
SALEVER@126.COM

Eclipse Rich Client Application 开发自学教程

For The Eclipser

salever

2011-2-17

根据最新版本的 Eclipse 3.6 重新编写，每章都可独立于其他章节，内附的代码均可直接运行，适合 Eclipse 开发者参考。

序

Eclipse RCP 允许开发者使用 eclipse 结构风格设计弹性的可扩展的应用程序，可重用 Eclipse 中已存在的方法和编码模式。俗话说，万事开头难。Eclipse RCP 入手可能会比较困难、费时。接下来我们将主要讲述如何让 RCP 工作。

基于 Eclipse 3.6。

将涉及以下内容：创建第一个 RCP 程序，创建菜单和工具栏，查看，编辑，对话，外部 JAR 的用法，向一个 RCP 应用程序产品中添加标志和帮助。

每一章都基本独立于其他章节。

目录

1 富客户端平台.....	9
1.1 概述.....	9
1.2 ECLIPSE RCP 建设风格——插件，扩展和扩展点.....	9
1.3 RCP与PLUGIN.....	9
2 创建第一个RCP程序.....	11
2.1 创建一个RCP程序.....	11
2.2 启动RCP程序.....	15
2.3 程序VS 产品.....	17
2.4 维护LAUNCH配置.....	17
2.5 可能的APPLICATION ID 错误:.....	20
2.6 应用程序的PLUGIN ID.....	21
3 ACTIONS的用法（菜单栏和工具栏）.....	22
3.1 概述.....	22
3.2 通过编码添加.....	22
3.3 “扩展”方式添加菜单和工具栏.....	25
3.4 添加全局快捷键.....	31
3.4.1 Command+Handler+Binding 绑定.....	31
3.4.2 Action+Command+Binding绑定.....	33
4 系统托盘.....	37
5 视图.....	42
5.1 添加示例视图.....	42
5.2 添加自定义视图.....	47
5.3 向VIEW里添加ACTION.....	52

6 编辑器.....	58
6.1 概述.....	58
6.2 创建工程.....	58
6.3 添加编辑器.....	59
6.4 调用编辑器.....	67
6.5 实例：文本编辑器实现.....	68
6.5.1 新建工程.....	68
6.5.2 添加菜单栏和工具栏.....	70
6.5.3 添加编辑器.....	75
7 对话框.....	86
7.1 概述.....	86
7.2 预定义的对话框.....	87
7.2.1 概述.....	87
7.2.2 创建工程.....	87
7.2.3 添加菜单.....	87
7.2.4 调用对话框.....	87
7.3 用户自定义对话框.....	89
7.3.1 概述.....	89
7.3.2 创建SWT/JFace工程.....	89
7.3.3 自定义Dialog.....	91
7.3.4 说明.....	97
8 向导.....	98
8.1 概述.....	98
8.2 示例.....	98
9 首选项.....	106

9.1 首选项	106
9.2 使用首选项	106
9.3 首选项页	112
10 添加状态栏.....	117
10.1 简介	117
10.2 安装状态栏	117
10.3 初始化状态条	118
10.4 控制状态栏	120
11 透视图.....	123
11.1 简介	123
11.2 添加透视图	123
11.3 显示透视图工具栏	128
11.4 显示透视图菜单	130
12 进度条.....	133
12.1 简介	133
12.2 进度条对话框	133
12.3 JOB进度条.....	135
13 使用第三方JAR.....	140
13.1 概述	140
13.2 向构建路径中添加JAR	140
13.3 使JAR在你的运行路径里有效.....	141
14 提示和策略.....	143
14.1 控制台日志	143
14.2 保存用户的布局	145

14.3 获得DISPLAY	146
14.4 使用ECLIPSE的“保存”ACTION	146
14.5 向你的程序添加错误日志视图	146
15 产品.....	149
15.1 概述	149
15.2 PRODUCT配置文件.....	149
15.3 测试你的产品	153
15.4 欢迎页面	153
15.5 商标	154
15.6 风格化LAUNCHING.....	155
15.7 发布你的产品	156
15.8 创建一个帮助插件工程	159
16 专题一 ECLIPSE的版本和发行包.....	164
16.1 版本 VERSION	164
16.1.1 版本的理解.....	164
16.1.2 Eclipse的版本	164
16.1.3 版本的选择和下载.....	164
16.2 发行包 EDITION	166
16.2.1 发行包的理解.....	166
16.2.2 Eclipse的发行包	166
17 专题二 ECLIPSE的国际化与语言包.....	169
17.1 国际化	169
17.1.1 Externalize Strings	169
17.1.2 中文属性文件.....	175
17.1.3 国际化文件.....	176
17.1.4 指定语言环境.....	176

17.1.5 Propedit工具	177
17.2 语言包	177
17.2.1 Babel小组	177
17.2.2 中文语言包的下载.....	178
18 专题三 DECORATOR与MARKER的使用.....	180
18.1 简介	180
18.2 扩展DECORATOR.....	180
18.3 扩展MARKER.....	184
19 专题四 RUN/DEBUG LAUNCHER实现.....	190
19.1 ECLIPSE RUN/DEBUG 实现流程.....	190
19.2 扩展CONFIGURATIONTYPE	192
19.3 扩展ILAUNCHSHORTCUT	194
19.4 创建RUN/DEBUG CONFIGURATION界面	197
19.5 指定RUN/DEBUG显示图片	204
19.6 说明	205
20 专题五 EQUINOX P2 方式实现RCP自动更新.....	207
20.1 概述	207
20.2 示例	207
20.2.1 Feature概念.....	207
20.2.2 配置Product	210
20.2.3 配置Feature.....	211
20.2.4 product导出	213
20.2.5 配置Equinox P2.....	214
20.2.6 配置 Update Site.....	215
21 专题六 COMMON NAVIGATOR FRAMEWORK初探.....	216

1 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 (Rich Client Application) 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；
- 所有的插件必须拥有一个名为“`plugin.xml`”的配置文件。
- 同时还需要一些核心插件，例如 `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 (Plugin) 是 Eclipse 最小的可开发可安装元件。一个插件可以使用扩展 (Extension)，例如，向扩展点 (Extension Point) 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 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。

1.3 RCP 与 Plugin

很多朋友在一开始会对这两者比较迷茫，弄不清楚它们是什么关系，有什么区别，这里也简单讲一讲。

RCP, Rich Client Application, 富客户端程序，首先它是一个程序。而 plugin, 插件，指的是 RCP 环境中的最小可安装原件，事实上，就是许多 Plugin 组成了 RCP, Eclipse 本身就是一个完美的 RCP, 而它包含的 Resource、JDT、CVS、Team 等 Plugin 则各自负责完成它的各种功能。

所以我们常常说的 RCP 开发和 Plugin 开发，其实就是一个意思，无论做哪一种，Plugin 开发

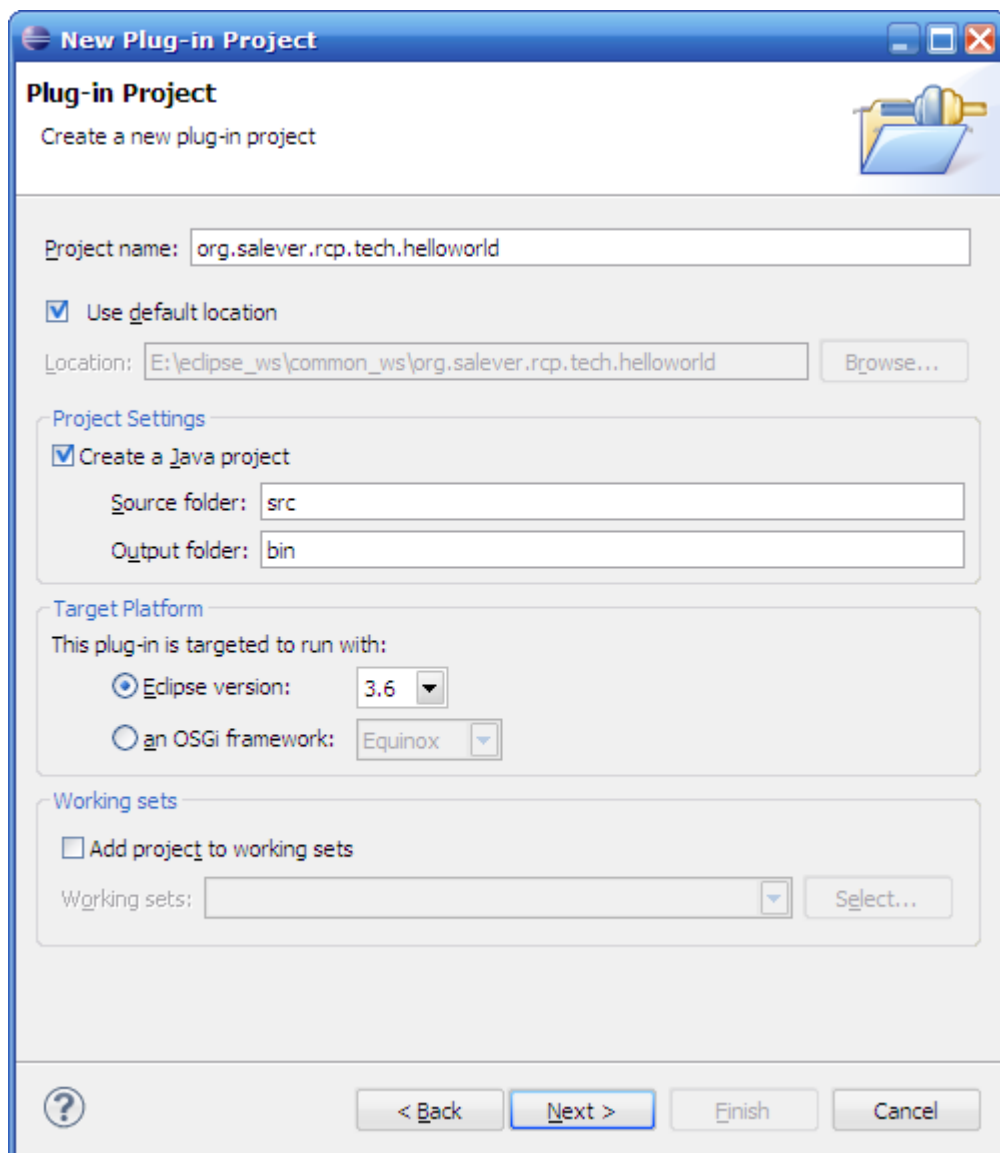
都是基础。

2 创建第一个 RCP 程序

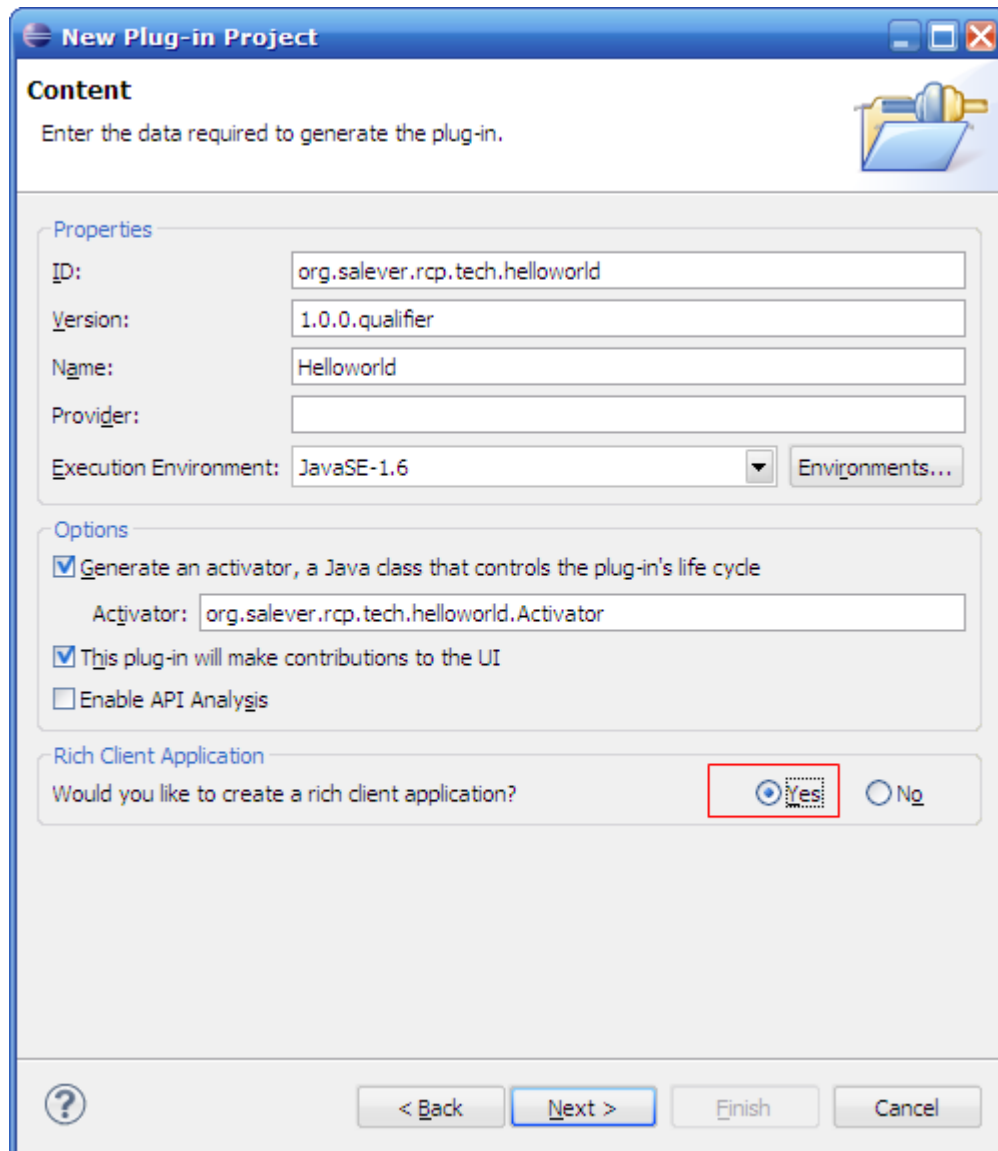
接下来给出快速指南帮助你创建一个简单的 RCP 程序。

2.1 创建一个 RCP 程序

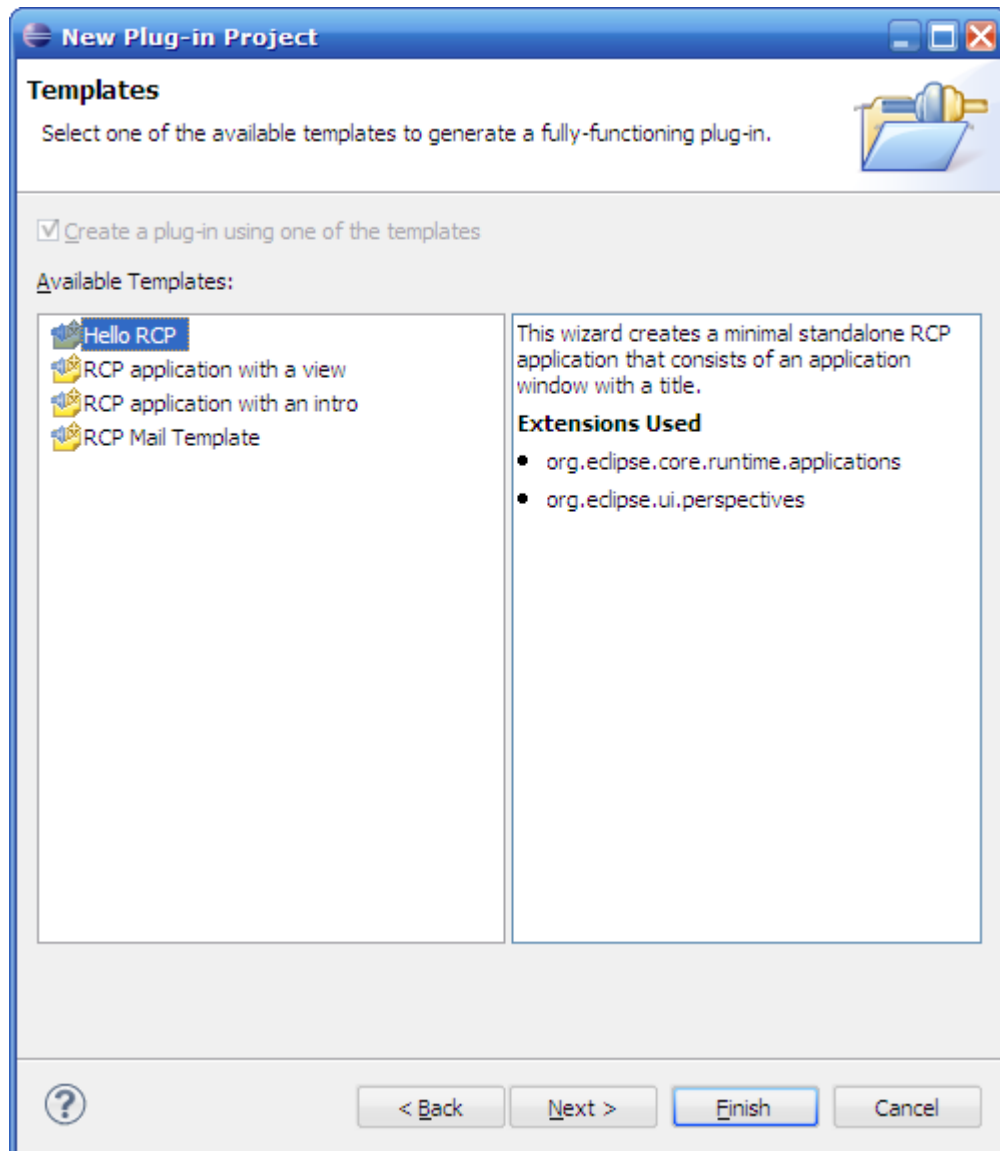
在 Eclipse 里 选择 File-> New Project, 在列表中选择 Plug-in Project,
接下来: 给你的 RCP 工程命名, 例如: “org.salever.rcp.tech.helloworld”



