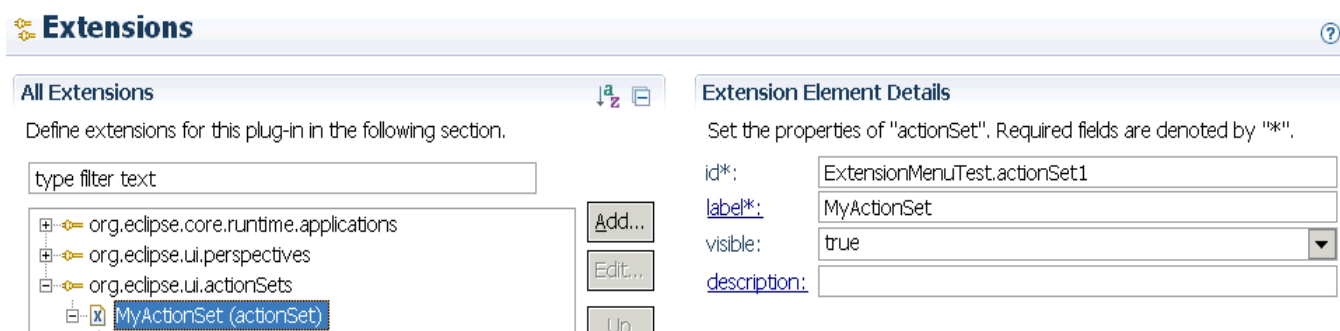
    public ActionBarAdvisor createActionBarAdvisor(IActionBarConfigurer configurer) {
        return new ApplicationActionBarAdvisor(configurer);
    }

    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowCoolBar(true);
        configurer.setShowStatusLine(false);
        configurer.setTitle("Hello RCP!");
    }
}
```

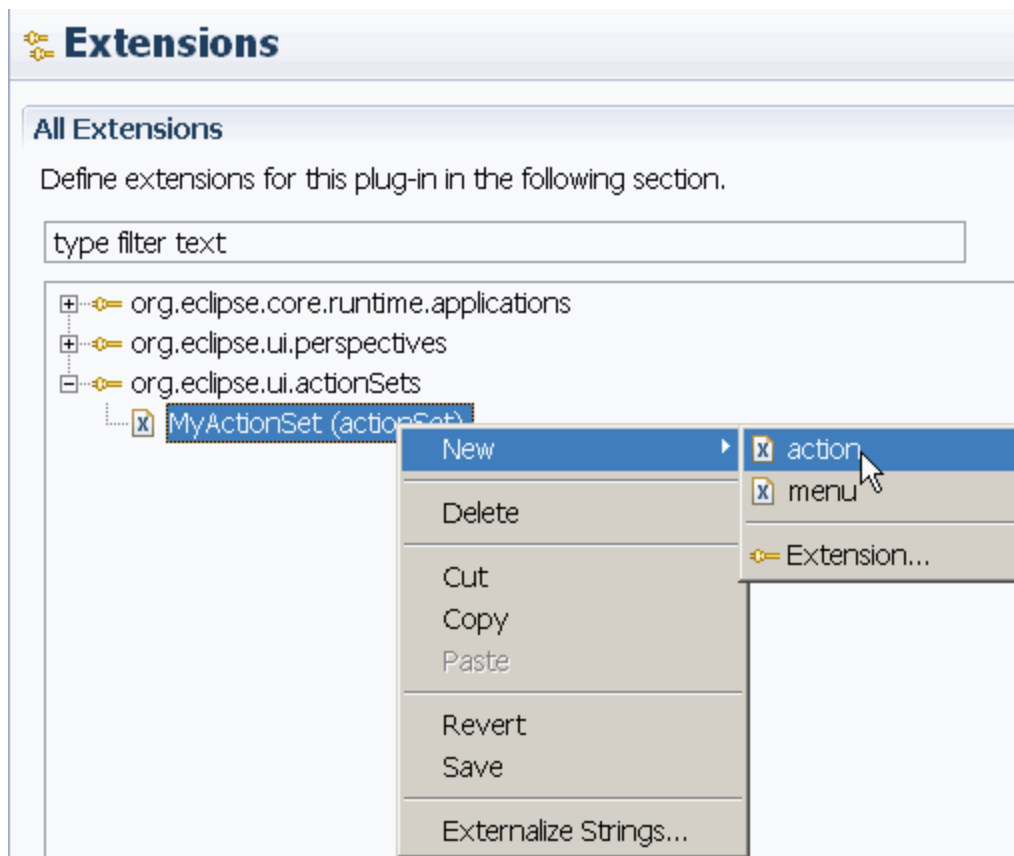
双击 `plugin.xml`，打开 PDE 修改文件，选择扩展标签。



点击 Add...弹出 New Extension 窗口，选择 org.eclipse.ui.actionSets 扩展点，点击完成给 actionset 命名、设置可见为 true，保存，如图



右击新的 actionSet 选择 new->action



为你的新 action 添加数据：label 是显示在用户接口上的文字，要使 action 可见在菜单或者工具栏可见，menubarPath 和 toolbarPath 是必须的

Extension Element Details

The selected element has no properties to set.

id\*:

ExtensionMenuTest.action2

label\*:

MyLabel

accelerator:

definitionId:

menubarPath:

lars

toolbarPath:

lars

icon:

Browse...

disabledIcon:

Browse...

hoverIcon:

Browse...

tooltip:

helpContextId:

style:

▼

state:

▼

pulldown:

▼

class:

Browse...

retarget:

▼

allowLabelUpdate:

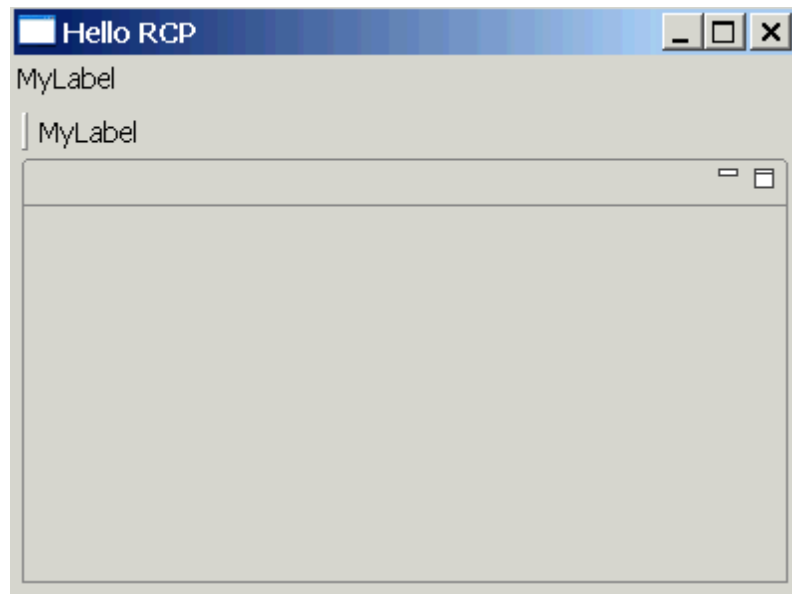
▼

enablesFor:

运行程序，将显示有菜单和工具栏，点击 action，将弹出消息框 “the chosen operation is not currently available” 提示操作不可用。这是因为我们还没有定义接有 action 接口行为的类。Actions 由扩展声明 so-called lazy-loading principle

这表明类是在需要的时候才被加载。

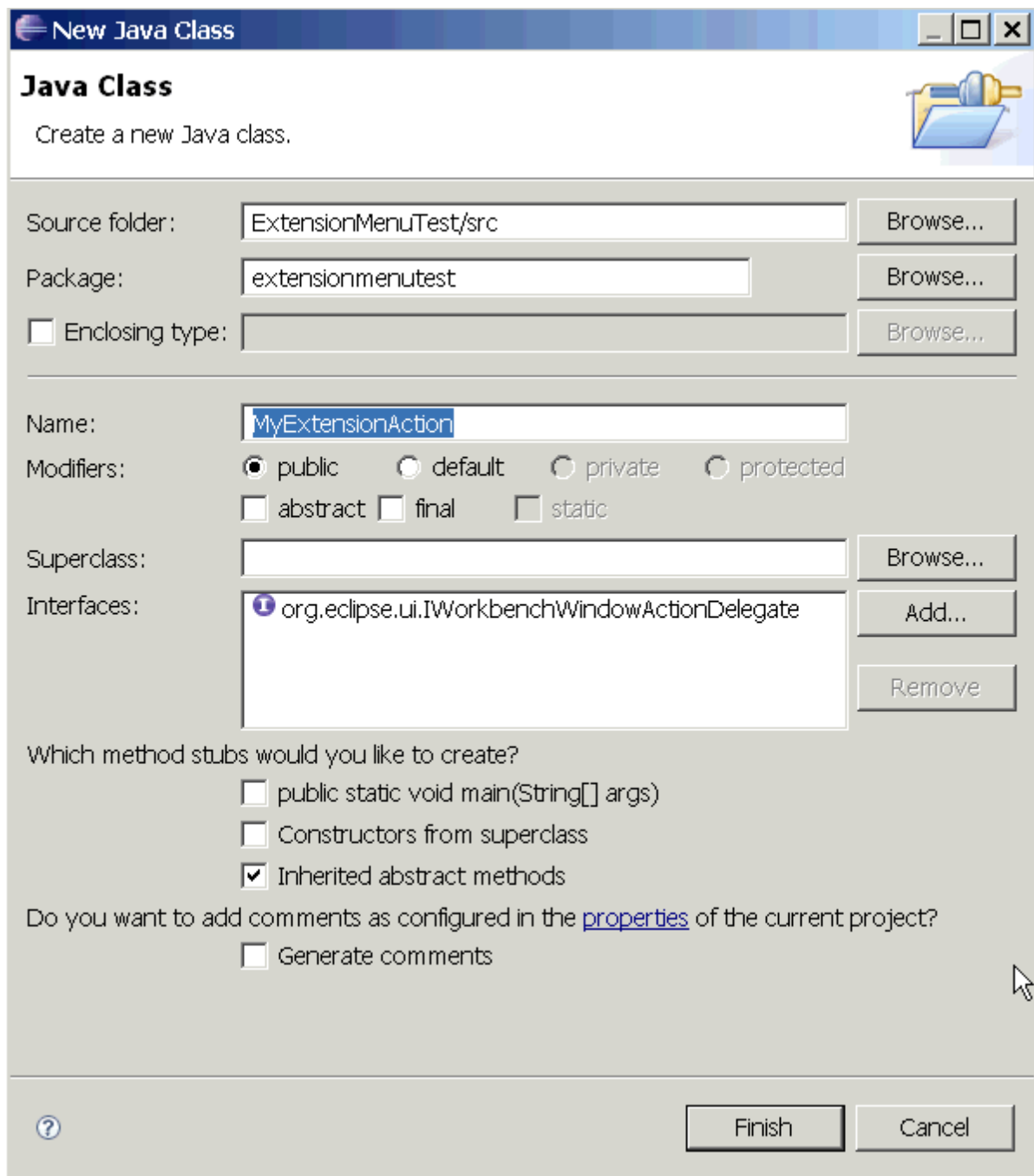
The definition of the class is done via the extension but the real java class is only loaded the first time the extension is activated



回到 `plugin.xml` 的扩展标签，修改类名 “`extensionmenutest.MyExtensionAction`”，点击兰色的连接 “`class`” 直接从 PDEC 创建类。

Extension Element Details	
The selected element has no properties to set.	
<code>id*</code> :	ExtensionMenuTest.action2
<code>label*</code> :	MyLabel
<code>accelerator</code> :	
<code>definitionId</code> :	
<code>menubarPath</code> :	lars
<code>toolbarPath</code> :	lars
<code>icon</code> :	<input type="text"/> Browse...
<code>disabledIcon</code> :	<input type="text"/> Browse...
<code>hoverIcon</code> :	<input type="text"/> Browse...
<code>tooltip</code> :	<input type="text"/>
<code>helpContextId</code> :	<input type="text"/>
<code>style</code> :	<input type="text"/> ▼
<code>state</code> :	<input type="text"/> ▼
<code>pulldown</code> :	<input type="text"/> ▼
<code>class</code> :	extensionmenutest.MyExtensionAction Browse...
<code>retarget</code> :	<input type="text"/> ▼
<code>allowLabelUpdate</code> :	<input type="text"/> ▼
<code>enablesFor</code> :	<input type="text"/>

接受默认设置，点击“Finish”。



使用方法 run 为拥护创建一个精彩的消息框

```
package extensionmenutest;
```

```
import org.eclipse.jface.action.IAction;
```

```
import org.eclipse.jface.dialogs.MessageDialog;
```

```
import org.eclipse.jface.viewers.ISelection;
```

```
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

public class MyExtensionAction implements IWorkbenchWindowActionDelegate {
    private Shell shell;

    public void dispose() {
        // TODO Auto-generated method stub
    }

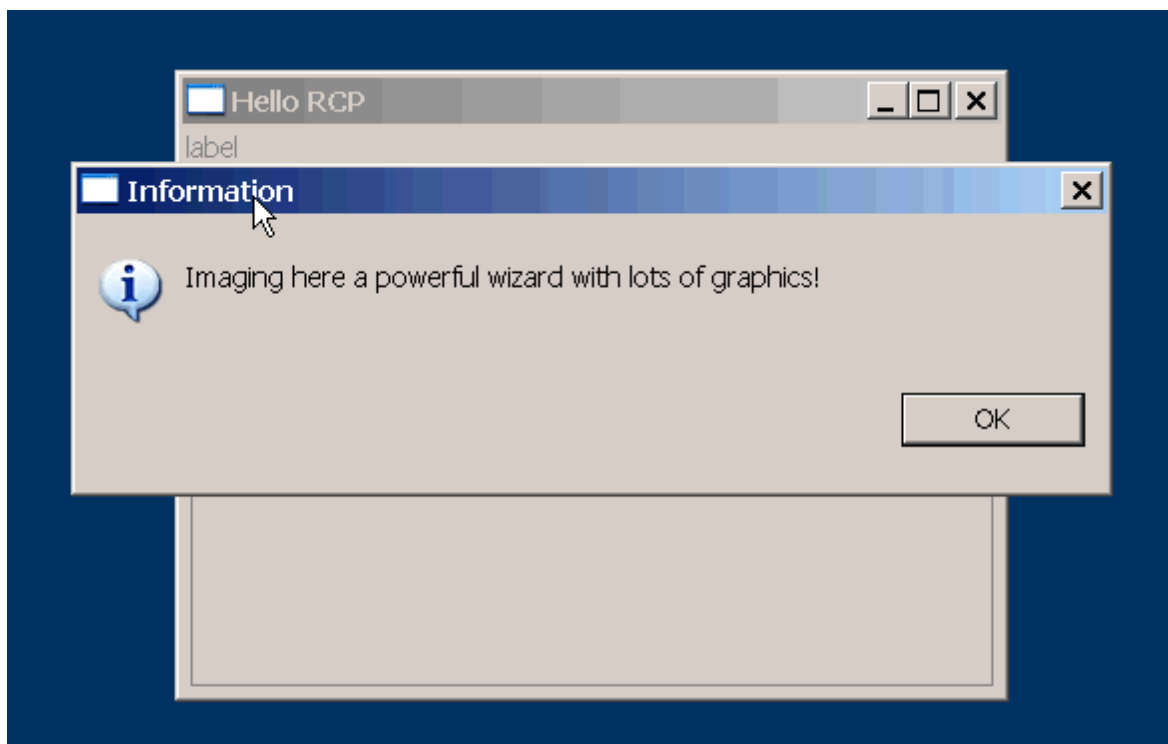
    public void init(IWorkbenchWindow window) {
        // TODO Auto-generated method stub
        shell = window.getShell();
    }

    public void run(IAction action) {
        MessageDialog.openInformation(shell, "Information",
            "Imagine here a powerful wizard with lots of graphics!");
    }

    public void selectionChanged(IAction action, ISelection selection) {
        // TODO Auto-generated method stub
    }
}
```

运行程序，选择你的 action，结果如图





## 第4章添加组合键

### 4.1 概述

用户可以通过键盘进入 actions，例如，用户可以通过 Ctrl+S 保存应用程序里的数据。接下来的章节，我们介绍如何向 actions 指定快捷键

组合键是以命令方式指定给一个 action 的。Commands are declarative components and not related to the implementation of an action.

### 4.2 声明actions的组合键

接下来将描述如何通过“扩展”向 actions 添加组合键。

#### Tip

对预定义 actions 由 ActionFactory 定义的请看一下 ActionFactory 类，and then in classes for the creation of the individual elements，例如，QuitAction。这些类使用的 setActionDefinitionId 必须使用命令 ID。

创建一个新工程，命名为“TestKeybinding”，使用“Hello RCP”模板

添加一个陈述性 action，ID 为“testkeybinding.helloaction”，到程序菜单里，此程序可以弹出一个消息框

为这个 action 创建一个类“testkeybinding.HelloAction”，可以调出一个小的对话框

### Extension Element Details

Set the properties of "action". Required fields are denoted by "\*".

id*:	<input type="text" value="testkeybinding.helloaction"/>	
label*:	<input type="text" value="HelloAction"/>	
accelerator:	<input type="text"/>	
definitionId:	<input type="text"/>	
menubarPath:	<input type="text" value="hello"/>	
toolbarPath:	<input type="text"/>	
icon:	<input type="text"/>	<input type="button" value="Browse..."/>
disabledIcon:	<input type="text"/>	<input type="button" value="Browse..."/>
hoverIcon:	<input type="text"/>	<input type="button" value="Browse..."/>
tooltip:	<input type="text"/>	
helpContextId:	<input type="text"/>	
style:	<input type="text"/>	<input type="button" value="▼"/>
state:	<input type="text"/>	<input type="button" value="▼"/>
pulldown:	<input type="text"/>	<input type="button" value="▼"/>
class:	<input type="text" value="testkeybinding.HelloAction"/>	<input type="button" value="Browse..."/>
retarget:	<input type="text"/>	<input type="button" value="▼"/>
allowLabelUpdate:	<input type="text"/>	<input type="button" value="▼"/>
enablesFor:	<input type="text"/>	

```
package testkeybinding;
```

```
import org.eclipse.jface.action.IAction;
```

```
import org.eclipse.jface.dialogs.MessageDialog;
```

```
import org.eclipse.jface.viewers.ISelection;
```

```
import org.eclipse.swt.widgets.Shell;
```

```
import org.eclipse.ui.IWorkbenchWindow;
```

```
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
```

```
public class HelloAction implements IWorkbenchWindowActionDelegate {
```

```
    private Shell shell;
```

```
    public void dispose() {
```

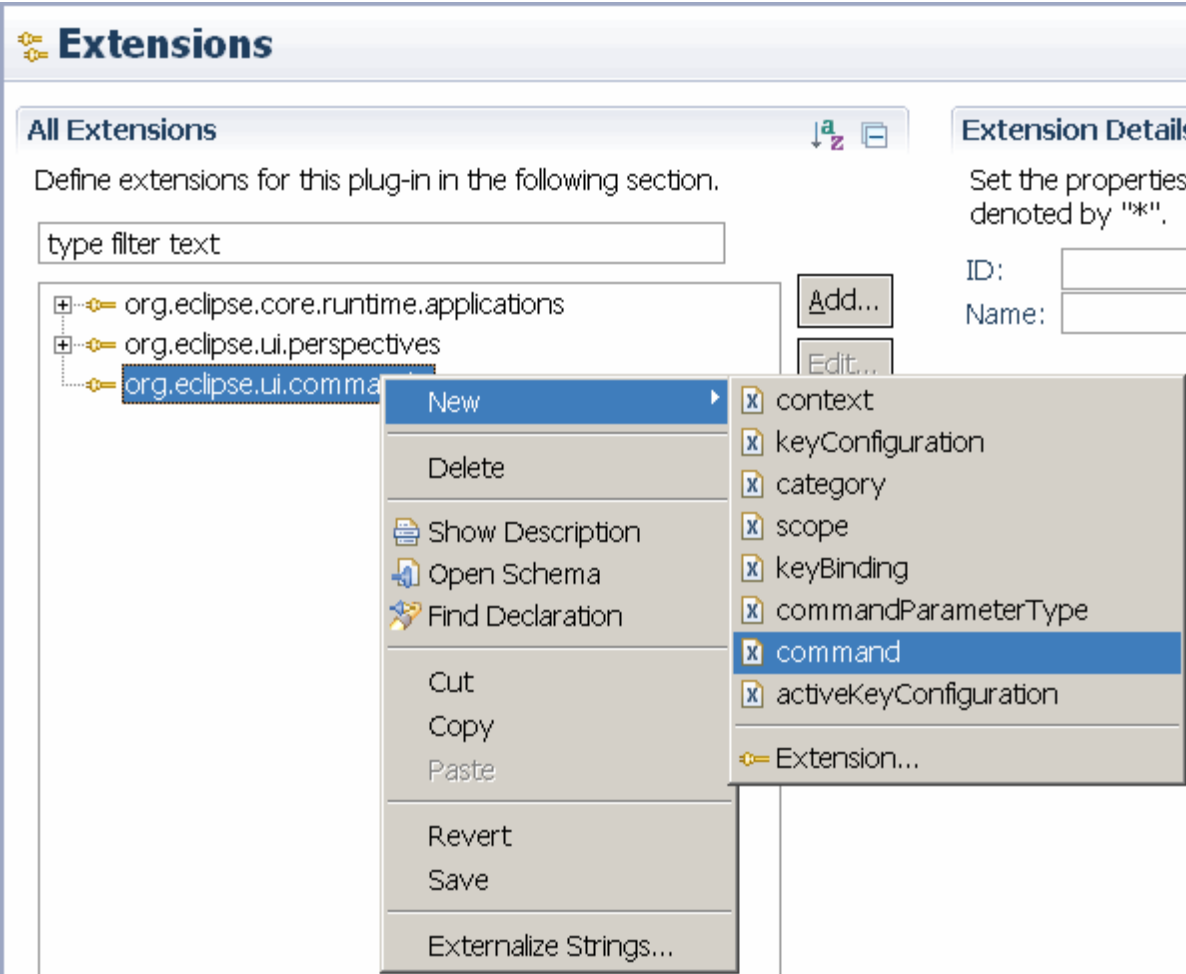
```
        // TODO Auto-generated method stub
    }

    public void init(IWorkbenchWindow window) {
        // TODO Auto-generated method stub
    }

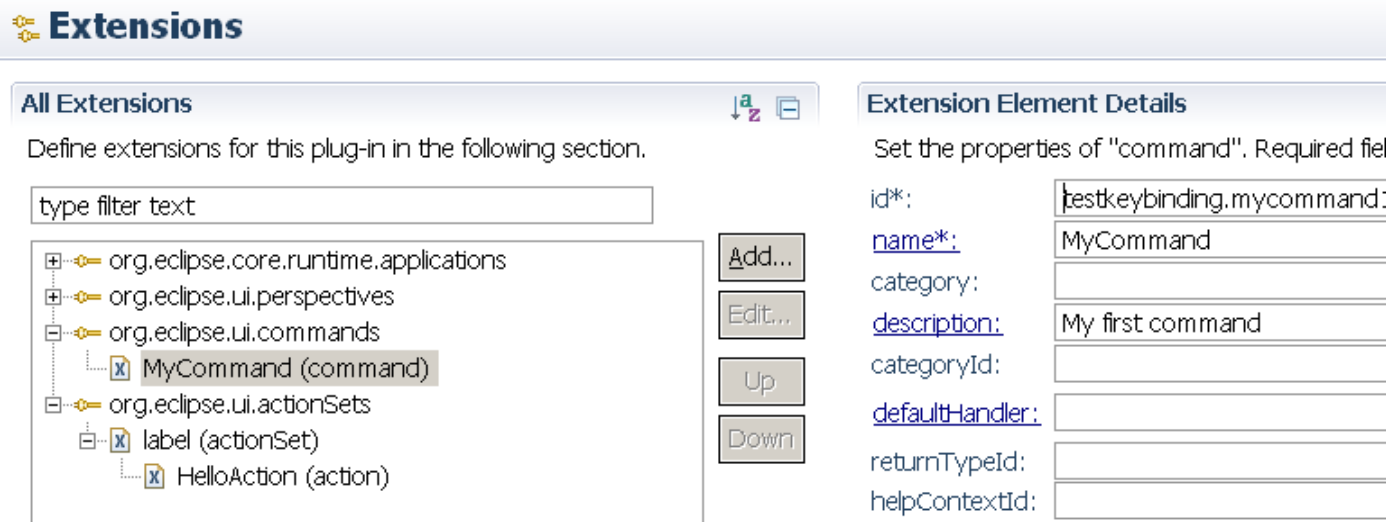
    public void run(IAction action) {
        // TODO Auto-generated method stub
        MessageDialog.openInformation(shell, "Information",
            "Imagine here a powerful wizard with lots of graphics!");
    }

    public void selectionChanged(IAction action, ISelection selection) {
        // TODO Auto-generated method stub
    }
}
```

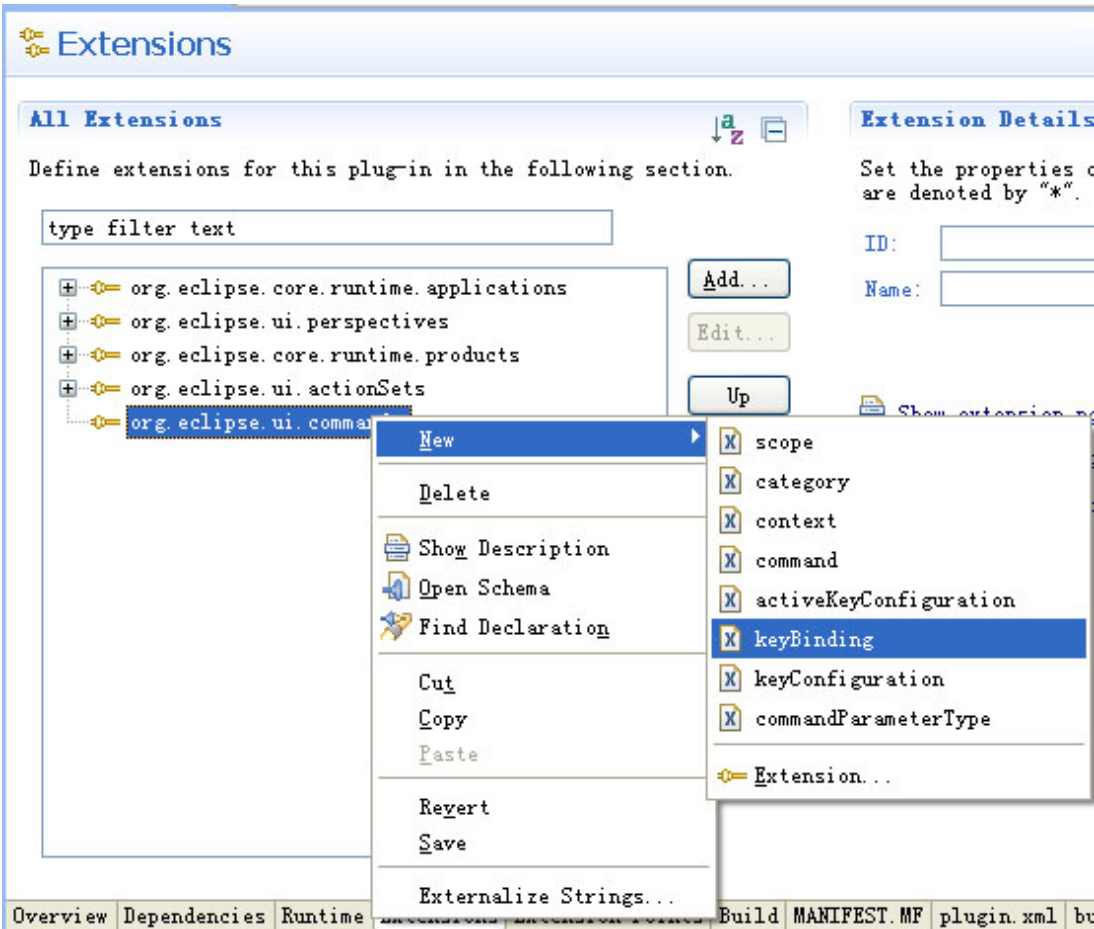
选择 `plugin.xml` 文件，在“扩展”标签里，添加“`org.eclipse.ui.commands`”进入工程。右键点击这个命令扩展，选择 `New->command`



设定下列值：

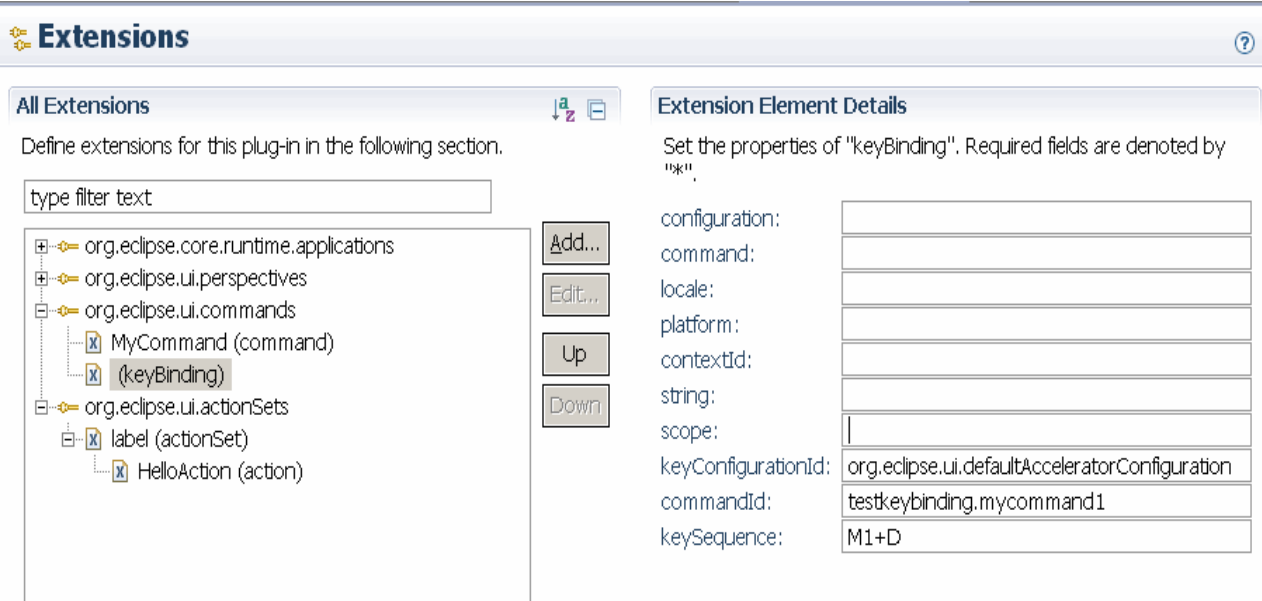


右键点击 command 扩展，选择 New->keyBinding



做如下设定。

“keyConfigurationId” 定义了组合键的有效性 “org.eclipse.ui.defaultAcceleratorConfiguration” 是工作台默认，它将确保组合键在整个程序中有效。commandID 就是你刚才创建的那个 command 的 ID（在这个例子里是 testkeybinding.mycommand1），keysequence 是指调用 command 的快捷键，M1 是指 Ctrl 键



### Extension Element Details

Set the properties of "action". Required fields are denoted by "\*".

id*:	<input type="text" value="testkeybinding.helloaction"/>	
<a href="#">label*</a> :	<input type="text" value="HelloAction"/>	
accelerator:	<input type="text"/>	
definitionId:	<input type="text" value="testkeybinding.mycommand1"/>	
menubarPath:	<input type="text" value="hello"/>	
toolbarPath:	<input type="text"/>	
<a href="#">icon:</a>	<input type="text"/>	<input type="button" value="Browse..."/>
<a href="#">disabledIcon:</a>	<input type="text"/>	<input type="button" value="Browse..."/>
<a href="#">hoverIcon:</a>	<input type="text"/>	<input type="button" value="Browse..."/>
<a href="#">tooltip:</a>	<input type="text"/>	
helpContextId:	<input type="text"/>	
style:	<input type="text"/>	<input type="button" value="▼"/>
state:	<input type="text"/>	<input type="button" value="▼"/>
pulldown:	<input type="text"/>	<input type="button" value="▼"/>
<a href="#">class:</a>	<input type="text" value="testkeybinding.HelloAction"/>	<input type="button" value="Browse..."/>
retarget:	<input type="text"/>	<input type="button" value="▼"/>
allowLabelUpdate:	<input type="text"/>	<input type="button" value="▼"/>
enablesFor:	<input type="text"/>	

## 第5章 系统托盘

接下来我们将添加一个系统托盘图标。如果窗口最小化，程序将在任务面栏上不可见（只有通过任务图标），而且，我们还将为系统托盘图标添加一个菜单

你需要一个 `Display` 来得到一个托盘图标。`Display` 是一个 `SWT` 对象，支持图形系统的存在。这个对象可以被当作 `postWindowOpen()` 方法，在 `ApplicationWorkbenchWindowAdvisor` 中使用。

创建新工程 “TrayTest”，使用 “Hello RCP application” 模板

在 `Application.java` 里定义你的程序 ID “TrayTest”

```
package traytest;

import org.eclipse.equinox.app.IApplication;

public class Application implements IApplication {

    public static final String PLUGIN_ID = "TrayTest";

    .....
```

你的程序一定已经有一个放图标的文件夹了。查看你可用的图标 “alt\_about.gif” 可用性。

在 `ApplicationWorkbenchWindowAdvisor` 输入如下代码

```
package traytest;

import org.eclipse.jface.action.MenuManager;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ShellAdapter;
import org.eclipse.swt.events.ShellEvent;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.graphics.Point;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.Shell;
```



```
import org.eclipse.swt.widgets.Tray;
import org.eclipse.swt.widgets.TrayItem;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;
import org.eclipse.ui.plugin.AbstractUIPlugin;

public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    private TrayItem trayItem;
    private Image trayImage;
    private IWorkbenchWindow window;
    public ApplicationActionBarAdvisor actionBarAdvisor;

    public ApplicationWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurer) {
        super(configurer);
    }

    public ActionBarAdvisor createActionBarAdvisor(IActionBarConfigurer configurer) {
        actionBarAdvisor = new ApplicationActionBarAdvisor(configurer);
        return actionBarAdvisor;
    }

    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowCoolBar(false);
        configurer.setShowStatusLine(false);
    }
}
```

```
    }  
    @Override  
  
    public void postWindowOpen() {  
        super.postWindowOpen();  
        window = getWindowConfigurer().getWindow();  
        trayItem = initTaskItem(window);  
        if (trayItem != null) {  
            // The following coding will will the program if the program is minimized  
            // Not always desired  
            createMinimize();  
            // Create exit and about action on the icon  
            hookPopupMenu(window);  
        }  
    }  
  
    private void hookPopupMenu(IWorkbenchWindow window2) {  
  
        trayItem.addListener(SWT.MenuDetect, new Listener(){  
            public void handleEvent(Event event) {  
                MenuManager trayMenu = new MenuManager();  
                Menu menu = trayMenu.createContextMenu(window2.getShell());  
                actionBarAdvisor.fillTrayItem(trayMenu);  
                menu.setVisible(true);  
            }  
        });  
    }  
  
    private void createMinimize() {  
        window2.getShell().addShellListener(new ShellAdapter(){  
            public void shellIconified(ShellEvent e){
```

```
        window.getShell().setVisible(false);
    }
});
trayItem.addListener(SWT.DefaultSelection, new Listener() {
    public void handleEvent(Event event) {
        Shell shell = window.getShell();
        if (!shell.isVisible()) {
            shell.setVisible(true);
            window.getShell().setMinimized(false);
        }
    }
});
}

private TrayItem initTaskItem(IWorkbenchWindow window) {
    final Tray tray = window.getShell().getDisplay().getSystemTray();
    TrayItem trayItem = new TrayItem(tray, SWT.NONE);
    trayImage = AbstractUIPlugin.imageDescriptorFromPlugin(
        Application.PLUGIN_ID, "/icons/alt_about.gif").createImage();
    trayItem.setImage(trayImage);
    trayItem.setToolTipText("TrayItem");
    return trayItem;
}

public void dispose() {
    if (trayImage != null) {
        trayImage.dispose();
        trayItem.dispose();
    }
}
}
```

此时会提示一些错误，先不要在意，继续编辑其他类文件  
现在创建 actions 和 fillTrayItem 方法