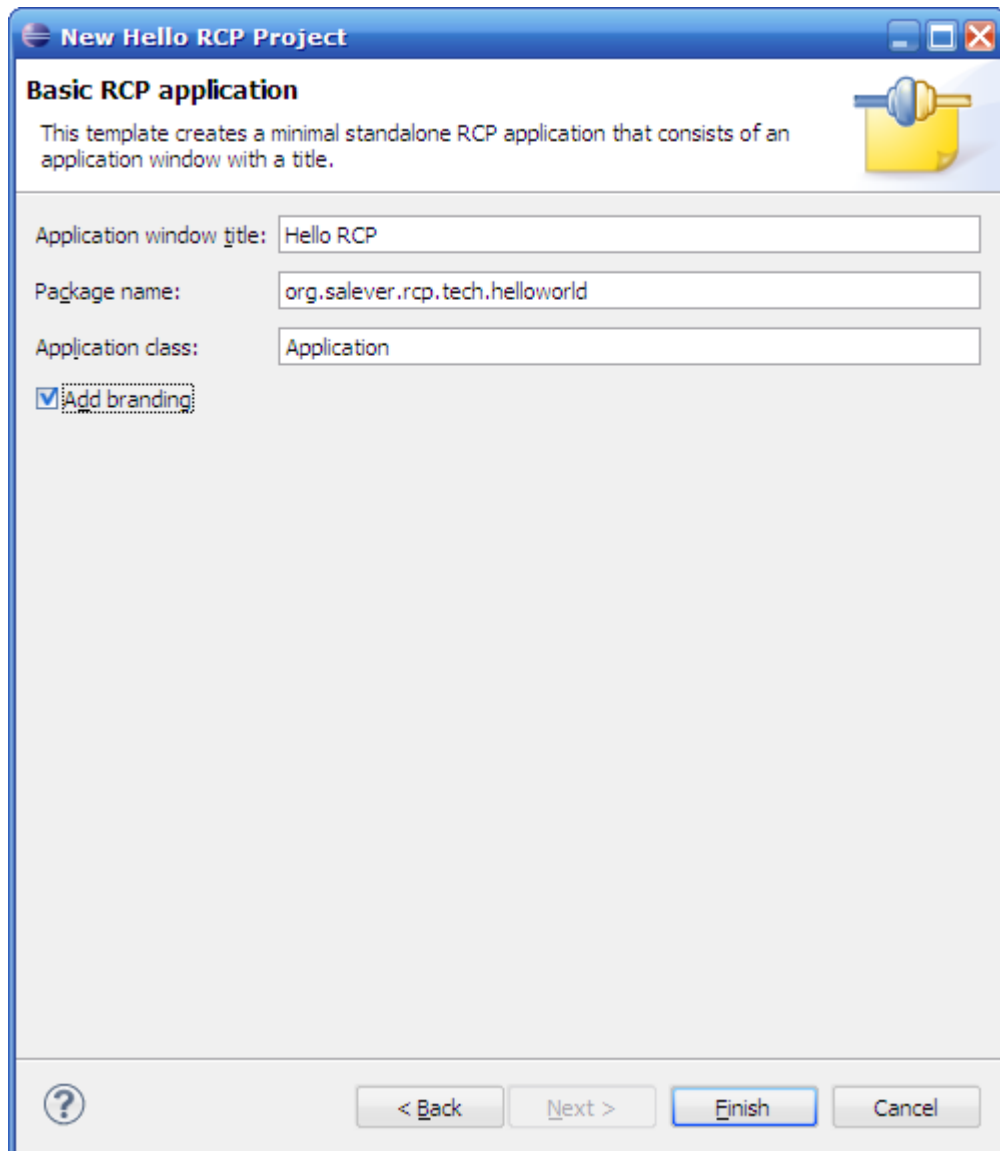
Next, 注意选择 Rich Client Application 一栏中选择“Yes”,



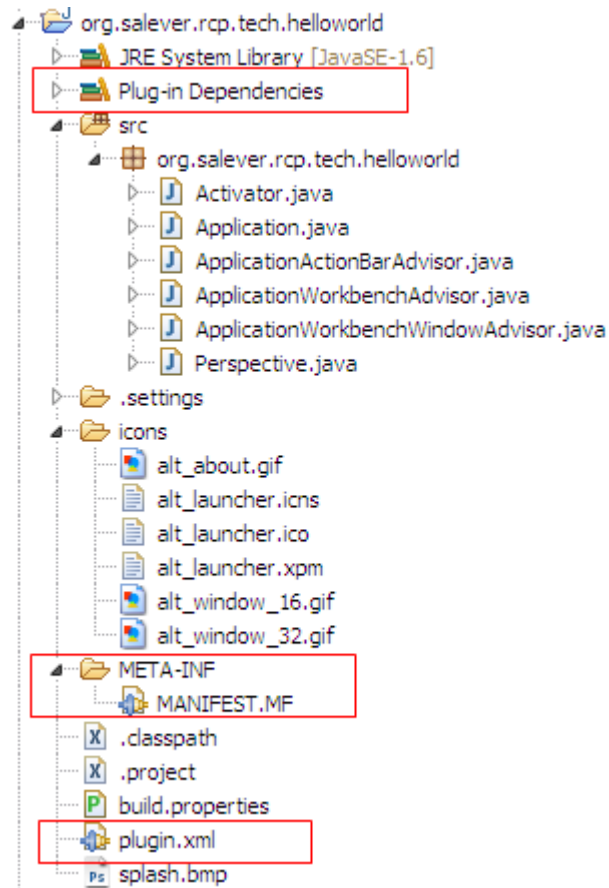
选择“Hello RCP”模版



在下一屏选择“Add branding”，点击“Finish”

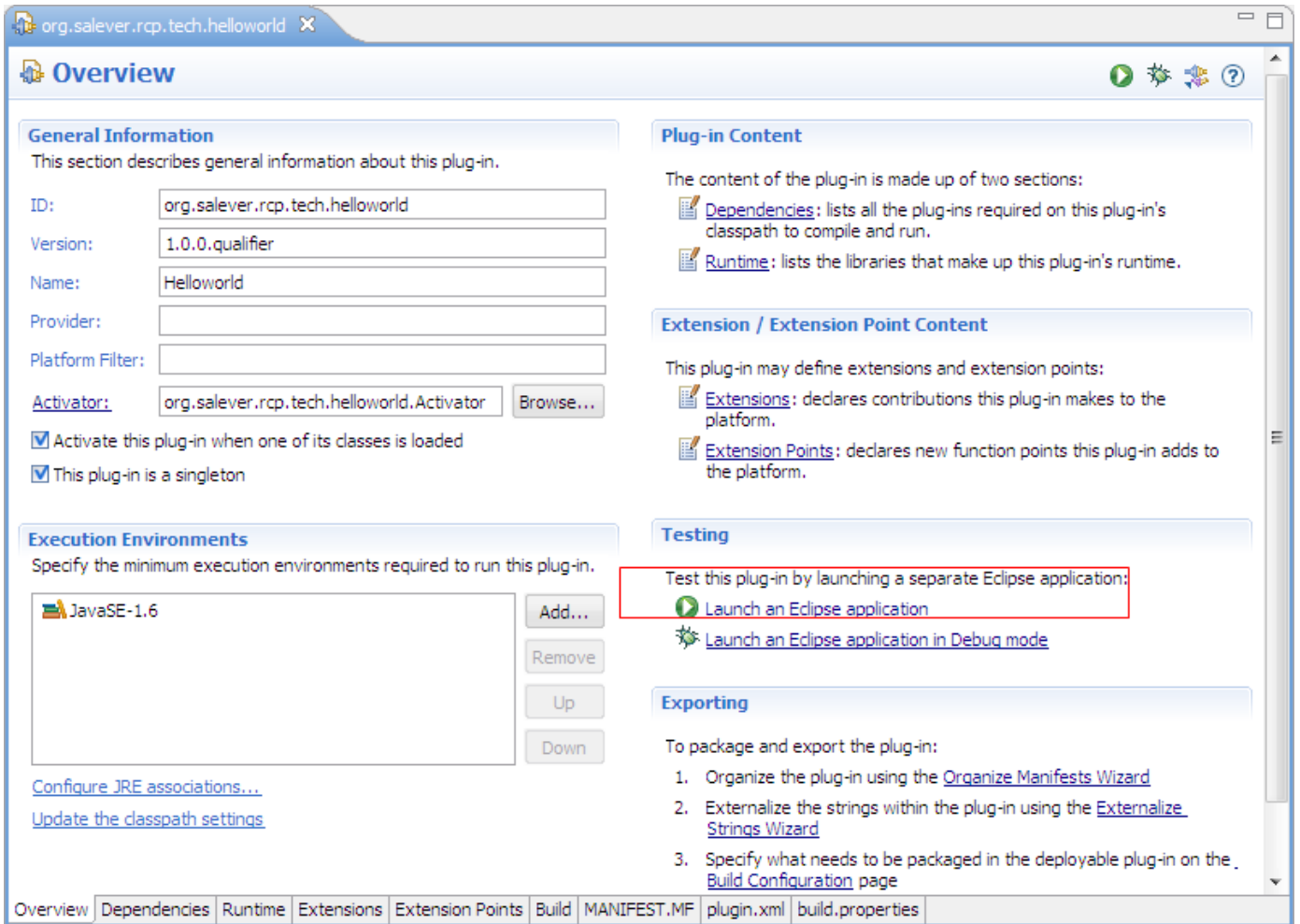


工程“org.salever.rcp.tech.helloworld”创建完毕，将会切换到 Plugin Development Perspective 透视图下。在 Package Explorer 视图下，看一看这些不同文件，找一下对工程结构的第一感觉。Plugin 工程与一般的 Java 工程的区别为，拥有 META-INF 文件夹，里面有 MANIFEST.MF 文件，以及 plugin.xml（可能没有）。

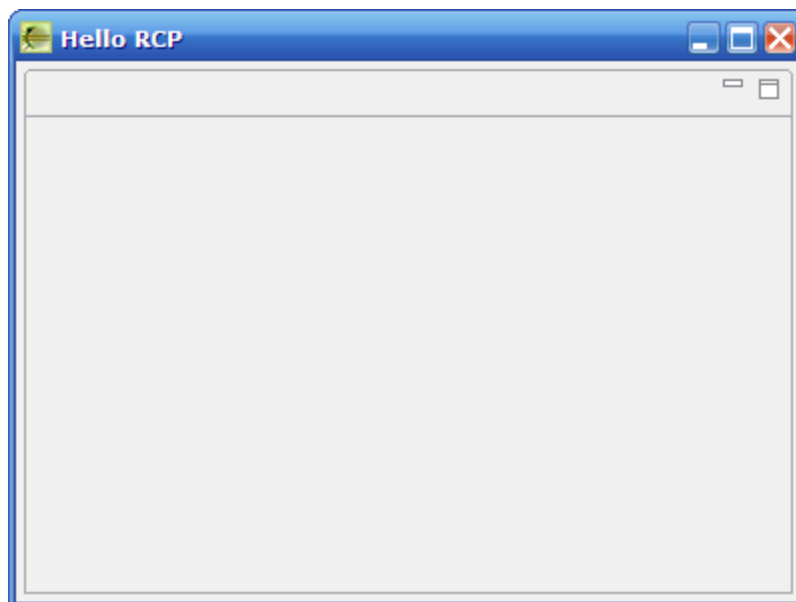


2.2 启动 RCP 程序

找到 `plugin.xml`，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”



可以看到如下结果：



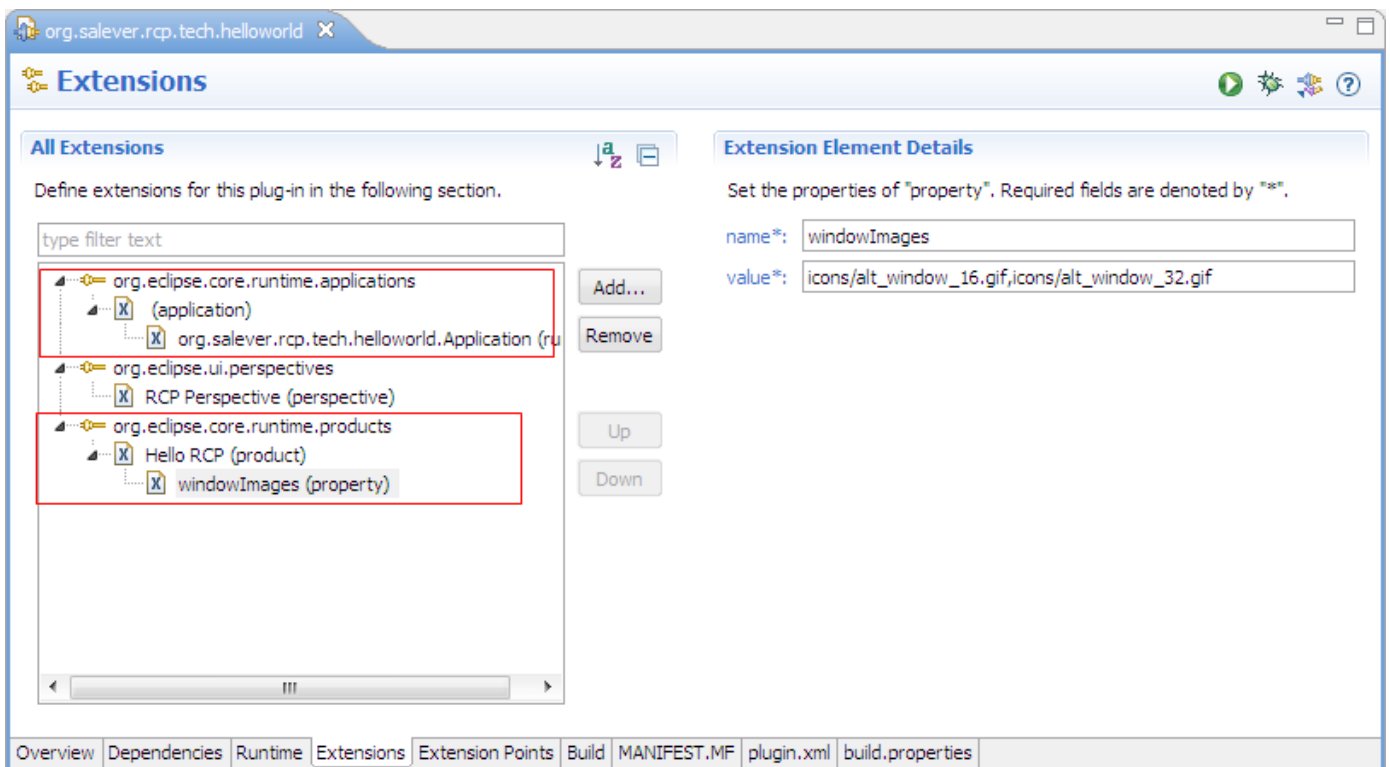
恭喜你，你已经创建了你的第一个 RCP 程序了

2.3 程序 VS 产品

程序（Application）：Plugin 工程可以以 Eclipse Application 的方式运行，一个 RCP 必须拥有一个 Application，否则无法运行。

产品（Product）：RCP 程序以 Product 的方式打包，然后单独运行。

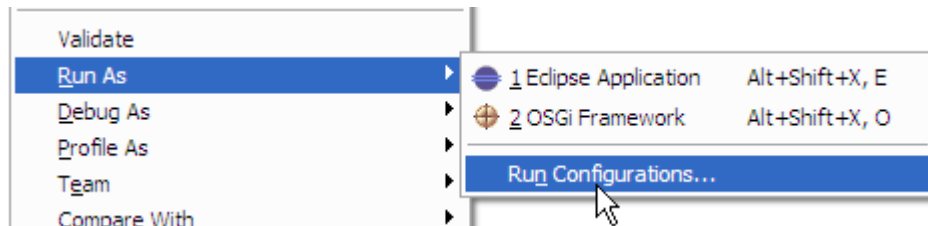
在我们刚刚新建的工程“org.salever.rcp.tech.helloworld”中，同时进行了 Application 和 Product 的定义，打开 plugin.xml 文件，转到 Extension Tab 页，可以看到 application 和 product 的扩展：



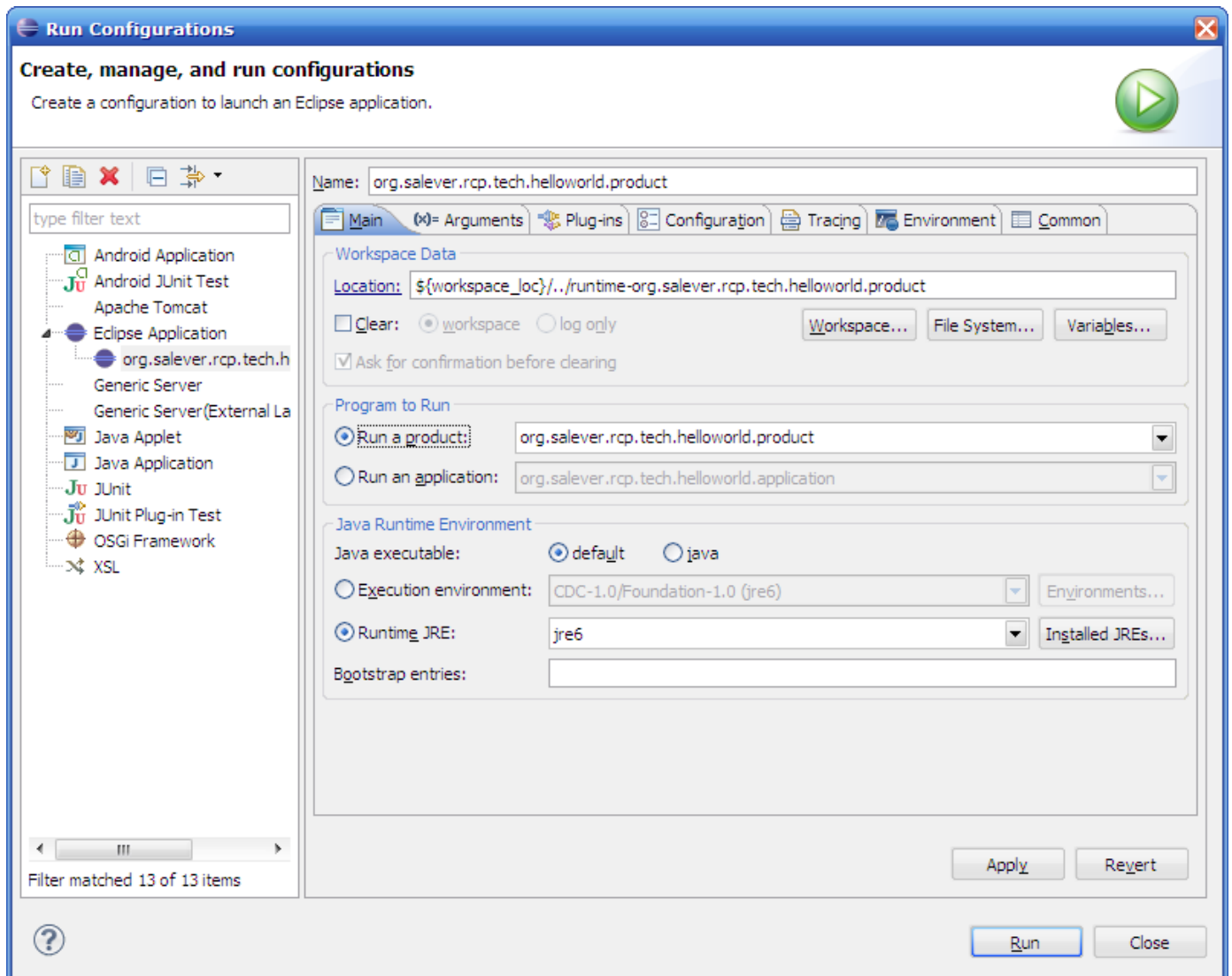
Product 中可以定义欢迎界面、窗口图标、程序图标等。关于 Product 的使用和导出，将在后面的章节详细讲解。

2.4 维护 Launch 配置

选择 plugin.xml->Run as->Run Configuration



这个例子里，我们将运行产品。试着也运行应用程序。



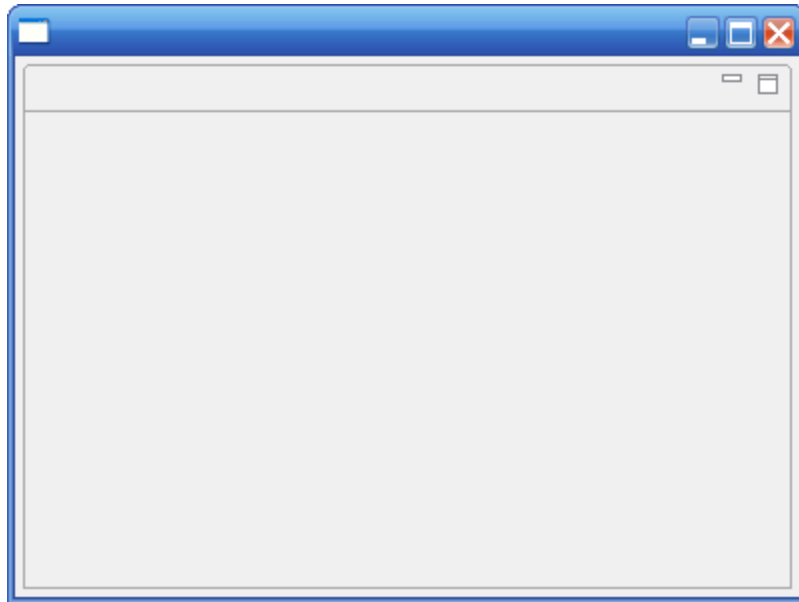
这里有几项重要设置：

- 首先，Workspace Data，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。

- 第三: Program to run, 你可以选择运行一个 Application, 或者运行一个 Product, 然后查看它们之间的异同。

运行 Application:



运行 Product:



可以发现, Application 运行的时候没有图标。

2.5 可能的 Application Id 错误:

有时候运行会出现如下错误，尤其是切换 Application 和 Product 的时候:

```
ENTRY org.eclipse.osgi 4 0 2011-02-15 13:41:38.328
```

```
!MESSAGE Application error
```

```
!STACK 1
```

[java.lang.RuntimeException](#): Application "org.salever.rcp.tech.helloworld.application" could not be found in the registry. The applications available are: org.eclipse.equinox.app.error.

```
at org.eclipse.equinox.internal.app.EclipseAppContainer.startDefaultApp(EclipseAppContainer.java:248)
```

```
at org.eclipse.equinox.internal.app.MainApplicationLauncher.run(MainApplicationLauncher.java:29)
```

```
at org.eclipse.core.runtime.internal.adaptor.EclipseAppLauncher.runApplication(EclipseAppLauncher.java:110)
```

```
at org.eclipse.core.runtime.internal.adaptor.EclipseAppLauncher.start(EclipseAppLauncher.java:79)
```

```
at org.eclipse.core.runtime.adaptor.EclipseStarter.run(EclipseStarter.java:369)
```

```
at org.eclipse.core.runtime.adaptor.EclipseStarter.run(EclipseStarter.java:179)
```

```
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

```
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
```

```
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
```

```
at java.lang.reflect.Method.invoke(Unknown Source)
```

```
at org.eclipse.equinox.launcher.Main.invokeFramework(Main.java:619)
```

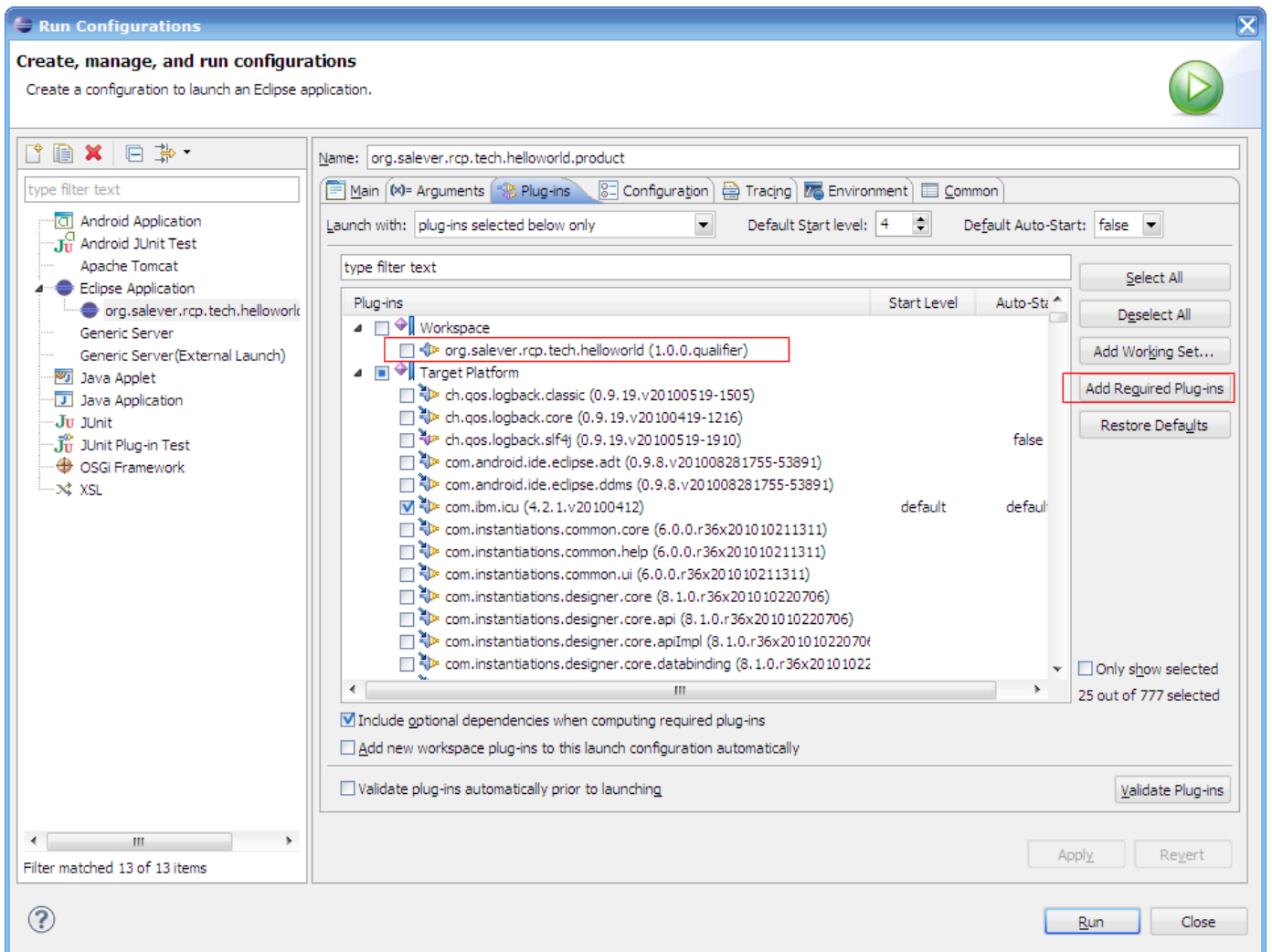
```
at org.eclipse.equinox.launcher.Main.basicRun(Main.java:574)
```

```
at org.eclipse.equinox.launcher.Main.run(Main.java:1407)
```

```
at org.eclipse.equinox.launcher.Main.main(Main.java:1383)
```

这是初学者最容易遇见的错误之一，往往也不知道从何着手去解决，这里讲解一下。

在 Plugin 运行中，需要一些依赖的 Plugin，如果这些 Plugin 缺失或者未正确引用，就会出现这个问题。解决方法为：打开 plugin.xml->Run as->Run Configuration，转到 Plugins Tab 页，检查一下，发现 workspace 下的“org.salever.rcp.tech.helloworld”没有选择，问题就在这里了，我们运行的就是这个插件，但是它没有加载，所以就出错了，勾选，然后别忘了点击一下右侧的 Add Required Plugins 按钮，确定一下，问题解决。



2.6 应用程序的 Plugin 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“org.salever.rcp.tech.helloworld”，可以在 Activator.java 里添加以下声明

```
public static final String PLUGIN_ID = "org.salever.rcp.tech.helloworld";
```

工程源码见附件。

3 Actions 的用法（菜单栏和工具栏）

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

1. 编写代码
2. 扩展（Extensions）

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

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 通过编码添加

创建新工程“org.salever.rcp.tech.chapter3”使用“Hello RCP”模板

打开 ApplicationActionBarAdvisor.java 做如下更改

```
package org.salever.rcp.tech.chapter3;

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



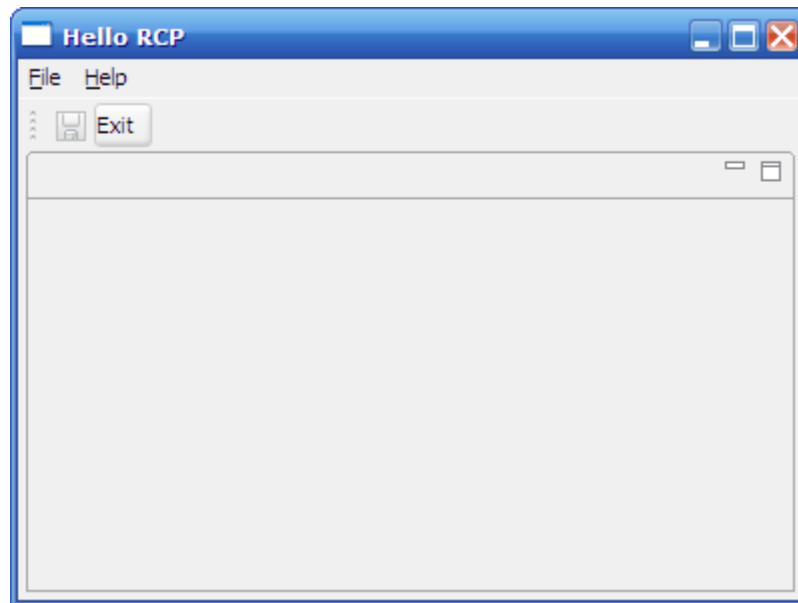
现在，通过方法 `fillCoolBar()` 方法，向 `ApplicationActionBarAdvisor.java` 添加工具栏，代码如下：
添加代码

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

你必须也得在 `ApplicationWorkbenchWindowAdvisor.java` 里设置显示 `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 “扩展”方式添加菜单和工具栏