```
package traytest;

import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;

public class ApplicationActionBarAdvisor extends ActionBarAdvisor {

    private IWorkbenchAction exitAction;
    private IWorkbenchAction aboutAction;

    public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
        super(configurer);
    }

    protected void makeActions(IWorkbenchWindow window) {
        exitAction = ActionFactory.QUIT.create(window);
        register(exitAction);
        aboutAction = ActionFactory.ABOUT.create(window);
        register(aboutAction);
    }

    protected void fillMenuBar(IMenuManager menuBar) {
    }
```

```
public void fillTrayItem(MenuManager trayMenu) {  
    trayMenu.add(aboutAction);  
    trayMenu.add(exitAction);  
}  
  
}
```

现在，运行你的程序，看一看程序最小化到系统托盘，右键实验一下菜单。

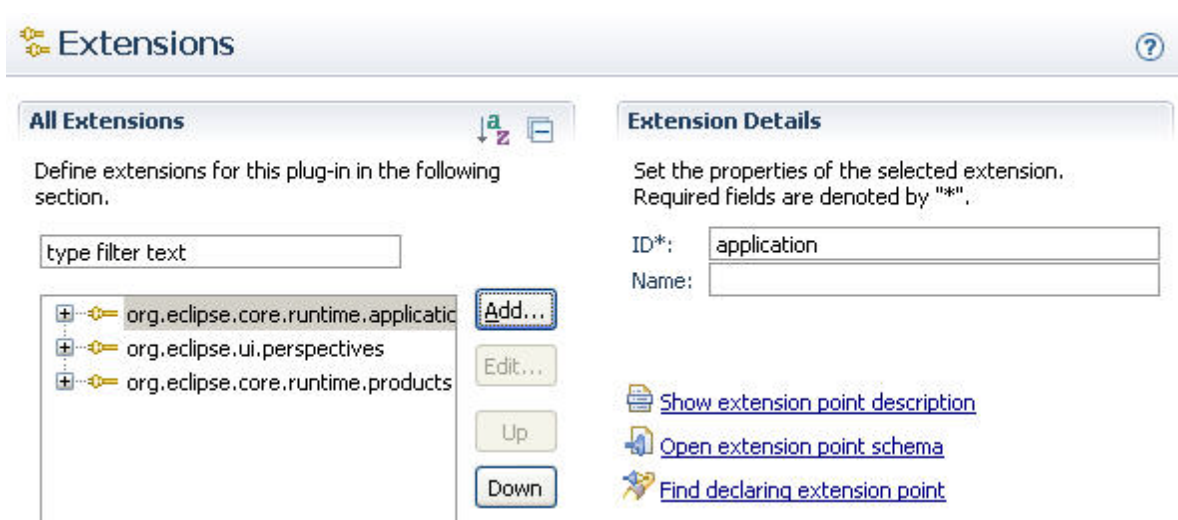
## 第6章外观

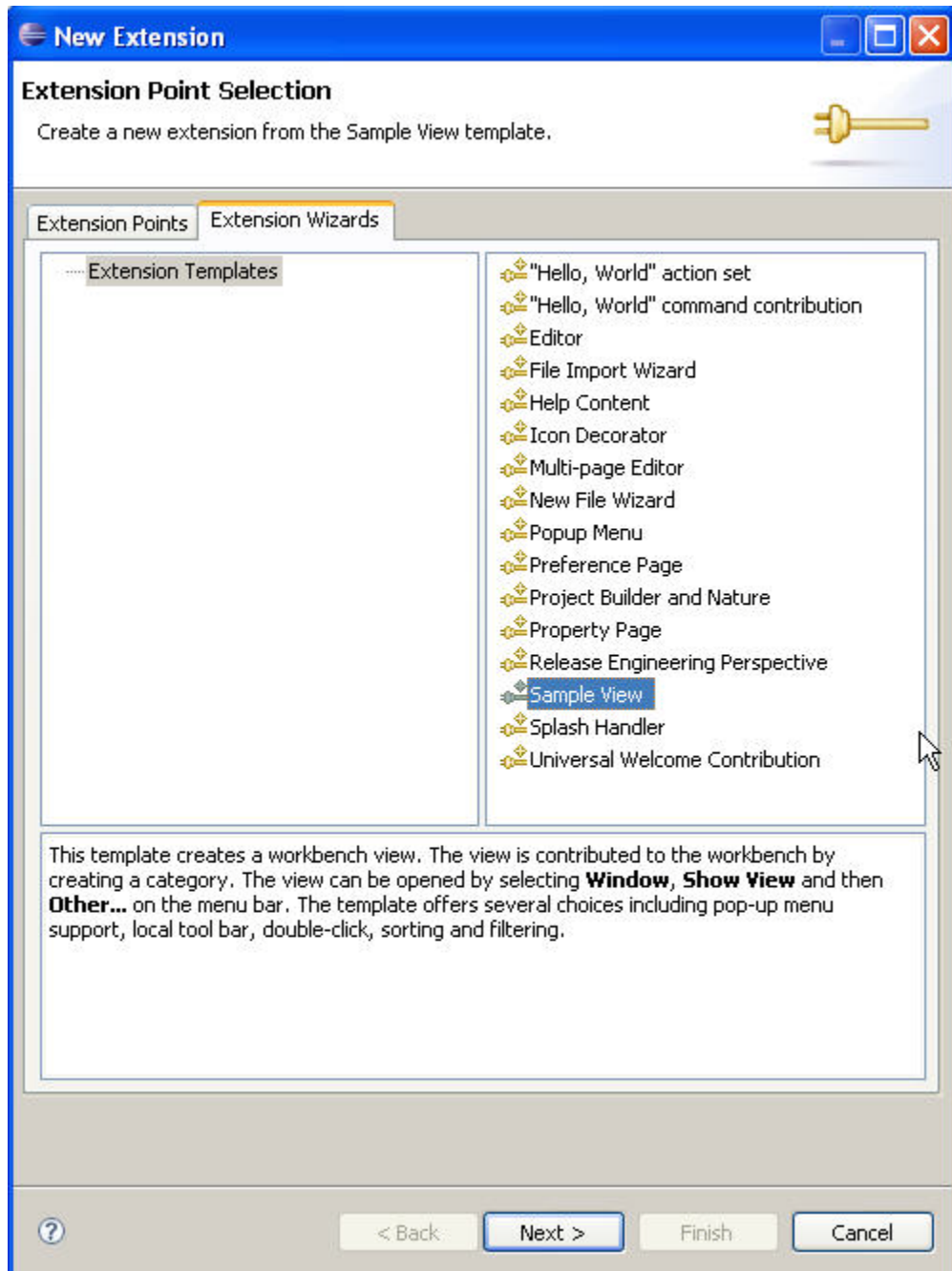
外观可以为任务提供信息，查看通常为信息层次提供导航，打开编辑器，或者浏览属性，下面将介绍，如何向你的应用程序里添加 VIEWS

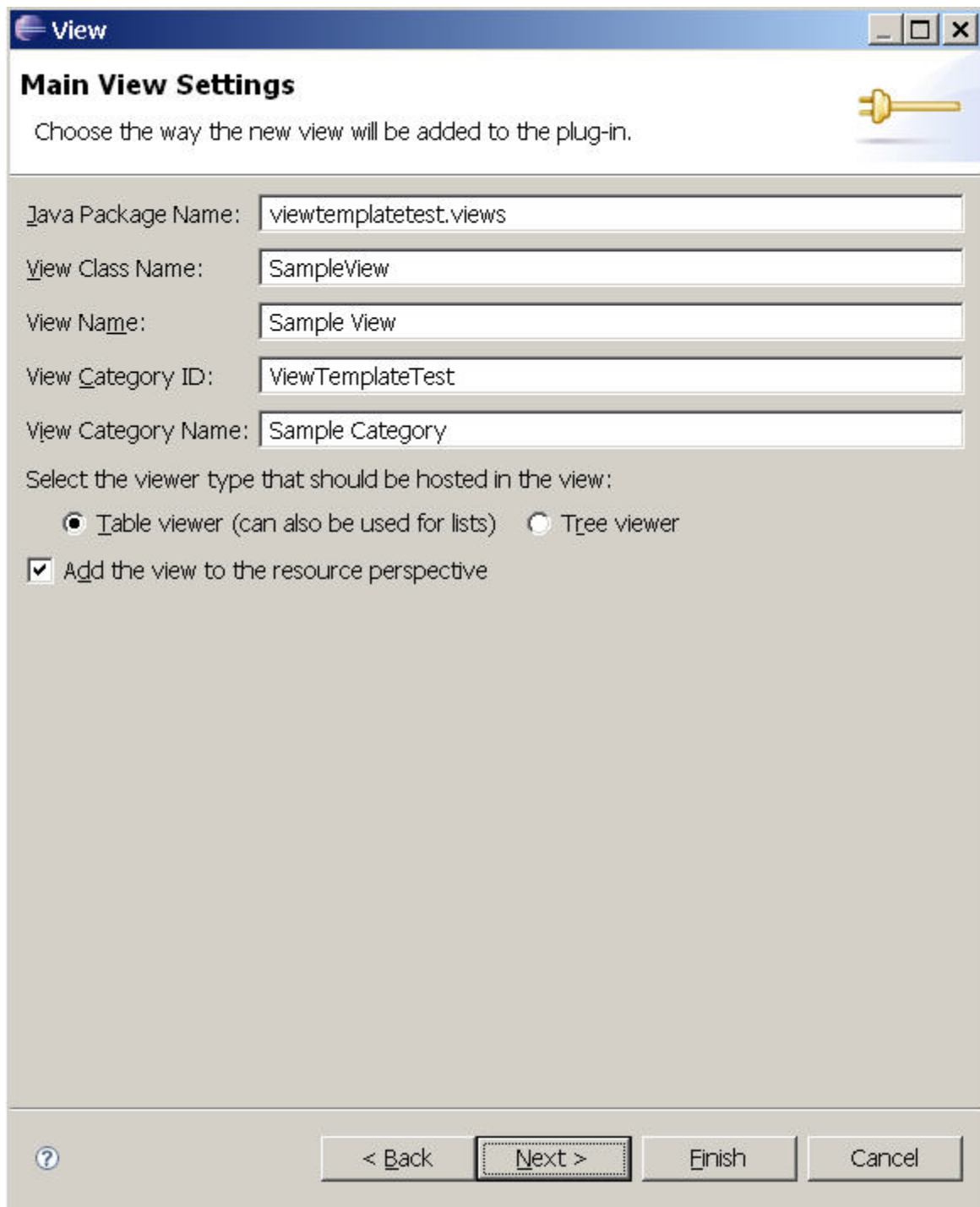
### 6.1 向程序中添加视图模板

创建一个新的工程“ViewTemplateTest”，使用“Hello RCP”模板

选择 plugin.xml 文件，在扩展标签里，点击“Add”按钮，选择“extension wizard”，步骤如下







The image shows a 'View' dialog box titled 'Main View Settings'. It contains several text input fields for configuration: 'Java Package Name' (viewtemplatetest.views), 'View Class Name' (SampleView), 'View Name' (Sample View), 'View Category ID' (ViewTemplateTest), and 'View Category Name' (Sample Category). Below these fields, there is a section for selecting the viewer type, with 'Table viewer (can also be used for lists)' selected by default. A checkbox for 'Add the view to the resource perspective' is also present and checked. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel', along with a help icon.

**View**

### Main View Settings

Choose the way the new view will be added to the plug-in.

Java Package Name:

View Class Name:

View Name:

View Category ID:

View Category Name:

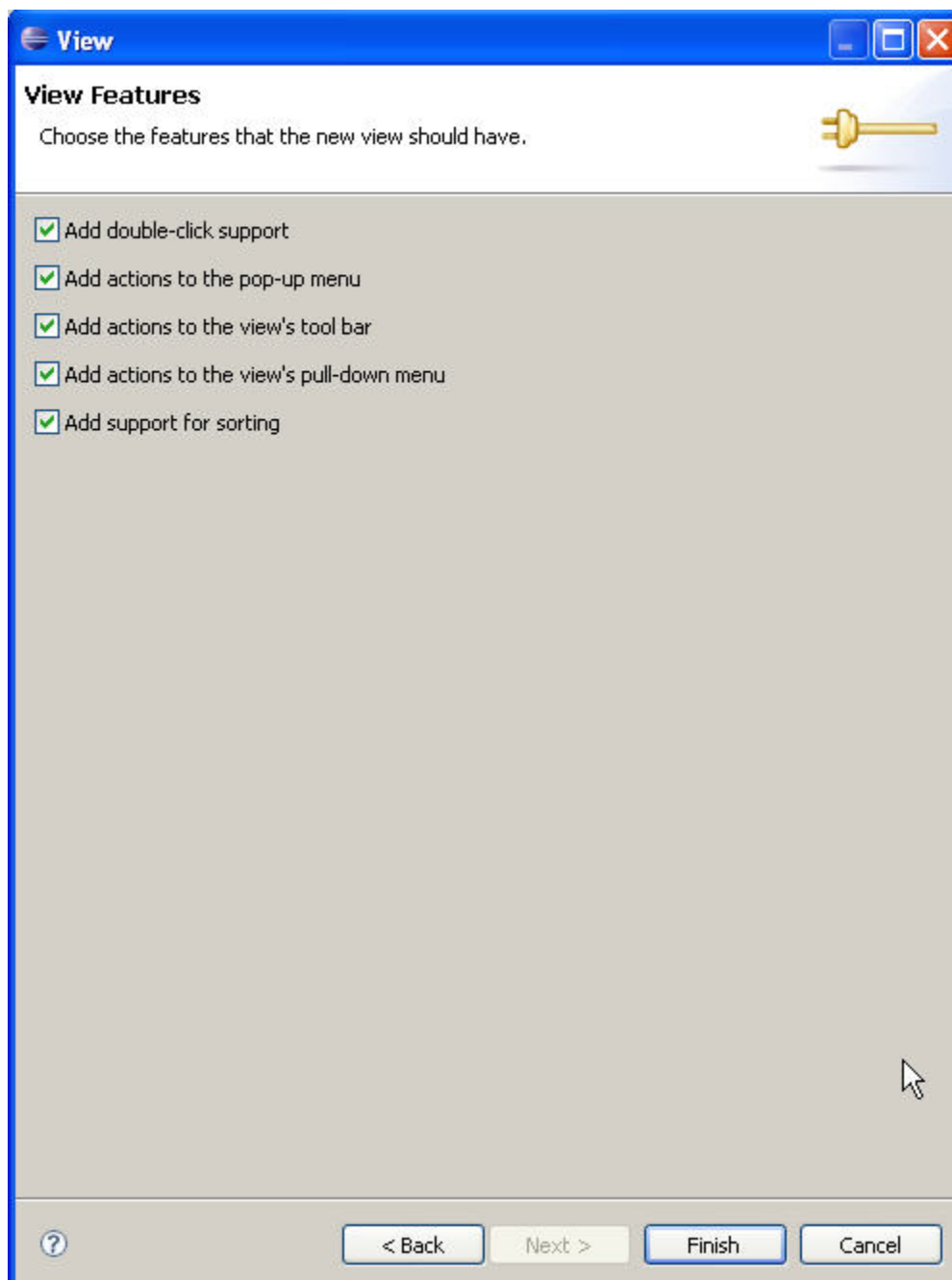
Select the viewer type that should be hosted in the view:

☒ Table viewer (can also be used for lists) ☐ Tree viewer

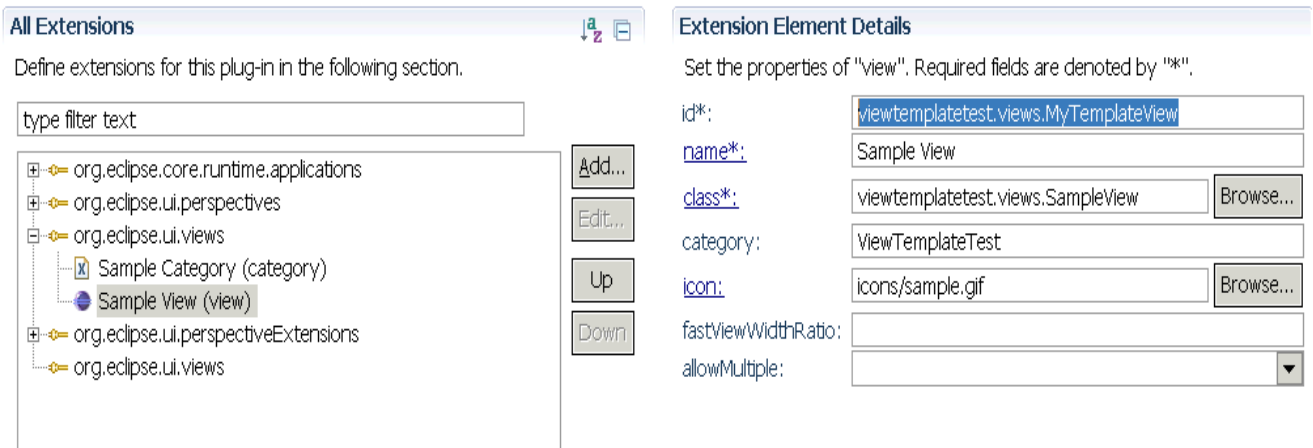
☒ Add the view to the resource perspective

? < Back Next > Finish Cancel





为新 VIEW 更改 ID `viewtemplatetest.views.MyTemplateView`



运行程序，新的 **VIEW** 并没有显示出来，为此，你还要将 **VIEW** 添加到透视图类的 `createInitialLayout` 方法中，`addView` 的第一个参数就是你在 `plugin.xml` 定义的 ID

```
package viewtemplatetest;
```

```
import org.eclipse.ui.IPageLayout;
```

```
import org.eclipse.ui.IPerspectiveFactory;
```

```
public class Perspective implements IPerspectiveFactory {
```

```
    public void createInitialLayout(IPageLayout layout) {
```

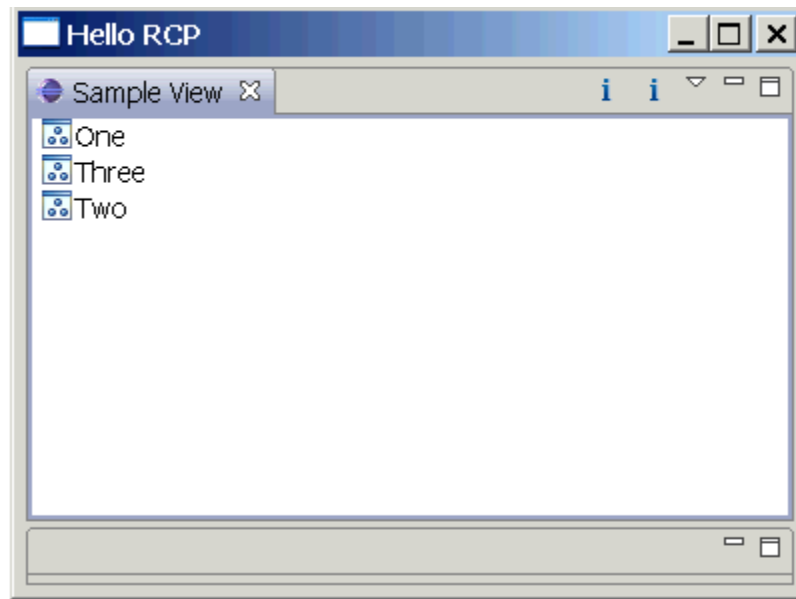
```
        layout.addView("viewtemplatetest.views.MyTemplate View", IPageLayout.TOP,
```

```
            IPageLayout.RATIO_MAX, IPageLayout.ID_EDITOR_AREA);
```

```
    }
```

```
}
```

运行结果如图



注：如果 VIEW 不可关闭，加入如下设定

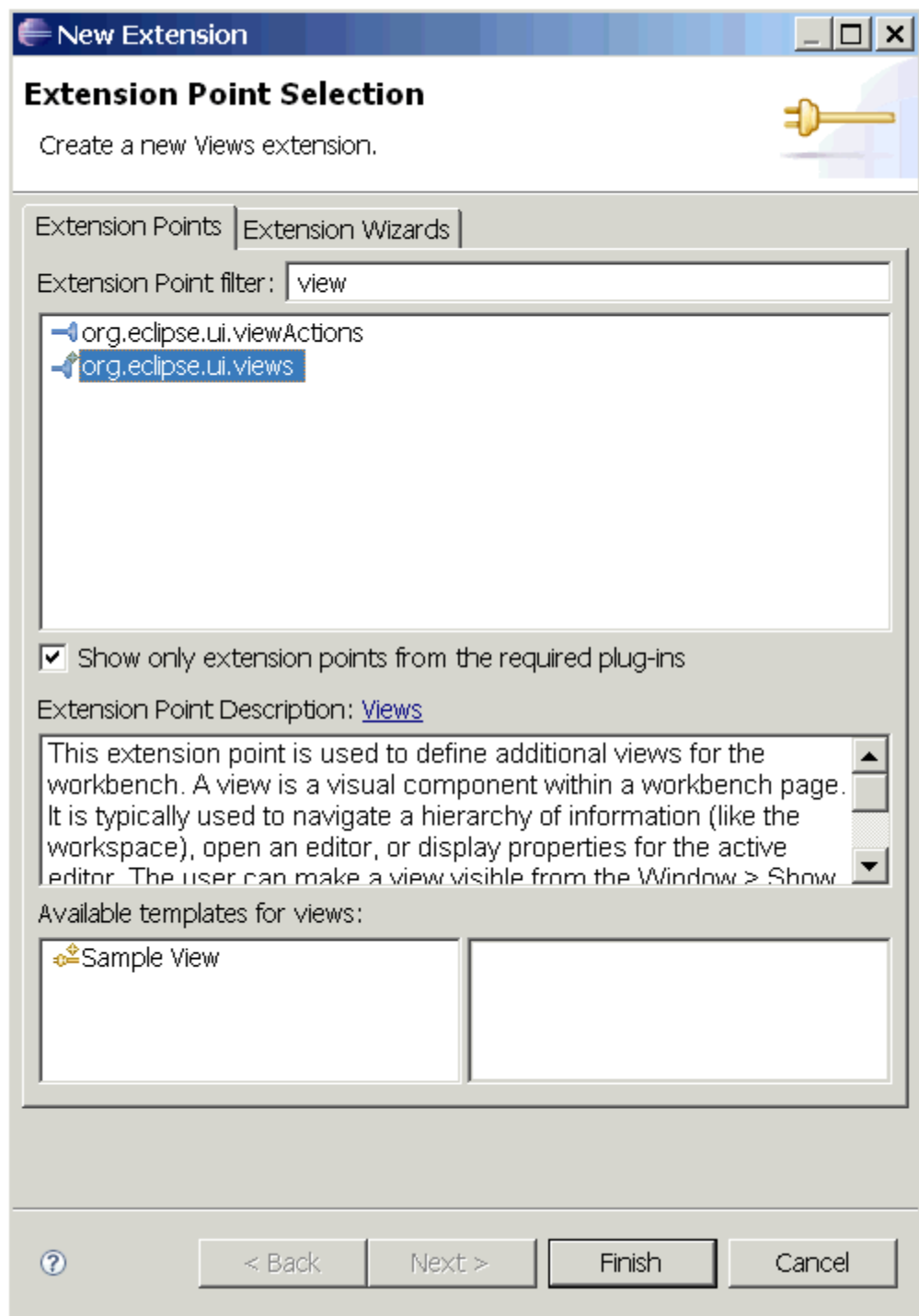
```
layout.getViewLayout("myplugin.views.MySampleView1").setCloseable(false);
```

或者如果所有 VIEWS 不可关闭，在方法 `createInitialLayout()` 在布局里调用 `setFixed(true)`。一个固定视图使得 VIEW 不可调整大小或关闭。也使 VIEWS 不能移动

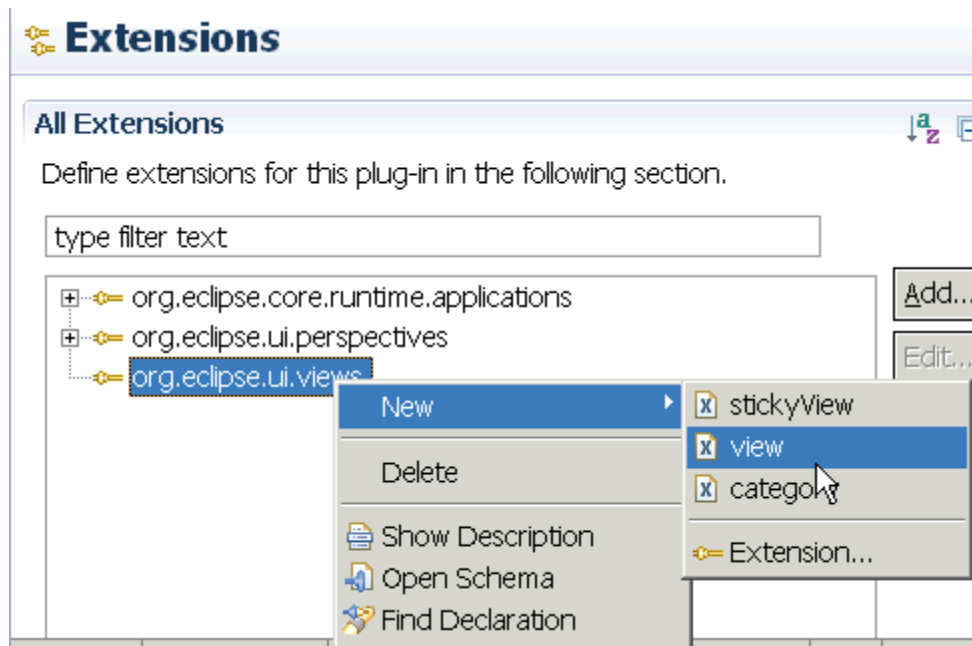
## 6.2 向应用程序添加VIEW

创建新工程“ViewTest”，使用“Hello RCP”模板

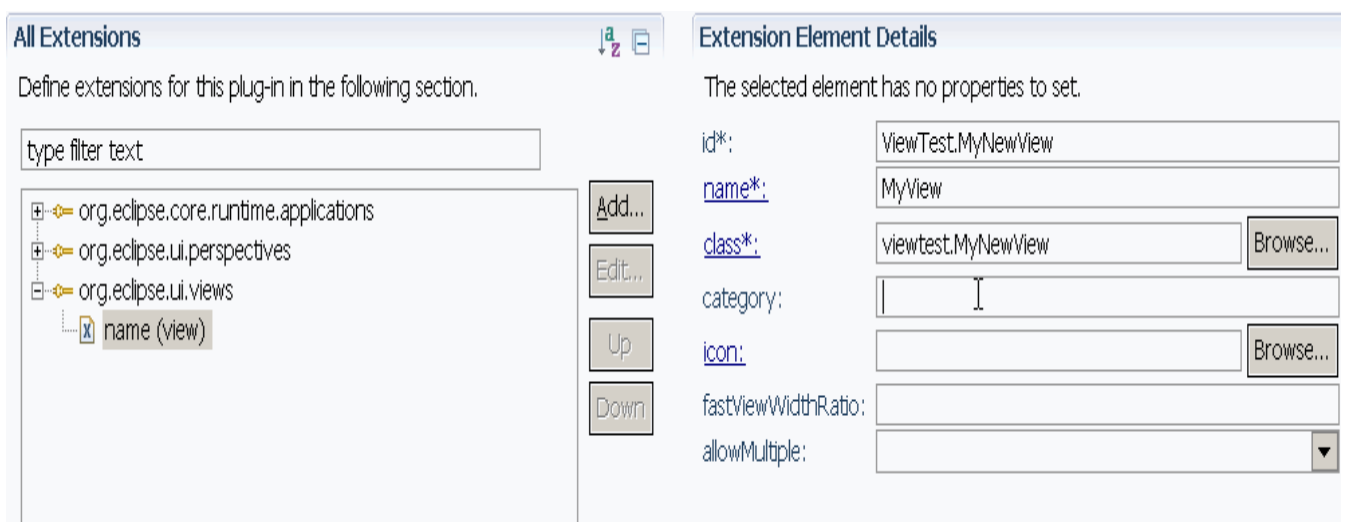
双击 `plugin.xml`，选择扩展标签。点击“Add”按钮，加入 `org.eclipse.ui.views` 扩展



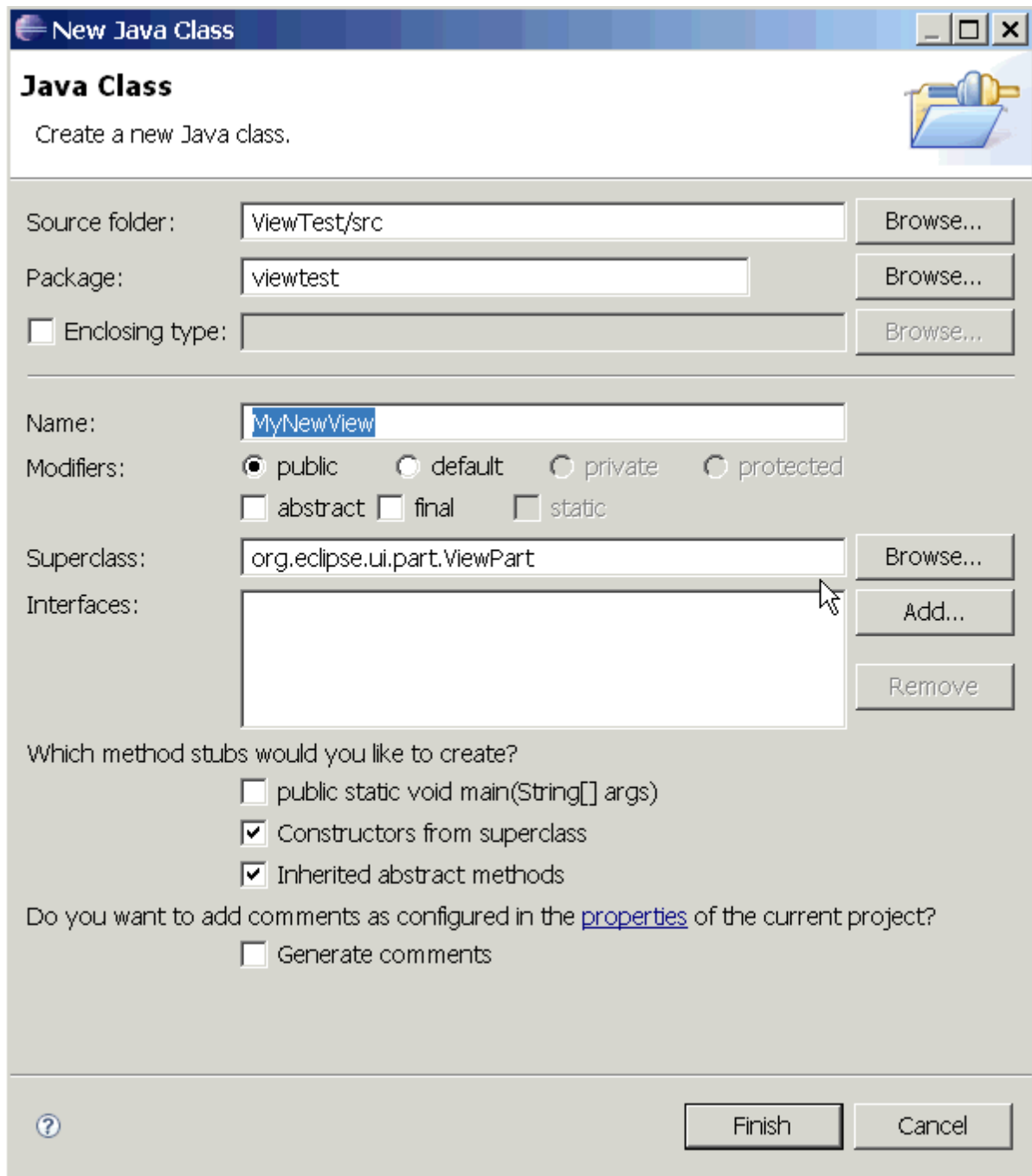
右键点击此扩展，选择 New->View



修改 ID 为 ViewTest.MyNewView 类 viewtest.MyNewView。



创建一个 view 的新类，点击 class 超连接。



在你的新类里如下修改代码：

```
package viewtest;
```

```
import org.eclipse.swt.SWT;
```

```
import org.eclipse.swt.widgets.Composite;
```

```
import org.eclipse.swt.widgets.Text;
```

```
import org.eclipse.ui.part.ViewPart;
```

```
public class MyNewView extends ViewPart {

    public MyNewView() {
        // TODO Auto-generated constructor stub
    }

    @Override
    public void createPartControl(Composite parent) {
        // TODO Auto-generated method stub
        Text text = new Text(parent, SWT.BORDER);
        text.setText("Imagine a fantastic user interface here");

    }

    @Override
    public void setFocus() {
        // TODO Auto-generated method stub

    }

}
```

在 Perspective.java 里添加 View

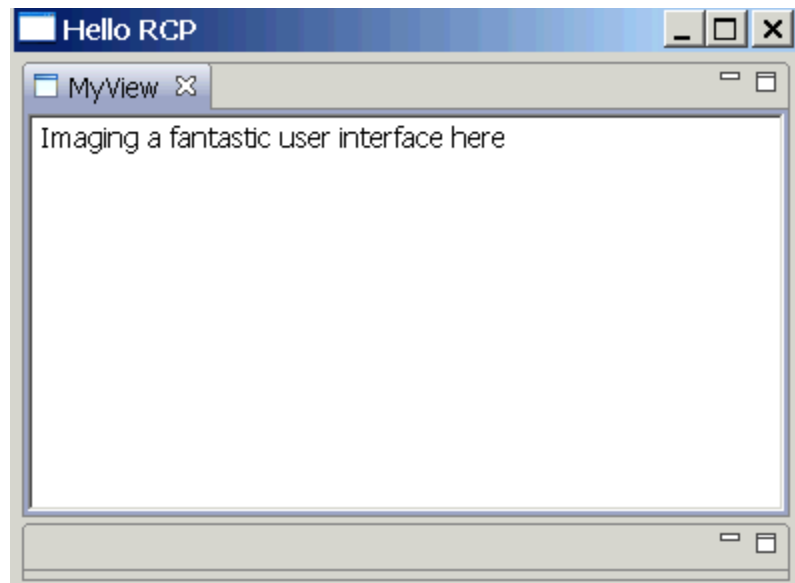
```
package viewtest;

import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;

public class Perspective implements IPerspectiveFactory {
```

```
public void createInitialLayout(IPageLayout layout) {  
    layout.addView("ViewTest.MyNewView", IPageLayout.TOP,  
        IPageLayout.RATIO_MAX, IPageLayout.ID_EDITOR_AREA);  
  