创建一个新工程，命名为“org.salever.rcp.tech.chapter3.ex”，使用“Hello RCP”模板。

双击 `ApplicationWorkbenchWindowAdvisor` 类，`ApplicationWorkbenchWindowAdvisor.java` 里设置显示 `coolbar` 属性

```
package org.salever.rcp.tech.chapter3.ex;
import org.eclipse.swt.graphics.Point;
public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

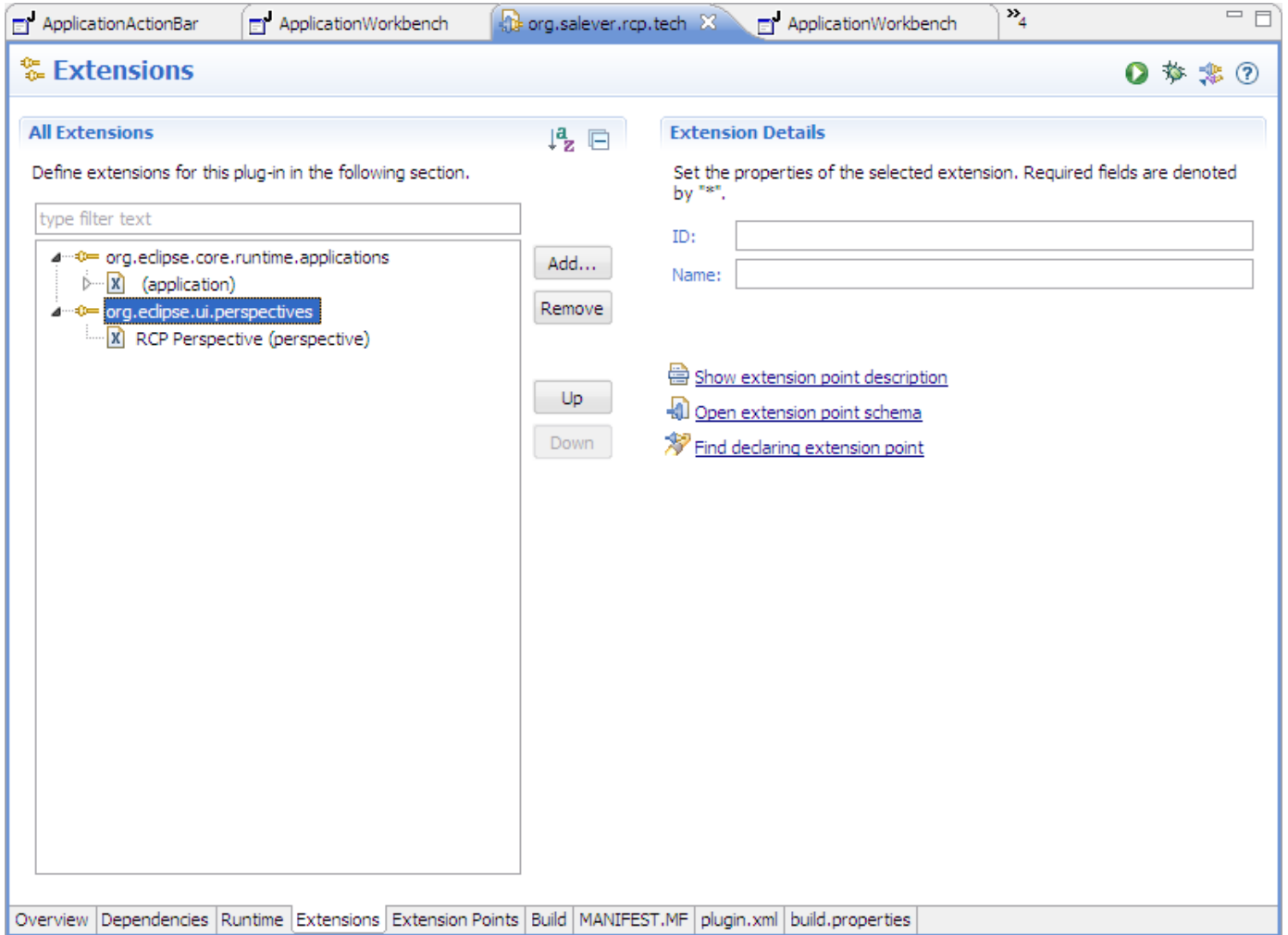
    public ApplicationWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurer) {
        super(configurer);
    }

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

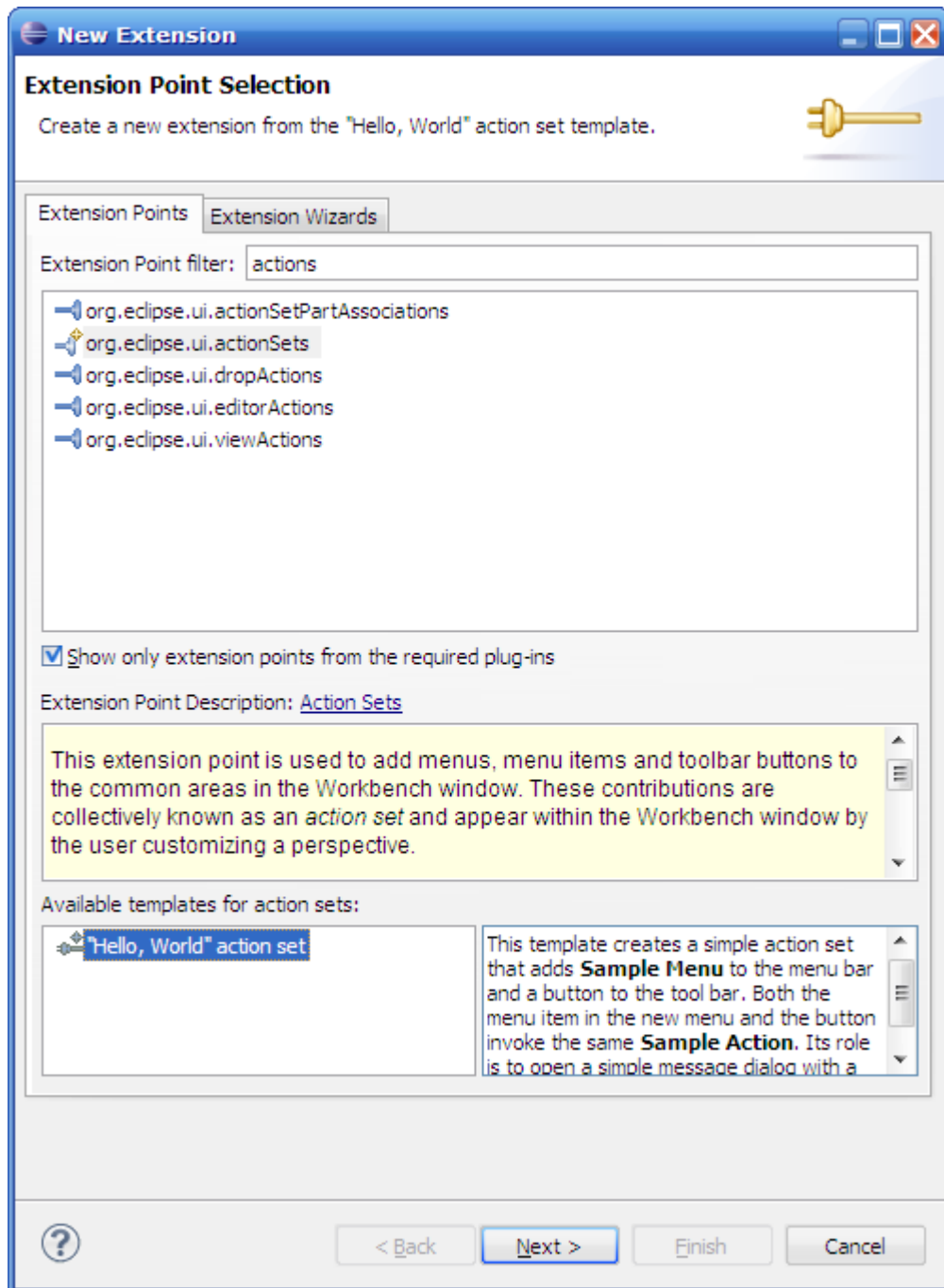
    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowCoolBar(true);
    }
}
```

```
    configurer.setShowStatusLine(false);  
    configurer.setTitle("Hello RCP With Extension Menu"); //$NON-NLS-1$  
  }  
}
```

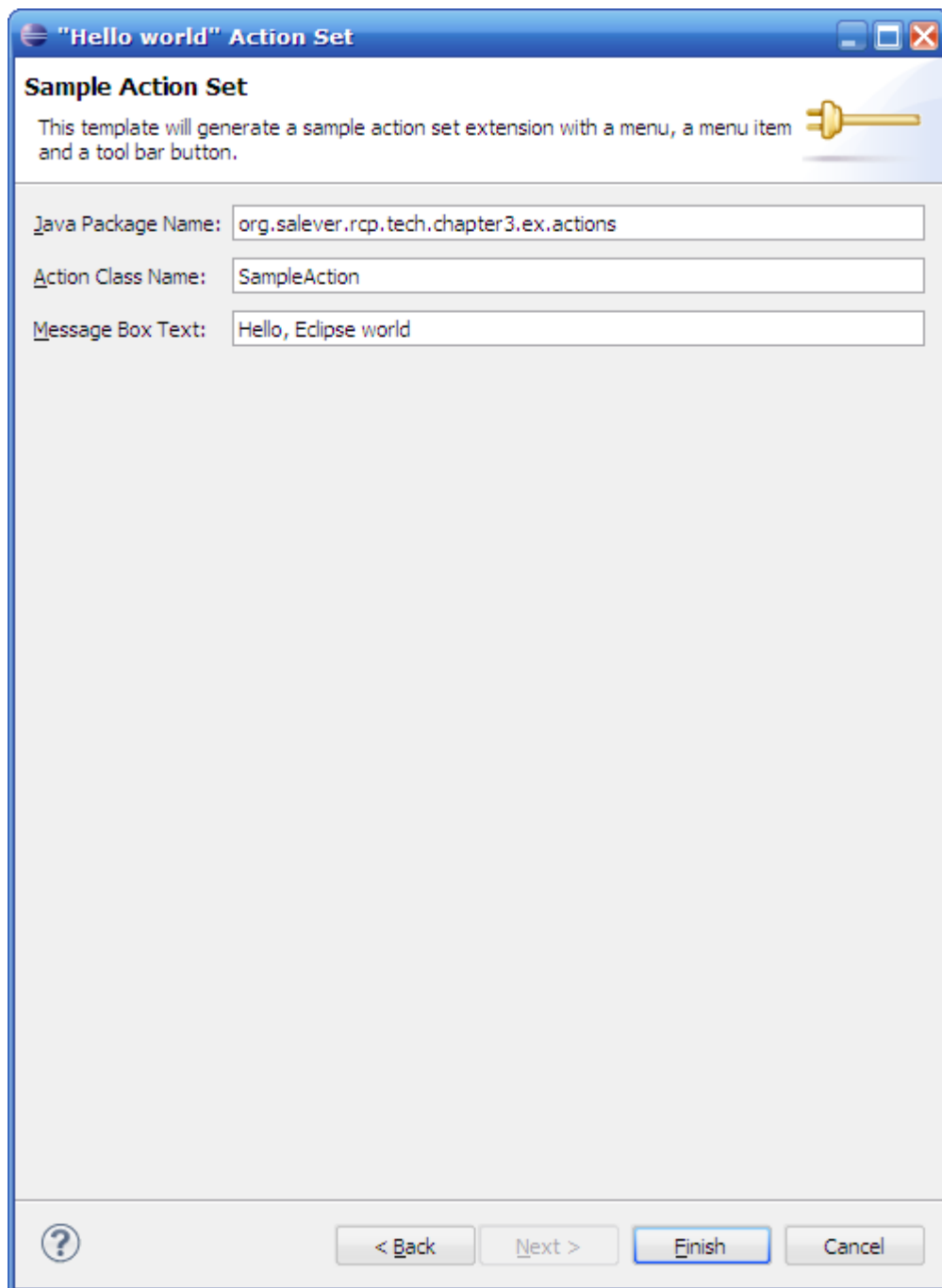
双击 plugin.xml, 转到 Extensions Tab 页,



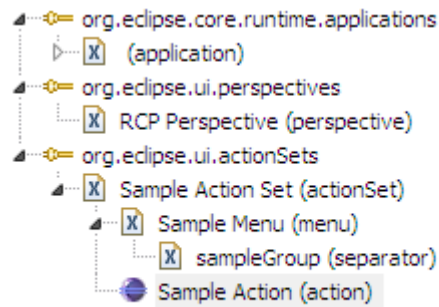
点击 Add...弹出 New Extension 窗口, 选择 org.eclipse.ui.actionSets 扩展点, 使用“Hello, world“ action set。



下一步设置菜单信息，



点击完成，新建的 Action set 如图：

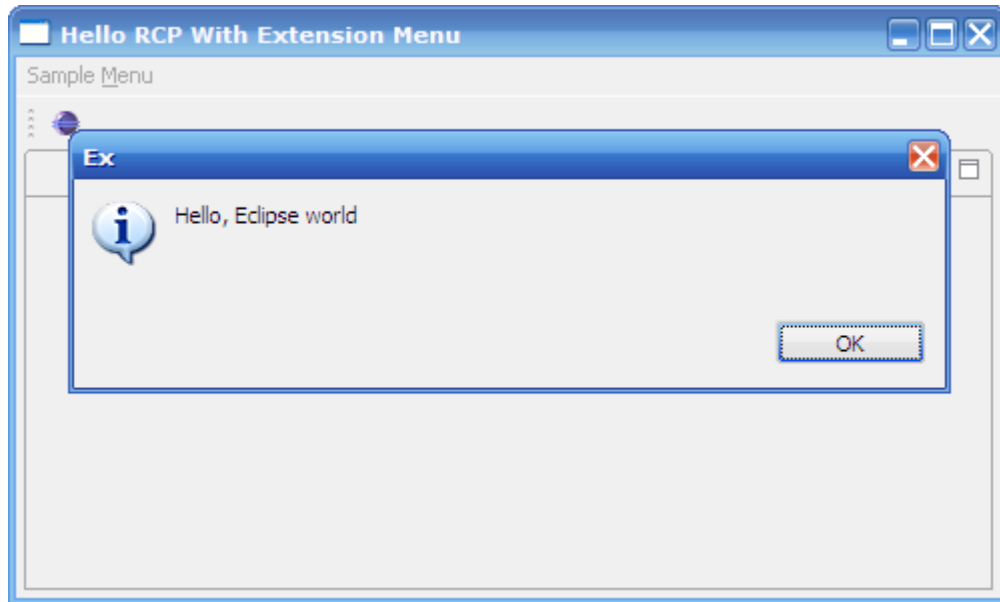


完整的 Extension 扩展为：

```
<extension
    point="org.eclipse.ui.actionSets">
    <actionSet
        id="org.salever.rcp.tech.chapter3.ex.actionSet"
        label="Sample Action Set"
        visible="true">
        <menu
            id="sampleMenu"
            label="Sample &Menu">
            <separator
                name="sampleGroup">
            </separator>
        </menu>
        <action
            class="org.salever.rcp.tech.chapter3.ex.actions.SampleAction"
            icon="icons/sample.gif"
            id="org.salever.rcp.tech.chapter3.ex.actions.SampleAction"
            label="&Sample Action"
            menubarPath="sampleMenu/sampleGroup"
            toolbarPath="sampleGroup"
            tooltip="Hello, Eclipse world">
        </action>
    </actionSet>
</extension>
```

我们发现在 `org.salever.rcp.tech.chapter3.ex.actions` 包下新建了一个新的类，`SampleAction.java`，里面定义了菜单的具体操作。

运行 `org.salever.rcp.tech.chapter3.ex`，可以看见新建的菜单，单击菜单，结果为：



下面讲解一下 Action set 个元素的作用：

- **Menu**，定义菜单栏，也就是菜单显示的位置，示例里面新建了一个 `Sample Menu` 菜单；
- **Action**，定义菜单项，菜单项包含许多属性，其中比较重要的是 `id`、`label`、`menubar path`、`toolbar path`、`class` 等。`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

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

id*:	org.salever.rcp.tech.chapter3.ex.actions.SampleAction
label*:	&Sample Action
accelerator:	
definitionId:	<input type="text"/> Browse...
menubarPath:	sampleMenu/sampleGroup
toolbarPath:	sampleGroup
icon:	icons/sample.gif Browse...
disabledIcon:	<input type="text"/> Browse...
hoverIcon:	<input type="text"/> Browse...
tooltip:	Hello, Eclipse world
helpContextId:	<input type="text"/>
style:	<input type="text"/> ▼
state:	<input type="text"/> ▼
pulldown:	<input type="text"/> ▼
class:	org.salever.rcp.tech.chapter3.ex.actions.S Browse...
retarget:	<input type="text"/> ▼
allowLabelUpdate:	<input type="text"/> ▼
enablesFor:	<input type="text"/>
mode:	<input type="text"/> ▼

3.4 添加全局快捷键

接下来，我们介绍如何向 actions 指定全局快捷键。用户可以通过按键操作菜单，例如，用户可以通过 Ctrl+S 保存应用程序里的数据。

全局快捷键一般为组合键，是以 Command 方式指定给一个 action 的。

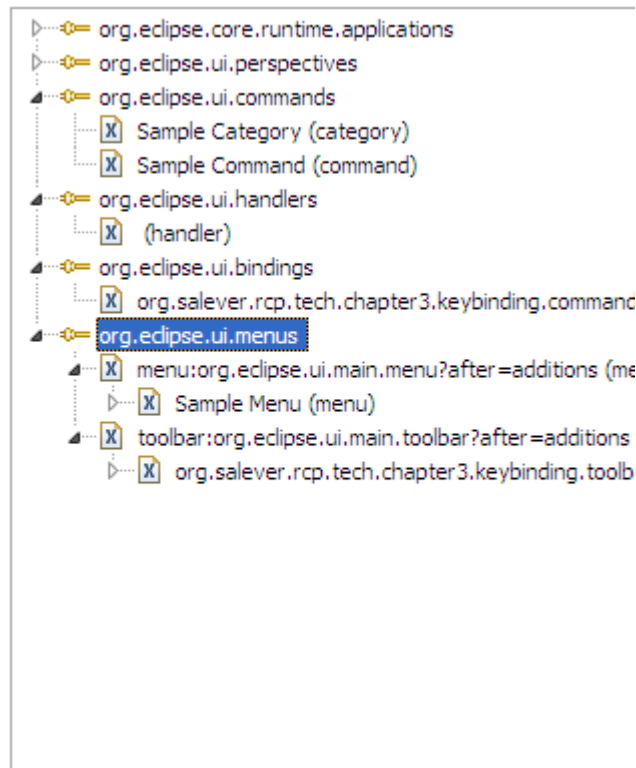
有两种配置全局快捷键的方式，下面为你一一介绍。

3.4.1 Command+Handler+Binding 绑定

首先，创建一个新工程，命名为“org.salever.rcp.tech.chapter3.keybinding”，使用“Hello RCP”模

板。

双击打开 plugin.xml，转到 Extension Tab 页，Add Extension，选择“org.eclipse.ui.commands”扩展点，使用“Hello, world” command contribution。这时候会发现新建了一系列的扩展点，



如果想使用快捷键，必须新建一个 Command，用于对应 Action，这里并没有新建 Action，而是采用了一种新的菜单扩展方法，handler。

3.4.1.1 配置 Command

```
<extension
  point="org.eclipse.ui.commands">
  <category
    id="org.salever.rcp.tech.chapter3.keybinding.commands.category"
    name="Sample Category">
  </category>
  <command
    categoryId="org.salever.rcp.tech.chapter3.keybinding.commands.category"
    id="org.salever.rcp.tech.chapter3.keybinding.commands.sampleCommand"
    name="Sample Command">
  </command>
```



```
</extension>
```

3.4.1.2 配置 Handler

```
<extension
    point="org.eclipse.ui.handlers">
    <handler
        class="org.salever.rcp.tech.chapter3.keybinding.handlers.SampleHandler"
        commandId="org.salever.rcp.tech.chapter3.keybinding.commands.sampleCommand">
    </handler>
</extension>
```

注意 commandId 一定要与要绑定的 Command 一致。

3.4.1.3 快捷键绑定 Binding

```
<extension
    point="org.eclipse.ui.bindings">
    <key
        commandId="org.salever.rcp.tech.chapter3.keybinding.commands.sampleCommand"
        contextId="org.eclipse.ui.contexts.window"
        schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"
        sequence="M1+6">
    </key>
</extension>
```

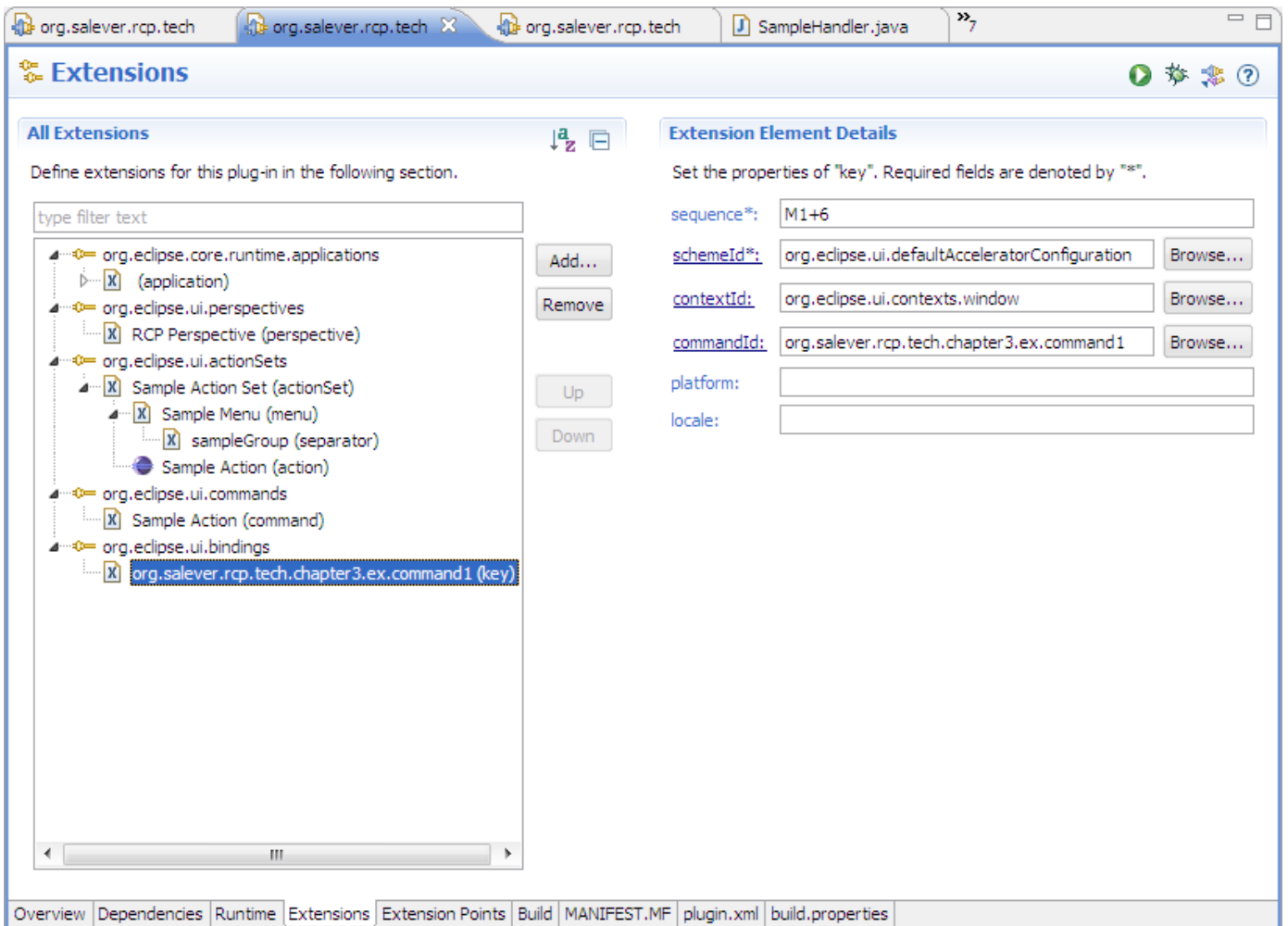
“sequence”是指定组合键的地方，M1 在 windows 系统里代表 Ctrl，详细参考 api 文档，其余的配置默认就行。

注意 commandId 一定要与要绑定的 Command 一致。

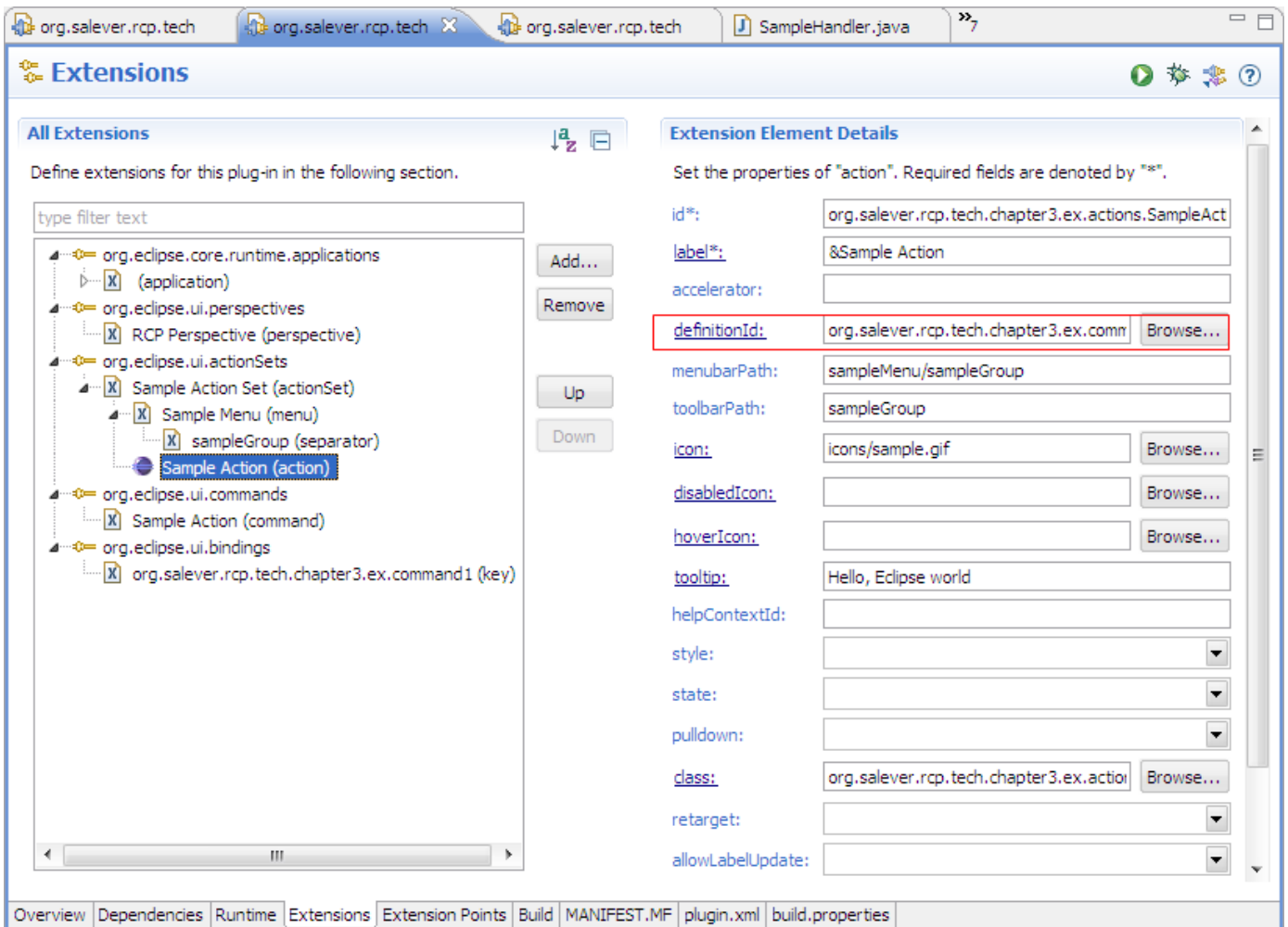
3.4.2 Action+Command+Binding 绑定

有人可能有疑问，如果我们已经使用 Action 方式添加了新的菜单，这时候需要添加绑定，怎么办呢？这里再介绍一种 Action 与 Command 结合使用的绑定方式。

回到示例“org.salever.rcp.tech.chapter3.ex”，里面我们已经新建了一个 ActionSets。打开 plugin.xml 文件，转到 Extension Tab 页，添加“org.eclipse.ui.commands”和“org.eclipse.ui.bindings”扩展点。



然后分别新建 `comand` 和 `binding` 的，配置类似。需要注意的是，`Sample Action` 里面需要配置 `definationId`，通过这个属性，将 `Action` 与 `Command` 连接起来。`Binding` 中仍然需要指定 `command id`。



完整代码为：

```
<extension
  point="org.eclipse.ui.actionSets">
  <actionSet
    id="org.salever.rcp.tech.chapter3.ex.actionSet"
    label="Sample Action Set"
    visible="true">
    <menu
      id="sampleMenu"
      label="Sample &Menu">
      <separator
        name="sampleGroup">
      </separator>
```

```
</menu>
<action
  class="org.salever.rcp.tech.chapter3.ex.actions.SampleAction"
  definitionId="org.salever.rcp.tech.chapter3.ex.command1"
  icon="icons/sample.gif"
  id="org.salever.rcp.tech.chapter3.ex.actions.SampleAction"
  label="&Sample Action"
  menubarPath="sampleMenu/sampleGroup"
  toolbarPath="sampleGroup"
  tooltip="Hello, Eclipse world">
  </action>
</actionSet>
</extension>
<extension
  point="org.eclipse.ui.commands">
  <command
    id="org.salever.rcp.tech.chapter3.ex.command1"
    name="Sample Action">
  </command>
</extension>
<extension
  point="org.eclipse.ui.bindings">
  <key
    commandId="org.salever.rcp.tech.chapter3.ex.command1"
    contextId="org.eclipse.ui.contexts.window"
    schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"
    sequence="M1+6">
  </key>
</extension>
```

4 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 “`org.salever.rcp.tech.chapter4`”，使用“Hello RCP application”模板

在 `Activator.java` 里定义你的程序 ID “`org.salever.rcp.tech.chapter4`”：

```
package org.salever.rcp.tech.chapter4;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

/**
 * The activator class controls the plug-in life cycle
 */
public class Activator extends AbstractUIPlugin {

    // The plug-in ID
    public static final String PLUGIN_ID = "org.salever.rcp.tech.chapter4"; //$NON-NLS-1$
```

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 org.salever.rcp.tech.chapter4;

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(window.getShell());  
            actionBarAdvisor.fillTrayItem(trayMenu);  
            menu.setVisible(true);  
        }  
    });  
}
```

```
private void createMinimize() {  
    window.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(
        Activator.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 org.salever.rcp.tech.chapter4;
```