}  
}
```

运行结果如图：



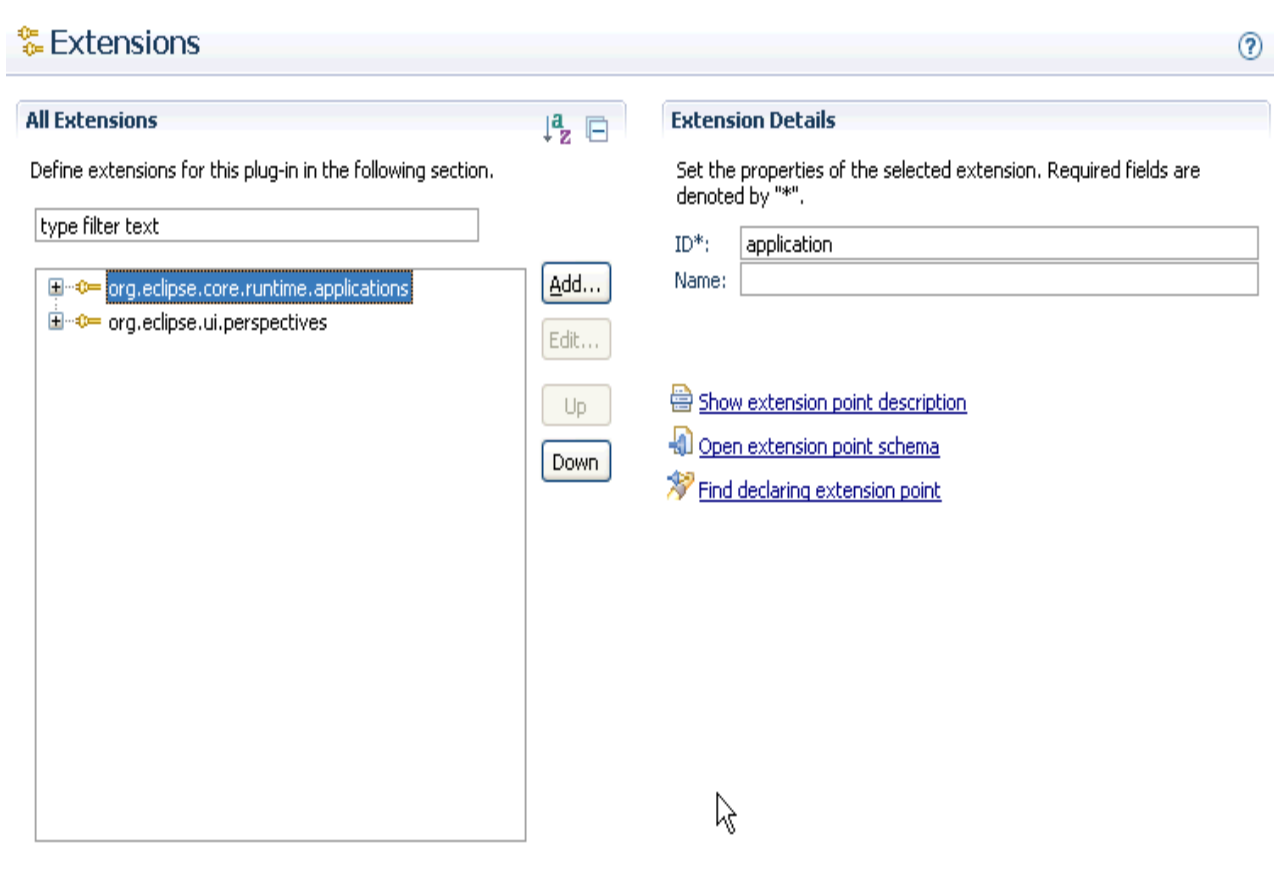
### 6.3 向VIEW里添加action

除了可以向程序添加菜单/工具条之外，你还可以向 VIEW 添加菜单和按钮。这些 action 具有 VIEW 的数据入口，因此你可以在你的 action 里直接使用 VIEW 数据

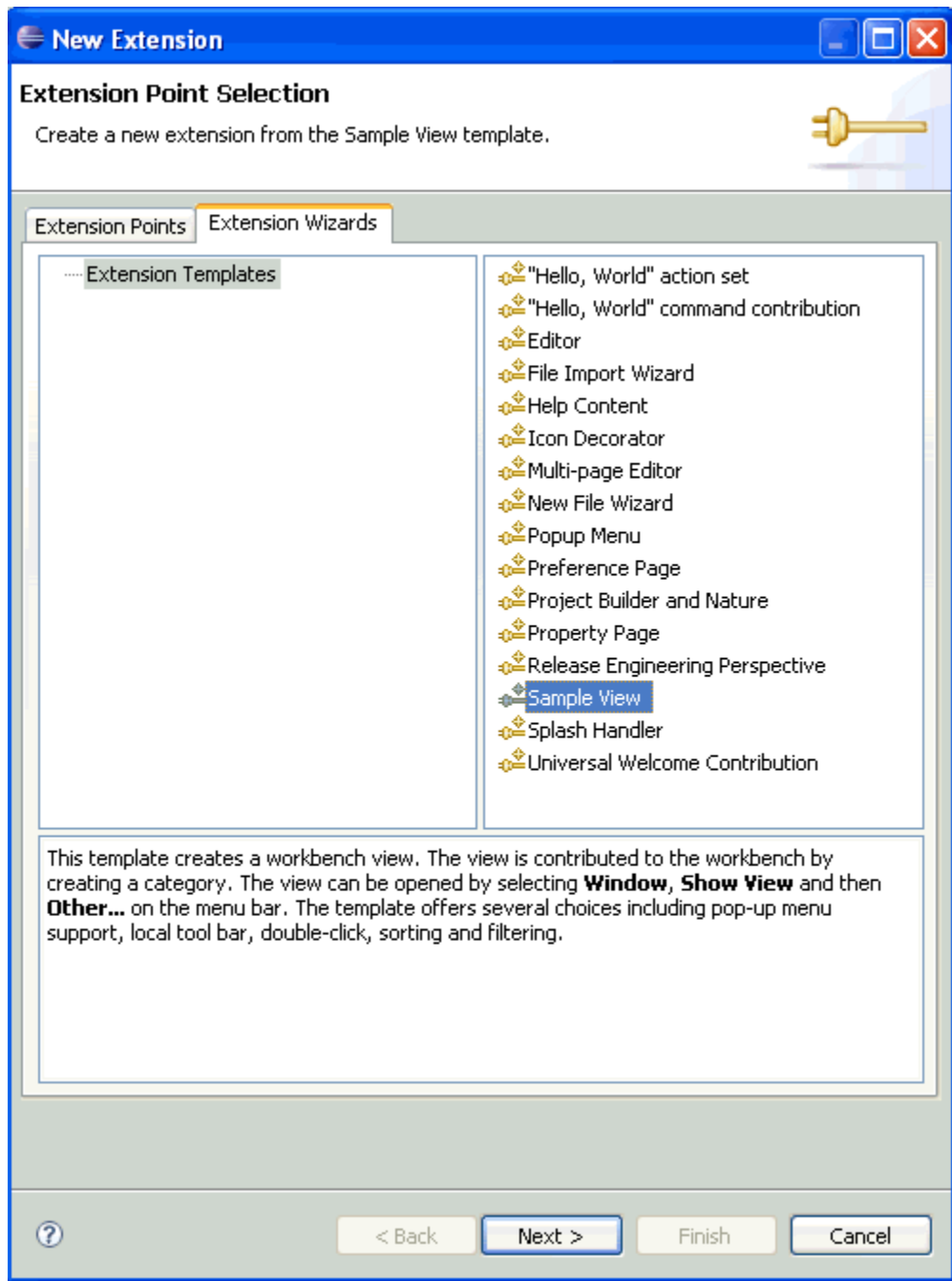
使用“Hello RCP”模板创建一个新工程“AddActiontoView”

双击 plugin.xml，选择“扩展”标签

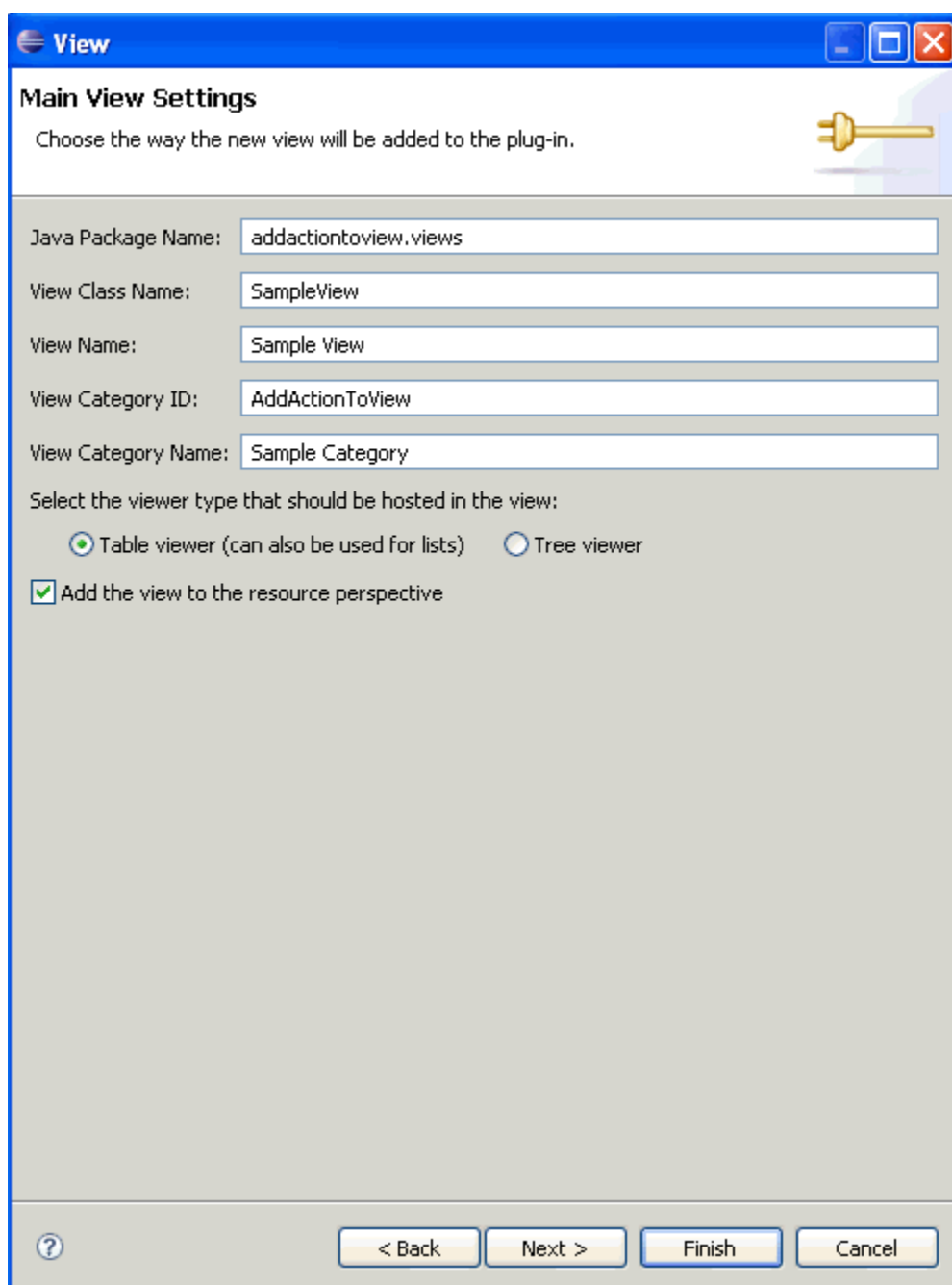




点击“Add”按钮，选择“Extension wizard”标签。选择 Sample View，用以下设定创建样本视图



然后 add->new action



The screenshot shows a dialog box titled "View" with a blue header bar. Below the header, the title "Main View Settings" is displayed. A subtitle reads "Choose the way the new view will be added to the plug-in." To the right of the subtitle is a yellow plug icon. The dialog contains several text input fields: "Java Package Name" with the value "addactiontoview.views", "View Class Name" with "SampleView", "View Name" with "Sample View", "View Category ID" with "AddActionToView", and "View Category Name" with "Sample Category". Below these fields, a section titled "Select the viewer type that should be hosted in the view:" contains two radio buttons: "Table viewer (can also be used for lists)" which is selected, and "Tree viewer". Below the radio buttons is a checked checkbox labeled "Add the view to the resource perspective". At the bottom of the dialog, there is a question mark icon on the left and four buttons: "< Back", "Next >", "Finish", and "Cancel".

**View**

**Main View Settings**

Choose the way the new view will be added to the plug-in.

Java Package Name:

View Class Name:

View Name:

View Category ID:

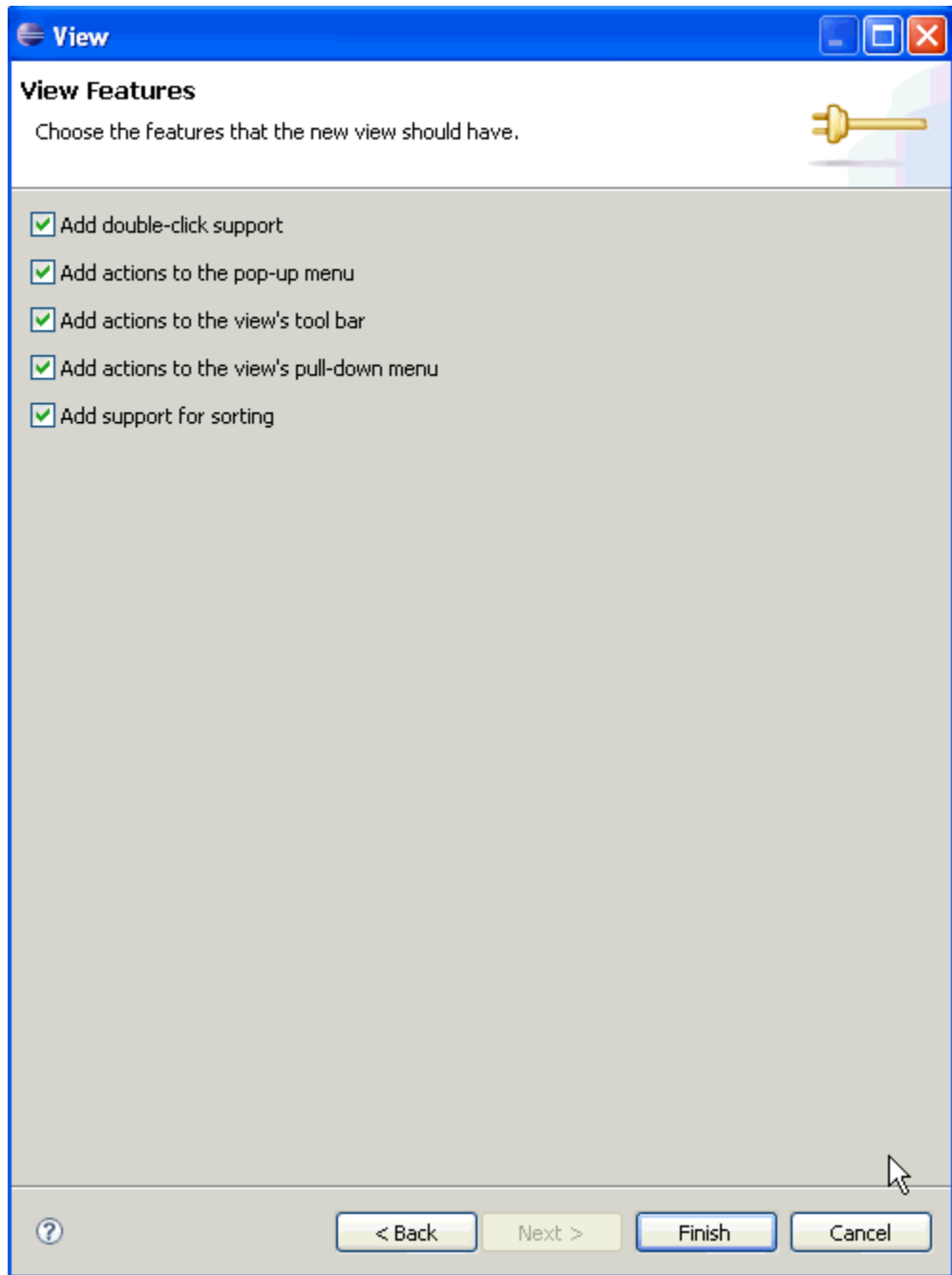
View Category Name:

Select the viewer type that should be hosted in the view:

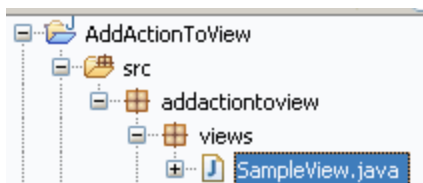
☒ Table viewer (can also be used for lists) ☐ Tree viewer

☒ Add the view to the resource perspective

? < Back Next > Finish Cancel



选择你新视图的代码，加入识别 ID 的静态变量。



```
public class SampleView extends ViewPart {  
    public static final String ID="addactiontoview.views.SampleView";  
    private TableViewer viewer;
```

```
private Action action1;  
private Action action2;  
private Action doubleClickAction;
```

将 VIEW 加入你的 RCP 程序中：选择 perspective.java，修改如下数据代码：

```
package addactionview;
```

```
import org.eclipse.ui.IPageLayout;  
import org.eclipse.ui.IPerspectiveFactory;
```

```
import addactionview.views.SampleView;
```

```
public class Perspective implements IPerspectiveFactory {
```

```
    public void createInitialLayout(IPageLayout layout) {
```

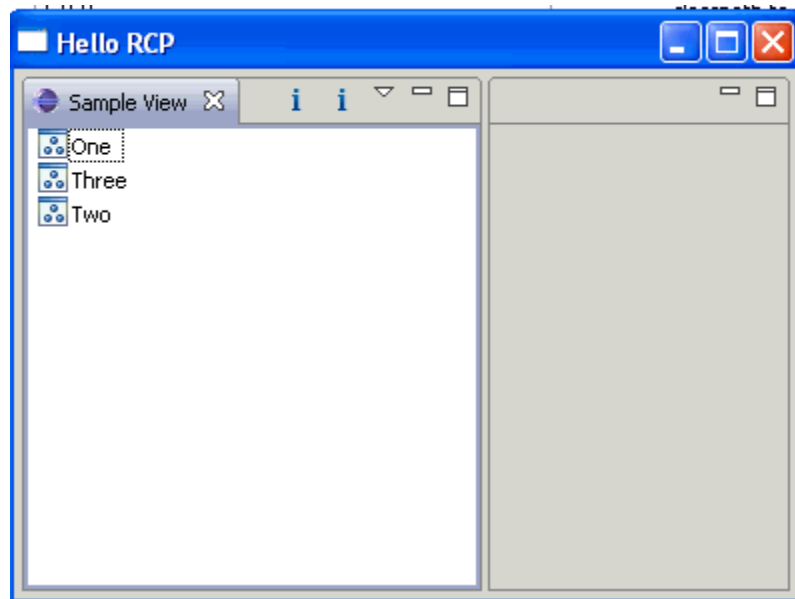
```
        String editorArea = layout.getEditorArea();
```

```
        layout.addView(SampleView.ID, IPageLayout.LEFT, 0.60f, editorArea);
```

```
    }
```

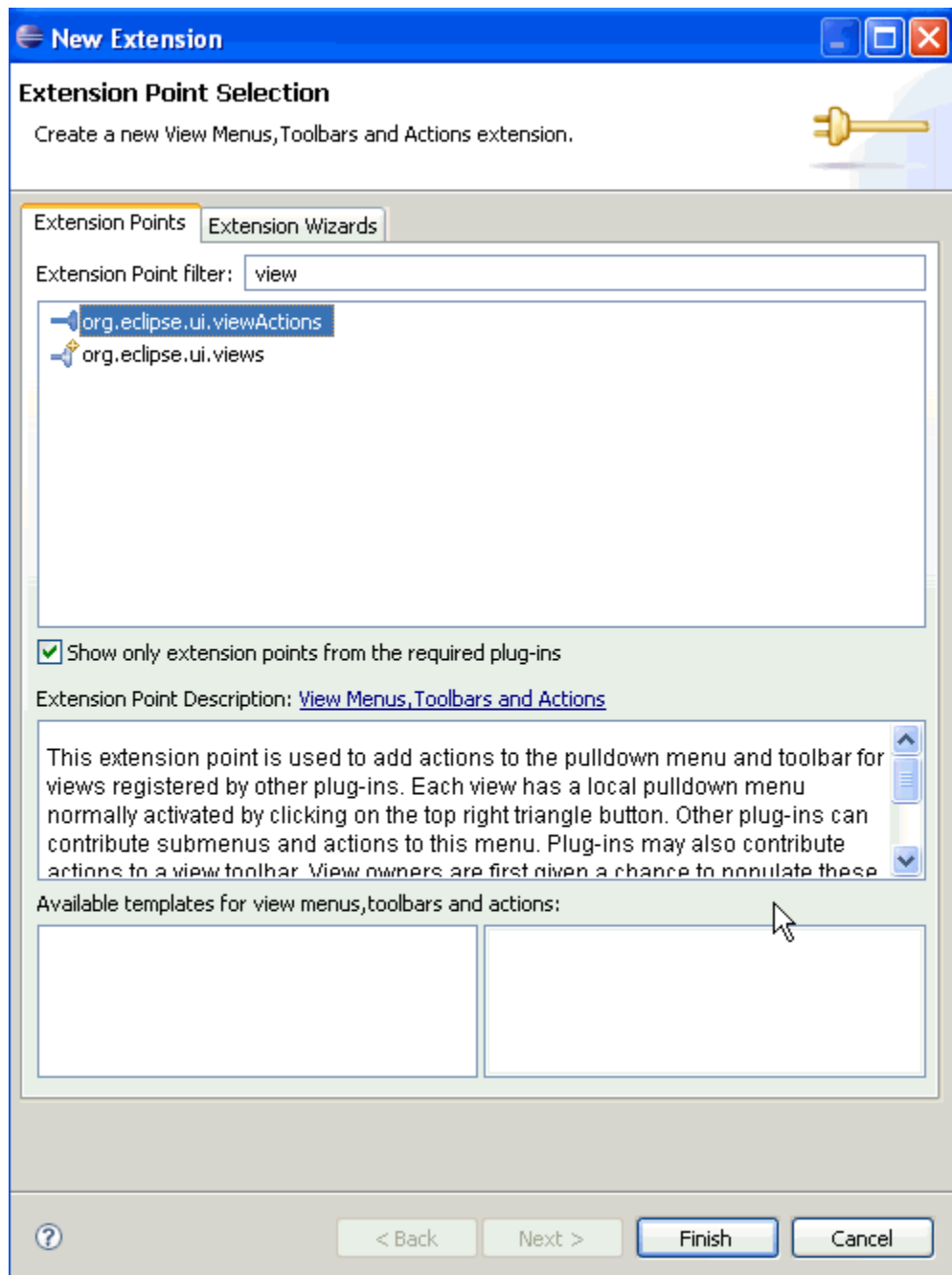
```
}
```

运行，如图

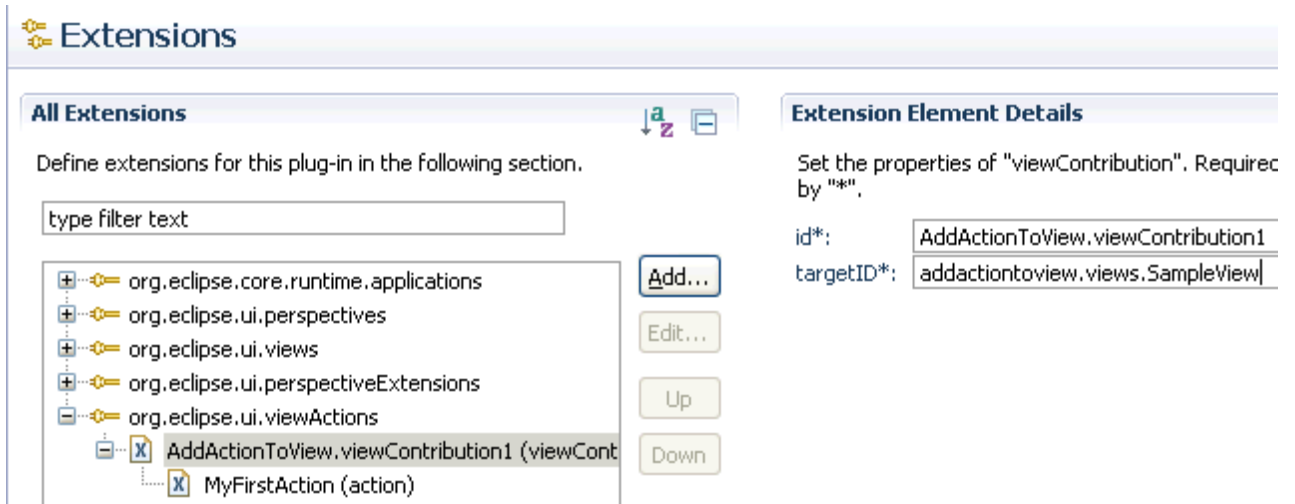


现在我们已经准备好一切向这个 VIEW 添加 ACTION 了，这是我们这一章的目的所在  
再次选择 plugin.xml 的 “Extensions”

点击 “Add” 按钮，选择 `org.eclipse.ui.ViewActions`

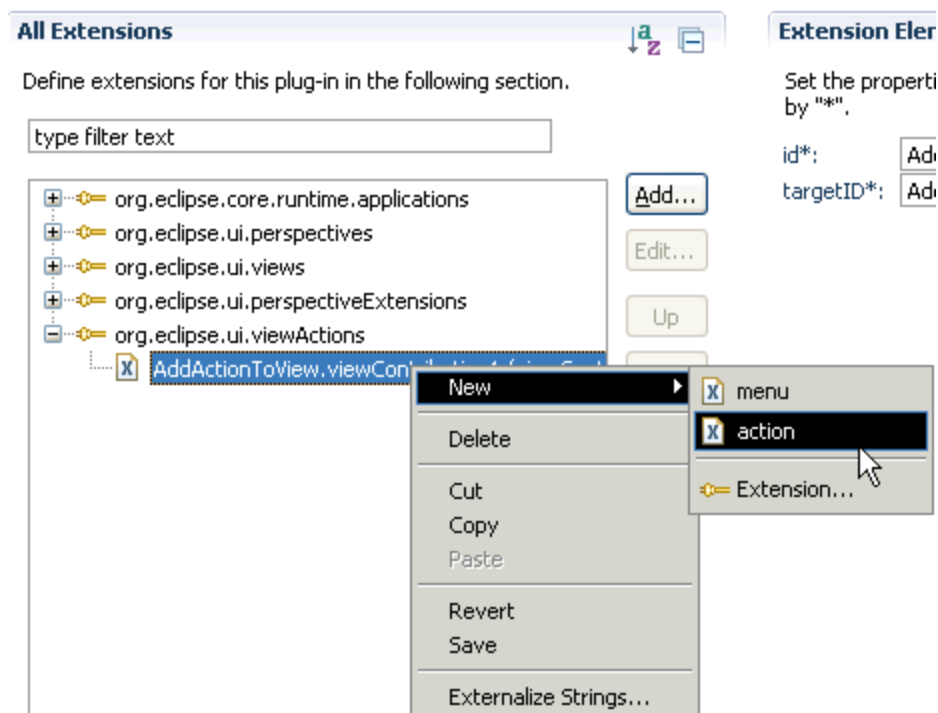


现在是最重要的了，将 `targetID` 改为你早先创建的 `view` 名称。`targetID` 是 `view action` 和 `view` 之间的连接

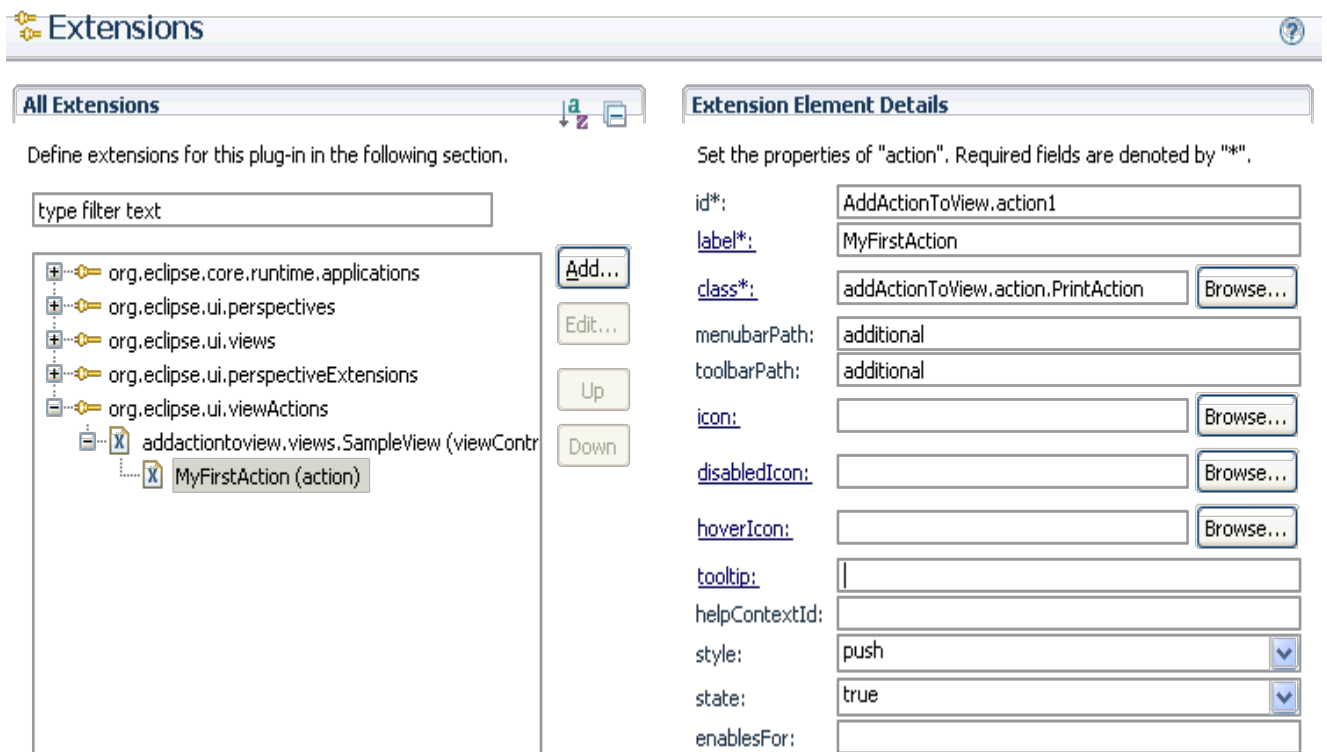


右键点击新建的 viewaction，选择 New->action，改变标签为 MyFirstAction。确认你填写了 menubarPath 和 toolbarPath。

如果不是特别的 menubarPath 或者 toolbarPath，你的 action 将不被显示在菜单或者工具栏内。这里我也同样修改了 action 的名字为 “addactiontoview.action.PrintAction”

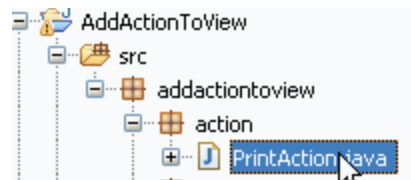






再次运行你的程序。菜单和工具栏里出现 action 了。如果你点击 action 你将得到一个弹出，你的 action 不可用。这是因为他不具有行为类

创建你所指定的类，下面这段代码将是 action 产生一个消息框



```
package addactiontoview.action;
```

```
import org.eclipse.jface.action.IAction;
```

```
import org.eclipse.jface.dialogs.MessageDialog;
```

```
import org.eclipse.jface.viewers.ISelection;
```

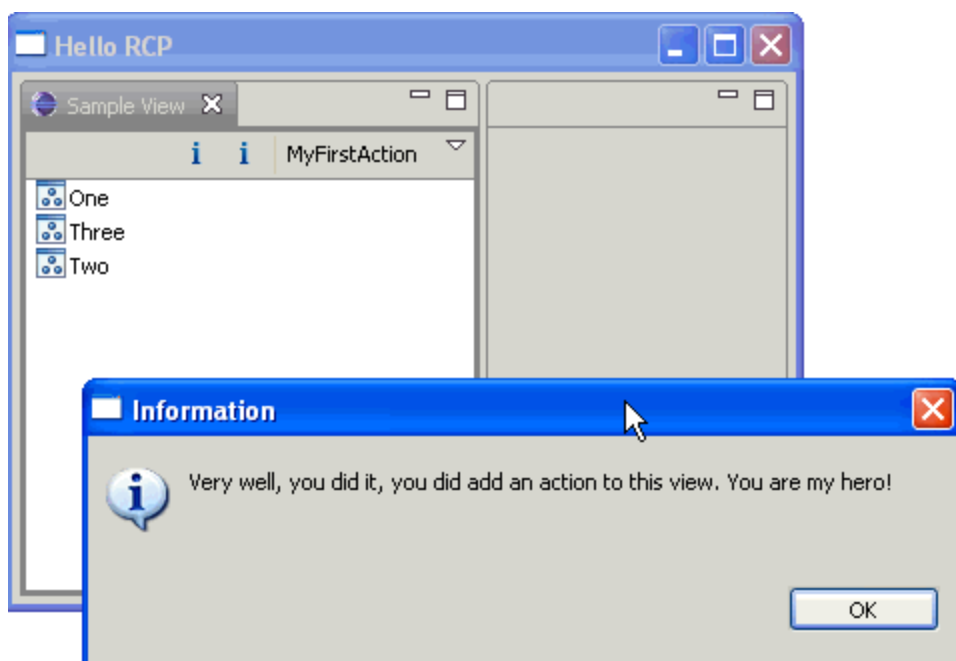
```
import org.eclipse.ui.IViewActionDelegate;
```

```
import org.eclipse.ui.IViewPart;
```

```
import addactiontoview.views.SampleView;
```

```
public class PrintAction implements IViewActionDelegate {  
    SampleView myView;  
  
    public void init(IViewPart view) {  
        // TODO Auto-generated method stub  
        this.myView = (SampleView) view;  
    }  
  
    public void run(IAction action) {  
        // TODO Auto-generated method stub  
        MessageDialog.openInformation(myView.getViewSite().getShell(),  
            "Information",  
            "Very well, you did it, you did add an action to this view. You are my hero!");  
    }  
  
    public void selectionChanged(IAction action, ISelection selection) {  
        // TODO Auto-generated method stub  
    }  
}
```

现在，运行你的程序，你的视口里将出现一个新按钮，它连接着 **action**，点击它，一个消息框弹出，内容为刚才在代码中设定的文字。



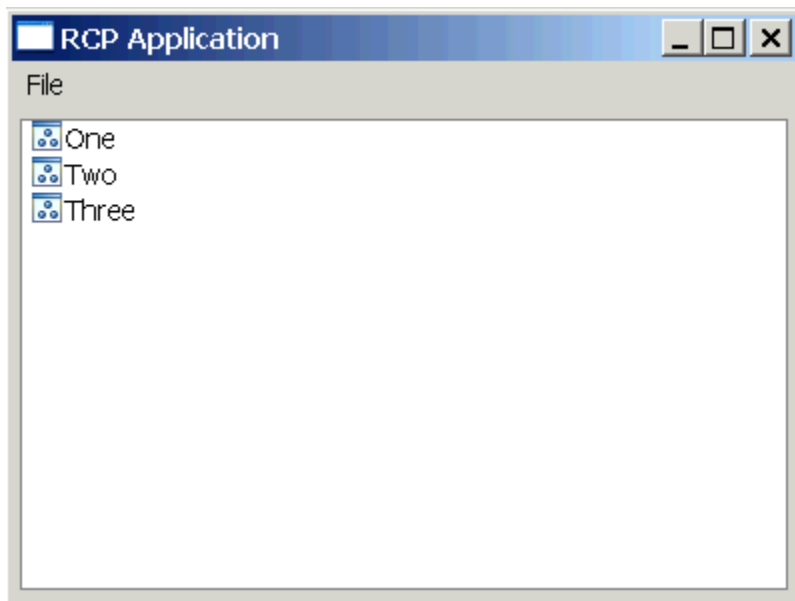


## 第7章 和编辑器一起工作

### 7.1 概述

### 7.2 创建工程

使用“RCP application with view”模板创建一个新工程



### 7.3 创建并准备domain 模型

创建一个包 mydomain。在包里建一个新类 Content.java

```
package mydomain;
```

```
import java.util.ArrayList;
```

```
public class Content extends ArrayList {  
    public class Person {  
        private String firstName;  
        private String lastName;  
  
        public Person(String firstName, String lastName) {
```

```
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}

private ArrayList<Person> persons = new ArrayList<Person>();

public Content() {
    // Just for testing we hard-code the persons here:
    persons.add(new Person("Lars", "Vogel"));
}
```

```
        persons.add(new Person("Jim", "Knopf"));

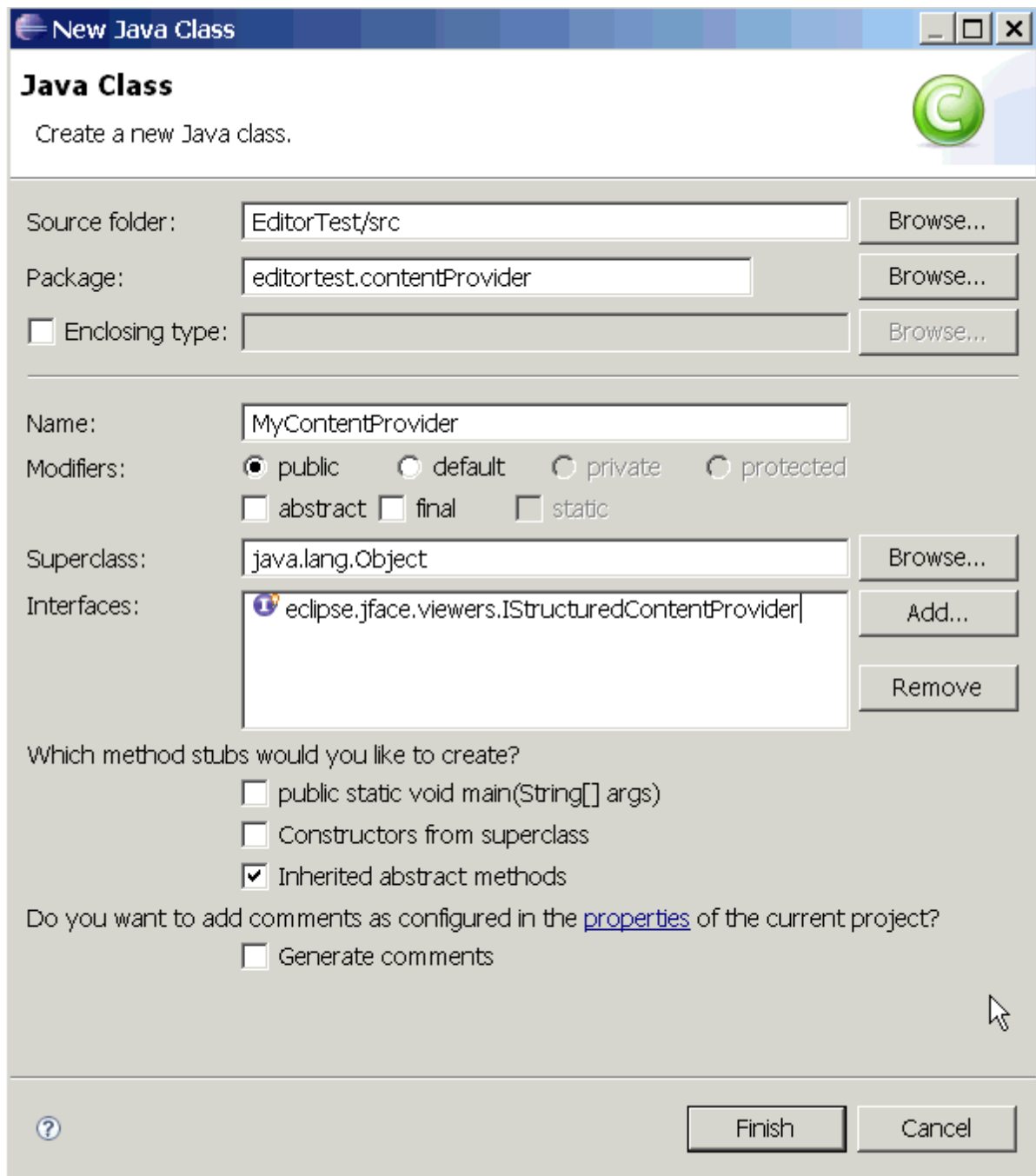
    }

    public Object[] toArray() {
        return persons.toArray();
    }

}
```

创建一个新包 `editortest.contentProvider`

创建一个新类 “`MyContentProvider`”，并接入接口 `IstructuredContentProvider`



调整如下代码:

```
package editortest.contentProvider;
```

```
import mydomain.Content;
```

```
import org.eclipse.jface.viewers.IStructuredContentProvider;
```

```
import org.eclipse.jface.viewers.Viewer;
```



```
public class MyContentProvider implements IStructuredContentProvider {  
    private Content content;  
  
    public MyContentProvider(){  
        content = new Content();  
    }  
    public Object[] getElements(Object inputElement) {  
        // TODO Auto-generated method stub  
        return content.toArray();  
    }  
  
    public void dispose() {  
        // TODO Auto-generated method stub  
    }  
  
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {  
        // TODO Auto-generated method stub  
    }  
}
```

创建一个新类 “MyLabelProvider” ， 接入借口 ILabelProvider。使用如下代码

```
package editortest.contentProvider;  
  
import mydomain.Content.Person;  
import org.eclipse.jface.viewers.ILabelProvider;  
import org.eclipse.jface.viewers.ILabelProviderListener;  
import org.eclipse.swt.graphics.Image;  
import org.eclipse.ui.ISharedImages;
```

```
import org.eclipse.ui.PlatformUI;

public class MyLabelProvider implements ILabelProvider {

    public Image getImage(Object element) {
        return PlatformUI.getWorkbench().getSharedImages().getImage(
            ISharedImages.IMG_OBJ_ELEMENT);
    }

    public String getText(Object element) {
        Person person = (Person) element;
        return (person.getLastName());
    }

    public void addListener(ILabelProviderListener listener) {
        // TODO Auto-generated method stub
    }

    public void dispose() {
        // TODO Auto-generated method stub
    }

    public boolean isLabelProperty(Object element, String property) {
        // TODO Auto-generated method stub
        return false;
    }

    public void removeListener(ILabelProviderListener listener) {
        // TODO Auto-generated method stub
    }
}
```

```
}  
  
}
```

## 7.4 在视口中使用domain模型

改变 VIEW，使用你新的 content providers

```
package editortest;  
  
import org.eclipse.jface.viewers.TableViewer;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.widgets.Composite;  
import org.eclipse.ui.part.ViewPart;  
  
import editortest.contentProvider.MyContentProvider;  
import editortest.contentProvider.MyLabelProvider;  
  
public class View extends ViewPart {  
    public static final String ID = "EditorTest.view";  
  
    private TableViewer viewer;  
  
    /**  
     * This is a callback that will allow us to create the viewer and initialize  
     * it.  
     */  
    public void createPartControl(Composite parent) {  
        viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL  
            | SWT.V_SCROLL);
```

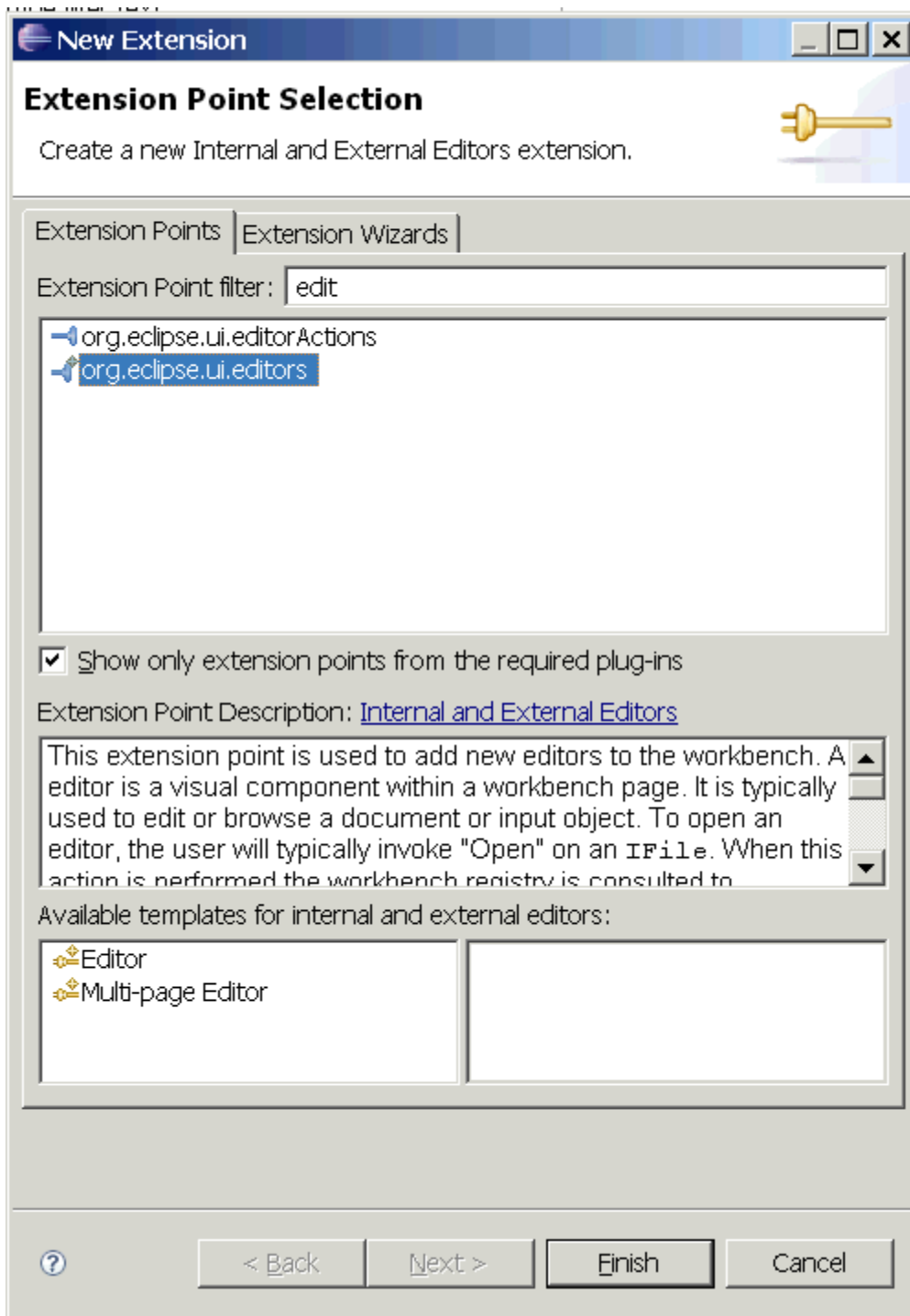
```
viewer.setContentProvider(new MyContentProvider());
viewer.setLabelProvider(new MyLabelProvider());
viewer.setInput(getViewSite());
}

/**
 * Passing the focus request to the viewer's control.
 */
public void setFocus() {
    viewer.getControl().setFocus();
}
}
```

## 7.5 加入编辑器

双击 `plugin.xml` 选择 “Extension” 标签。加入扩展 `org.eclipse.ui.editors`。不要使用模板。使用 ID “`editortest.editors.MyNameEditor`”, 或者任何你想要的名字, class 为 “`editortest.editors.MyNameEditor`”

确认你选择了一个图标, 否则你的编辑器将不会工作



### Extension Element Details

Set the properties of "editor". Required fields are denoted by "\*".

<b>id*:</b>	editortest.editors.MyNameEditor	
<b>name*:</b>	Name	
<b>icon:</b>	icons/alt_window_16.gif	Browse...
<b>extensions:</b>		
<b>class:</b>	editortest.editors.MyNameEdit	Browse...
<b>command:</b>		
<b>launcher:</b>		Browse...
<b>contributorClass:</b>		Browse...
<b>default:</b>	false	▼
<b>filenames:</b>		
<b>symbolicFontName:</b>		
<b>matchingStrategy:</b>		Browse...

点击 class 超连接以创建一个类，这个 ID 非常重要，稍后我们将用它来指示一个类 package editortest.editors;

```
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorSite;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.EditorPart;

public class MyNameEditor extends EditorPart {
    public static final String ID = "editortest.editors.MyNameEditor";
```

```
public MyNameEditor() {  
    // TODO Auto-generated constructor stub  
}  
  
public void doSave(IProgressMonitor monitor) {  
    // TODO Auto-generated method stub  
}  
  
public void doSaveAs() {  
    // TODO Auto-generated method stub  
}  
  
public void init(IEditorSite site, IEditorInput input)  
    throws PartInitException {  
    setSite(site);  
    setInput(input);  
    setPartName(input.getName());  
}  
  
public boolean isDirty() {  
    // TODO Auto-generated method stub  
    return false;  
}  
  
public boolean isSaveAsAllowed() {  
    // TODO Auto-generated method stub  
    return false;  
}
```

```
}

public void createPartControl(Composite parent) {
    GridLayout layout = new GridLayout();
    layout.numColumns = 2;
    parent.setLayout(layout);
    Label label1 = new Label(parent, SWT.BORDER);
    label1.setText("First Name");
    Text text1 = new Text(parent, SWT.BORDER);
    Label label2 = new Label(parent, SWT.BORDER);
    label2.setText("Last Name");
    Text text2 = new Text(parent, SWT.BORDER);

}

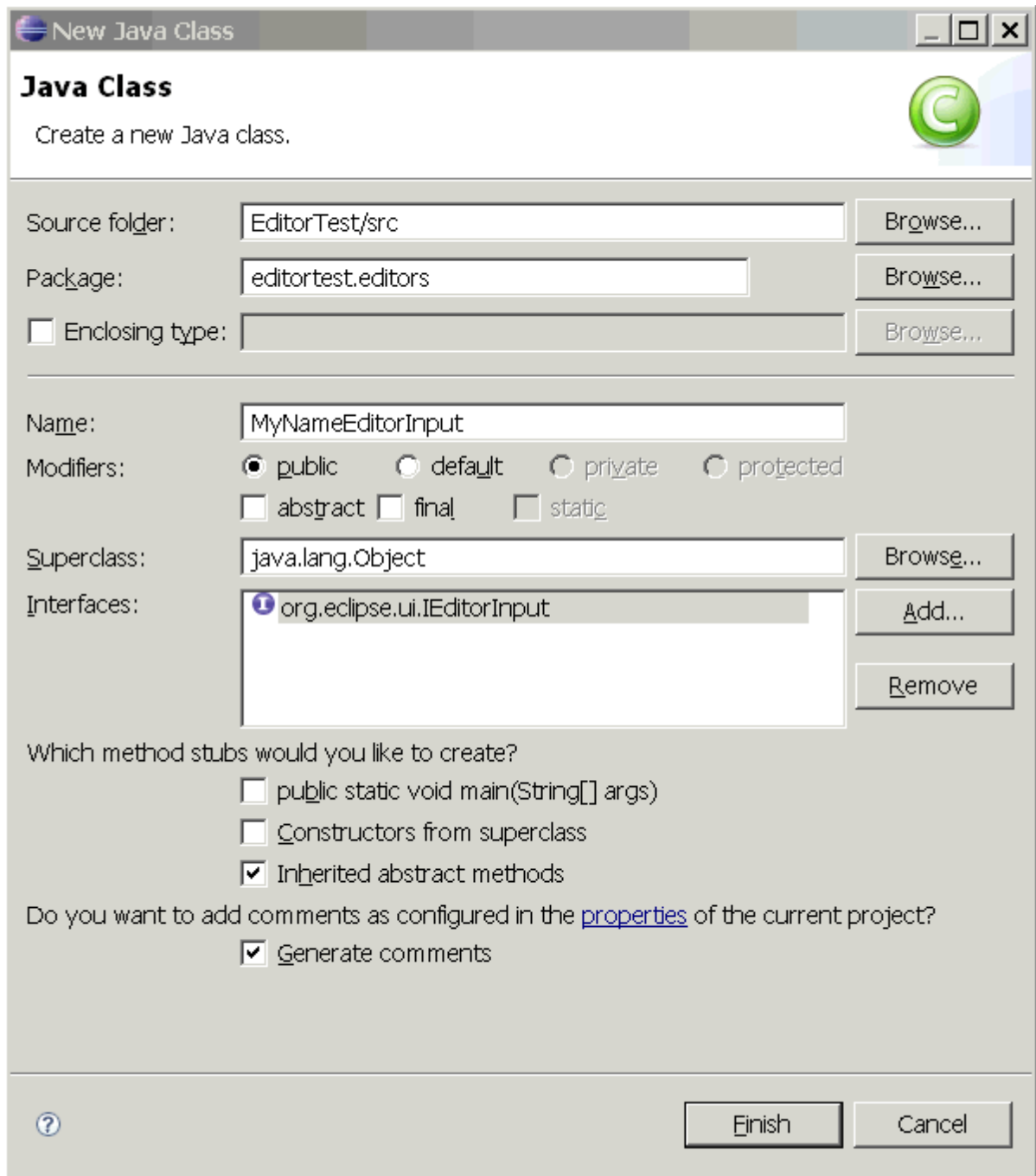
public void setFocus() {
    // TODO Auto-generated method stub

}

}
```

在 `editors` 里创建一个新类 “`MyNameEditorInput`”，并扩展一个 `IEditorInput`





使用如下代码。这将表明接有 `IEditorInput` 借口的类是轻量级模型的代表。It should not contain the model. We cover this later (via view and editor linking).

一些重要的评论。方法 `equals` 和 `hashCode` 总被重写。基于 `equals` 方法，系统能够决定编辑器是已经打开还是没有

```
package editortest.editors;
```

```
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IPersistableElement;

public class MyNameEditorInput implements IEditorInput {
    private final String lastname;

    public MyNameEditorInput(String lastname) {
        this.lastname = lastname;
    }

    public boolean exists() {
        // TODO Auto-generated method stub
        return false;
    }

    public ImageDescriptor getImageDescriptor() {
        // TODO Auto-generated method stub
        return null;
    }

    public String getName() {
        return lastname;
    }

    public IPersistableElement getPersistable() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

```
}

/*
 * (non-Javadoc)
 *
 * @see org.eclipse.ui.IEditorInput#getToolTipText()
 */

public String getToolTipText() {
    return "My Tool Tip";
}

public Object getAdapter(Class adapter) {
    // TODO Auto-generated method stub
    return null;
}

public boolean equals(Object obj) {

    if (super.equals(obj)) {
        return true;
    }

    if (obj instanceof MyNameEditorInput) {
        return lastname.equals(((MyNameEditorInput) obj).getName());
    }

    return false;
}
```

```
    }

    public int hashCode() {
        return lastname.hashCode();
    }

}
```

## 7.6 调用编辑器

向 view 里添加一个 action，每当双击一个新条目时就能打开一个新的编辑器。

```
package editortest;

import mydomain.Content.Person;
import org.eclipse.jface.action.Action;
import org.eclipse.jface.viewers.DoubleClickEvent;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.IViewSite;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.ViewPart;
import editortest.contentProvider.MyContentProvider;
import editortest.contentProvider.MyLabelProvider;
```

```
import editortest.editors.MyNameEditor;
import editortest.editors.MyNameEditorInput;

public class View extends ViewPart {
    public static final String ID = "EditorTest.view";
    private Action doubleClickAction;
    private TableViewer viewer;
    private IViewSite viewSite;

    /**
     * This is a callback that will allow us to create the viewer and initialize
     * it.
     */

    public void createPartControl(Composite parent) {
        viewSite = getViewSite();
        viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL
            | SWT.V_SCROLL);
        viewer.setContentProvider(new MyContentProvider());
        viewer.setLabelProvider(new MyLabelProvider());
        viewer.setInput(getViewSite());
        // New
        hookDoubleClickAction();
        contributeActions();
    }

    /**
     * Passing the focus request to the viewer's control.
     */
}
```

```
public void setFocus() {
    viewer.getControl().setFocus();
}

// New
private void hookDoubleClickAction() {
    viewer.addClickListener(new IDoubleClickListener() {
        public void doubleClick(DoubleClickEvent event) {
            doubleClickAction.run();
        }
    });
}

// New
private void contributeActions() {

    doubleClickAction = new Action() {

        public void run() {
            ISelection selection = viewer.getSelection();
            Object obj = ((IStructuredSelection) selection)
                .getFirstElement();

            Person person = (Person) obj;

            MyNameEditorInput input = new MyNameEditorInput(person
                .getLastName());

            IWorkbenchPage page = PlatformUI.getWorkbench()
```

```
        .getActiveWorkbenchWindow().getActivePage();

        try {
            page.openEditor(input, MyNameEditor.ID);
        } catch (PartInitException e) {
            System.out.println(e.getMessage());
        }
    }

}

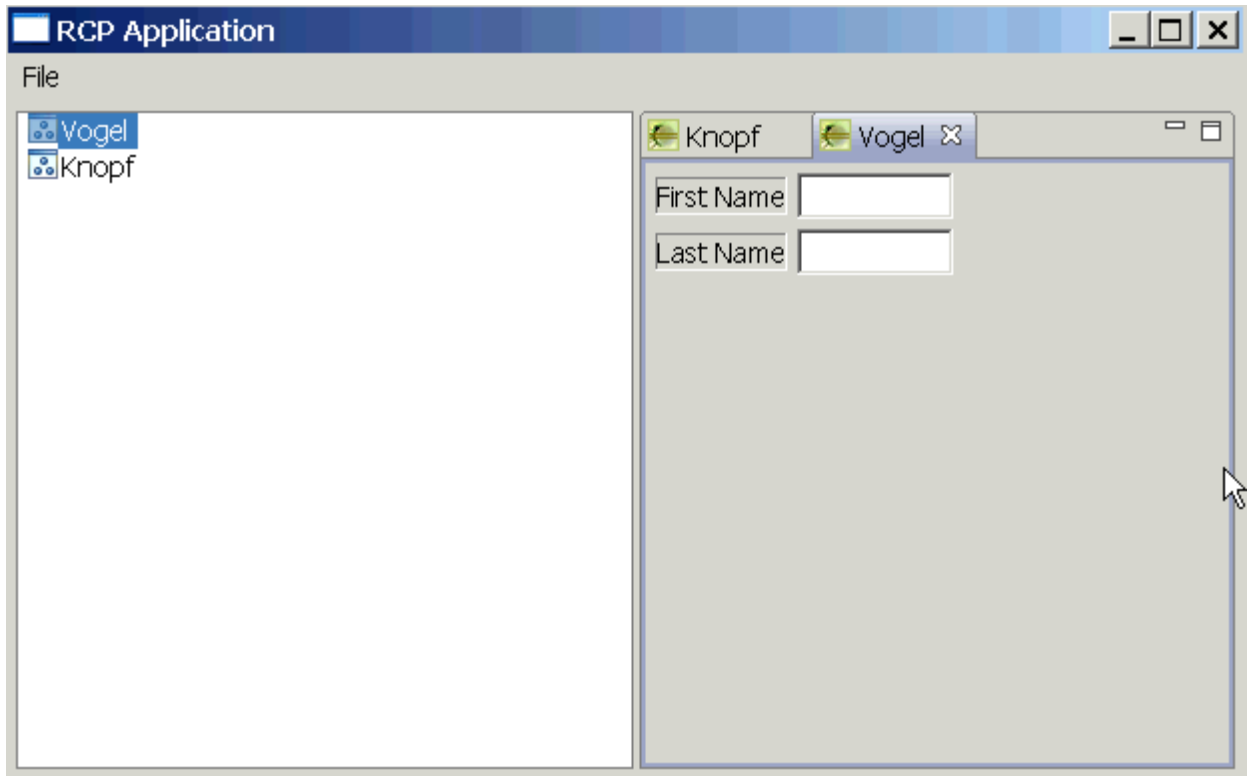
};

}

}
```

在 Perspective.java 里，将方法 createInitialLayout 中的 layout.setEditorAreaVisible(true)参数设为 true;

运行程序，结果如图：



## 7.7 向编辑器提供内容

为了使编辑器得到内容，我们使用 observer pattern。在程序运行期间，这个 pattern 将改变 View.java 和 MyNameEditor.java。在这章的最后将完整的代码提供给大家

//////////



## 第8章对话框

### 8.1 概述

Eclipse 提供了几种预定义的对话框：

```
org.eclipse.swt.widgets.FileDialog  
org.eclipse.swt.widgets.DirectoryDialog  
org.eclipse.swt.widgets.MessageDialog  
org.eclipse.jface.dialogs.ErrorDialog
```

Eclipse 同时也支持用户自定义的对话框。通过继承类 `TitleAreaDialog` 得到新的对话框。

### 8.2 预定义的对话框

#### 8.2.1 概述

接下来将描述如何使用预定义的对话框；

#### 8.2.2 创建工程

使用 “Hello RCP” 模板创建一个新工程 `StandardDialogTest`；

#### 8.2.3 声明 action

添加一个 action “`OpenDialogAction`” 给菜单或者工具栏，为这个 action 创建一个类 `standarddialogtest.CallStandardDialog`

#### 8.2.4 调用对话框

在 `CallStandardDialog` 里使用如下代码调用一个标准对话框

```
package standarddialogtest;  
  
import org.eclipse.jface.action.IAction;  
import org.eclipse.jface.dialogs.MessageDialog;  
import org.eclipse.jface.viewers.ISelection;  
import org.eclipse.swt.graphics.FontData;
```

```
import org.eclipse.swt.graphics.RGB;
import org.eclipse.swt.widgets.ColorDialog;
import org.eclipse.swt.widgets.DirectoryDialog;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.FontDialog;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

public class CallStandardDialog implements IWorkbenchWindowActionDelegate {

    private IWorkbenchWindow window;

    public void dispose() {
        // TODO Auto-generated method stub
    }

    public void init(IWorkbenchWindow window) {

        this.window = window;

    }

    public void run(IAction action) {

        // File standard dialog
        FileDialog fileDialog = new FileDialog(window.getShell());

        // Set the text
        fileDialog.setText("Select File");
```

```
// Set filter on .txt files
fileDialog.setFilterExtensions(new String[] { "*.txt" });

// Put in a readable name for the filter
fileDialog.setFilterNames(new String[] { "Textfiles(*.txt)" });

// Open Dialog and save result of selection
String selected = fileDialog.open();

// Directly standard selection
DirectoryDialog dirDialog = new DirectoryDialog(window.getShell());
dirDialog.setText("Select your home directory");
String selectedDir = dirDialog.open();

// Select Font
FontDialog fontDialog = new FontDialog(window.getShell());
fontDialog.setText("Select your favorite font");
FontData selectedFond = fontDialog.open();

// Select Color
ColorDialog colorDialog = new ColorDialog(window.getShell());
fontDialog.setText("Select your favorite color");
RGB selectedColor = colorDialog.open();

// Now a few messages
MessageDialog.openConfirm(window.getShell(), "Confirm",
    "Please confirm");
MessageDialog.openError(window.getShell(), "Error", "Error occured");
MessageDialog
    .openInformation(window.getShell(), "Info", "Info for you");
MessageDialog.openQuestion(window.getShell(), "Question",
```

```
        "Really, really?");  
        MessageDialog.openWarning(window.getShell(), "Warning", "I warn you");  
  
    }  
  
    public void selectionChanged(IAction action, ISelection selection) {  
        // TODO Auto-generated method stub  
  
    }  
  
}
```

运行程序。如果点击菜单或按钮，将依次弹出所有标准对话框

## 8.3 用户自定义对话框

### 8.3.1 概述

接下来将描述如何自定义对话框，以及如何使用它

### 8.3.2 创建工程

使用“Hello RCP”模板创建

### 8.3.3 声明 action

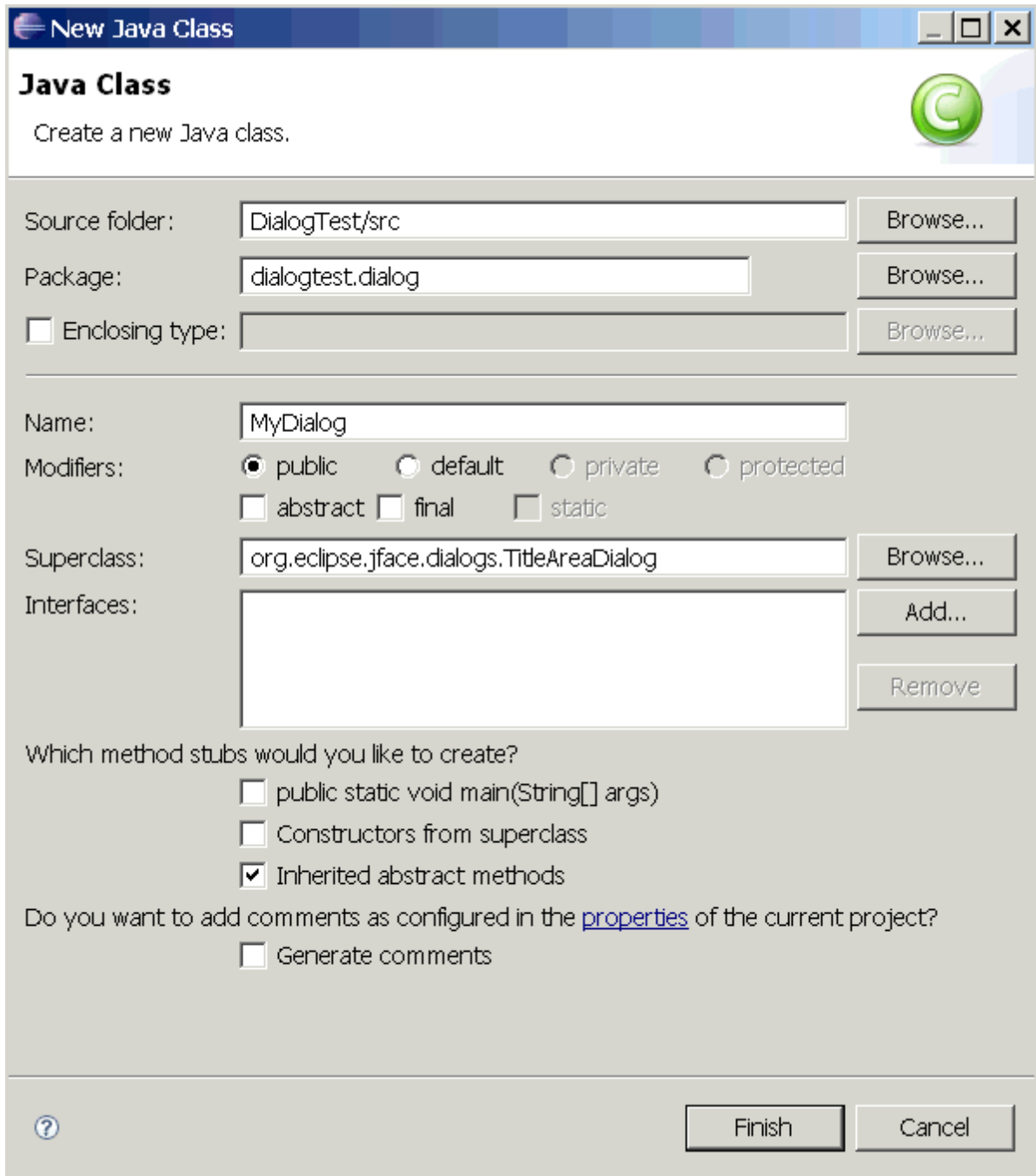
Add one action to the menu and / or the toolbar and create the class  
standarddialogtest.CallStandardDialog for this action.

### 8.3.4 声明 action

向菜单或工具栏里添加一个 action，并且为这个 action 创建类 dialogtest.CallMyDialog。

### 8.3.5 创建对话框

创建一个类，用以描述对话框，取名为 `dialogtest.dialog.MyDialog`。此类继承了 `TitleAreaDialog`。



输入如下代码

```
package dialogtest;
```

```
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

public class CallMyDialog implements IWorkbenchWindowActionDelegate {

    public void dispose() {
        // TODO Auto-generated method stub

    }

    public void init(IWorkbenchWindow window) {
        // TODO Auto-generated method stub

    }

    public void run(IAction action) {
        // TODO Auto-generated method stub

    }

    public void selectionChanged(IAction action, ISelection selection) {
        // TODO Auto-generated method stub

    }

}
```

运行程序，将弹出一个对话框，你必须在 `last name` 里修改数据，否则对话框将弹出一个出错消息框。



## 第9章 向导（wizard）

### 9.1 概述

向导框提供一个灵活的方法收集软件使用者系统的输入并且准确的执行输入。一个向导能够在任务中一步步的指导用户的工作进程

eclipse 可以由 WizardDialog 类实现了一个向导框。向导框控制着进程（导航，进度条，area for error 和消息框）。Wizard 类可以提供向导内容，WizardPages 类提供向导页面的设计。

### 9.2 例子

创建一个新工程，使用“Hello RCP”模板

添加一个 action “CallWizardAction” 给菜单或者工具栏，为这个 action 创建一个 wizardtest.CallWizardDialog 类

创建两个新类 MyPageOne 和 MyPageTwo 继承 org.eclipse.jface.wizard.WizardPage 扩展 org.eclipse.jface.wizard.IWizardPage 接口。

创建一个新类 MyWizard，继承 org.eclipse.jface.wizard.Wizard。

修改数据如下：

```
package wizardtest;

import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.KeyListener;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;

public class MyPageOne extends WizardPage {
```



```
private Text text1;

private Composite container;

public MyPageOne() {
    super("First Page");
    setTitle("First Page2");
    setDescription("This wizard does not really do anything. But this is the first page");
}

public void createControl(Composite parent) {
    container = new Composite(parent, SWT.NULL);
    GridLayout layout = new GridLayout();
    container.setLayout(layout);
    layout.numColumns = 2;
    Label label1 = new Label(container, SWT.NULL);
    label1.setText("Put here a value");

    text1 = new Text(container, SWT.BORDER | SWT.SINGLE);
    text1.setText("");
    text1.addListener(new KeyListener() {

        public void keyPressed(KeyEvent e) {
            // TODO Auto-generated method stub
        }

        public void keyReleased(KeyEvent e) {
            if (!text1.getText().equals(new String())){
                setPageComplete(true);
            }
        }
    });
}
```

```
        }

    }

});

GridData gd = new GridData(GridData.FILL_HORIZONTAL);
text1.setLayoutData(gd);
// Required to avoid an error in the system
setControl(container);
setPageComplete(false);

}

public String getText1() {
    return text1.getText();
}

// You need to overwrite this method otherwise you receive a
// AssertionError
// This method should always return the top widget of the application

public Control getControl() {
    return container;
}

}
```

```
package wizardtest;

import org.eclipse.jface.wizard.IWizardPage;
import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.KeyListener;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;

public class MyPageTwo extends WizardPage implements IWizardPage {
    private Text text1;
    private Composite container;
    public MyPageTwo(){
        super("Second Page");
        setTitle("Second Page");
        setDescription("Now this is the second page");
        setControl(text1);
    }

    public void createControl(Composite parent) {
        container = new Composite(parent, SWT.NULL);
        GridLayout layout = new GridLayout();
        container.setLayout(layout);
        layout.numColumns = 2;
        Label label1 = new Label(container, SWT.NULL);
```

```
label1.setText("Put here a value");

text1 = new Text(container, SWT.BORDER | SWT.SINGLE);
text1.setText("");
text1.addKeyListener(new KeyListener() {

    public void keyPressed(KeyEvent e) {
        // TODO Auto-generated method stub

    }

    public void keyReleased(KeyEvent e) {

        if (!text1.getText().equals(new String())){
            setPageComplete(true);

        }

    }

});

GridData gd = new GridData(GridData.FILL_HORIZONTAL);
text1.setLayoutData(gd);

// Required to avoid an error in the system
setControl(container);
setPageComplete(false);

}
```

```
public String getText1() {
    return text1.getText();

}

// You need to overwrite this method otherwise you receive a AssertionError
// This method should always return the top widget of the application

public Control getControl() {
    return container;

}

}

package wizardtest;

import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.wizard.Wizard;

public class MyWizard extends Wizard {

    private MyPageOne one;
    private MyPageTwo two;

    public MyWizard() {
        super();
        setNeedsProgressMonitor(true);

    }
```

```
public void addPages() {
    one = new MyPageOne();
    two = new MyPageTwo();
    addPage(one);
    addPage(two);
}

public boolean performFinish() {

    // just put the result to the console, imagine here much more
    // intelligent stuff.
    System.out.println(one.getText1());
    System.out.println(two.getText1());

    return true;
}
}

package wizardtest;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.wizard.WizardDialog;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
```

```
public class CallWizardDialog implements IWorkbenchWindow ActionDelegate {  
    private IWorkbenchWindow window;  
  
    public void dispose() {  
  
        // TODO Auto-generated method stub  
  
    }  
  
    public void init(IWorkbenchWindow window) {  
        this.window = window;  
  
    }  
  
    public void run(IAction action) {  
        MyWizard wizard = new MyWizard();  
        WizardDialog dialog = new WizardDialog(window.getShell(), wizard);  
        dialog.open();  
  
    }  
  
    public void selectionChanged(IAction action, ISelection selection) {  
  
        // TODO Auto-generated method stub  
  
    }  
  
}
```

## 第10章 首选项

### 10.1 首选项

eclipse 首选项和 `java.util.prefs.Preferences` 非常相似。首选项对一些类似查找、存储的附加特征提供支持

首选项是 `key / values pairs`，`key` 是一个 `arbitrary` 名字的首选。`Value` 可以是 `boolean`, `string`, `int` 或者其他简单类型。首选项被接受且保存通过 `get` 和 `put` 方法，在 `preferences` 还没有被设定的时候，`get` 方法可以支持一个默认值

eclipse 首选项通过 eclipse 的框架保存和恢复数据，因此更容易被重用

运行时定义了三个 so-called 范围。这个范围定义了如何首选数据，如何改变

**Instance scope:** 如果用户运行同一个程序两次，两次之间的的设定可能不一样

**Configuration scope:** 如果用户运行同一个程序两次，两次之间的设定是一样的

**Default scope:**默认值不可被更改，由插件里的设置文件和产品定义。

你能使用以下代码存储优先项

```
Preferences preferences = new  
ConfigurationScope().getNode(Application.PLUGIN_ID)
```

### 10.2 使用首选项

使用“Hello RCP”模板创建一个新工程

修改应用程序的 ID 为“PreferenceRCP”

为菜单或工具栏添加两个 action “PreferenceAction” “DeleteAction”，创建与之同步的类 `preferencetest.PreferenceAction` 和 `preferencetest.DeleteAction`