```
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);
    }
}
```

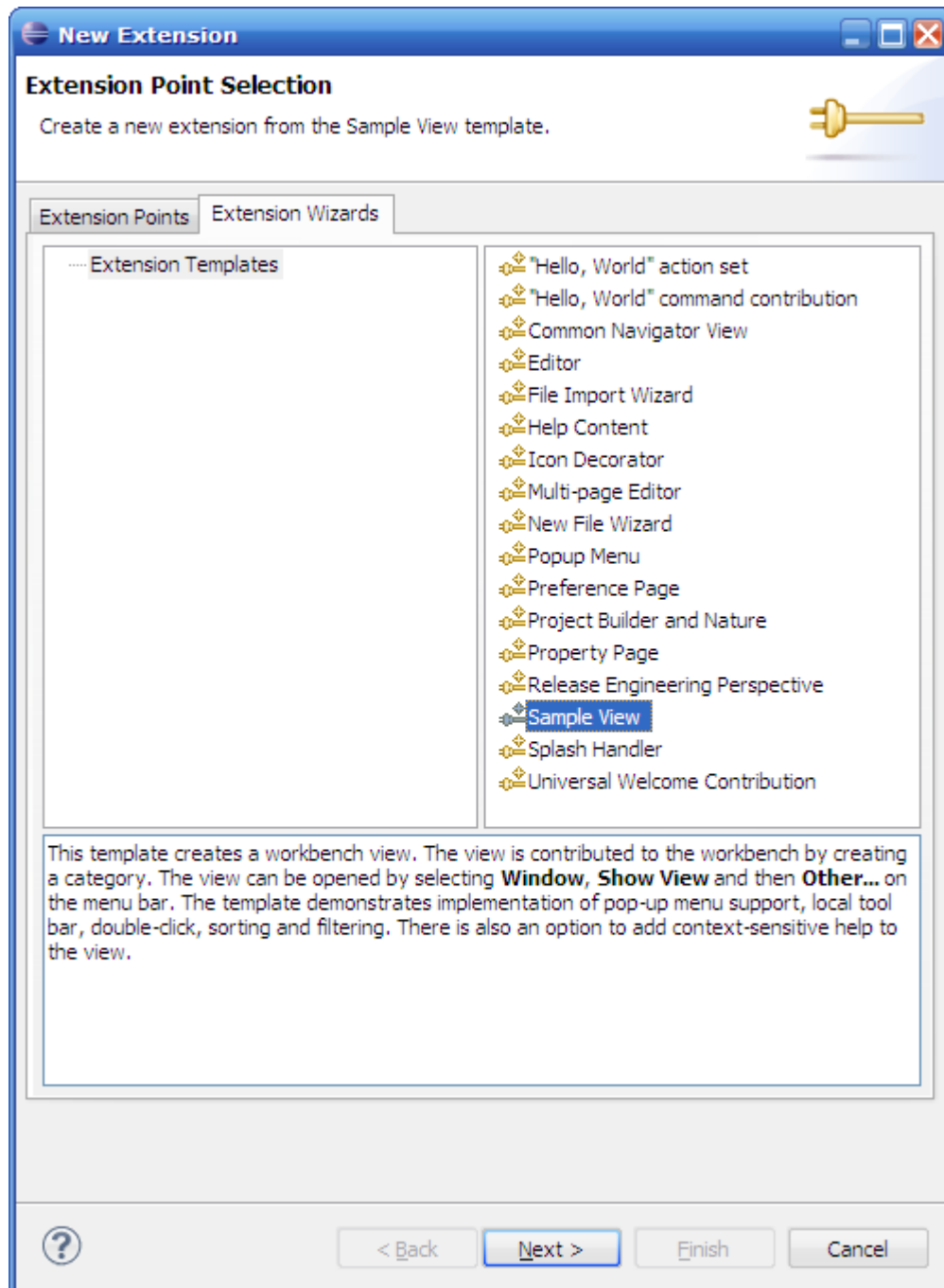
现在，运行你的程序，看一看程序最小化到系统托盘，右键实验一下菜单。

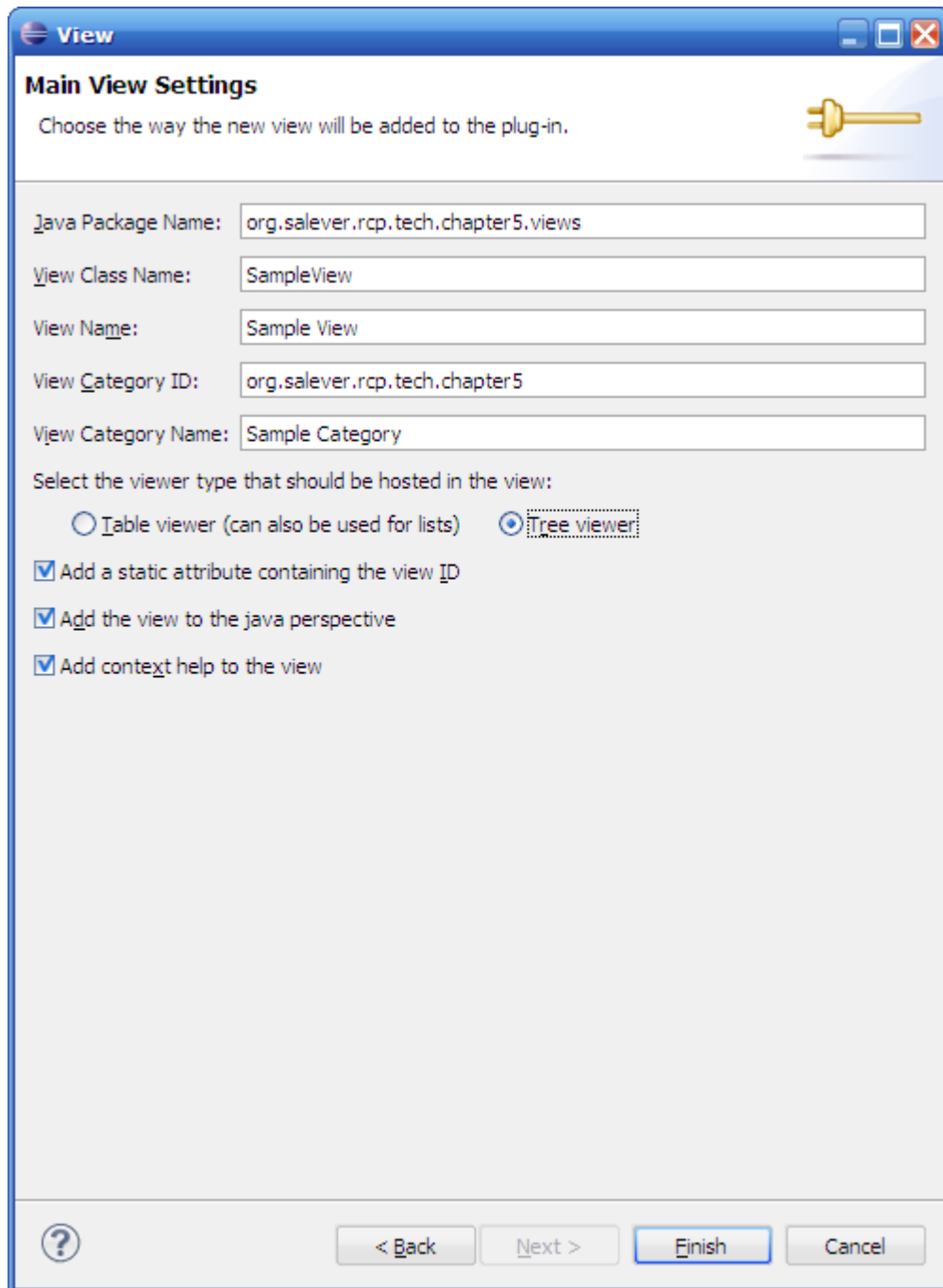
5 视图

视图（View）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 View

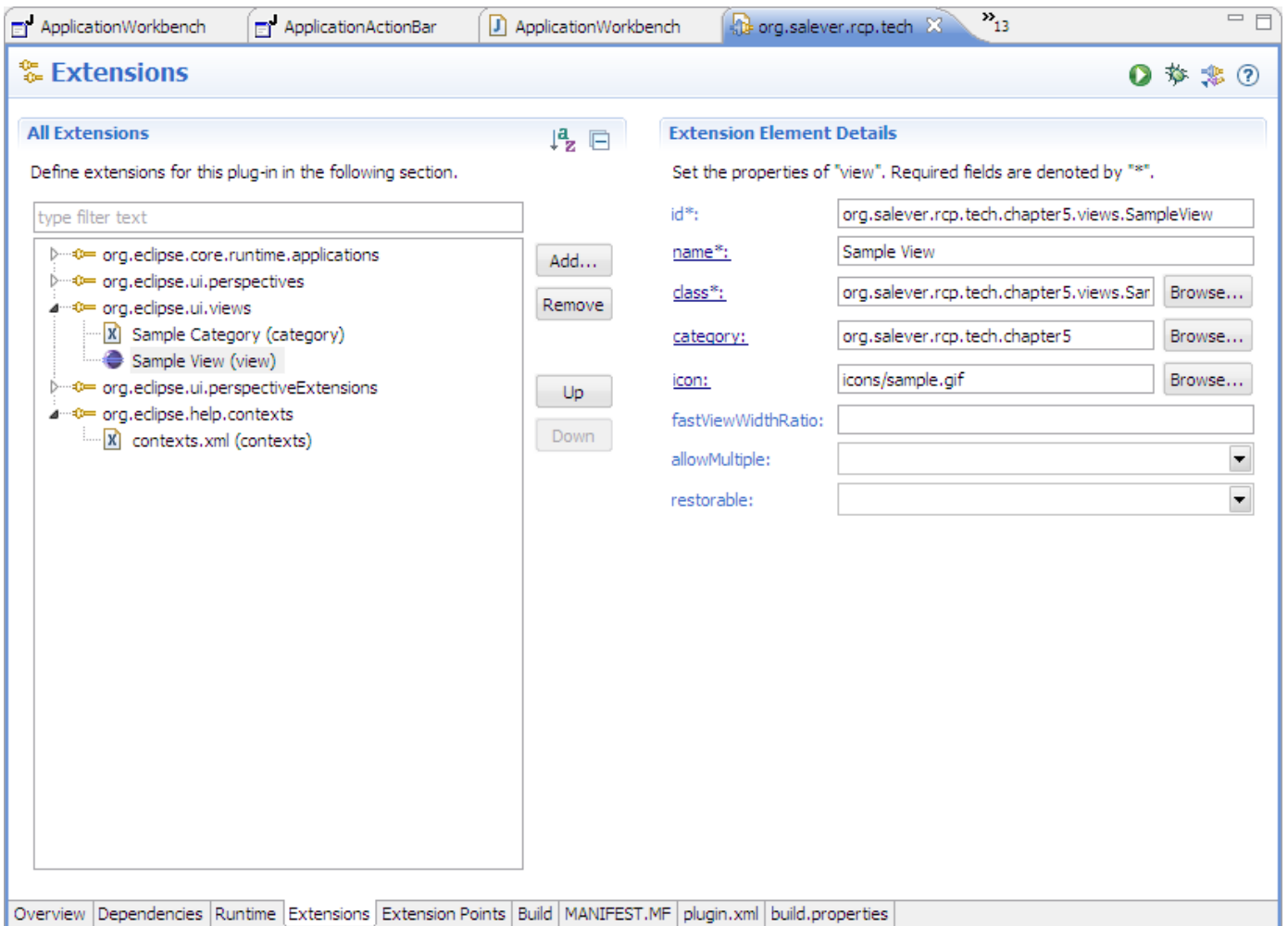
5.1 添加示例视图

创建一个新的工程“org.salever.rcp.tech.chapter5”，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”。





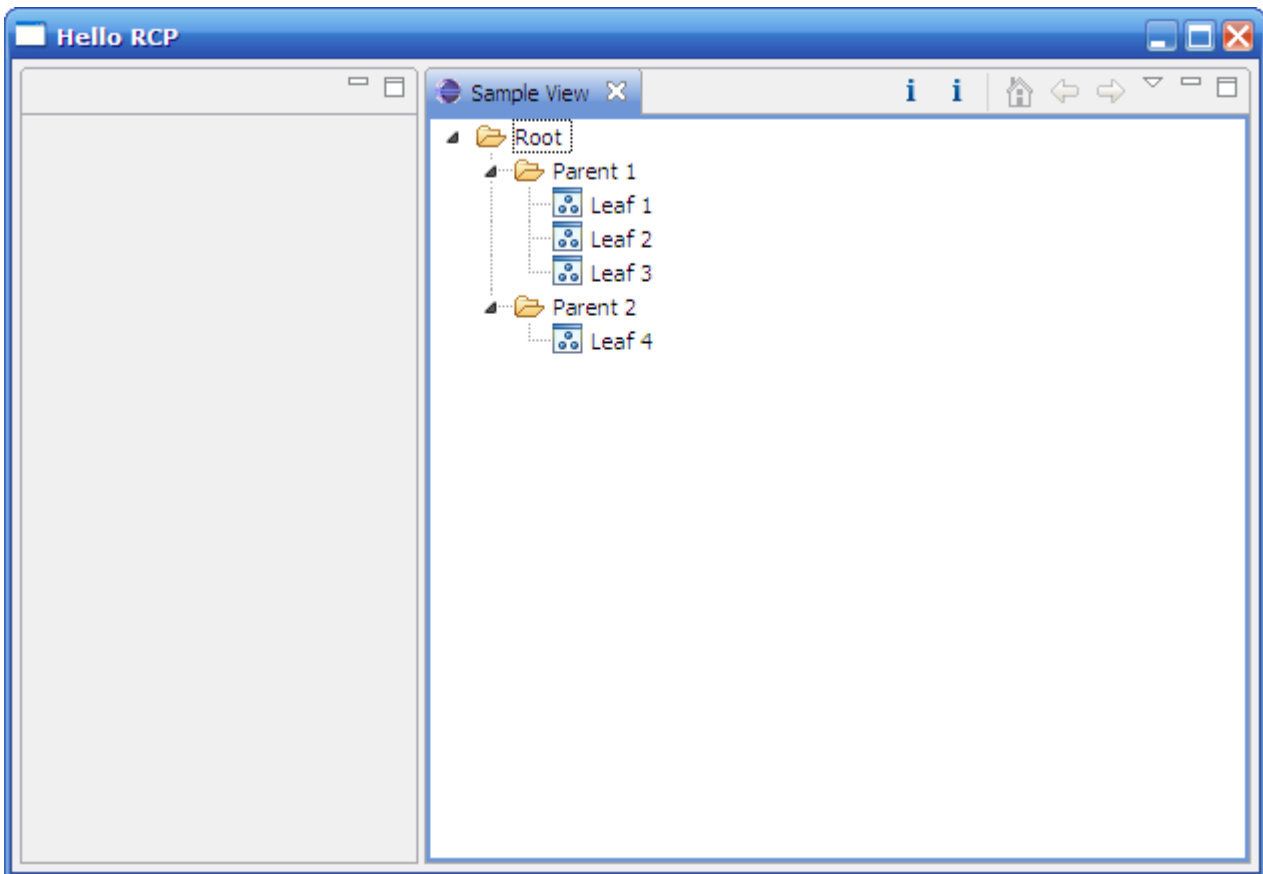
可以发现 Extension 中添加了几个新的扩展点，这时候运行程序，发现 View 无法显示。



修改 “org.eclipse.ui.perspectiveExtensions” 中的 targetID 为：

org.salever.rcp.tech.chapter5.perspective（RCP 默认的透视图），

再次运行，新的视图就出现了。



注：如果 View 不可关闭，加入如下设定：

```
<extension
  point="org.eclipse.ui.perspectiveExtensions">
  <perspectiveExtension
    targetID="org.salever.rcp.tech.chapter5.perspective">
    <view
      closeable="false"
      id="org.salever.rcp.tech.chapter5.views.SampleView"
      ratio="0.5"
      relationship="right"
      relative="org.eclipse.ui.views.TaskList">
    </view>
  </perspectiveExtension>
</extension>
```

类似的还可以控制 View 的移动、最大化最小化等：

Extension Element Details

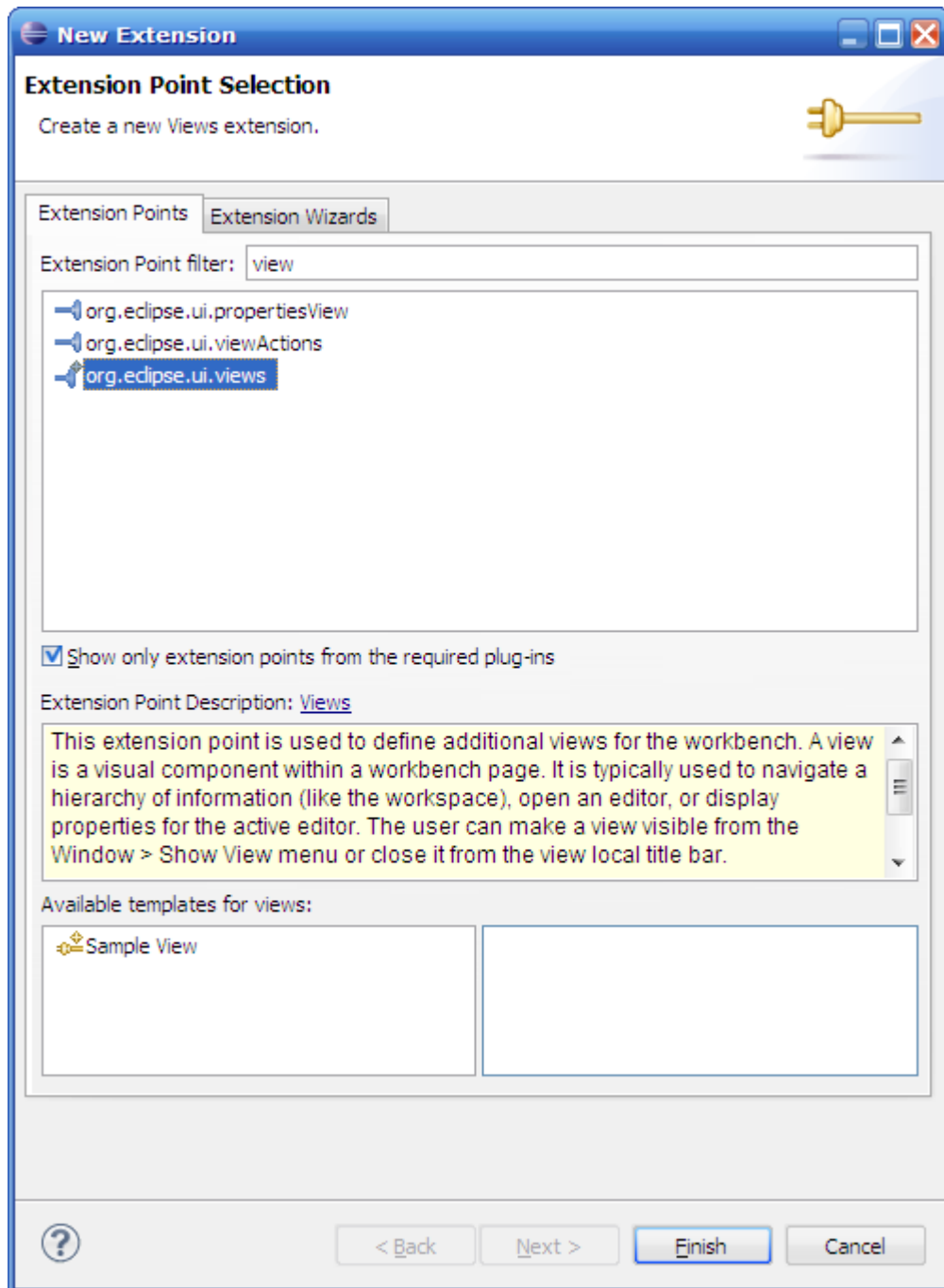
Set the properties of "view". Required fields are denoted by "**".

id**:	<input type="text" value="org.salever.rcp.tech.chapter5.views.SampleVie"/>	<input type="button" value="Browse..."/>
relationship**:	<input type="text" value="right"/>	<input type="button" value="▼"/>
relative:	<input type="text" value="org.eclipse.ui.views.TaskList"/>	<input type="button" value="Browse..."/>
ratio:	<input type="text" value="0.5"/>	
visible:	<input type="text"/>	<input type="button" value="▼"/>
closeable:	<input type="text" value="false"/>	<input type="button" value="▼"/>
moveable:	<input type="text" value="false"/>	<input type="button" value="▼"/>
standalone:	<input type="text"/>	<input type="button" value="▼"/>
showTitle:	<input type="text"/>	<input type="button" value="▼"/>
minimized:	<input type="text" value="false"/>	<input type="button" value="▼"/>

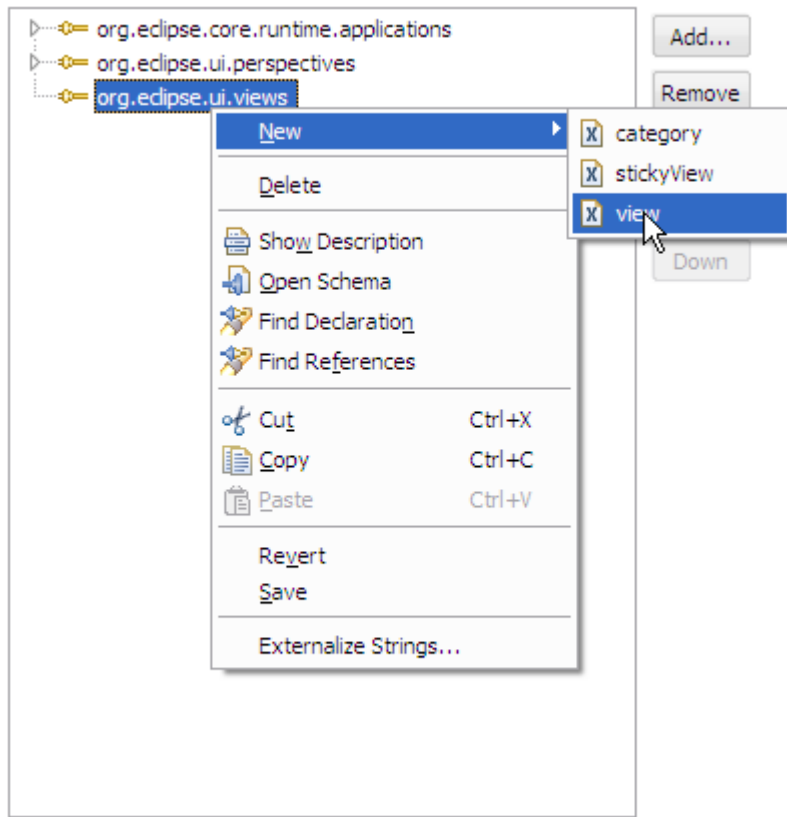
5.2 添加自定义视图

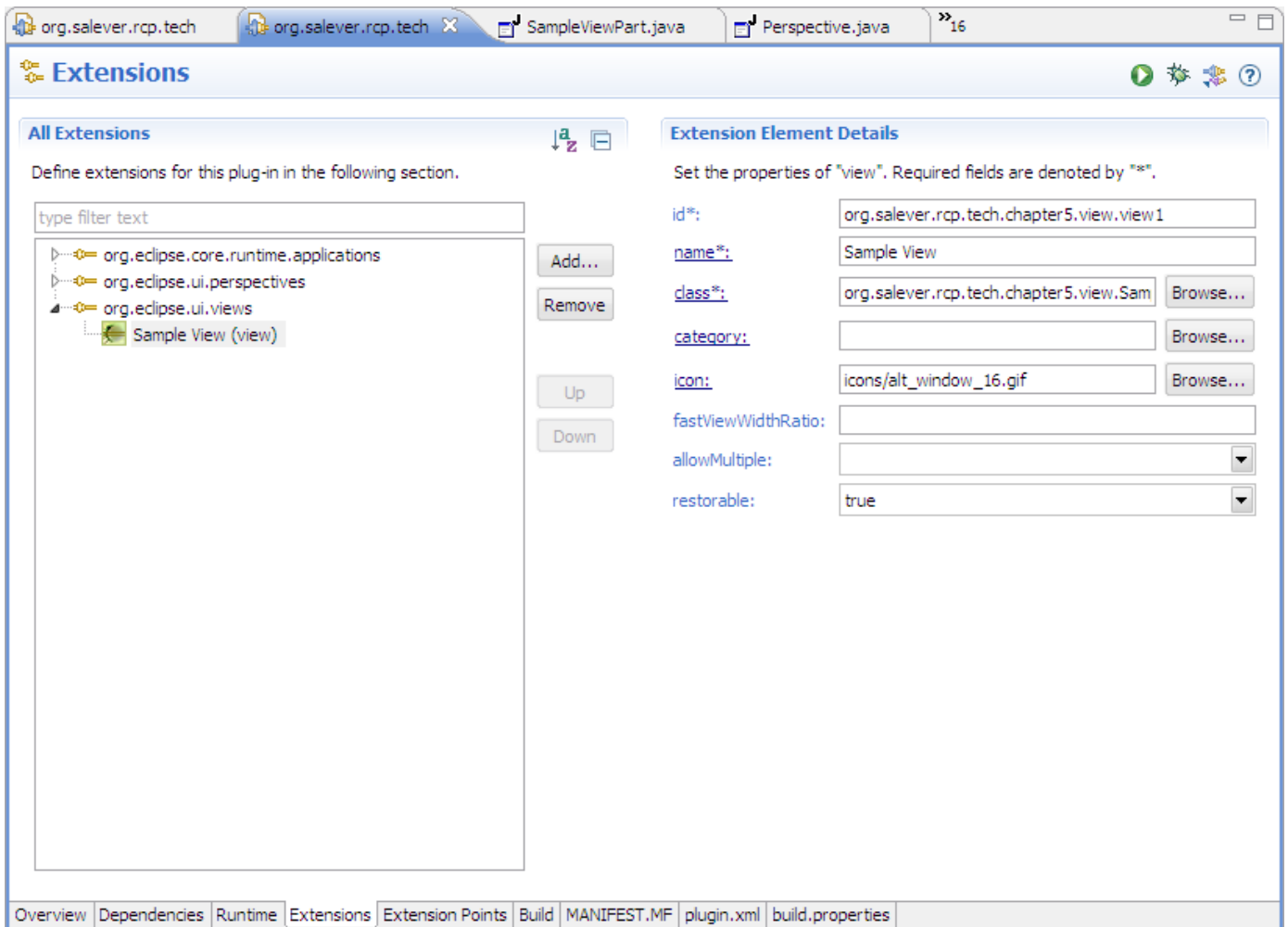
创建新工程“org.salever.rcp.tech.chapter5.view”，使用“Hello RCP”模板。

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



右键点击此扩展，选择 New->View





创建一个 view 的新类，点击 class 超连接，新建类 SampleViewPart.java。在你的新类里如下修改代码：