接下来为两个 action 编码。我们用以存储 `preferences` 和删除他们。设定之后重起软件，然后删除他们

```
package preferencetest;  
  
import org.eclipse.core.runtime.preferences.ConfigurationScope;  
import org.eclipse.jface.action.IAction;  
import org.eclipse.jface.dialogs.MessageDialog;
```



```
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;
import org.osgi.service.prefs.Preferences;

public class PreferenceAction implements IWorkbenchWindowActionDelegate {

    private IWorkbenchWindow window;

    public void dispose() {
        // TODO Auto-generated method stub
    }

    public void init(IWorkbenchWindow window) {
        this.window = window;
    }

    public void run(IAction action) {

        // This would be using instance scope
        // Preferences preferences = new InstanceScope()
        // .getNode(Application.PLUGIN_ID);
        // This is using configuration scope

        Preferences preferences = new ConfigurationScope()
            .getNode(Application.PLUGIN_ID);

        // This would be using default n scope
```

```
// Preferences preferences = new DefaultScope()
// .getNode(Application.PLUGIN_ID);

Preferences sub1 = preferences.node("note1");
Preferences sub2 = preferences.node("node2");

sub1.put("h1", "Hello");
sub1.put("h2", "Hello again");
sub2.put("h1", "Moin");

MessageDialog.openInformation(window.getShell(), "Info", sub1.get("h1",
    "de fault"));

MessageDialog.openInformation(window.getShell(), "Info", sub1.get("h2",
    "de fault"));

MessageDialog.openInformation(window.getShell(), "Info", sub2.get("h1",
    "de fault"));

// Forces the application to save the preferences

try {
    preferences.flush();

} catch (BackingStoreException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

}
```

```
public void selectionChanged(IAction action, ISelection selection) {  
    // TODO Auto-generated method stub  
  
}  
  
}  
  
package preferencetest;  
  
import org.eclipse.core.runtime.preferences.ConfigurationScope;  
import org.eclipse.jface.action.IAction;  
import org.eclipse.jface.dialogs.MessageDialog;  
import org.eclipse.jface.viewers.ISelection;  
import org.eclipse.ui.IWorkbenchWindow;  
import org.eclipse.ui.IWorkbenchWindowActionDelegate;  
import org.osgi.service.prefs.BackingStoreException;  
import org.osgi.service.prefs.Preferences;  
  
public class DeleteAction implements IWorkbenchWindowActionDelegate {  
  
    private IWorkbenchWindow window;  
  
    public void dispose() {  
        // TODO Auto-generated method stub  
  
    }  
  
    public void init(IWorkbenchWindow window) {  
        this.window = window;  
    }  
}
```

```
}
```

```
public void run(IAction action) {
```

```
    Preferences preferences = new ConfigurationScope()
```

```
        .getNode(Application.PLUGIN_ID);
```

```
    Preferences sub1 = preferences.node("note1");
```

```
    Preferences sub2 = preferences.node("node2");
```

```
    // Show the values before deleting them
```

```
    MessageDialog.openInformation(window.getShell(), "Info", sub1.get("h1",  
        "default"));
```

```
    MessageDialog.openInformation(window.getShell(), "Info", sub1.get("h2",  
        "default"));
```

```
    MessageDialog.openInformation(window.getShell(), "Info", sub2.get("h1",  
        "default"));
```

```
    // Delete the existing settings
```

```
    try {
```

```
        sub1.clear();
```

```
        sub2.clear();
```

```
    } catch (BackingStoreException e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
    // Now show the values
```

```
        MessageDialog.openInformation(window.getShell(), "Info", sub1.get("h1",
            "de fault"));

        MessageDialog.openInformation(window.getShell(), "Info", sub1.get("h2",
            "de fault"));

        MessageDialog.openInformation(window.getShell(), "Info", sub2.get("h1",
            "de fault"));

        // Forces the application to save the preferences
        try {
            preferences.flush();

        } catch (BackingStoreException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    public void selectionChanged(IAction action, ISelection selection) {
        // TODO Auto-generated method stub

    }

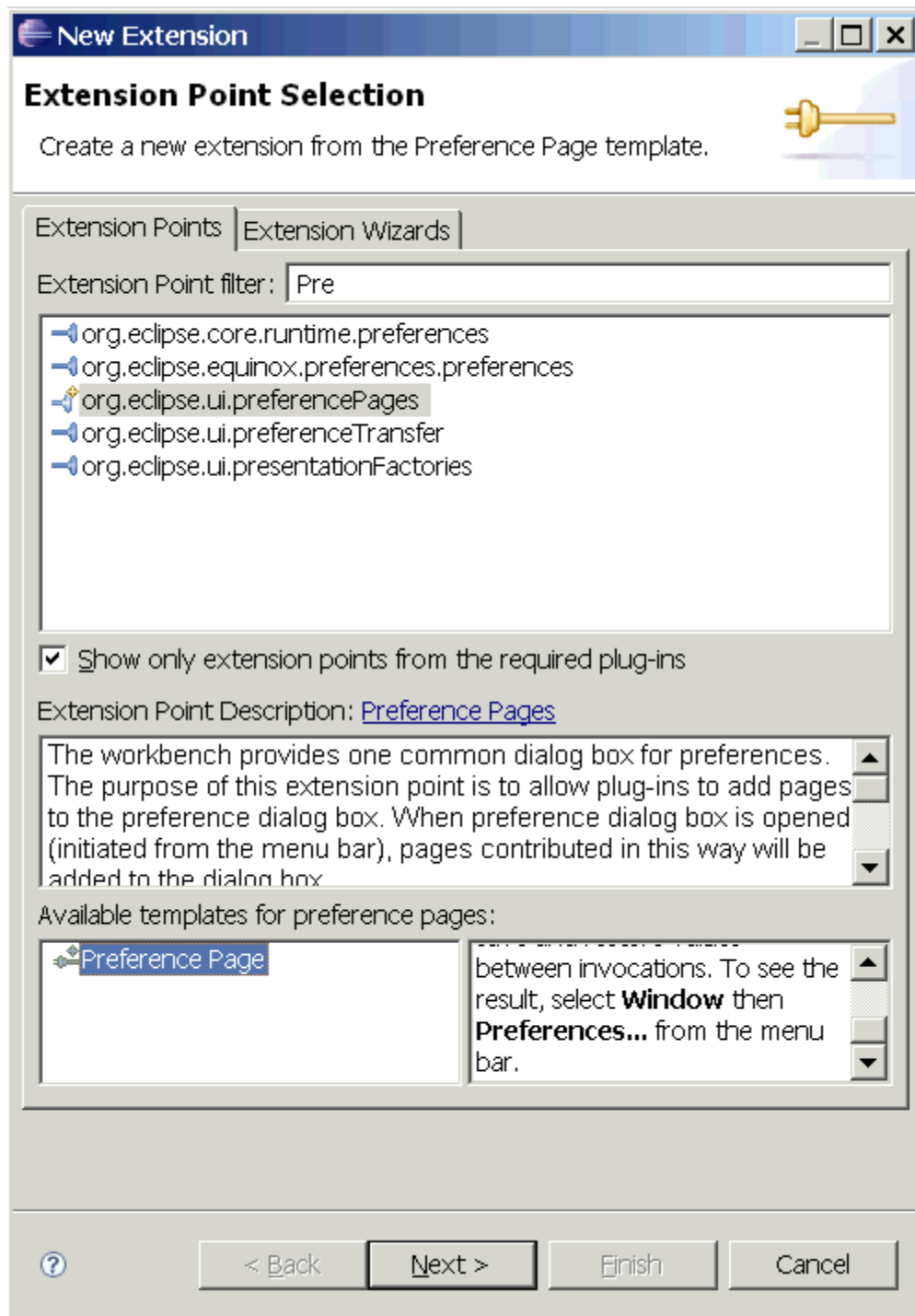
}
```

## 10.3 首选项页

首选项页允许修改/浏览用户或系统的设定。他们需要增加扩展点 `org.eclipse.ui.preferencePages`。

使用“Hello RCP”创建一个新的工程。

在上面的例子中的 `plugin.xml`，添加扩展点 `org.eclipse.ui.preferencePages`。选择模板 `PreferencePage`





创建一个新包

向你的菜单添加首选项

```
package preferencetest;
```

```
import org.eclipse.jface.action.IMenuManager;
```

```
import org.eclipse.jface.action.MenuManager;
```

```
import org.eclipse.ui.IWorkbenchActionConstants;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;

public class ApplicationActionBarAdvisor extends ActionBarAdvisor {
    private IWorkbenchAction preferenceAction;

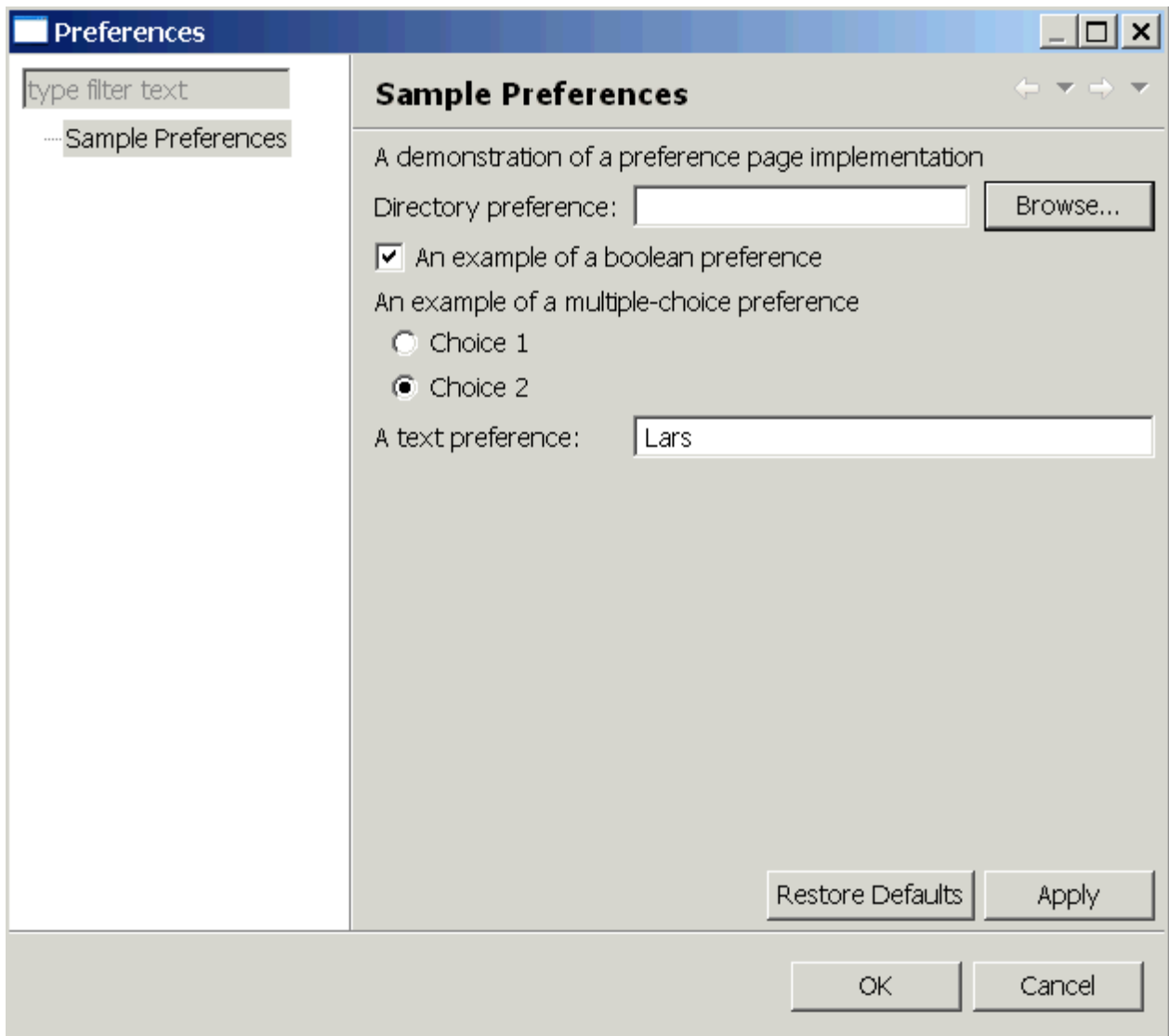
    public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
        super(configurer);
    }

    protected void makeActions(IWorkbenchWindow window) {
        preferenceAction = ActionFactory.PREFERENCES.create(window);
        register(preferenceAction);
    }

    protected void fillMenuBar(IMenuManager menuBar) {
        MenuManager helpMenu = new MenuManager("&Help",
            IWorkbenchActionConstants.M_HELP);
        menuBar.add(helpMenu);
        helpMenu.add(preferenceAction);
    }
}
```

运行程序，你可以通过菜单选择你的首选项。





向菜单和工具栏添加一个 action “ShowPrefAction”，创建同步的类 `preferencepagetest.ShowAction`。

```
package preferencepagetest;
```

```
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import preferencepagetest.preferences.PreferenceConstants;
```

```
public class ShowAction implements IWorkbenchWindowActionDelegate {
```

```
private IWorkbenchWindow window;

public void dispose() {

    // TODO Auto-generated method stub

}

public void init(IWorkbenchWindow window) {

    this.window = window;

}

public void run(IAction action) {

    String myPrefString = Activator.getDefault().getPreferenceStore()
        .getString(PreferenceConstants.P_STRING);
    MessageDialog.openInformation(window.getShell(), "Info", myPrefString);
    Boolean myPrefBoolean = Activator.getDefault().getPreferenceStore()
        .getBoolean(PreferenceConstants.P_BOOLEAN);
    MessageDialog.openInformation(window.getShell(), "Info", myPrefBoolean
        .toString());

    // I assume you get the rest by yourself

}

public void selectionChanged(IAction action, ISelection selection) {

    // TODO Auto-generated method stub

}
```

}

## 第11章 添加状态条

### 11.1 安装状态条

使用“Hello RCP”创建一个新工程“Statusline”

进入 `ApplicationWorkbenchWindowAdvisor`，改变 `preWindowOpen()` 方法。与此有关的代码是“`configurer.setShowStatusLine(true);`”

```
package statusline;

import org.eclipse.swt.graphics.Point;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;

public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    public ApplicationWorkbenchWindowAdvisor(
        IWorkbenchWindowConfigurer configurer) {
        super(configurer);
    }

    public ActionBarAdvisor createActionBarAdvisor(
        IActionBarConfigurer configurer) {
        return new ApplicationActionBarAdvisor(configurer);
    }

    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
```

```

        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowStatusLine(true);
        configurer.setShowCoolBar(false);
        configurer.setShowMenuBar(false);
        configurer.setTitle("Status Line Example");

    }

}

```

运行程序，可以看到一个状态条。这个状态条没有内容。

## 11.2 共享状态条

共享消息区可以使应用程序的不同部分使用，均向此区写入消息。

接下来将解释如何向程序添加状态条。以及程序中的 view 或者编辑器向其中添加内容。

提示

对于整个 RCP 程序，有进入共享状态条消息的入口。共享状态条的消息很可能被重写。

现在，让我们为状态条填写内容。可以通过 `ApplicationWorkbenchWindowAdvisor` 的 `postWindowOpen()` 方法做这个例子。你有两个方法添加这个方法

~可以向类中粘贴方法

~或者使用菜单“Source”实现或重写方法。然后选择 `postWindowOpen()` 创建方法，并改写成你所需要的

```

package statusline;

import org.eclipse.jface.action.IStatusLineManager;
import org.eclipse.swt.graphics.Point;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;

```

```
public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    public ApplicationWorkbenchWindowAdvisor(
        IWorkbenchWindowConfigurer configurer) {
        super(configurer);
    }

    public ActionBarAdvisor createActionBarAdvisor(
        IActionBarConfigurer configurer) {
        return new ApplicationActionBarAdvisor(configurer);
    }

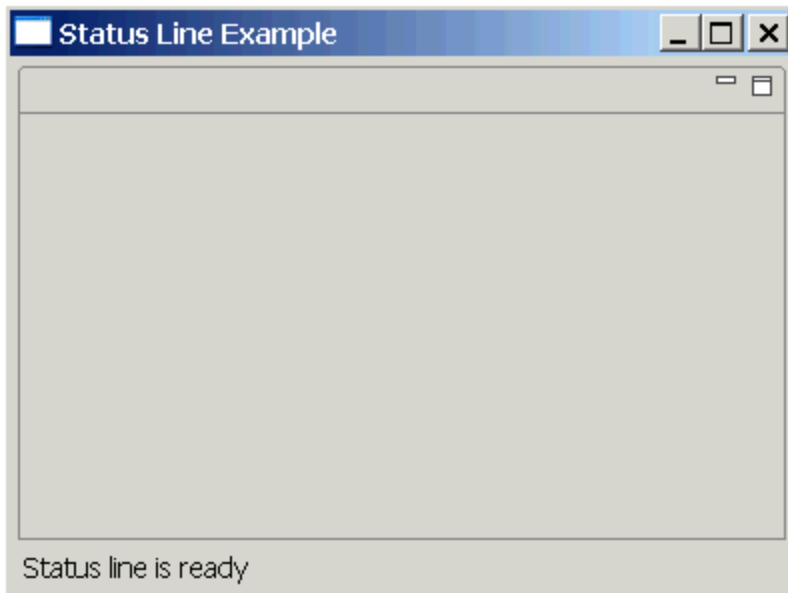
    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowCoolBar(false);
        configurer.setShowStatusLine(true);
        configurer.setTitle("Hello RCP");
    }

    // This is the new method

    public void postWindowOpen() {

        IStatusLineManager statusline = getWindowConfigurer()
            .getActionBarConfigurer().getStatusLineManager();
        statusline.setMessage(null, "Status line is ready");
    }
}
```

```
}  
  
}
```



向程序添加一个新的 VIEW，ID 为 “StatusLine.View1”。一旦你有 VIEW 后，你就可以进入状态条。如下

```
IActionBars bars = getViewSite().getActionBars();
```

```
bars.getStatusLineManager().setMessage("Hello");
```

例子中，我们将创建一个按钮，用来设置状态条。

```
package statusline;
```

```
import org.eclipse.swt.SWT;
```

```
import org.eclipse.swt.events.SelectionEvent;
```

```
import org.eclipse.swt.events.SelectionListener;
```

```
import org.eclipse.swt.widgets.Button;
```

```
import org.eclipse.swt.widgets.Composite;
```

```
import org.eclipse.ui.IActionBars;
```

```
import org.eclipse.ui.part.ViewPart;
```

```
public class View1 extends ViewPart {

    boolean pressed = false;

    public View1() {

    }

    public void createPartControl(Composite parent) {
        Button setStatusLine = new Button(parent, SWT.PUSH);
        setStatusLine.setText("Set Statusline ");
        setStatusLine.addSelectionListener(new SelectionListener() {

            public void widgetSelected(SelectionEvent e) {

                IActionBars bars = getViewSite().getActionBars();

                if (pressed) {
                    bars.getStatusLineManager().setMessage(
                        "I would like to say hello to you.");
                } else {
                    bars.getStatusLineManager().setMessage(
                        "Thank you for using me");
                }

                pressed = !pressed;
            }

            public void widgetDefaultSelected(SelectionEvent e) {
```



```
        }

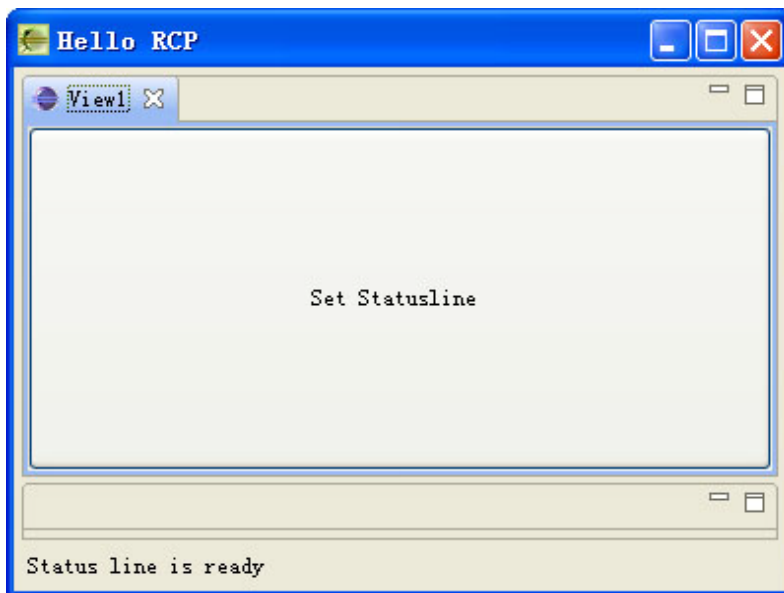
        });

    }

    public void setFocus() {

    }

}
```



提示:

通过如下代码你可以是一个编辑器进入状态条

```
IEditorPart.getEditorSite().getActionBarContributor();
```



## 第12章 透视图

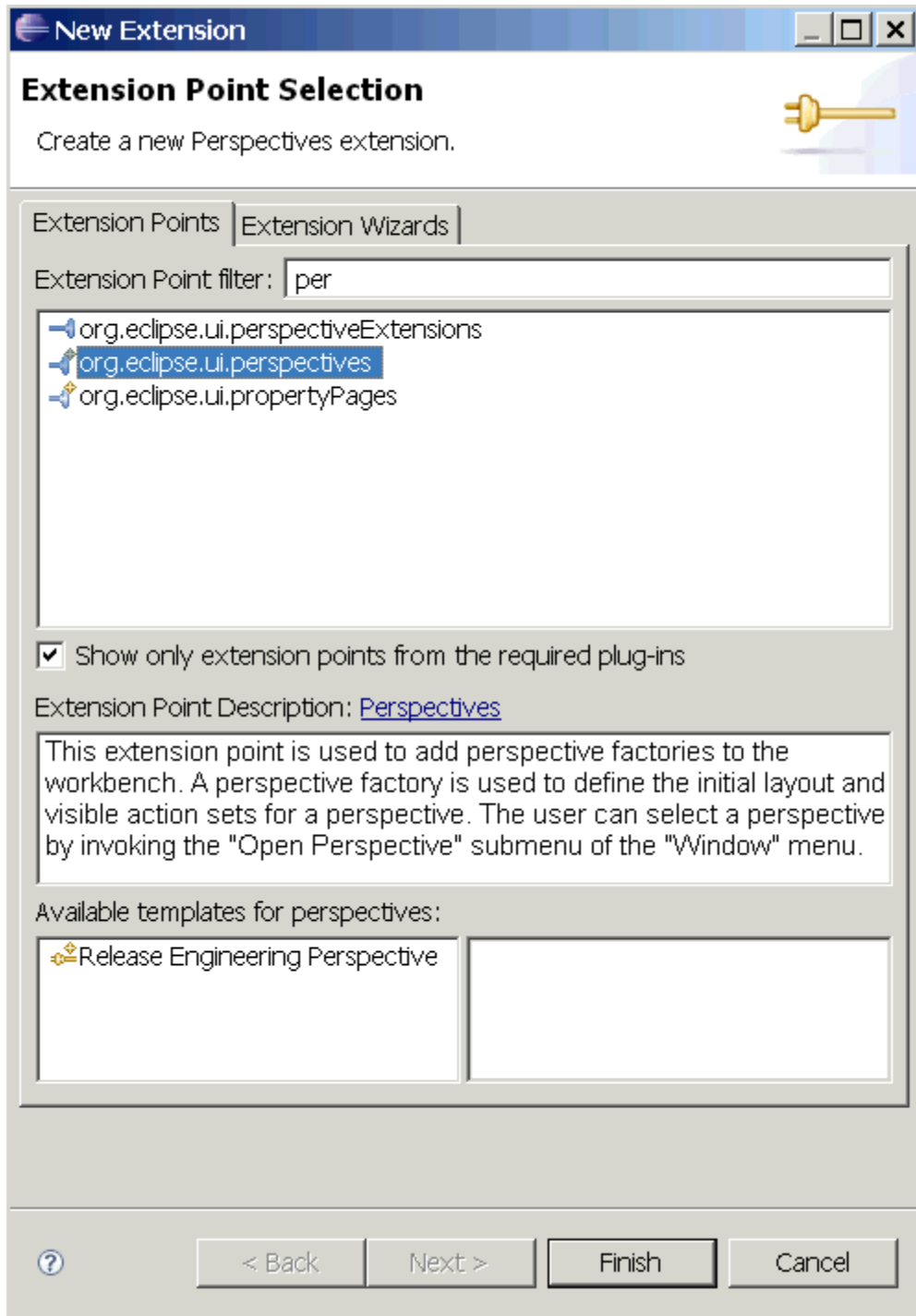
透视图将相关的 UI 元素集合并组织起来，更适合特殊的任务及工作流程。

Eclipse RCP 允许你添加简单的透视图到程序中。接下来将介绍如何做到

### 12.1 向你的程序中添加透视图

使用 “RCP application with a view” 模板创建一个新的 RCP 工程，命名为 “PerspectiveTest”，运行，看结果

在 `plugin.xml` 里添加一个新的透视图扩展点



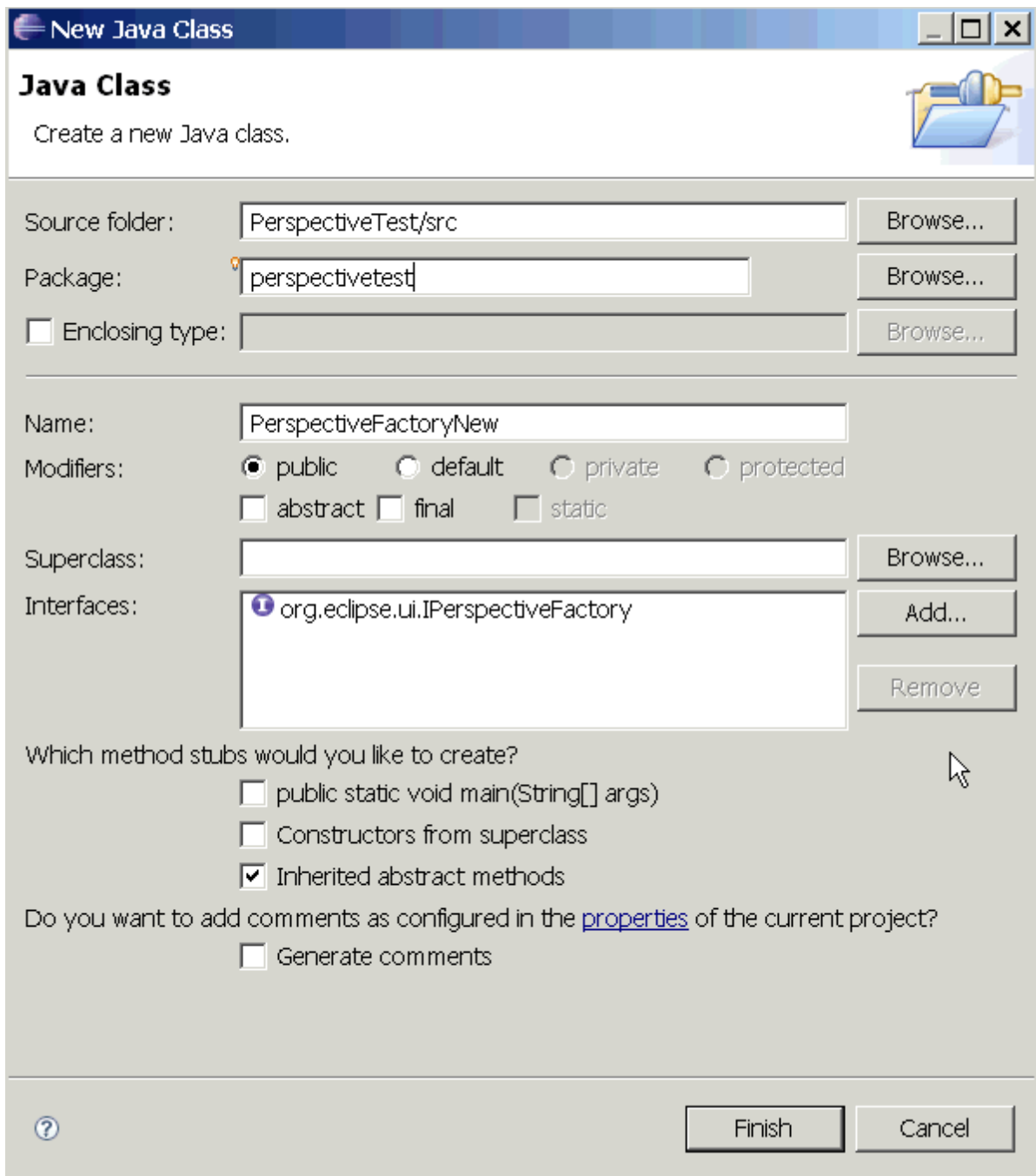
给它起个好的 ID 和名字。名字将会显示在透视图下面。修改类名。添加一个图标，使用户可以选择透视图。点击“class\*”

连接，创建一个类。确认这个包名正确

**Extension Element Details**  
Set the properties of "perspective". Required fields are denoted by "\*".

id*:	<input type="text" value="PerspectiveTest.NewPerspective"/>		
<a href="#">name*</a> :	<input type="text" value="New Perspective"/>		
<a href="#">class*</a> :	<input type="text" value="perspectivetest.PerspectiveFactory"/>	<input type="button" value="Browse..."/>	
<a href="#">icon:</a>	<input type="text"/>	<input type="button" value="Browse..."/>	
fixed:	<input type="text"/> ▼		

你新类里的 `createInitialLayout()` 方法负责创建一个新的透视图。因此在这个例子里，我们使用相同的 **VIEW**，在先创建的透视图里创建一个已定义的独立运行的 **VIEW**。它是一个正常的视口



```
package perspectivetest;
```

```
import org.eclipse.ui.IPageLayout;
```

```
import org.eclipse.ui.IPerspectiveFactory;
```

```
public class PerspectiveFactory1 implements IPerspectiveFactory {
```

```
public void createInitialLayout(IPageLayout layout) {  
    // TODO Auto-generated method stub  
    String editorArea = layout.getEditorArea();  
    layout.setEditorAreaVisible(true);  
    layout.setFixed(false);  
    layout.addView(View.ID, IPageLayout.LEFT, 0.33f, editorArea);  
  
}  
  
}
```

现在透视图已经定义了，但是还不能在应用程序里获得。

## 12.2使透视图可选。

### 12.2.1 使透视图可由一个 toolbar 可选

你可以激活开关 You can activate the switch between perspectives the ApplicationWorkbenchWindowAdvisor in method preWindowOpen() with configurer.setShowPerspectiveBar(true);

```
package perspectivetest;  
  
import org.eclipse.jface.preference.IPreferenceStore;  
import org.eclipse.swt.graphics.Point;  
import org.eclipse.ui.IWorkbenchPreferenceConstants;  
import org.eclipse.ui.PlatformUI;  
import org.eclipse.ui.application.ActionBarAdvisor;  
import org.eclipse.ui.application.IActionBarConfigurer;  
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;  
import org.eclipse.ui.application.WorkbenchWindowAdvisor;  
  
public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {
```

```
public ApplicationWorkbenchWindowAdvisor(
    IWorkbenchWindowConfigurer configurer) {
    super(configurer);
}

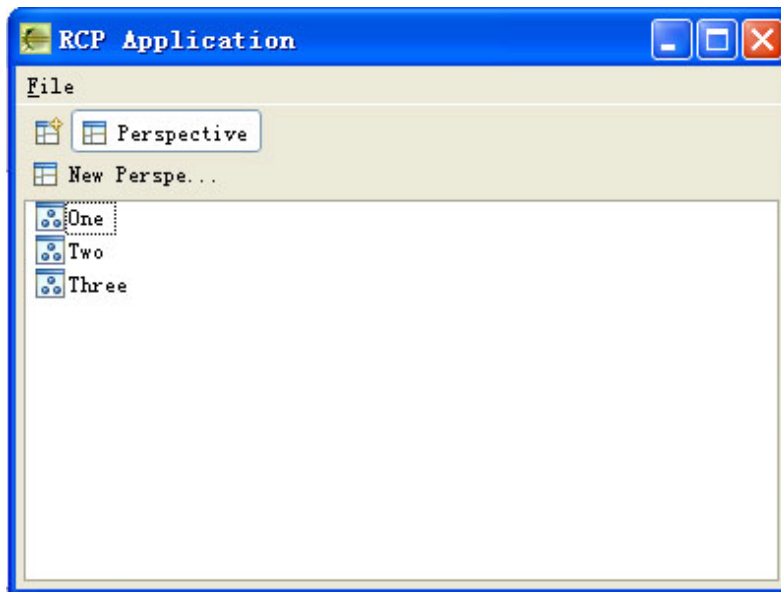
public ActionBarAdvisor createActionBarAdvisor(
    IActionBarConfigurer configurer) {
    return new ApplicationActionBarAdvisor(configurer);
}

public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setInitialSize(new Point(400, 300));
    configurer.setShowCoolBar(false);
    configurer.setShowStatusLine(false);
    configurer.setTitle("RCP Application");
    configurer.setShowPerspectiveBar(true);

    // Set the preference toolbar to the left place
    // If other menus exists then this will be on the left of them
    IPreferenceStore apiStore = PlatformUI.getPreferenceStore();
    apiStore.setValue(IWorkbenchPreferenceConstants.DOCK_PERSPECTIVE_BAR,
        "TOP_LEFT");
}
}
```

你现在将可以选择你的透视图了





### 12.2.2 使透视图可通过菜单选择

你可以通过如下代码激活菜单里的开关

```
package perspectivetest;
```

```
import org.eclipse.jface.action.IContributionItem;
import org.eclipse.jface.action.ICoolBarManager;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.IToolBarManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.ToolBarContributionItem;
import org.eclipse.jface.action.ToolBarManager;
import org.eclipse.swt.SWT;
import org.eclipse.ui.IWorkbenchActionConstants;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.ui.actions.ContributionItemFactory;
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
```

```
/**
 * An action bar advisor is responsible for creating, adding, and disposing of
 * the actions added to a workbench window. Each window will be populated with
 * new actions.
 */

public class ApplicationActionBarAdvisor extends ActionBarAdvisor {

    // Actions - important to allocate these only in makeActions, and then use
    // them
    // in the fill methods. This ensures that the actions aren't recreated
    // when fillActionBars is called with FILL_PROXY.

    private IWorkbenchAction exitAction;
    private IContributionItem perspectivesMenu;

    public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
        super(configurer);
    }

    protected void makeActions(final IWorkbenchWindow window) {
        // Creates the actions and registers them.
        // Registering is needed to ensure that key bindings work.
        // The corresponding commands keybindings are defined in the plugin.xml
        // file.
        // Registering also provides automatic disposal of the actions when
        // the window is closed.

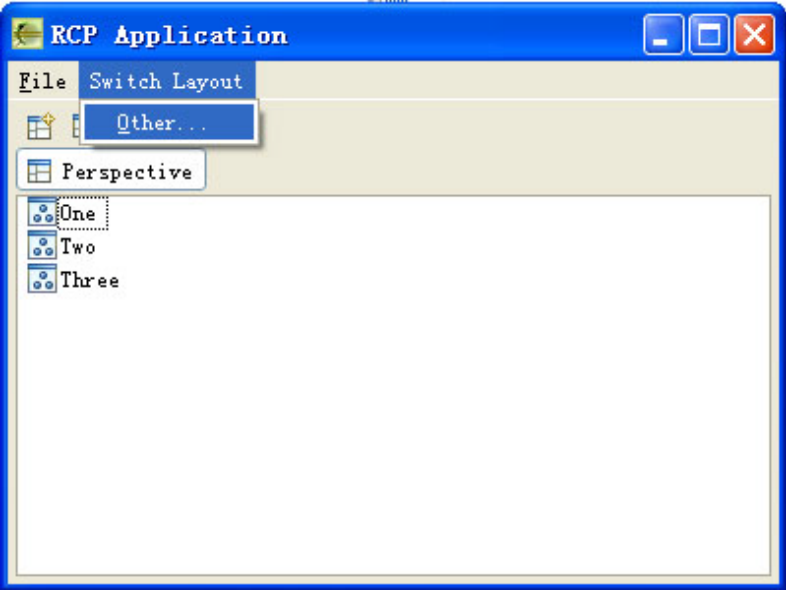
        exitAction = ActionFactory.QUIT.create(window);
    }
}
```

```
        register(exitAction);
        perspectivesMenu = ContributionItemFactory.PERSPECTIVES_SHORTLIST
            .create(window);
    }

    protected void fillMenuBar(IMenuManager menuBar) {
        MenuManager fileMenu = new MenuManager("&File",
            IWorkbenchActionConstants.M_FILE);
        menuBar.add(fileMenu);
        fileMenu.add(exitAction);
        MenuManager layoutMenu = new MenuManager("Switch Layout", "layout");
        layoutMenu.add(perspectivesMenu);
        menuBar.add(layoutMenu);
    }

    protected void fillCoolBar(ICoolBarManager coolBar) {
        IToolBarManager toolbar = new ToolBarManager(SWT.FLAT | SWT.RIGHT);
        toolbar.add(exitAction);
        coolBar.add(new ToolBarContributionItem(toolbar, "main"));
    }
}
```

运行结果如图



## 第13章 进度报告

如果你要进行一项需要很长时间运行的操作，你需要想用户提供关于运行工作的信息报告。一个好的方法就是使用一个进度指示器。或者进度对话框。接下来将展示这两个方法。

使用“Hello RCP”模板创建一个新的工程“ProgressTest”。并且添加一个 action “progresstest.progressdialog”给 coolbar。

为 action 创建如下的类“progresstest.ProgressDialog Action”

```
package progresstest;

import java.lang.reflect.InvocationTargetException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.ProgressMonitorDialog;
import org.eclipse.jface.operation.IRunnableWithProgress;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

public class ProgressDialogAction implements IWorkbenchWindowActionDelegate {
    private IWorkbenchWindow window;

    public void dispose() {
        // TODO Auto-generated method stub
    }

    public void init(IWorkbenchWindow window) {
        // TODO Auto-generated method stub
        this.window = window;
    }
}
```

```
}
```

```
public void run(IAction action) {  
    // TODO Auto-generated method stub  
    ProgressMonitorDialog dialog = new ProgressMonitorDialog(window.getShell());  
    try {  
        dialog.run(true, true, new IRunnableWithProgress() {  
            public void run(IPressMonitor monitor) {  
                monitor  
                    .beginTask("Doing something timeconsuming here",100);  
  
                for (int i = 0; i < 10; i++) {  
                    if (monitor.isCanceled())  
                        return;  
                    monitor.subTask("I'm doing something here " + i);  
                    sleep(1000);  
                    monitor.worked(i);  
                }  
                monitor.done();  
            }  
        });  
    } catch (InvocationTargetException e) {  
        e.printStackTrace();  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}  
  
public void selectionChanged(IAction action, ISelection selection) {  
    // TODO Auto-generated method stub  
}
```

```
private void sleep(Integer waitTime) {  
    try {  
        Thread.sleep(waitTime);  
    } catch (Throwable t) {  
        System.out.println("Wait time interrupted");  
    }  
}  
  
}
```

向工程里添加一个 action `progresstest.progressindicator`，并为他创建一个 `progresstest.ProgressIndicatorAction` 类。

```
package progresstest;  
  
import org.eclipse.core.runtime.IProgressMonitor;  
import org.eclipse.core.runtime.IStatus;  
import org.eclipse.core.runtime.Status;  
import org.eclipse.core.runtime.jobs.Job;  
import org.eclipse.jface.action.IAction;  
import org.eclipse.jface.viewers.ISelection;  
import org.eclipse.ui.IWorkbenchWindow;  
import org.eclipse.ui.IWorkbenchWindowActionDelegate;  
  
public class ProgressIndicatorAction implements IWorkbenchWindowActionDelegate {  
    public void dispose() {  
        // TODO Auto-generated method stub  
    }  
  
    public void init(IWorkbenchWindow window) {  
        // TODO Auto-generated method stub  
    }  
}
```

```
public void run(IAction action) {  
    // TODO Auto-generated method stub  
    Job job = new Job("myJob") {  
  
        public IStatus run(IProgressMonitor monitor) {  
            System.out.println("moin");  
            monitor.beginTask("Long running thing...", 100);  
            for (int i = 0; i < 10; i++) {  
                monitor.subTask("I'm doing something here " + i);  
                mysleep(1000);  
                monitor.worked(i);  
            }  
            monitor.done();  
            return Status.OK_STATUS;  
        }  
    };  
    job.setUser(true);  
    job.schedule();  
}  
  
private void mysleep(Integer waitTime) {  
    try {  
        System.out.println("Waiting");  
        Thread.sleep(waitTime);  
    } catch (Throwable t) {  
        System.out.println("Wait time interrupted");  
    }  
}  
  
public void selectionChanged(IAction action, ISelection selection) {
```



```
// TODO Auto-generated method stub
```

```
    }  
}
```

你必须在 `ApplicationWorkbenchWindowAdvisor.java` 类的 `preWindowOpen()` 方法里开启进度区

```
package progresstest;
```

```
import org.eclipse.swt.graphics.Point;  
import org.eclipse.ui.application.ActionBarAdvisor;  
import org.eclipse.ui.application.IActionBarConfigurer;  
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;  
import org.eclipse.ui.application.WorkbenchWindowAdvisor;
```

```
public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {
```

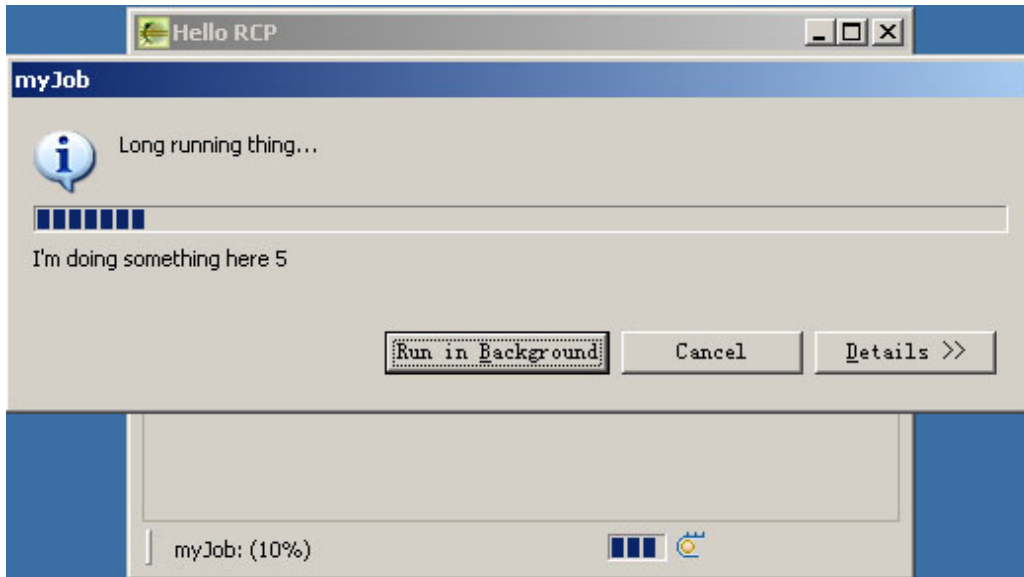
```
    public ApplicationWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurer) {  
        super(configurer);  
    }
```

```
    public ActionBarAdvisor createActionBarAdvisor(IActionBarConfigurer configurer) {  
        return new ApplicationActionBarAdvisor(configurer);  
    }
```

```
    public void preWindowOpen() {  
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();  
        configurer.setInitialSize(new Point(400, 300));  
        configurer.setShowCoolBar(false);  
        configurer.setShowStatusLine(false);  
        configurer.setShowProgressIndicator(true);  
    }
```

```
    configurer.setTitle("Hello RCP");  
  }  
}
```

运行结果如图：



## 第14章 将外部类包含进你的程序

### 14.1概述

使用 jars 有几个方面。对于你的产品来说，你需要将他们添加到你的构建路径中去，使编译器发现第一个类的定义。对于应用程序的运行测试，你必须把 jars 放入运行路径中。对于你的产品部署来说，你必须创建一个插件，以捆绑 jars

提示：

我个人习惯在工程里把所有的 jar 放在一个目录里，并命名为 `lib`，但这仅仅是我个人偏好。你可以将这些 jar 文件放入一个特别的目录里，并且指明他们为外部 jar。

提示：

打开 `plugin.xml`，如果在 “Dependencies” 添加依赖性，java 类路径将自动被更新。这将成为你所钟意的添加依赖性的方法

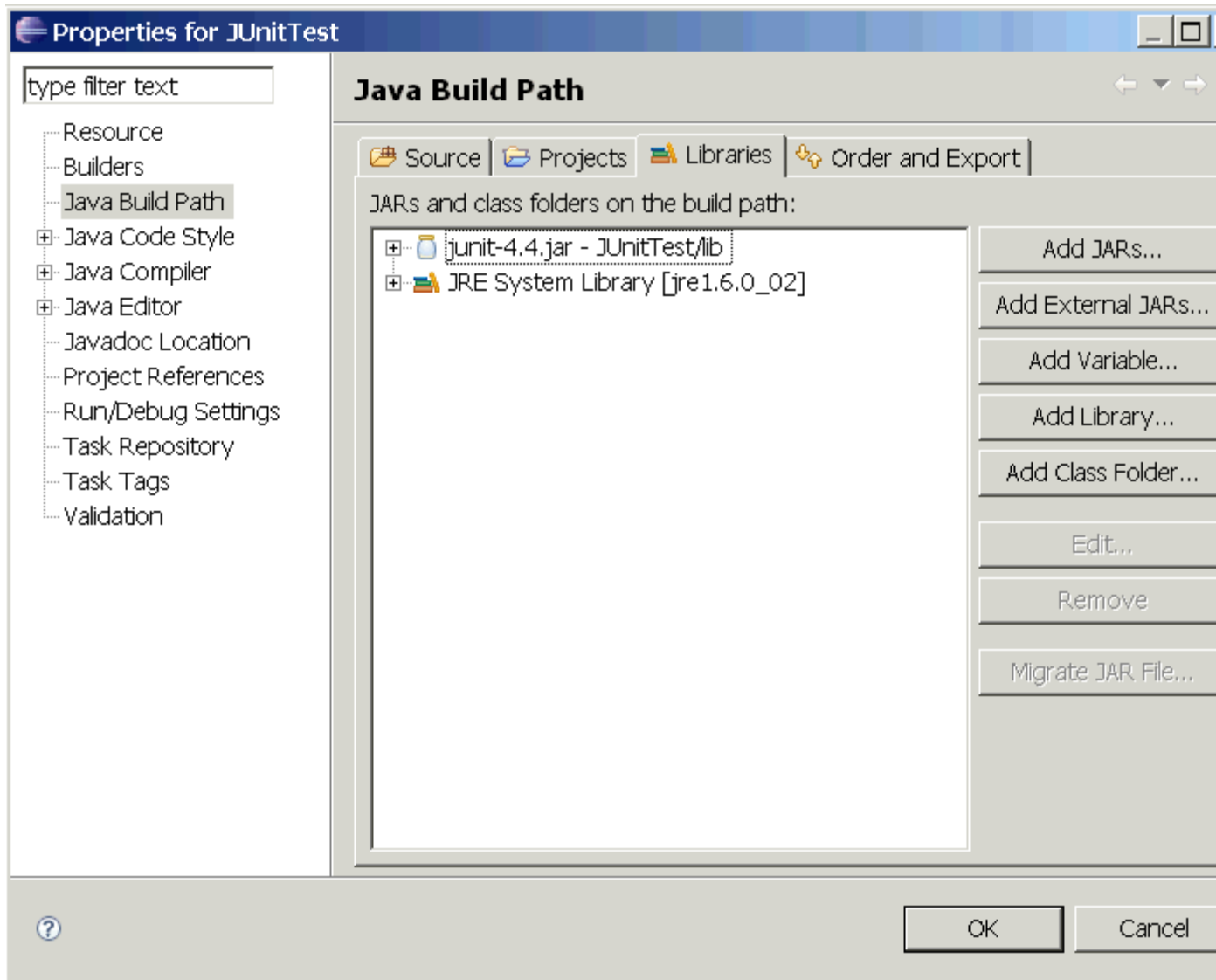
### 14.2向构建路径中添加jar

按下面步骤向工程中添加外部 jar

创建一个新文件夹，命名为 `lib`，或者使用已经存在的目录。

选择 `import->file system->import jar`


选择你的工程，鼠标右键点击，选择 “properties”，在 “libraries” 里选择 “Add JARs” 在 “Order and Export” 里将你的 jar 文件包含进来，并将他向上移动，避免冲突



### 14.3使jar在你的运行路径里有效

为了在你的 RCP 应用程序里使用外部类，你必须将他们添加如运行环境的 `classpath`。否则运行时，你将收到“`class not found exceptions`”异常。

双击 `plugin.xml` 文件，选择 `Runtime` 标签，在其中修改就可以了

 **Runtime**

**Exported Packages**  
Enumerate all the packages that this plug-in exposes to clients. All other packages will be hidden from clients at all times.

Add...

Remove

Properties...

Calculate Uses

Total: 0

**Package Visibility (Eclipse 3.1 or later)**  
When the runtime is in strict mode, the selected packages will be:

☐ visible to downstream plug-ins

☐ hidden from all plug-ins except:

**Classpath**  
Specify the libraries and folders that constitute the plug-in's classpath. The classes and resources are assumed to be at the root of the classpath.

lib/derby.jar

.

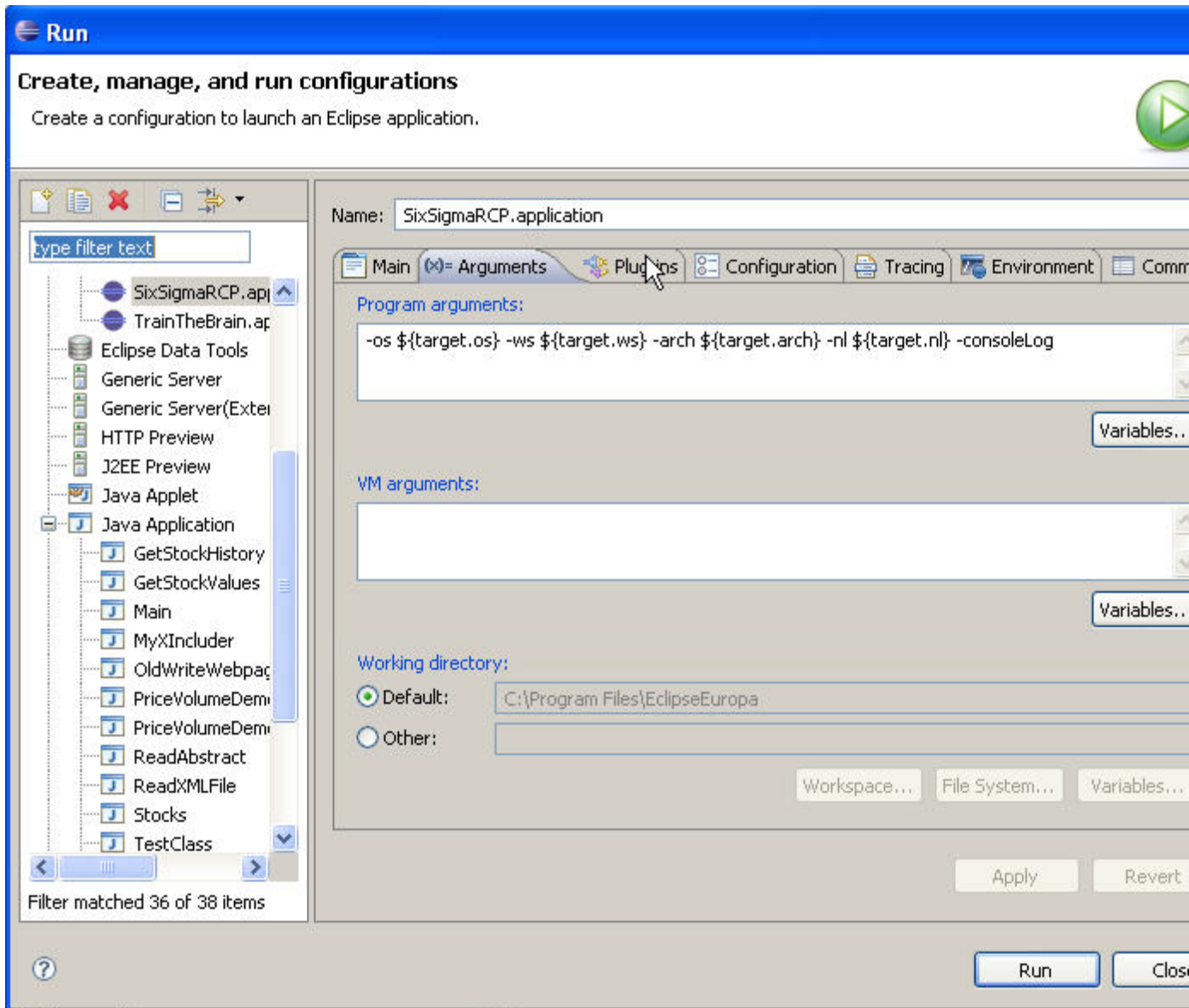
Overview | Dependencies | **Runtime** | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties

133 / 170

## 第15章 提示和策略

### 15.1 控制台日志

如果你想向控制台输出 eclipse 日志。使用 `-consoleLog` 作为程序参数



优点:

你不用寻找日志文件

你可以惦记一个堆栈跟踪**通信的**源码文件

### 15.2 保存用户的布局

为了记住用户的布局和窗口大小，以便下次启动时有同样设置。你可以向 `ApplicationWorkbenchAdvisor` 类的里添加 `configurer.setSaveAndRestore(true)`方法

```
package addactiontoview;

import org.eclipse.ui.application.IWorkbenchConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchAdvisor;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;

public class ApplicationWorkbenchAdvisor extends WorkbenchAdvisor {

    private static final String PERSPECTIVE_ID = "AddActiontoView.perspective";

    public WorkbenchWindowAdvisor createWorkbenchWindowAdvisor(
        IWorkbenchWindowConfigurer configurer) {
        return new ApplicationWorkbenchWindowAdvisor(configurer);
    }

    public String getInitialWindowPerspectiveId() {
        return PERSPECTIVE_ID;
    }

    public void initialize(IWorkbenchConfigurer configurer) {
        super.initialize(configurer);
        configurer.setSaveAndRestore(true);
    }
}
```

Eclipse 也有一个预定义的 action 可以重设 perspective。向你的程序中添加 `action` `ActionFactory.RESET_PERSPECTIVE.create(window)`

## 15.3 获得 display

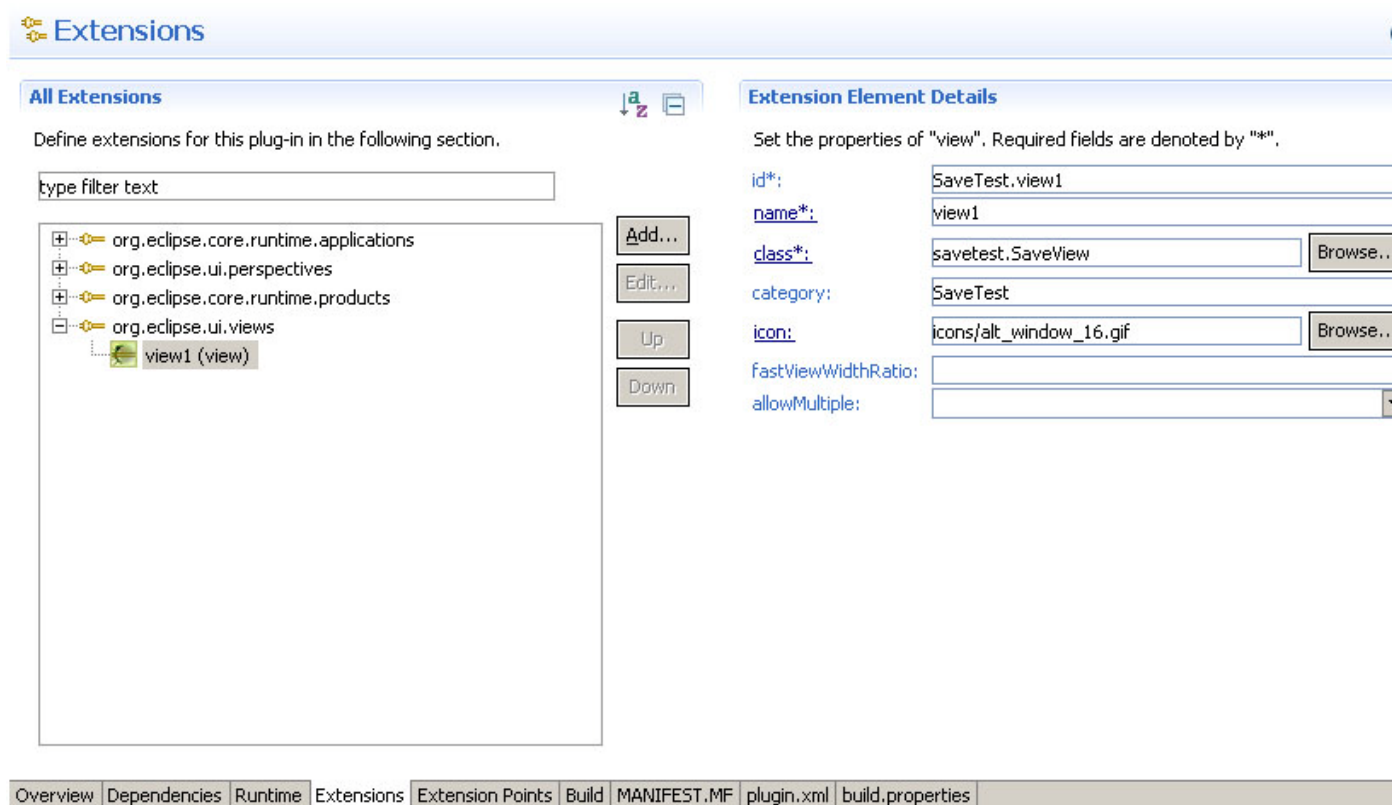
使用 `getSite().getShell().getDisplay();` 可以获得 display。

### 16. 4. 使用 eclipse 的 “保存” action

接下来将解释如何使用默认的 eclipse action “Save”

使用 “Hello RCP” 创建一个新的工程。

添加一个视口 “Save View”



向 `ApplicationActionBarAdvisor` 添加 Eclipse Save action 进入你的菜单

package savetest;

```
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.ui.IWorkbenchActionConstants;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.actions.ActionFactory;
```



```
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;

public class ApplicationActionBarAdvisor extends ActionBarAdvisor {

    private IWorkbenchAction saveAction;

    public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
        super(configurer);
    }

    protected void makeActions(IWorkbenchWindow window) {
        saveAction = ActionFactory.SAVE.create(window);
        register(saveAction);
    }

    protected void fillMenuBar(IMenuManager menuBar) {
        MenuManager fileMenu = new MenuManager("&File",
            IWorkbenchActionConstants.M_FILE);
        fileMenu.add(saveAction);
        menuBar.add(fileMenu);
    }
}
```

为了是“Save”按钮与你的视口相连，你需要添加 `ISaveablePart`。无论何时，一旦视口中的数据发生变化，`isDirty()`将返回“true”。这将告诉系统视口中的内容必须要保存了。

```
package savetest;
```

```
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.KeyListener;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.ISaveablePart;
import org.eclipse.ui.ISaveablesSource;
import org.eclipse.ui.Saveable;
import org.eclipse.ui.part.ViewPart;

public class SaveView extends ViewPart implements ISaveablePart {

    protected boolean dirty= false;

    public SaveView() {
        // TODO Auto-generated constructor stub
    }

    public void createPartControl(Composite parent) {
        Text text = new Text(parent, SWT.BORDER);
        text.setText("Hello");
        text.addKeyListener(new KeyListener(){

            public void keyPressed(KeyEvent e) {
                // TODO Auto-generated method stub
            }

            public void keyReleased(KeyEvent e) {
```

```
        dirty = true;

        // Make sure that Eclipse knows that the view is changed.
        firePropertyChange(ISaveablePart.PROP_DIRTY);

    }

});

}

public void setFocus() {
    // TODO Auto-generated method stub
}

public void doSave(IProgressMonitor monitor) {
    System.out.println("Imagine a very complex saving here...");
    dirty= false;
    firePropertyChange(ISaveablePart.PROP_DIRTY);
}

public void doSaveAs() {

}

public boolean isDirty() {
    return dirty;
}

public boolean isSaveAsAllowed() {
    return true;
}
```

```
public boolean isSaveOnCloseNeeded() {  
    return true;  
}  
  
}
```

运行应用程序，如果你改变了视口中的输入，视口将被标记已经做了改动，并且 **save action** 就由灰变亮，可以使用了。在你点击了保存之后，**save action** 又不可用了。如果在视口内容改变之后关闭视口或者应用程序，系统将提示，是否将改变保存

## 15.4 装载模型

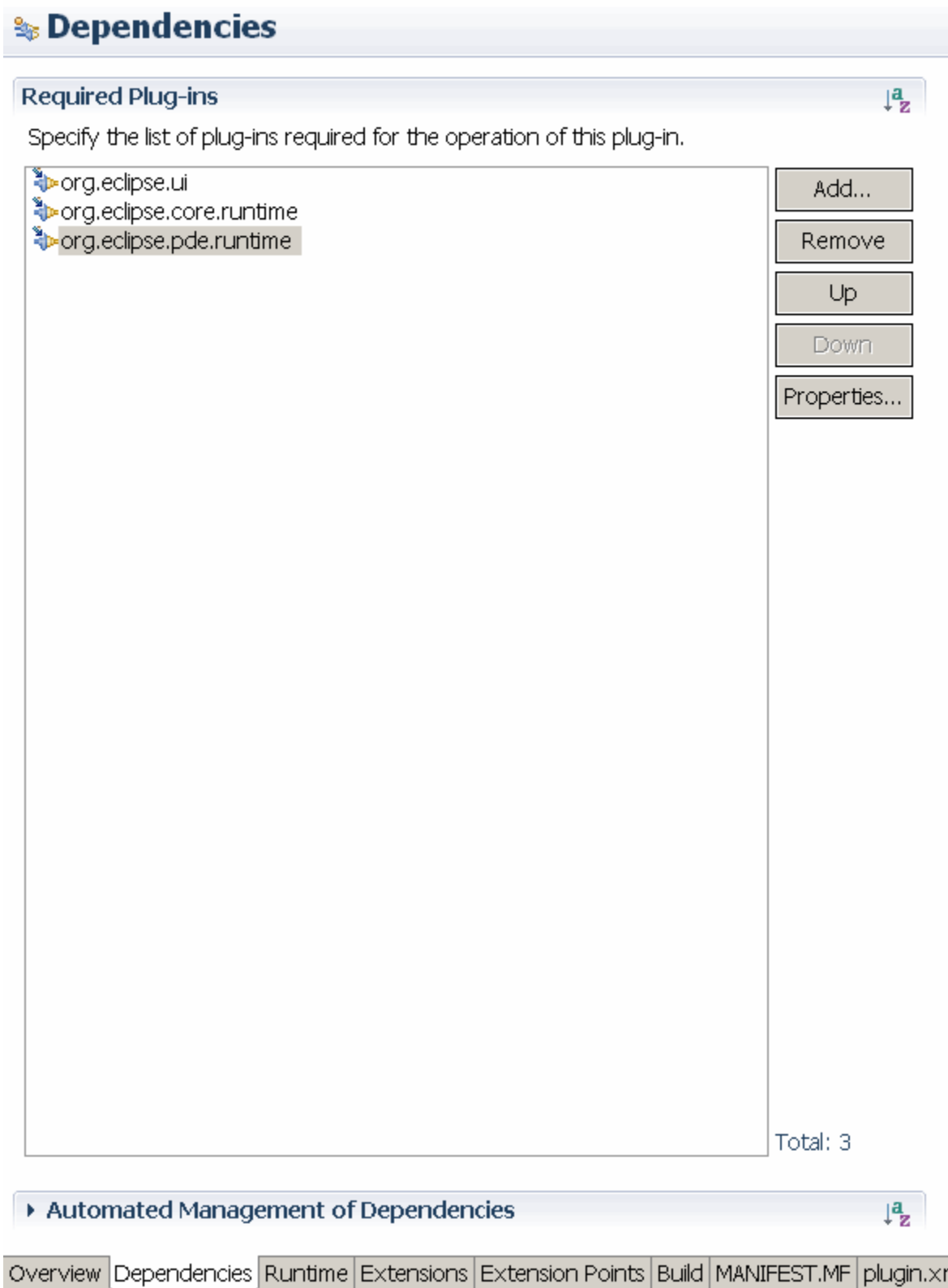
If the RCP application has to read data from a source this should usually be done before the user interface is created. Use method `initialize()` in `ApplicationWorkbenchAdvisor.java` for this. This method is called at the very beginning of the life cycle of your application. Alternatively you can use method `preStartup()` in `ApplicationWorkbenchAdvisor.java` which is called after `initialize` but before the first window is opened.

## 15.5 向你的程序添加错误日志视口

在一些出错的情况下 `eclipse` 会向你给出一些错误信息。在你的程序中使用已存在的错误日志，可以使这些错误变的用户可见。

创建一个新工程 “ErrorTest” 做这个例子，使用 “Hello RCP” 模板。

选择 `plugin.xml` 文件，添加依赖性 `org.eclipse.pde.runtime`



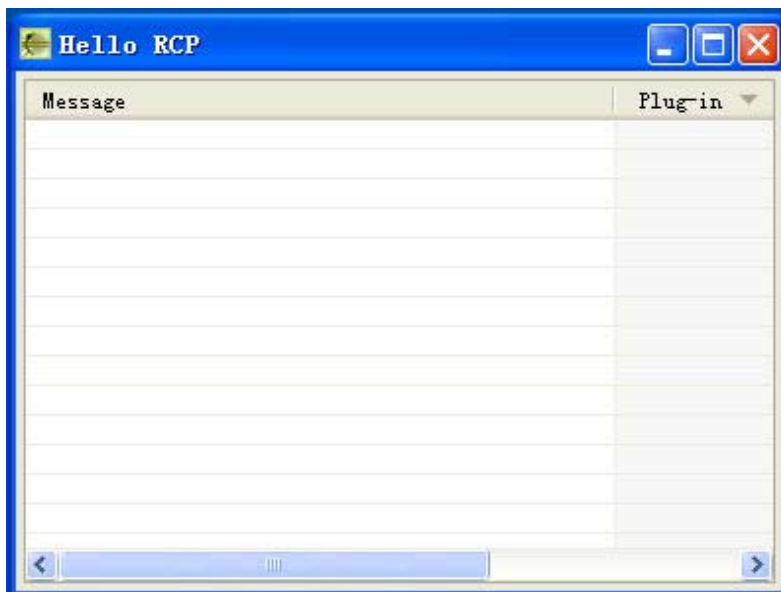
在 `Perspective` 类中包含如错误日志视口。使用 `org.eclipse.pde.runtime.LogView` 的 ID，代码如下

```
package errortest;
```

```
import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;

public class Perspective implements IPerspectiveFactory {
    public void createInitialLayout(IPageLayout layout) {
        String editorArea = layout.getEditorArea();
        layout.setEditorAreaVisible(false);
        layout.addStandaloneView("org.eclipse.pde.runtime.LogView", false,
            IPageLayout.LEFT, 0.25f, editorArea);
    }
}
```

运行程序， 得到结果如图



## 第16章 制造一个产品

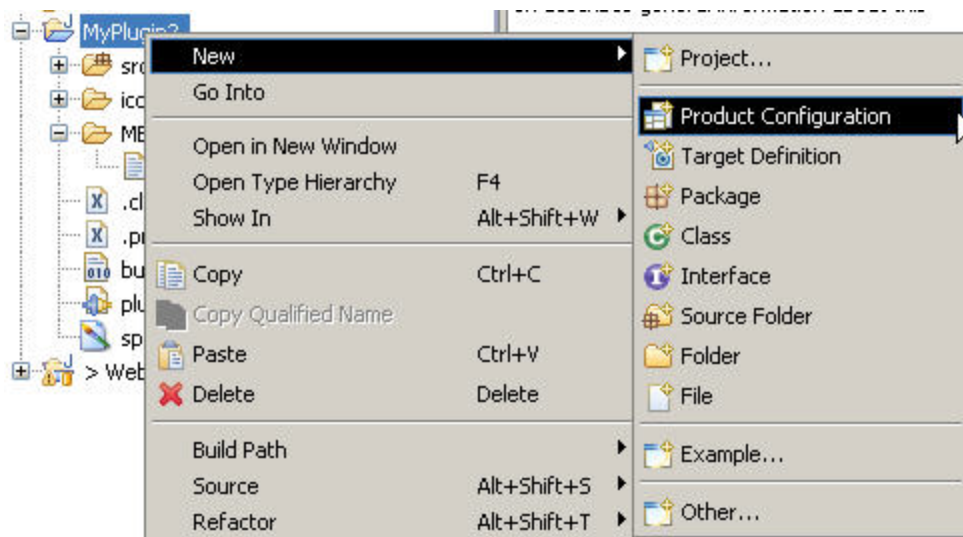
### 16.1 概述

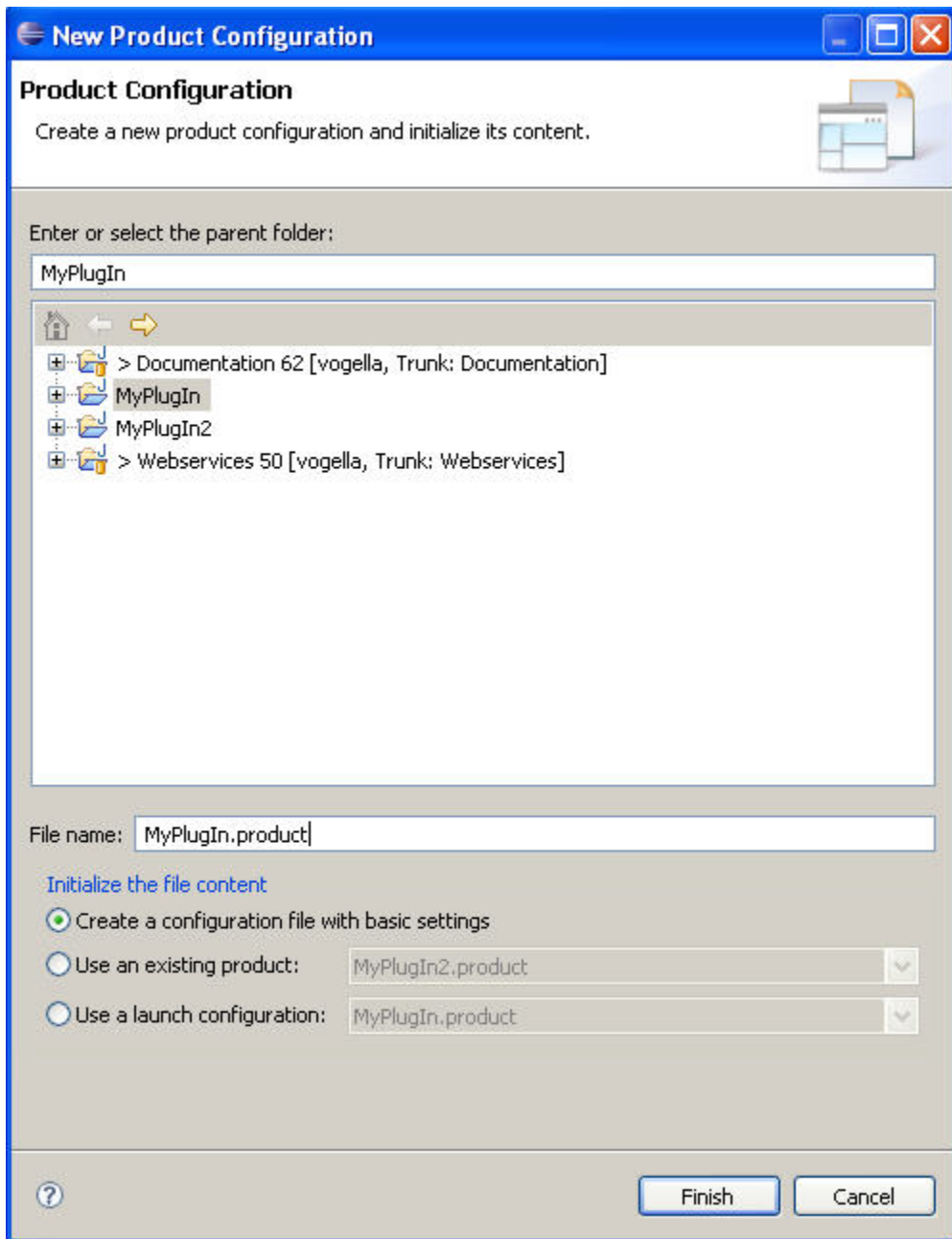
如果你想发布你的程序。你可以向产品中打包所有需要的包、设置文件等。

在 eclipse 中，一个产品是你程序用到的所有东西，包括所依赖的其他插件，一个运行命令，程序商标（图标）等

### 16.2 创建一个工程

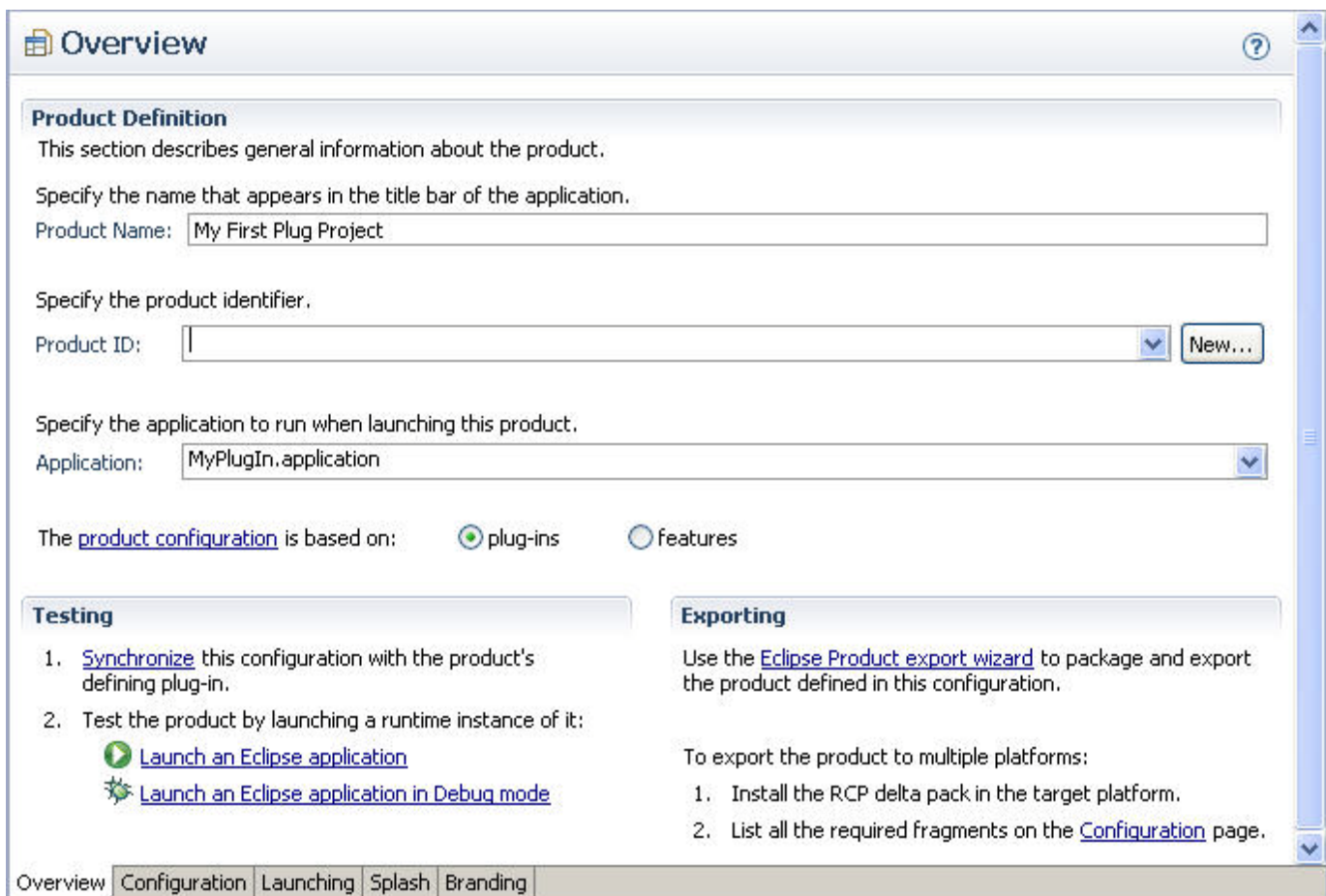
右键点击工程，选择 New->Product Configuration。键入一个名字，点击完成。





为你的产品键入一个名字，在你的应用程序下选择 **application** 类。产品的名称将显示在窗口的标题位置（这个名字将仅仅是显示，如果你没有在 `ApplicationWorkbenchWindowAdvisor` 类的 `preWindowOpen()` 中通过 `configurer.setTitle()` 设定名称。）







The screenshot shows the 'Overview' window in Eclipse, specifically the 'Product Definition' section. The window has a title bar with a question mark icon. The 'Product Definition' section contains the following fields and options:

- Product Name:** My First Plug Project
- Product ID:** (empty field) with a 'New...' button.
- Application:** MyPlugIn.application
- The product configuration is based on:** ☒ plug-ins ☐ features

Below the 'Product Definition' section, there are two tabs: 'Testing' and 'Exporting'.