```

package org.salever.rcp.tech.chapter5.view;

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 SampleViewPart extends ViewPart {

    public SampleViewPart() {
        // 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
    }
}

}

```

接下来在 Perspective.java 里添加 View

```
package org.salever.rcp.tech.chapter5.view;

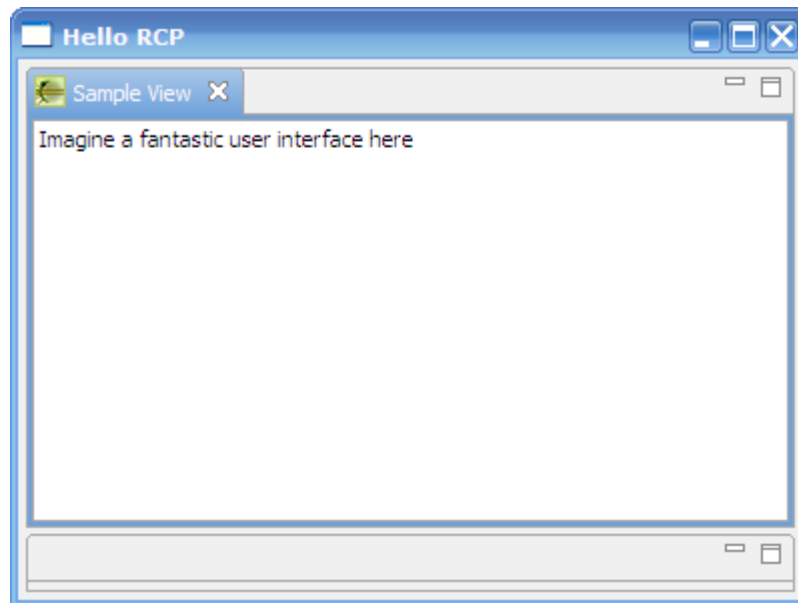
import org.eclipse.ui.IPageLayout;

public class Perspective implements IPerspectiveFactory {

    public void createInitialLayout(IPageLayout layout) {
        layout.addView("org.salever.rcp.tech.chapter5.view.view1", IPageLayout.TOP,
            IPageLayout.RATIO_MAX, IPageLayout.ID_EDITOR_AREA);
    }
}

```

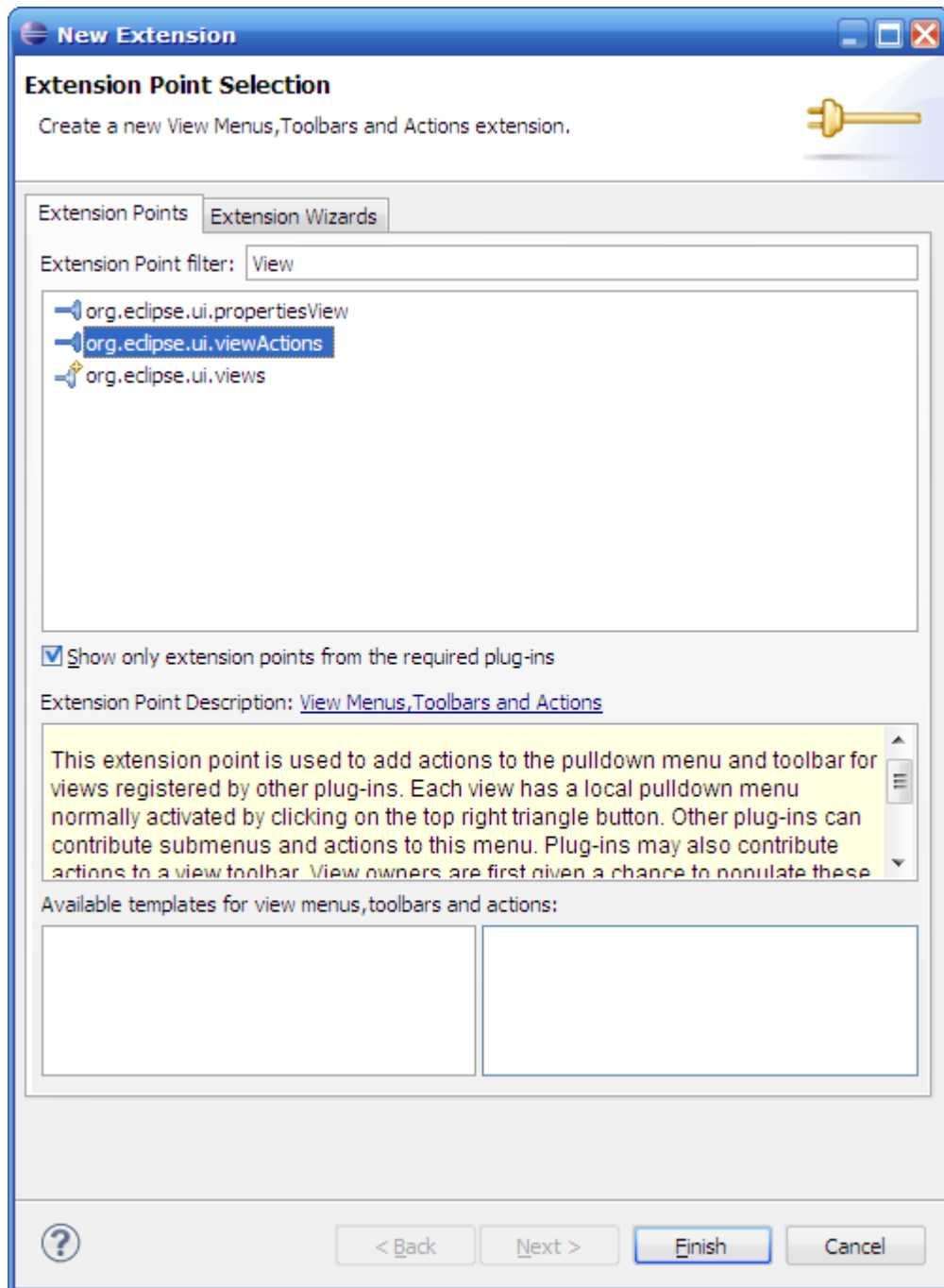
运行结果如图：



5.3 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

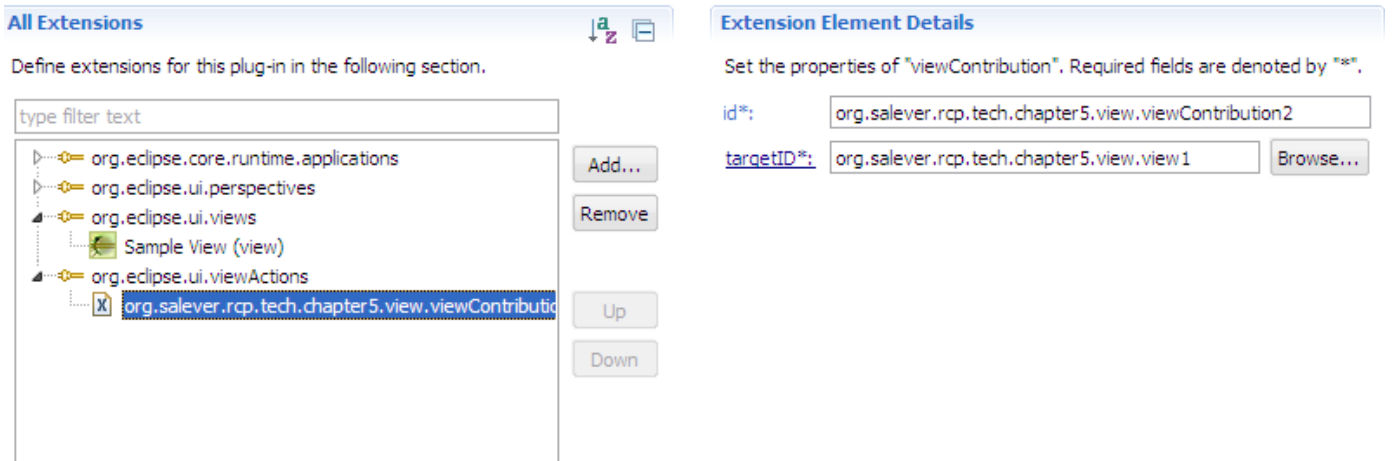
继续 “org.salever.rcp.tech.chapter5.view” 示例。选择 plugin.xml 的“Extensions”，点击“Add”按钮，选择 org.eclipse.ui.ViewActions。



现在是最重要的了，将 targetID 改为你早先创建的 view 名称：

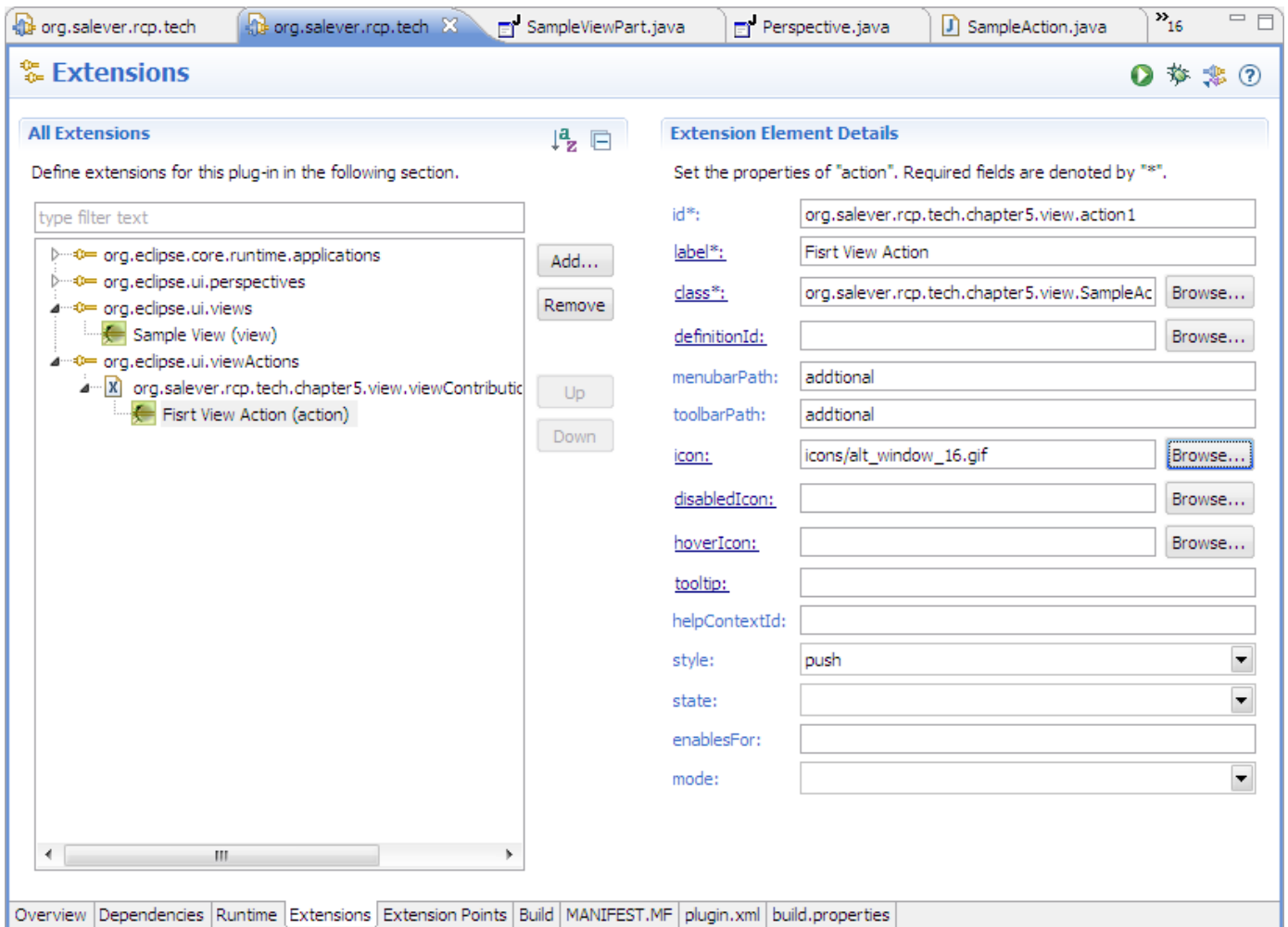
“org.salever.rcp.tech.chapter5.view.view1”

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 “org.salever.rcp.tech.chapter5.view.SampleAction”。



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 org.salever.rcp.tech.chapter5.view;
```

```
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;
```