**Testing**

- [Synchronize](#) this configuration with the product's defining plug-in.
- Test the product by launching a runtime instance of it:
  -  [Launch an Eclipse application](#)
  -  [Launch an Eclipse application in Debug mode](#)

**Exporting**

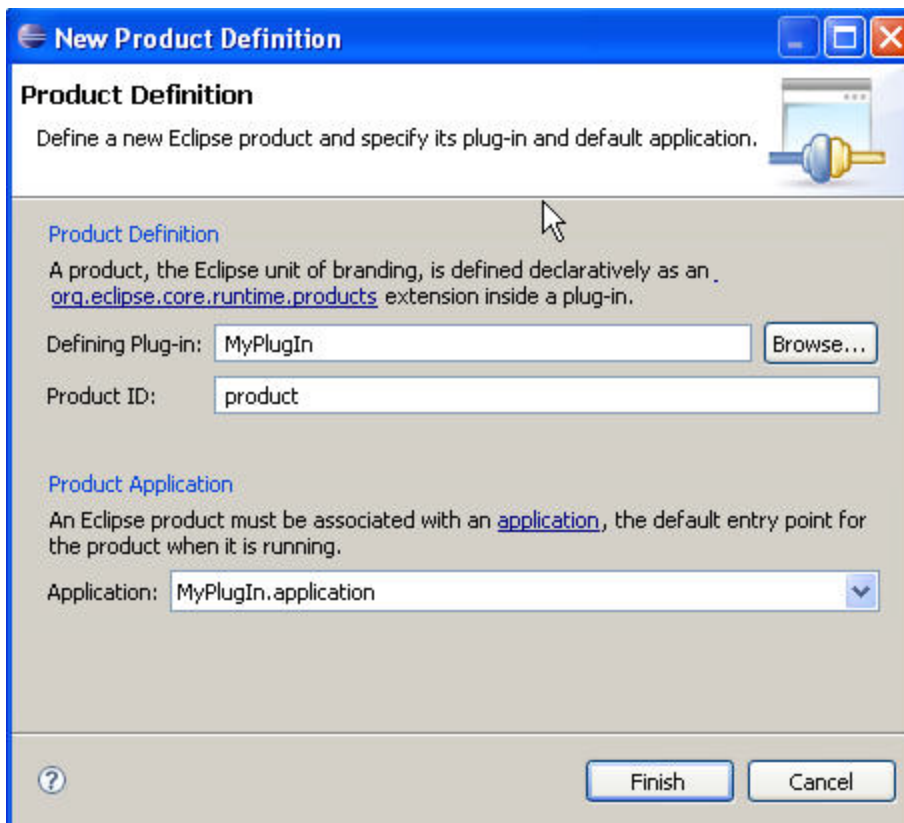
Use the [Eclipse Product export wizard](#) to package and export the product defined in this configuration.

To export the product to multiple platforms:

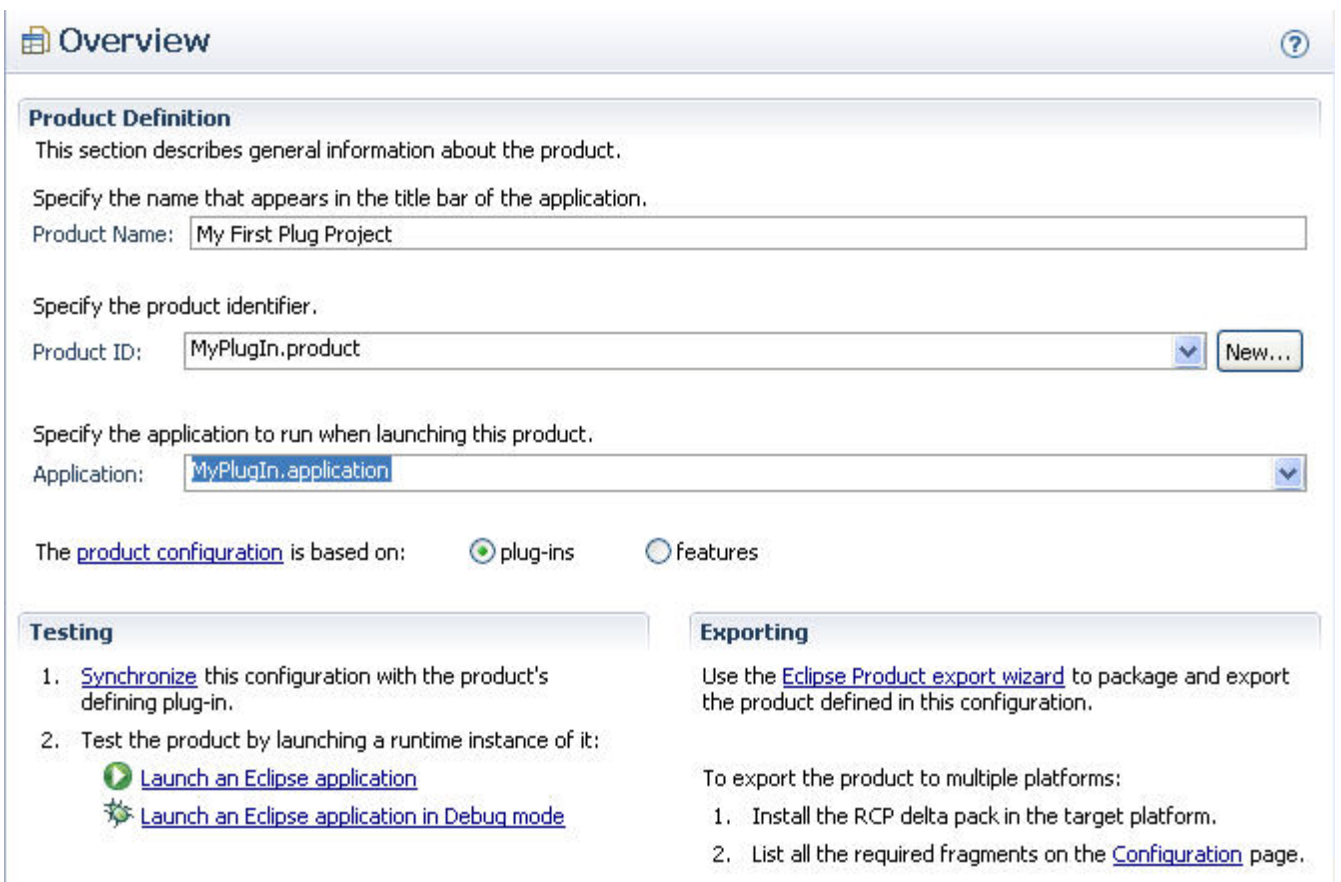
- Install the RCP delta pack in the target platform.
- List all the required fragments on the [Configuration](#) page.

At the bottom of the window, there is a tabbed interface with the following tabs: Overview, Configuration, Launching, Splash, Branding.

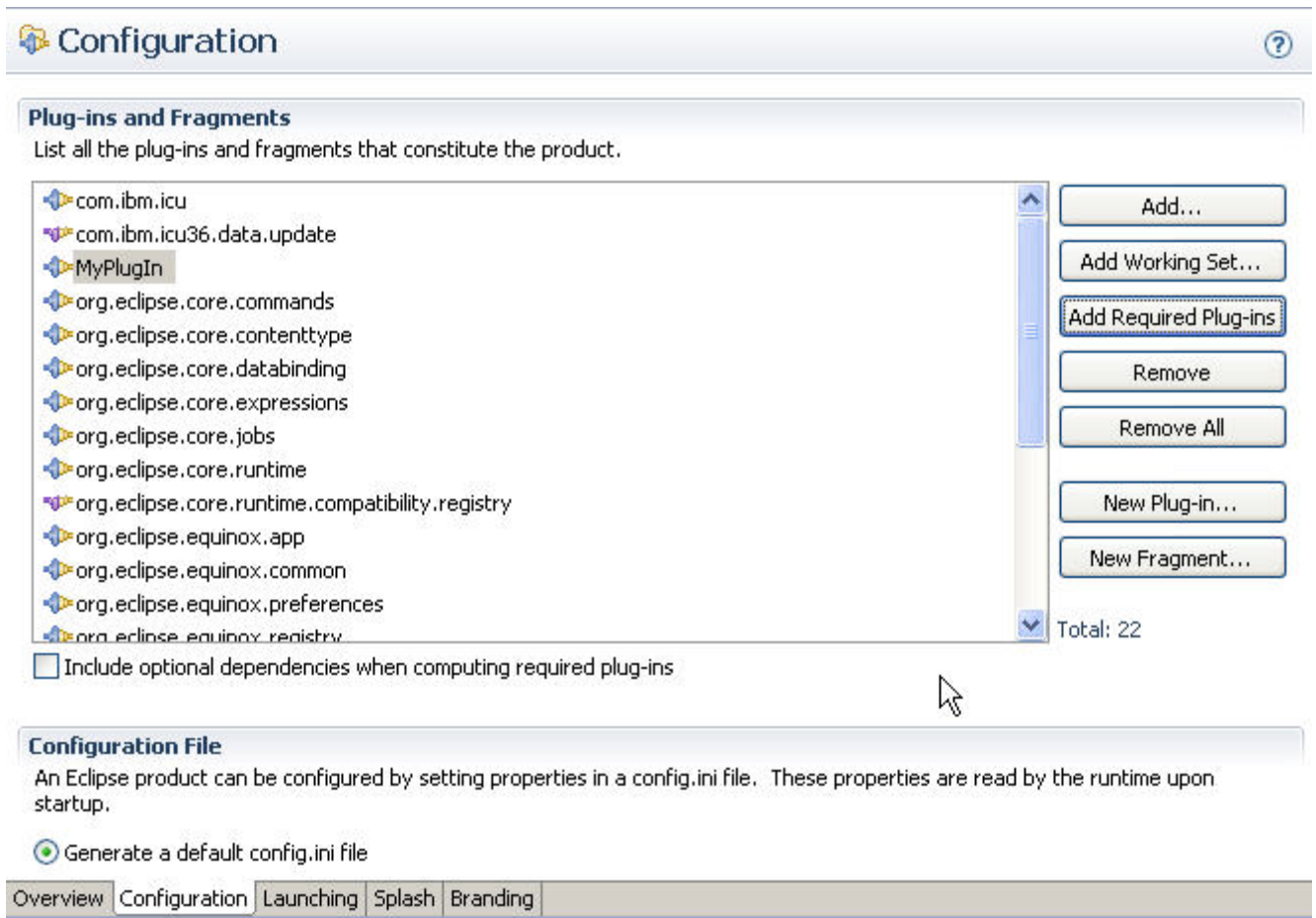
为了给你的插件创建一个产品，点击 Product ID 后的“New”按钮，为你的插件创建一个产品 ID



显示结果如下



进入“Configuration”标签，点击“Add”，选择你创建的插件（包括为外部 jar 的插件），并且点击“Add Required Plugins”。保存之后在返回到“overview”画面



## 16.3测试你的产品

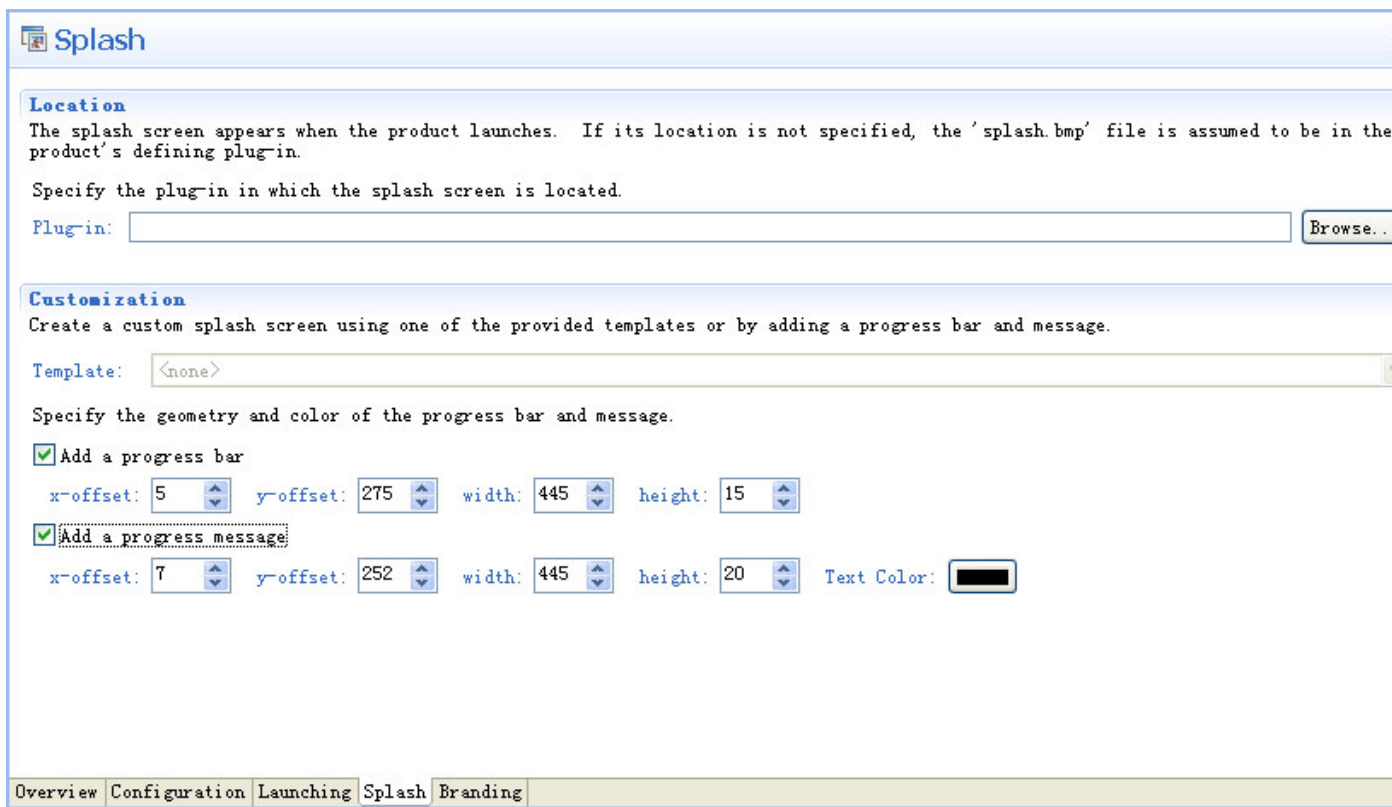
在“overview”面上点击“synchronize”，然后点击“launch application”。

## 第17章 商标

### 17.1 欢迎页面


打开产品文件\*.product，切换到“Splash”标签页，在这里你可以指定一个欢迎页面。你的应用程序将使用一个必须放在主目录下的 splash.bmp 文件。这个文件必须是 BMP 类型的。并且，必须命名为 splash.bmp

你可以添加一个信息，或者进度条给你的欢迎页面，通过选择这些相对应的设定。



### 17.2 商标

标准 eclipse 的“About Dialog”可以被注明商标。你可以向你的程序添加一个图标或者文字。

 Branding

1 warning detected

Window Images

Specify the images that will be associated with the application window. These GIF images are typically located in the product's defining plug-in.

16x16 Image:

Browse

32x32 Image:

Browse

48x48 Image:

Browse

64x64 Image:

Browse

128x128 Image:

Browse

About Dialog

Customize the text and image of the About dialog. The GIF image is typically located in the product's defining plug-in and its size must not exceed 500x330 pixels. The text is not shown if the image size exceeds 250x330 pixels.

Image:

/MyPlugin2/icons/ttb.GIF

Browse

Text:

Train-the-Brain is written by Lars Vogel

Welcome Page

The welcome page appears the first time the product is launched. It is intended to introduce the features of the product to new users.

Specify this product's welcome page.

Intro ID:

New

Overview

Configuration

Launching

Splash

Branding

## 17.3 风格化 launcher


launcher 是产品开发期间所创建的可执行程序，每一个带有“eclipse”图标的默认的“eclipse.exe”文件也被创建。可进入 **Launching** 标签，改变你的产品设置。

在这，你可以指定应用程序的名字、图标。确认图标深度被修改正确，否则这些图标将不被 eclipse 使用

## Launching

### Java Runtime Environment

Specify the JRE to be bundled with the product. Platform-specific JREs should be entered on their respective tabs.

linux | macosx | solaris |  win32

☒ JRE Name:

☐ Execution Environment:

### Program Launcher

Customize the executable that is used to launch the product.

Launcher Name:

Customizing the launcher icon varies per platform.

linux | macosx | solaris |  win32

☒ Specify 6 separate BMP images:

16x16 (8-bit):

16x16 (32-bit):

32x32 (8-bit):

32x32 (32-bit):

48x48 (8-bit):

48x48 (32-bit):

☐ Use a single ICO file containing the 6 images:

File:

### Launching Arguments

Specify the program arguments to be passed to the product. Platform-specific arguments should be entered on their respective tabs.

 All Platforms

Program Arguments:

VM Arguments:



## 第18章 发布你的产品

为了创建一个可以在 eclipse IDE 环境之外独立运行的计算机程序，你需要发布产品。可使他人共享你的程序。

下面假定你没有使用外部 jar。如果你想在你的发布产品中添加第三方库（jars），必须将他们捆绑进插件。

Eclipse 会自动将已编译的类包含进构建输出中。你必须手动控制其他文件。

如果你使用了图标或者欢迎页面，也必须将他们假如构建输出中

选择你的 `plugin.xml` 并且选择“build”标签。确认 META-INF 目录被选。选择你的图标目录或者其他可被包含进输出的图形文件。

## Build Configuration

☐ Custom Build

### Runtime Information

Define the libraries, specify the order in which they should be built, and list the source folders that should be con



Add Library...

Up

Down

### Binary Build

Select the folders and files to include in the binary build.

<input type="checkbox"/>	<input checked="" type="checkbox"/>	.classpath
<input type="checkbox"/>	<input checked="" type="checkbox"/>	.project
<input checked="" type="checkbox"/>	<input type="checkbox"/>	.settings
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	META-INF
<input type="checkbox"/>	<input checked="" type="checkbox"/>	SixSigmaRCP.product
<input checked="" type="checkbox"/>	<input type="checkbox"/>	bin
<input type="checkbox"/>	<input checked="" type="checkbox"/>	build.properties
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	icons
<input checked="" type="checkbox"/>	<input type="checkbox"/>	lib
<input type="checkbox"/>	<input type="checkbox"/>	logo16.bmp
<input type="checkbox"/>	<input type="checkbox"/>	logo16a.bmp
<input checked="" type="checkbox"/>	<input type="checkbox"/>	logo16transparent.bmp
<input type="checkbox"/>	<input type="checkbox"/>	logo32.bmp
<input type="checkbox"/>	<input type="checkbox"/>	logo32a.bmp
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	plugin.xml
<input type="checkbox"/>	<input type="checkbox"/>	plugin_customization.ini
<input checked="" type="checkbox"/>	<input type="checkbox"/>	splash.bmp
<input checked="" type="checkbox"/>	<input type="checkbox"/>	src
<input type="checkbox"/>	<input type="checkbox"/>	user.sql

### Source Build

Select the folders

<input type="checkbox"/>	<input checked="" type="checkbox"/>	.class
<input type="checkbox"/>	<input checked="" type="checkbox"/>	.proj
<input checked="" type="checkbox"/>	<input type="checkbox"/>	.settin
<input checked="" type="checkbox"/>	<input type="checkbox"/>	META
<input type="checkbox"/>	<input checked="" type="checkbox"/>	SixSig
<input checked="" type="checkbox"/>	<input type="checkbox"/>	bin
<input type="checkbox"/>	<input checked="" type="checkbox"/>	build.
<input checked="" type="checkbox"/>	<input type="checkbox"/>	icons
<input checked="" type="checkbox"/>	<input type="checkbox"/>	lib
<input type="checkbox"/>	<input type="checkbox"/>	logo1
<input type="checkbox"/>	<input type="checkbox"/>	logo1
<input type="checkbox"/>	<input type="checkbox"/>	logo1
<input type="checkbox"/>	<input type="checkbox"/>	logo3
<input type="checkbox"/>	<input type="checkbox"/>	logo3
<input type="checkbox"/>	<input checked="" type="checkbox"/>	plugin
<input type="checkbox"/>	<input type="checkbox"/>	plugin
<input type="checkbox"/>	<input type="checkbox"/>	splash
<input checked="" type="checkbox"/>	<input type="checkbox"/>	src
<input type="checkbox"/>	<input type="checkbox"/>	user.s



### ▶ Extra Classpath Entries

提示

If after deployment your graphics are missing check this tab if you really include them in the deployment

点击 “Eclipse Product export wizard” 导出你的产品



 Overview 



### Product Definition

This section describes general information about the product.


Specify the name that appears in the title bar of the application.

Product Name:

Specify the product identifier.



Product ID:   

Specify the application to run when launching this product.

Application:  

The [product configuration](#) is based on: ☒ plug-ins ☐ features

### Testing

1. [Synchronize](#) this configuration with the product's defining plug-in.
2. Test the product by launching a runtime instance of it:
  -  [Launch an Eclipse application](#)
  -  [Launch an Eclipse application in Debug mode](#)

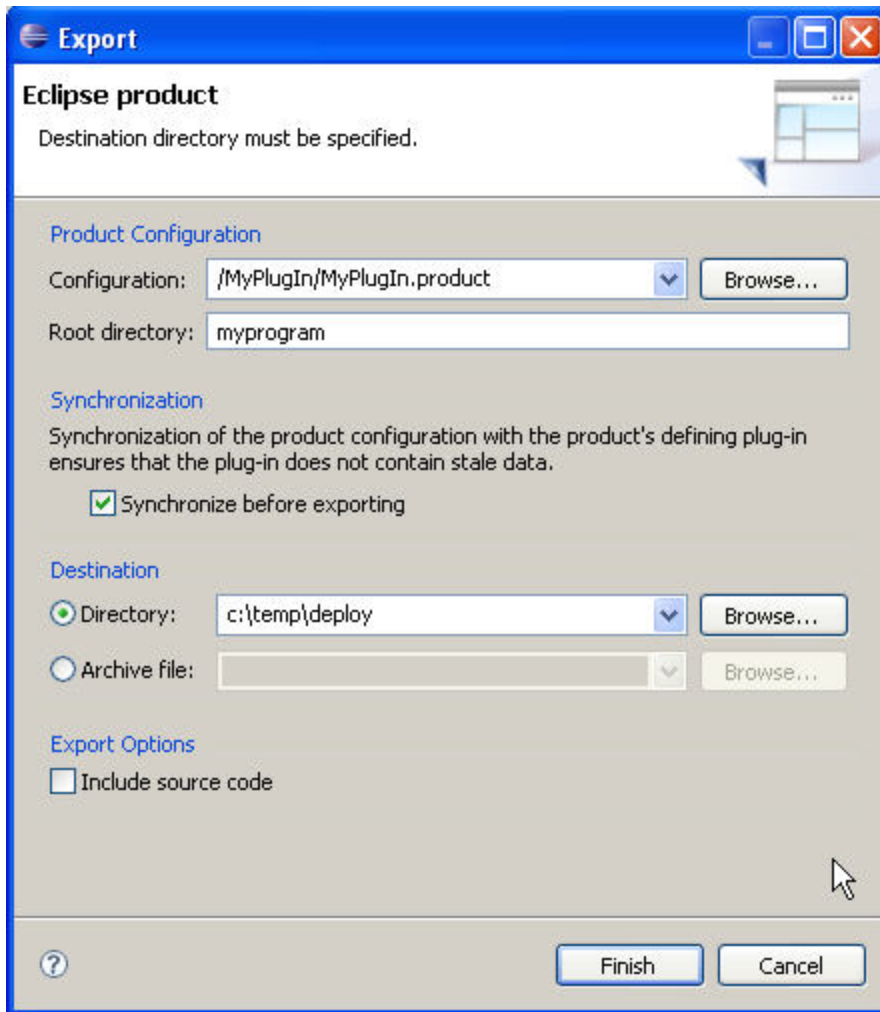
### Exporting

Use the [Eclipse Product export wizard](#) to package and export the product defined in this configuration.

To export the product to multiple platforms:

1. Install the RCP delta pack in the target platform.
2. List all the required fragments on the [Configuration](#) page.

修改指标，点击完成



这时，在你设定好的目录里会出现一系列文件及文件夹，其中有一个“eclipse.exe”，双击它，可以用来启动你的程序。（如果在“launching”里更改过名称之后，可执行文件叫不在被命名为“eclipse.exe”）

## 第19章 发布引入外部 jar 的产品

### 19.1 整合外部jar和第三方库

java 是将所有包捆绑进 jar 文件的。Eclipse RCP 需要将这些 jars 捆绑进插件工程。接下来介绍如何将这些 jar 整合入发布产品中。

### 19.2. 为你的jar创建一个插件工程

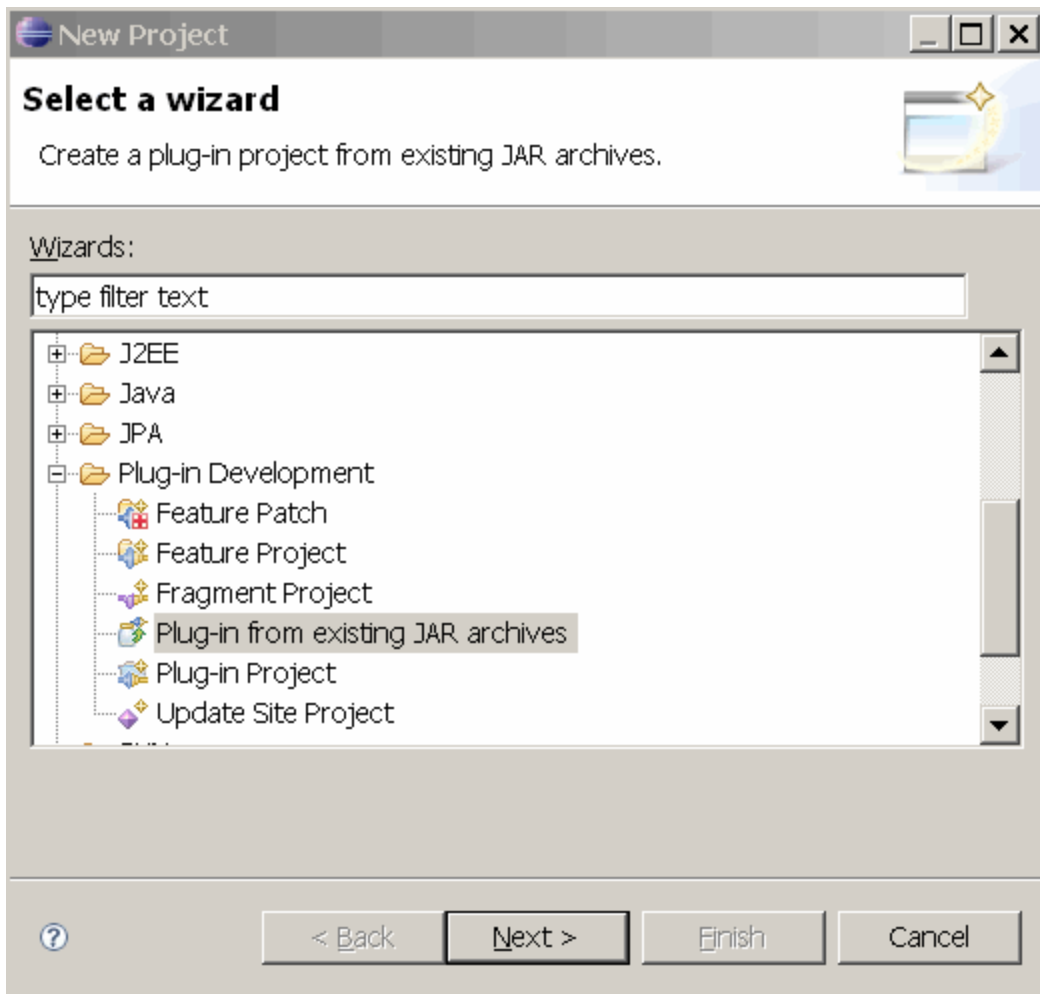
为了使你的 eclipse RCP 应用程序可被他人使用，你需要创建一个产品，并将之发布

如果你需要将外部 jar 添加入插件中，你需要将用到的 jar 文件重新打包，并且将他当作依赖性添加入产品中。这一章关于发布的描述，将告诉你如何添加依赖性。所以这里我先集中讲一下如何为外部 jar 创建一个插件。

#### Tip

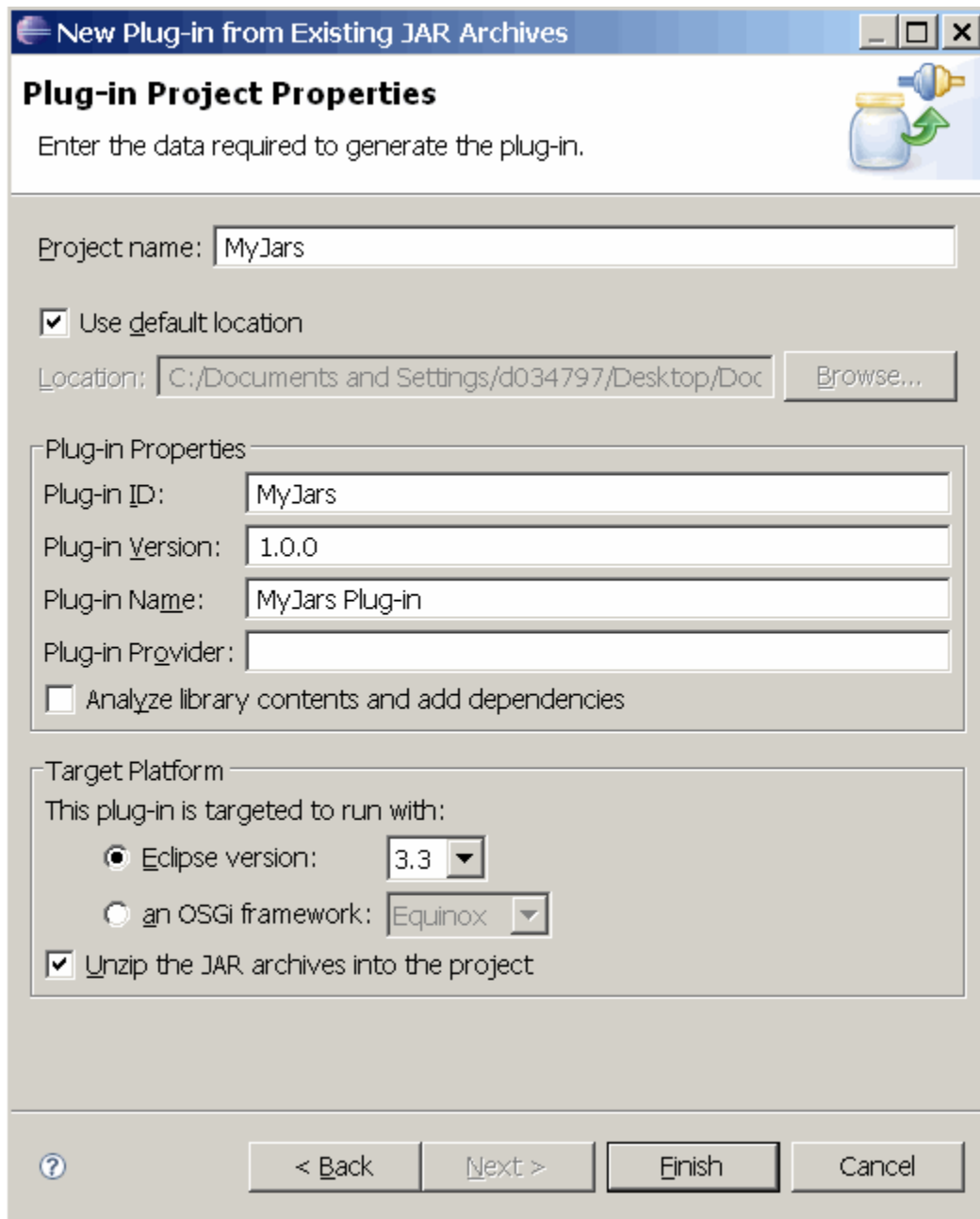
如果你需要将一个 jar 重新打包入一个插件中，最好先检查一下看他的许可证是否允许如此。

为外部 jar 创建一个插件。选择 File-> New -> Project...-> Plug-in Development -> Plug-in from existing JAR archives.



点击选择，添加你所需要的 jars 到你的新 plug-in，点击下一步。

为这个插件修改名字和版本。点击完成。



**New Plug-in from Existing JAR Archives**

**Plug-in Project Properties**

Enter the data required to generate the plug-in.

Project name:

☒ Use default location

Location:

Plug-in Properties

Plug-in ID:

Plug-in Version:

Plug-in Name:

Plug-in Provider:

☐ Analyze library contents and add dependencies

Target Platform

This plug-in is targeted to run with:

☒ Eclipse version:

☐ an OSGi framework:

☒ Unzip the JAR archives into the project

你现在已经将 jar 捆绑进新插件中，在“runtime”中核对 MANIFEST.MF。来自 jar 的所有包应该已经被包含进已导出的包，并且你的 RCP 应用程序也可以进入这些包。

## 第20章 向 RCP 应用程序中添加帮助

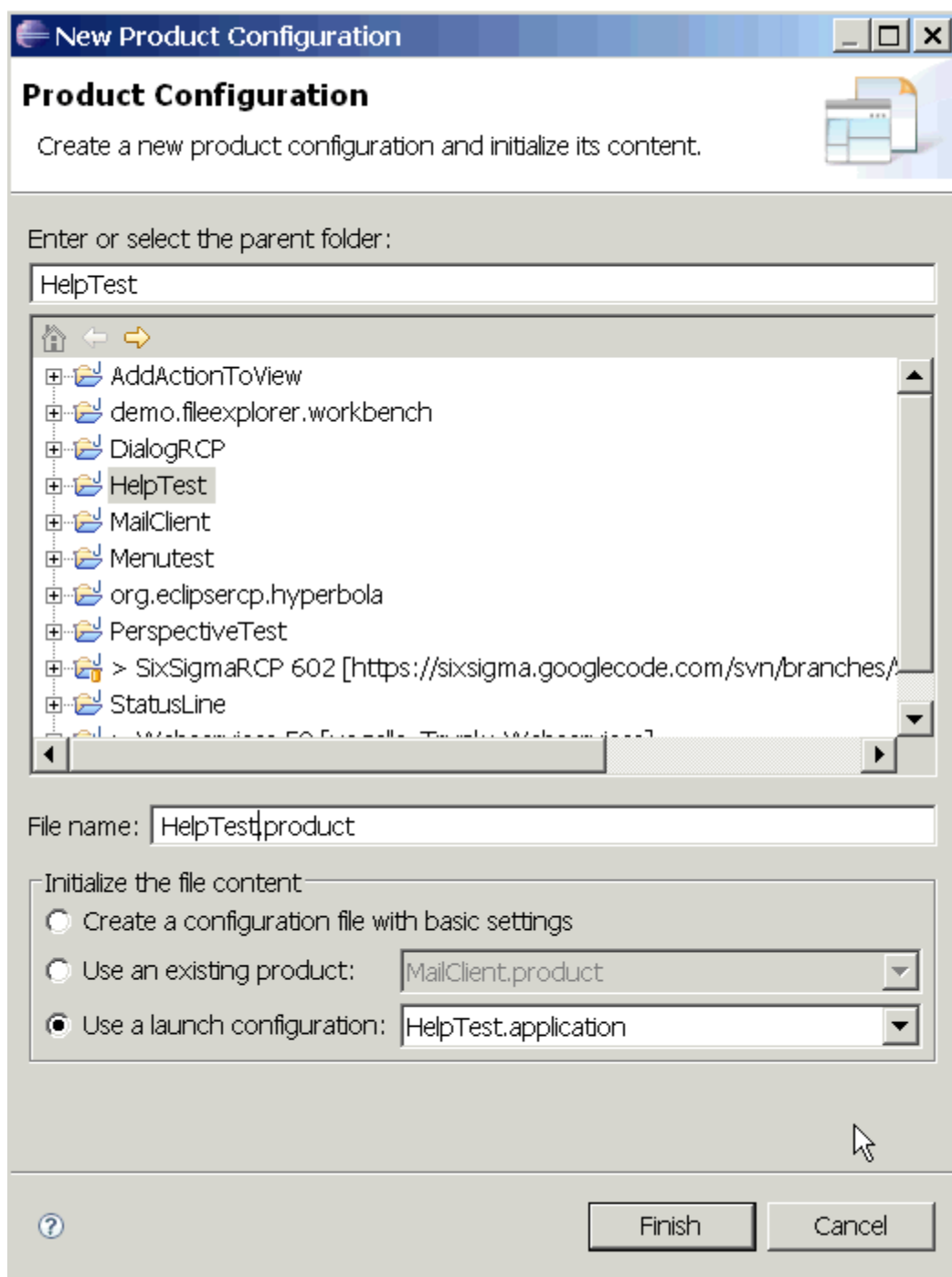
接下来将描述如何向 RCP 应用程序中添加帮助。Eclipse 的帮助是以 html 文档形式书写，会给出一个文档结构表格。向你的程序中添加帮助需要一个插件，这个插件不属于 RCP 软件开发工具包，因此，你需要向 RCP 中添加附加的插件。

### 20.1 创建一个新工程

使用“RCP application with a view”模板创建一个新工程，命名为“HelpTest”。运行他，先看一下他工作结果。

### 20.2 创建一个产品

创建产品“HelpTest.product”



 **Overview**

**Product Definition**

This section describes general information about the product.

Specify the name that appears in the title bar of the application.

Product Name:

Specify the product identifier.



Product ID:

Specify the application to run when launching this product.

Application:

The [product configuration](#) is based on: ☒ plug-ins ☐ features

**Testing**

- [Synchronize](#) this configuration with the product's defining plug-in.
- Test the product by launching a runtime instance of it:
  -  [Launch an Eclipse application](#)
  -  [Launch an Eclipse application in Debug mode](#)

**Exporting**

Use the [Eclipse Product export wizard](#) to export the product defined in this configuration.

To export the product to multiple platforms:

- Install the RCP delta pack in the target environment.
- List all the required fragments on the target environment.

## 20.3 添加依赖性

需要用到下面的插件。如果不添加进插件，将会得到一个运行时错误

org.apache.lucene

org.eclipse.help.appserver

org.eclipse.help.base

org.eclipse.help.ui

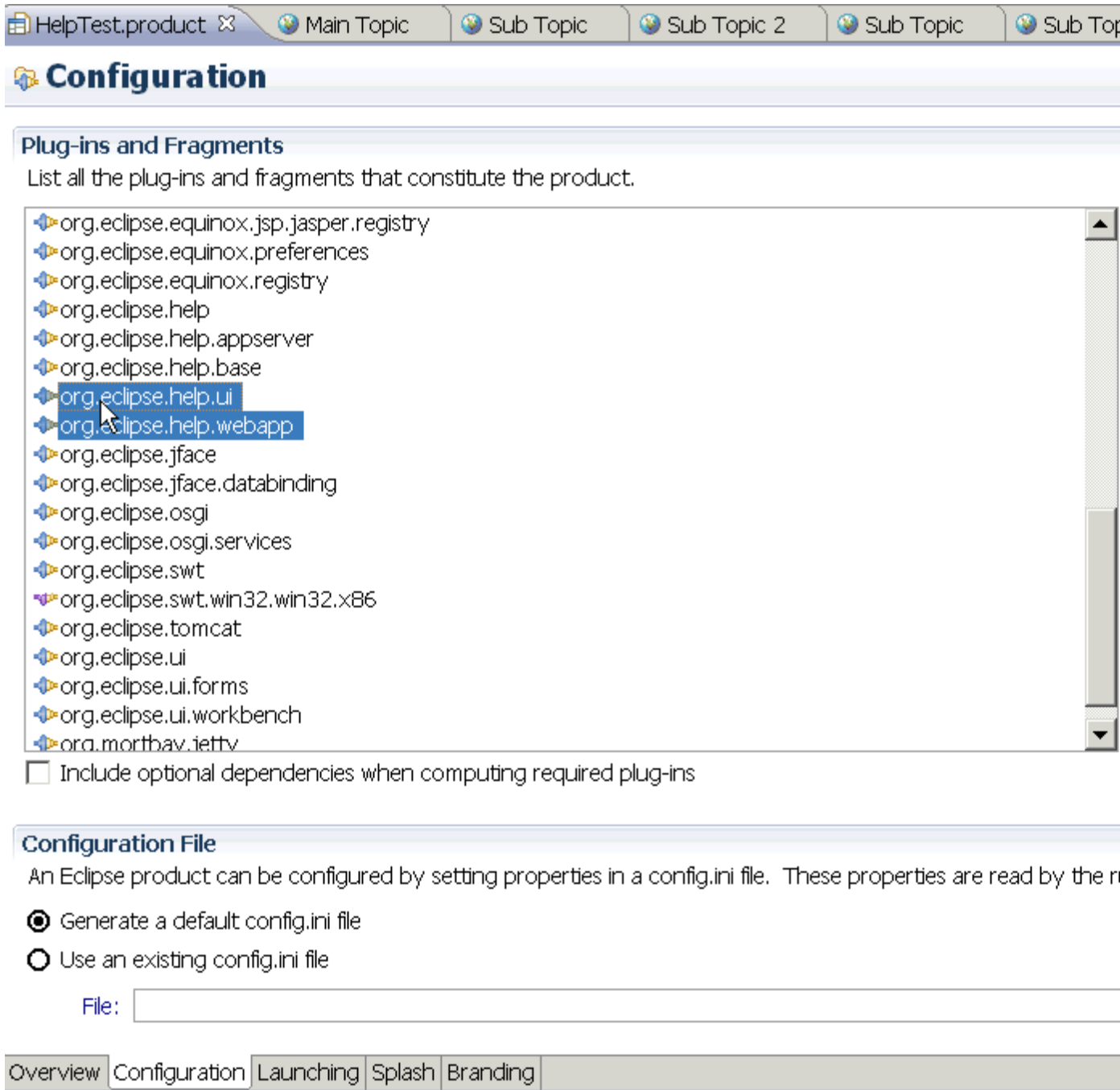
org.eclipse.help.webapp

org.eclipse.tomcat



org.eclipse.ui.forms

进入你的 HelpTest.product，选择 Configuration。点击“Add”按钮，添加 org.eclipse.help.webapp 和 org.eclipse.help.ui，然后点击 Add Required Plug-ins，从以上列表中选择所有插件



保存设置

#### 21. 4. 向程序中添加 action

向 ApplicationActionBarAdvisor 中添加 action 用以显示帮助

////需要自己写

```
package helptest;
```

```
import org.eclipse.jface.action.IMenuManager;
```

```
import org.eclipse.jface.action.MenuManager;
```

```
import org.eclipse.ui.IWorkbenchActionConstants;
```

```
import org.eclipse.ui.IWorkbenchWindow;
```

```
import org.eclipse.ui.actions.ActionFactory;
```

```
import org.eclipse.ui.actions.ActionFactory.IWorkbenchAction;
```

```
import org.eclipse.ui.application.ActionBarAdvisor;
```

```
import org.eclipse.ui.application.IActionBarConfigurer;
```

```
/**
```

```
 * An action bar advisor is responsible for creating, adding, and disposing of
```

```
 * the actions added to a workbench window. Each window will be populated with
```

```
 * new actions.
```

```
*/
```

```
public class ApplicationActionBarAdvisor extends ActionBarAdvisor {
```

```
    // Actions - important to allocate these only in makeActions, and then use
```

```
    // them
```

```
    // in the fill methods. This ensures that the actions aren't recreated
```

```
    // when fillActionBars is called with FILL_PROXY.
```

```
    private IWorkbenchAction exitAction;
```

```
    private IWorkbenchAction showHelpAction; // NEW
```

```
    private IWorkbenchAction searchHelpAction; // NEW
```

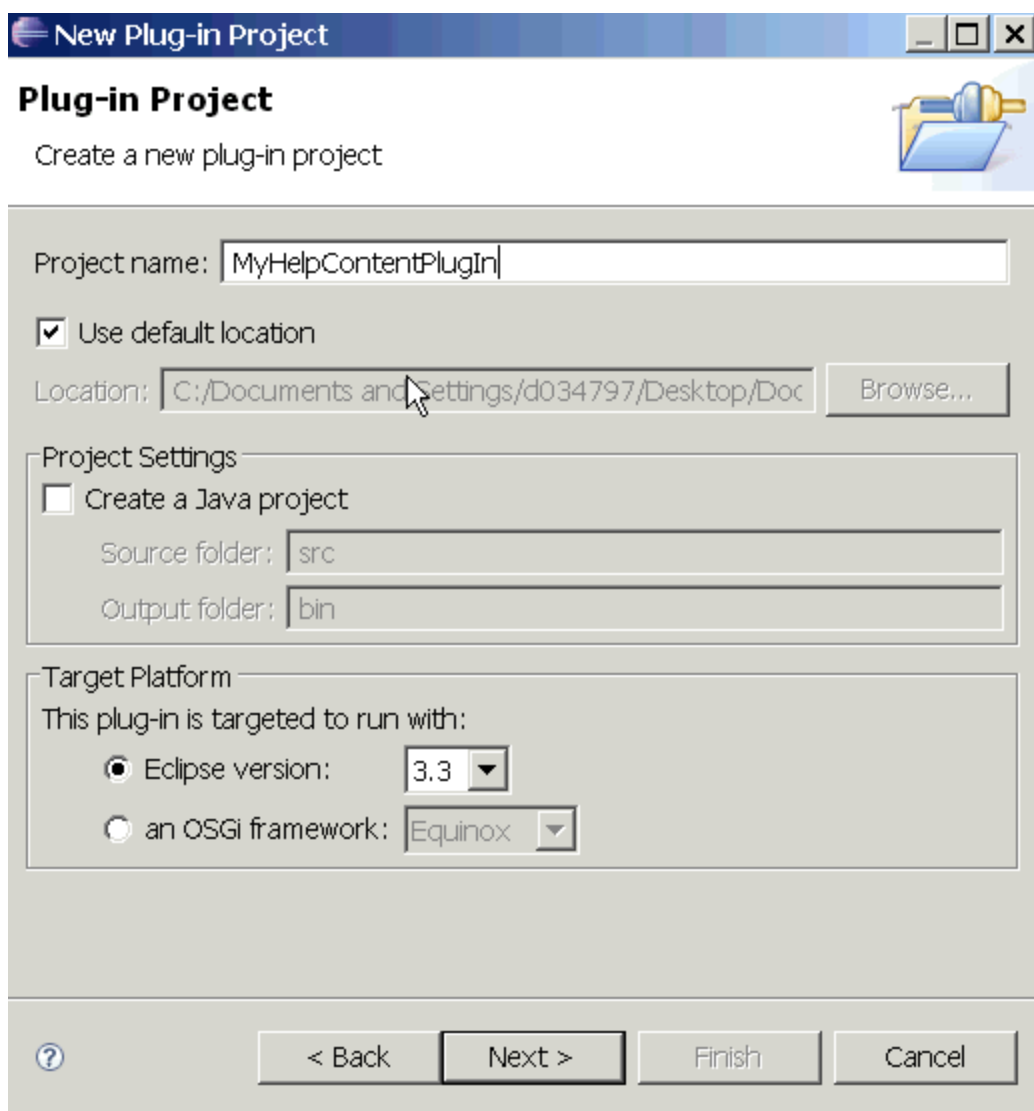
```
    public ApplicationActionBarAdvisor(IActionBarConfigurer configurer) {
```

```
        super(configurer);
```

```
    }
```

```
protected void makeActions(final IWorkbenchWindow window) {  
    exitAction = ActionFactory.QUIT.create(window);  
    register(exitAction);  
    showHelpAction = ActionFactory.HELP_CONTENTS.create(window); // NEW  
    register(showHelpAction); // NEW  
    searchHelpAction = ActionFactory.HELP_SEARCH.create(window); // NEW  
    register(searchHelpAction); // NEW  
}  
  
protected void fillMenuBar(IMenuManager menuBar) {  
    MenuManager fileMenu = new MenuManager("&File",  
        IWorkbenchActionConstants.M_FILE);  
    menuBar.add(fileMenu);  
    fileMenu.add(exitAction);  
  
    MenuManager helpMenu = new MenuManager("&Help",  
        IWorkbenchActionConstants.M_HELP); // NEW  
    helpMenu.add(showHelpAction); // NEW  
    helpMenu.add(searchHelpAction); // NEW  
    menuBar.add(helpMenu);  
}  
}
```

## 20.4 创建一个帮助插件工程////原著写的不好



The image shows the 'New Plug-in Project' dialog box in Eclipse. The title bar says 'New Plug-in Project'. The main heading is 'Plug-in Project' with a subtext 'Create a new plug-in project' and a plug icon. The 'Project name' field contains 'MyHelpContentPlugIn'. The 'Use default location' checkbox is checked. The 'Location' field shows a file path, and a 'Browse...' button is next to it. The 'Project Settings' section has an unchecked 'Create a Java project' checkbox, with 'Source folder' set to 'src' and 'Output folder' set to 'bin'. The 'Target Platform' section indicates the plug-in is targeted to run with either 'Eclipse version: 3.3' (selected) or 'an OSGi framework: Equinox'. At the bottom are buttons for '< Back', 'Next >', 'Finish', and 'Cancel', along with a help icon.

**New Plug-in Project**

**Plug-in Project**  
Create a new plug-in project

Project name:

☒ Use default location

Location:

**Project Settings**

☐ Create a Java project

Source folder:

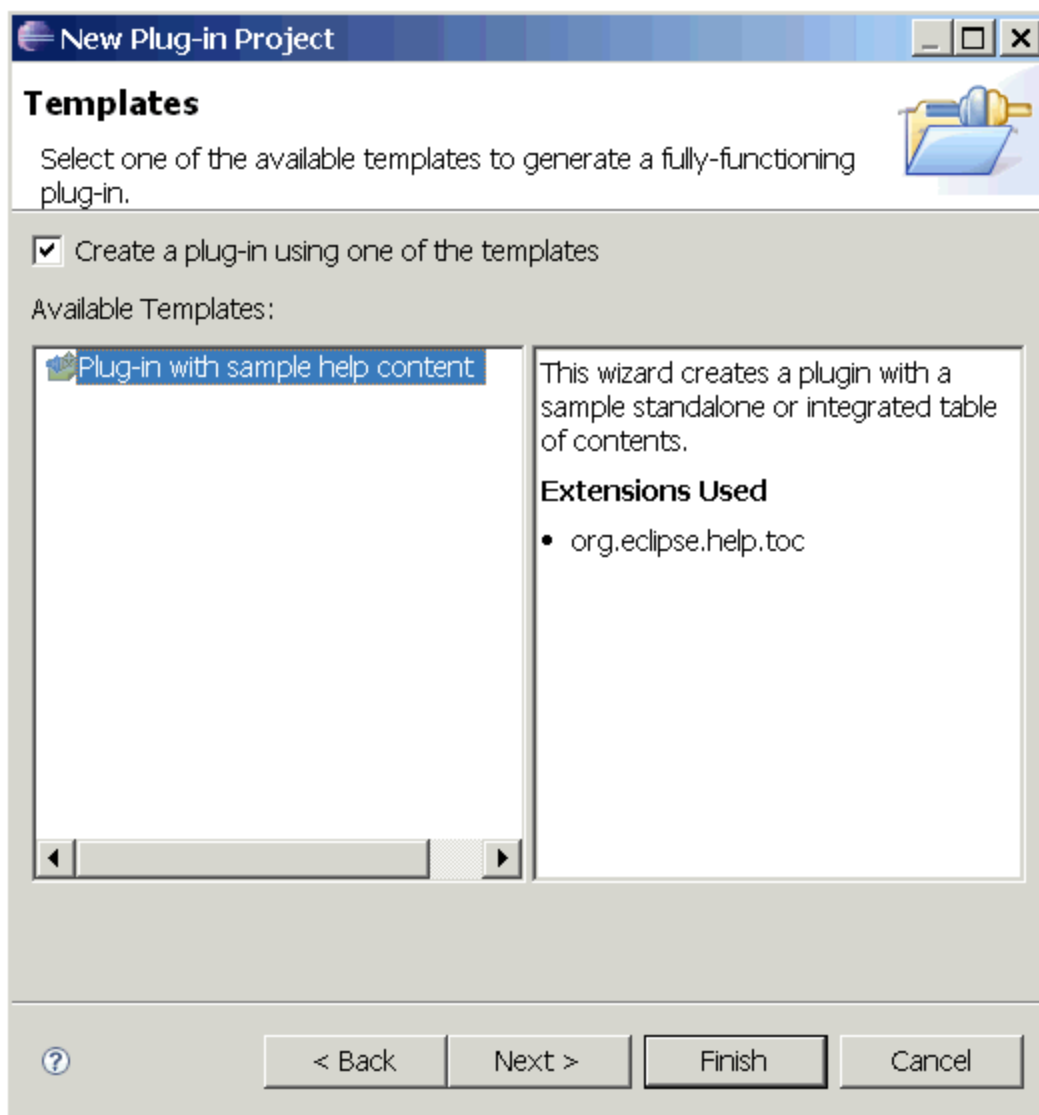
Output folder:

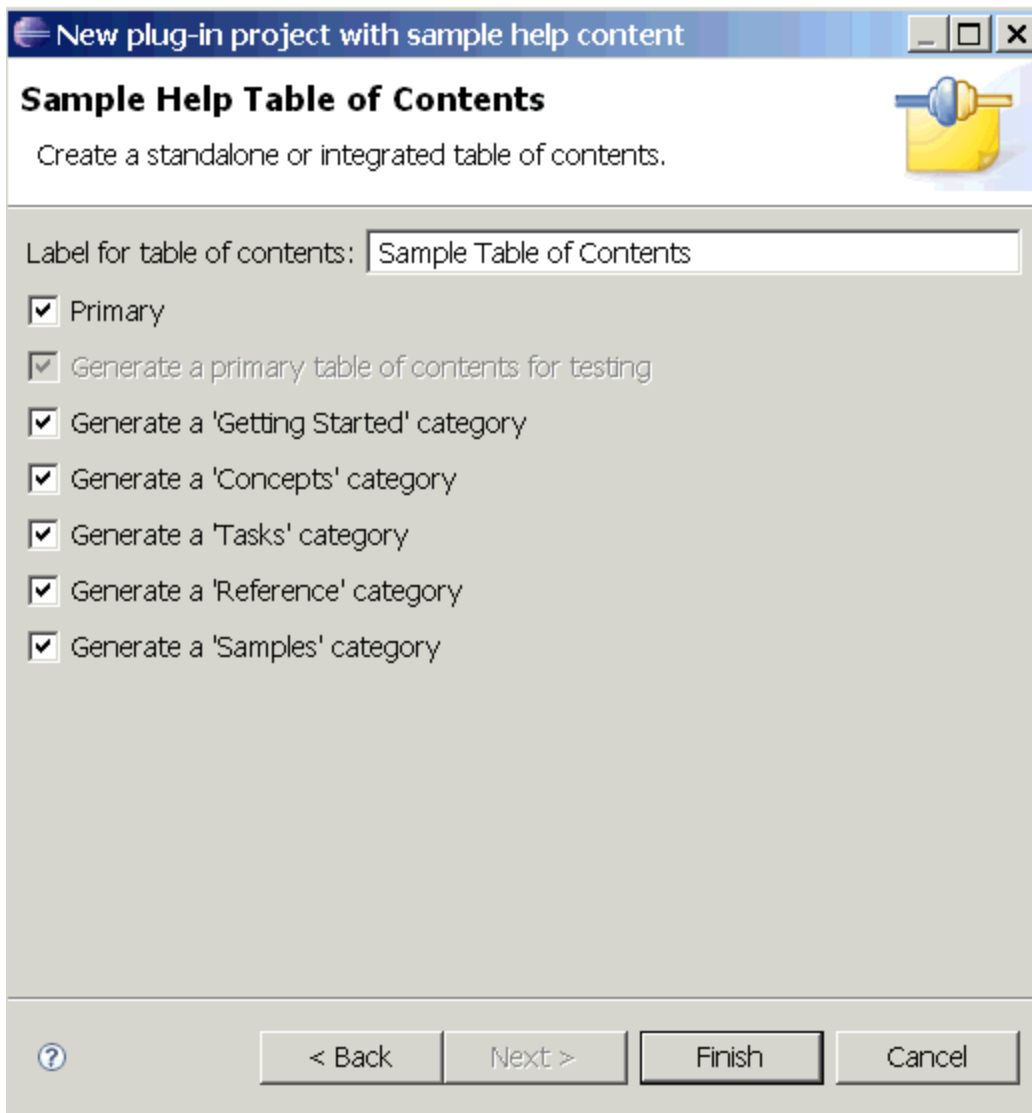
**Target Platform**

This plug-in is targeted to run with:

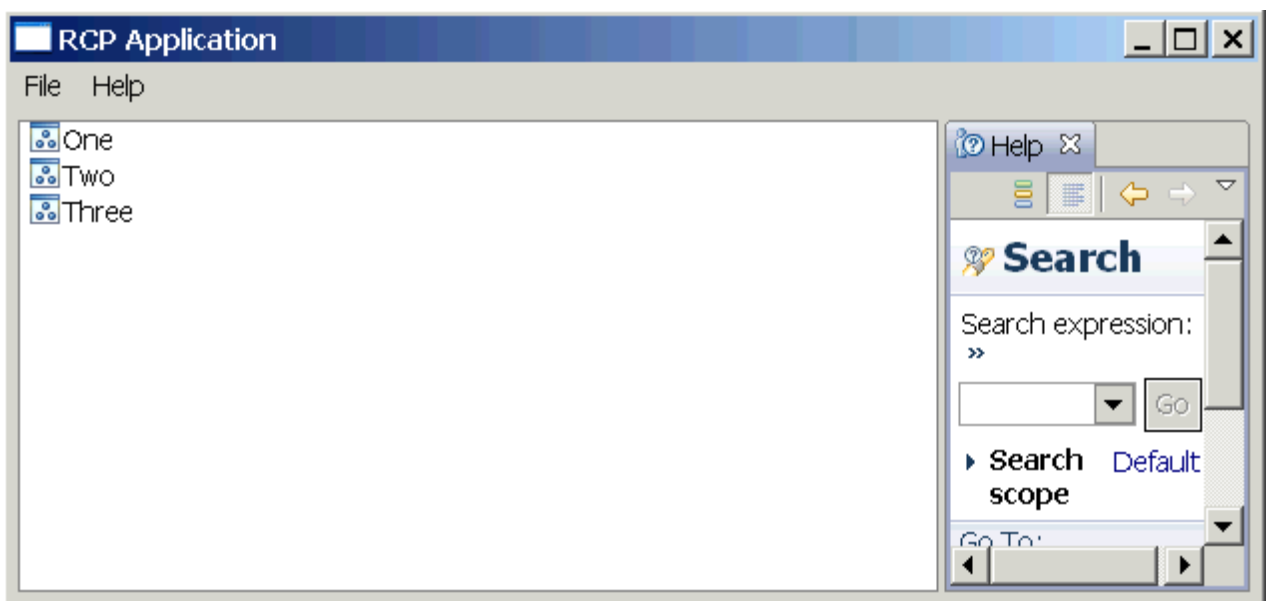
☒ Eclipse version:

☐ an OSGi framework:





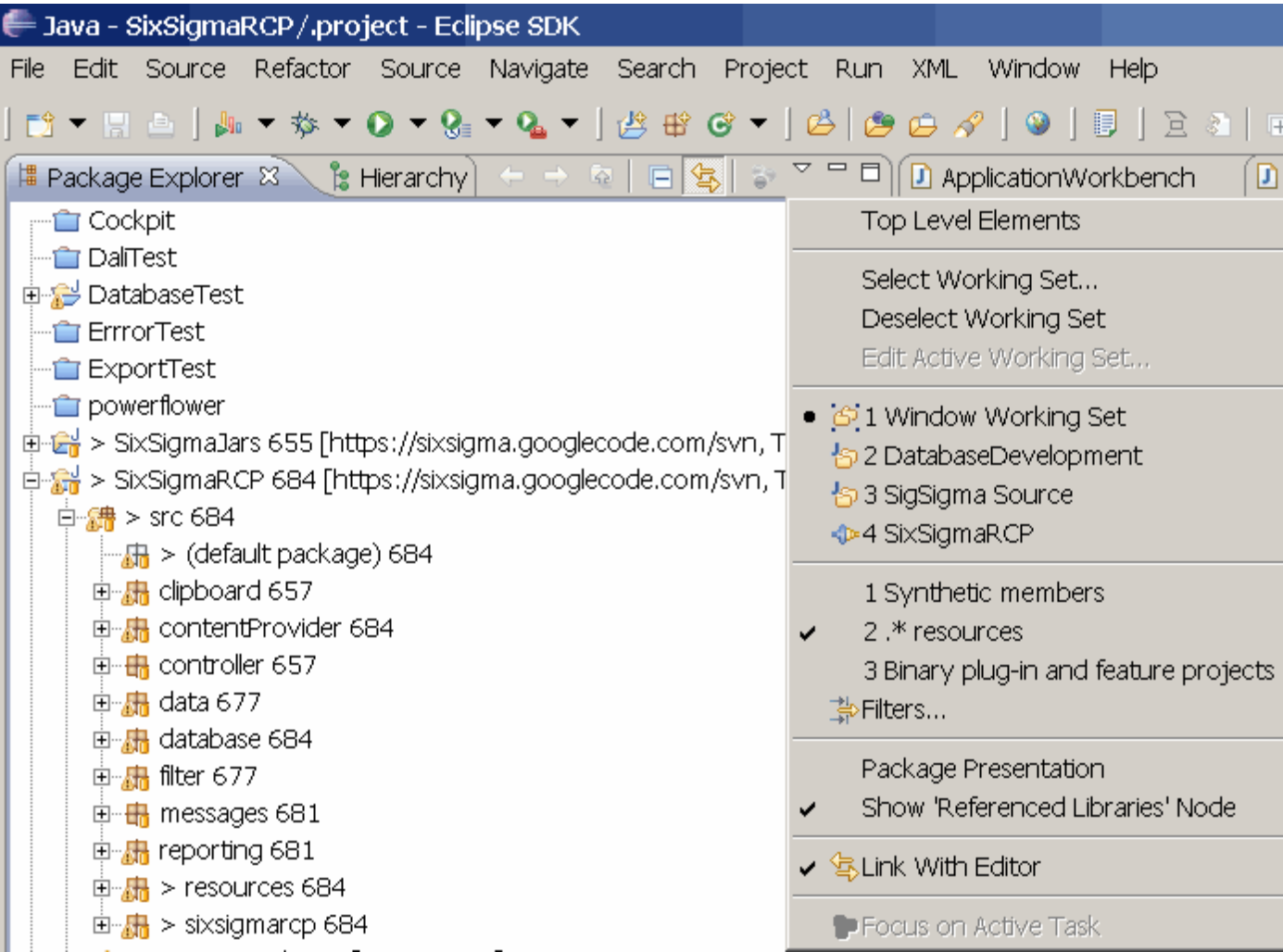
运行程序，点击 help 按钮就可以工作了

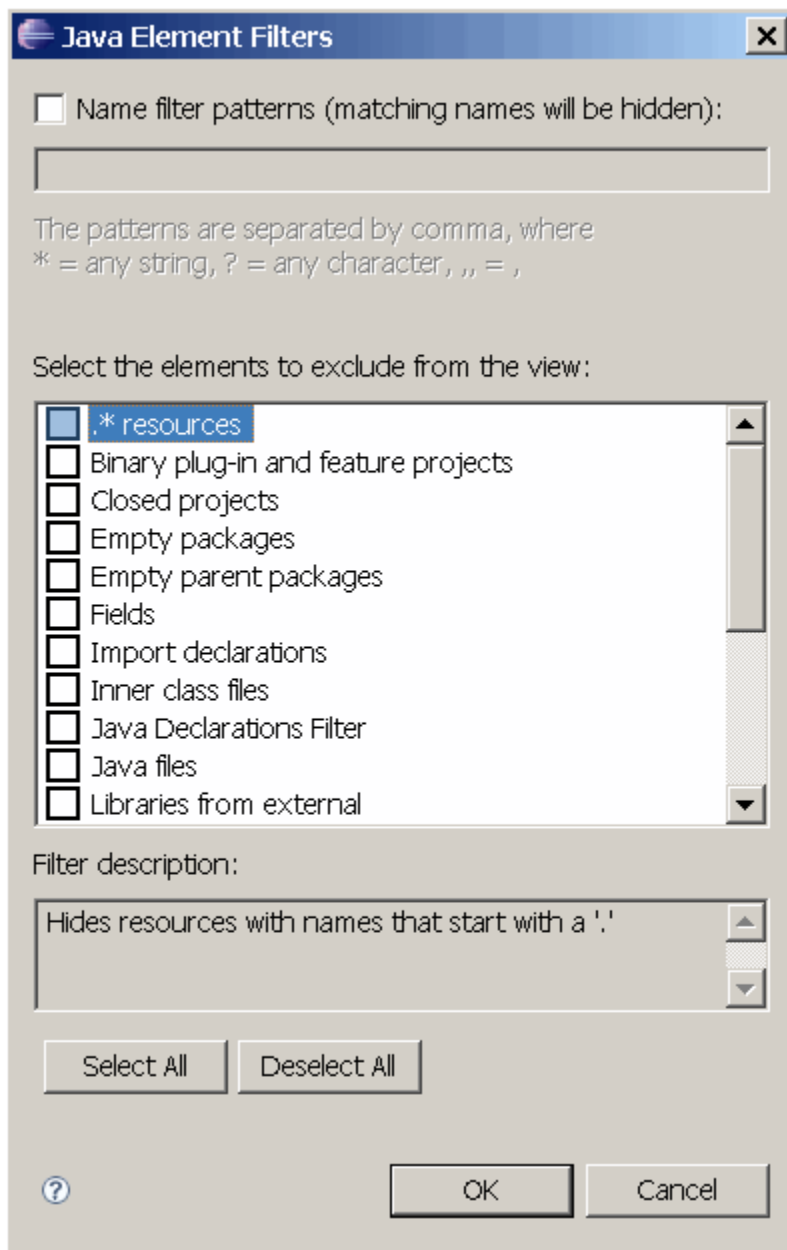


## 第21章 配置文件

### 21.1概述

一些设置文件可能被过滤器隐藏了。在包浏览器中选择 Filter 移动.\*resources 过滤器。





插件清单文件将所有的代码和资源连接起来。并且分为两个文件“MANIFEST.MF”和“plugin.xml”

开发环境为这两个文件提供了一个编辑器

## 21.2.project

.project 包含了对你工程再创造的描述，例如，导入到其他工作空间时。他包括一个描述工程性质的 XML 标签“natures”。

一个插件工程有属性“org.eclipse.pde.PluginNature”，一个 java 工程属性“org.eclipse.jdt.core.javanature”。这些标签将操作开发环境的某些行为，例如如果你改变插件工程的依赖性信息，一个具有 PluginNature 属性的工程将更新 java class 路径。



## 21.3Manifest.MF

OSGi 捆绑清单被存储在 MANIFEST.MF 中，OSGi 是指 eclipse 动态装载插件规范  
这个文件不用被手动编辑

## 第22章 使用接口技术

下面将讲述 SWT 和 Jface 的用法

**SWT:**是一套窗口小部件工具箱，提供一种便于移植的并且独立于操作系统的图形 API，需要基于系统固有工具部件。

**JFace:**一种模型 UI 工具包，简化了普通 UI 程序设计的工作

### 22.1 SWT

SWT 提供一种没有附加特点的未加工的 widget

SWT 使用一种程序上的通过层即 JNI (java 本地接口, java native interface) 使用操作系统的图形 API。JNI 层使得 SWT 可以控制系统固有的部件

### 22.2 Jface

Jface 是一种高端用户接口工具箱，使用原始的 SWT 部件提供模型驱动的部件。在一定范围，其功能不是很有效：例如高级编辑器，对话框，向导

窗口：org.eclipse.jface.window 包提供了对窗口的创建和设备管理。值得一提的是 ApplicationWindow 类，他提供了一个高层的应用程序窗口，并且封装了 SWT 事件循环。

视口：org.eclipse.jface.viewers 包提供了一个视口框架，例如，树型浏览器，表格型浏览器，他是由 SWT 部件构成的模型元件，

对话框：org.eclipse.jface.dialogs 包提供几种常用的对话框

Actions：org.eclipse.jface.actions 包提供一个 UI action 框架，与 Swing 的 action 框架非常相似 in order to implement shared behavior between two or more user interface components, such as a menu item and toolbar button.

向导：org.eclipse.jface.wizard 包提供创造向导的高级框架。（这个熟悉的对话框使重复复杂的工作变为自动化）。

文本：org.eclipse.jface.text 包及其子包提供了一个设计、操作、浏览、编辑文本的框架

资源：org.eclipse.jface.resource 包提供对资源管理的支持，例如 SWT 的字体或图形