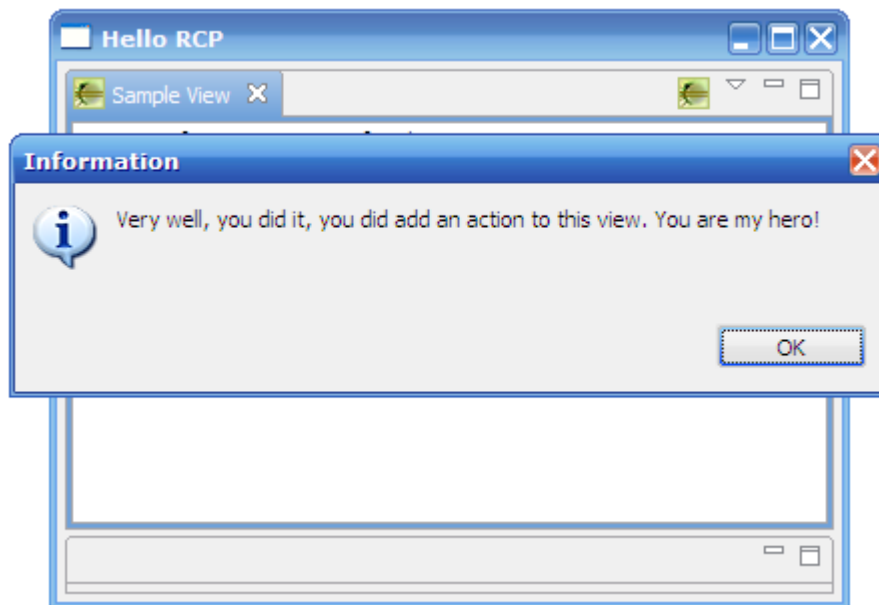
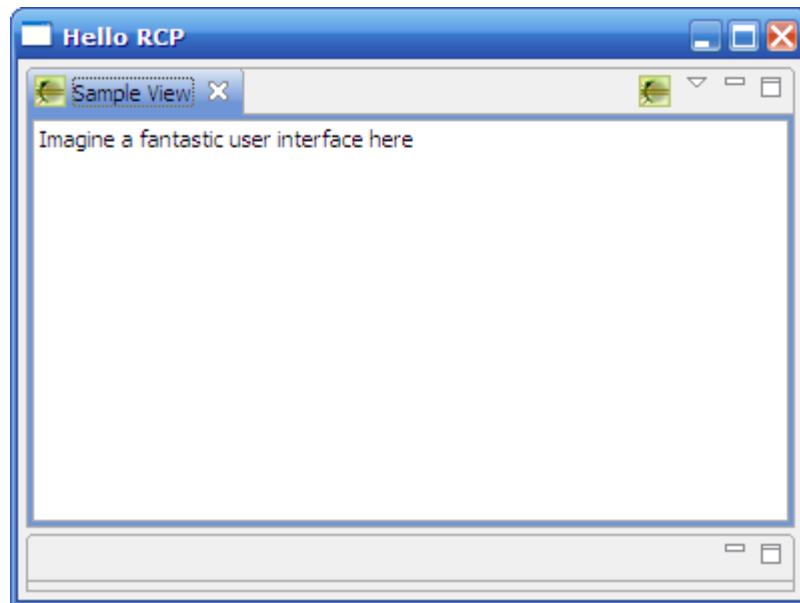
```
public class SampleAction implements IViewActionDelegate {
    SampleViewPart myView;
```

```
public void init(IViewPart view) {
    // TODO Auto-generated method stub
    this.myView = (SampleViewPart) 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。



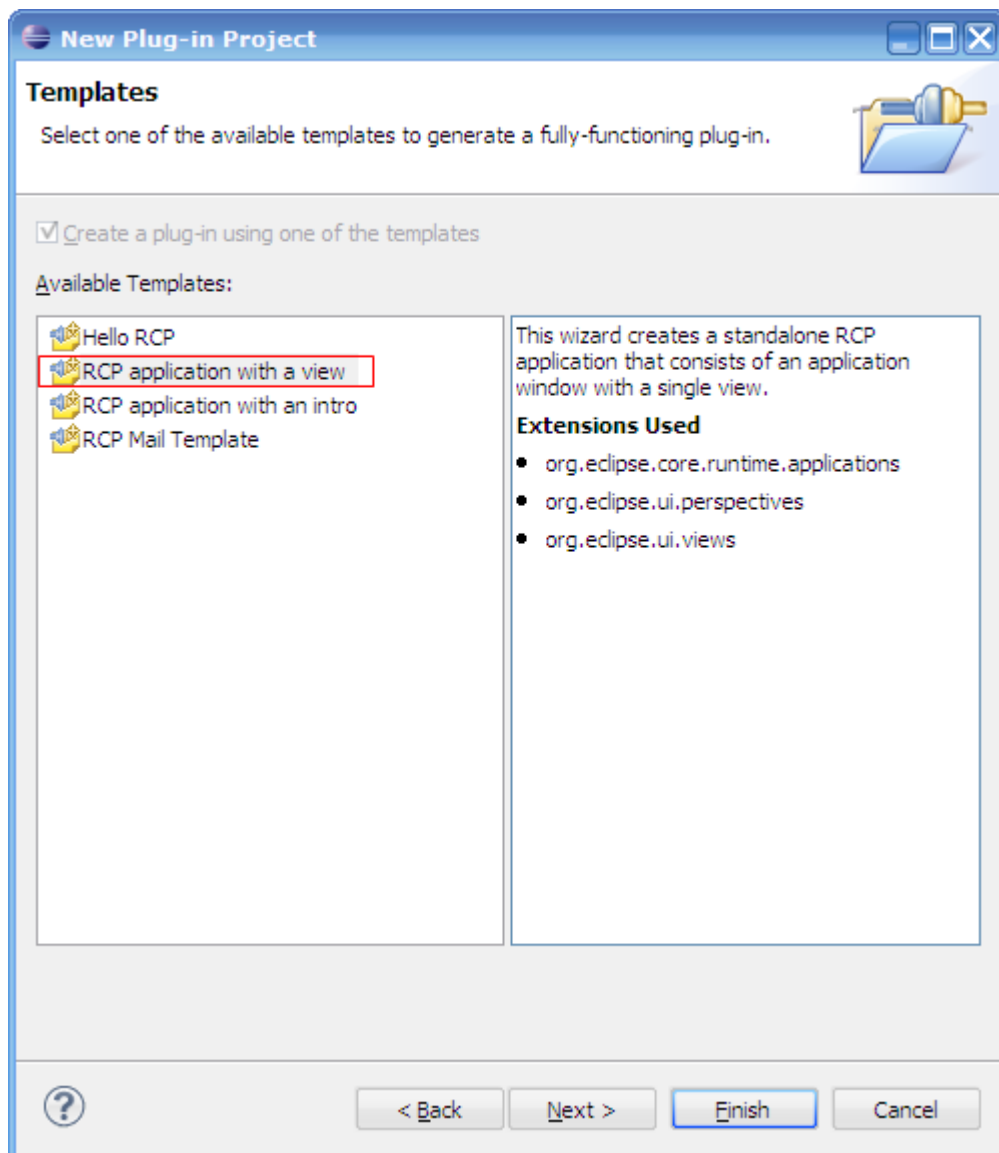
6 编辑器

6.1 概述

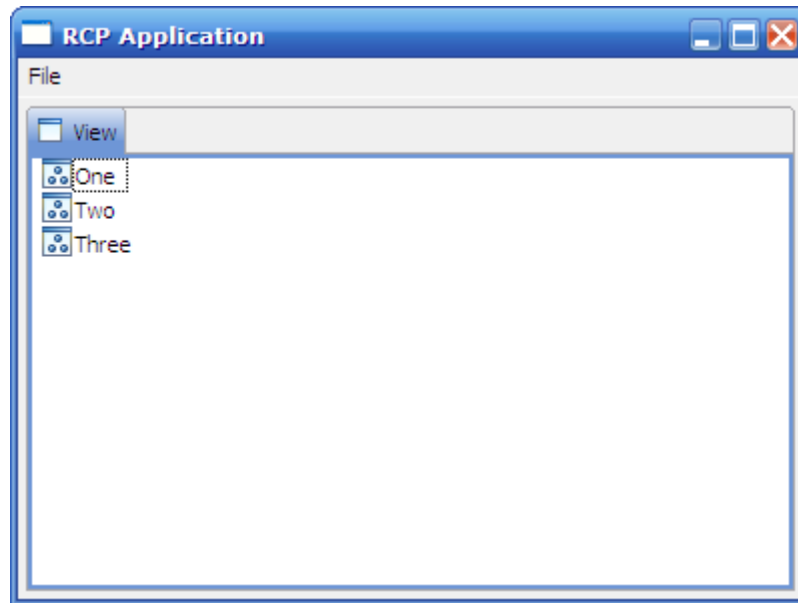
编辑器 (Editor) 的使用是 RCP 开发中非常重要的一个环节,基本上所有的 RCP 都会与它打交道,掌握编辑器是每个 Eclipse 必须的技能。

6.2 创建工程

新建 Plug-in 工程 “org.salever.rcp.tech.chapter6”,这次我们采用使用 “RCP application with view” 模板。

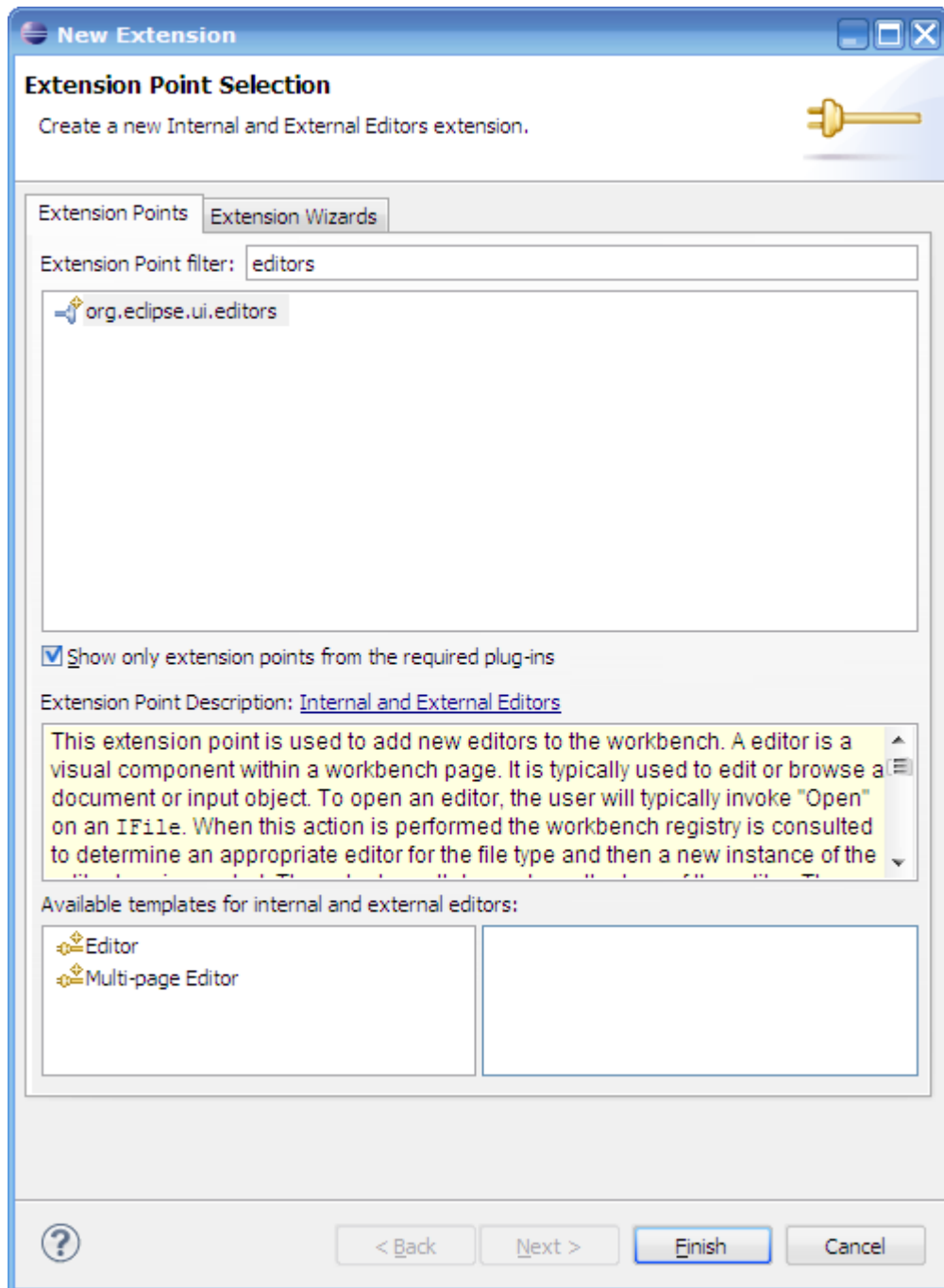


完成以后运行工程，得到如下结果：

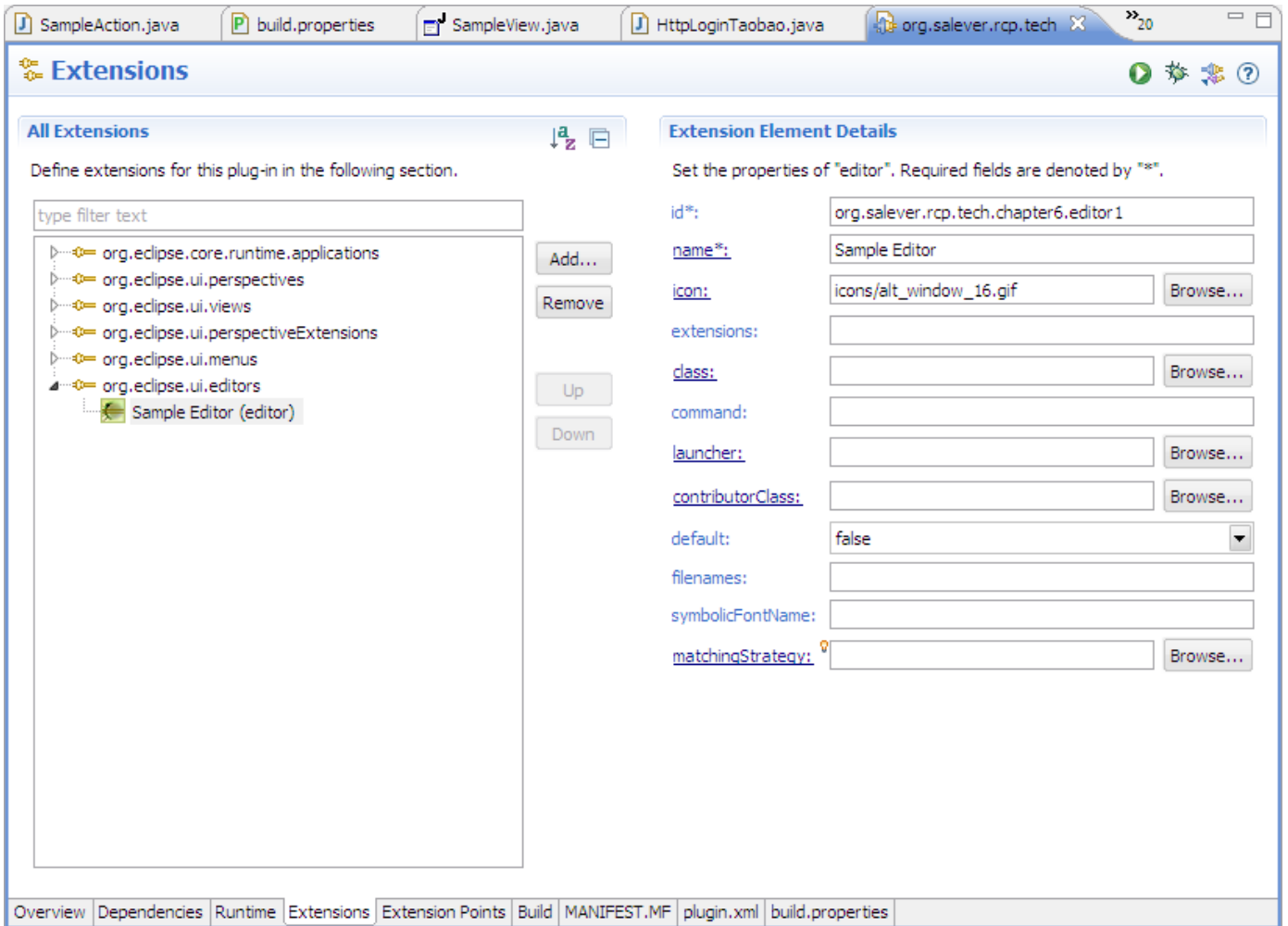


6.3 添加编辑器

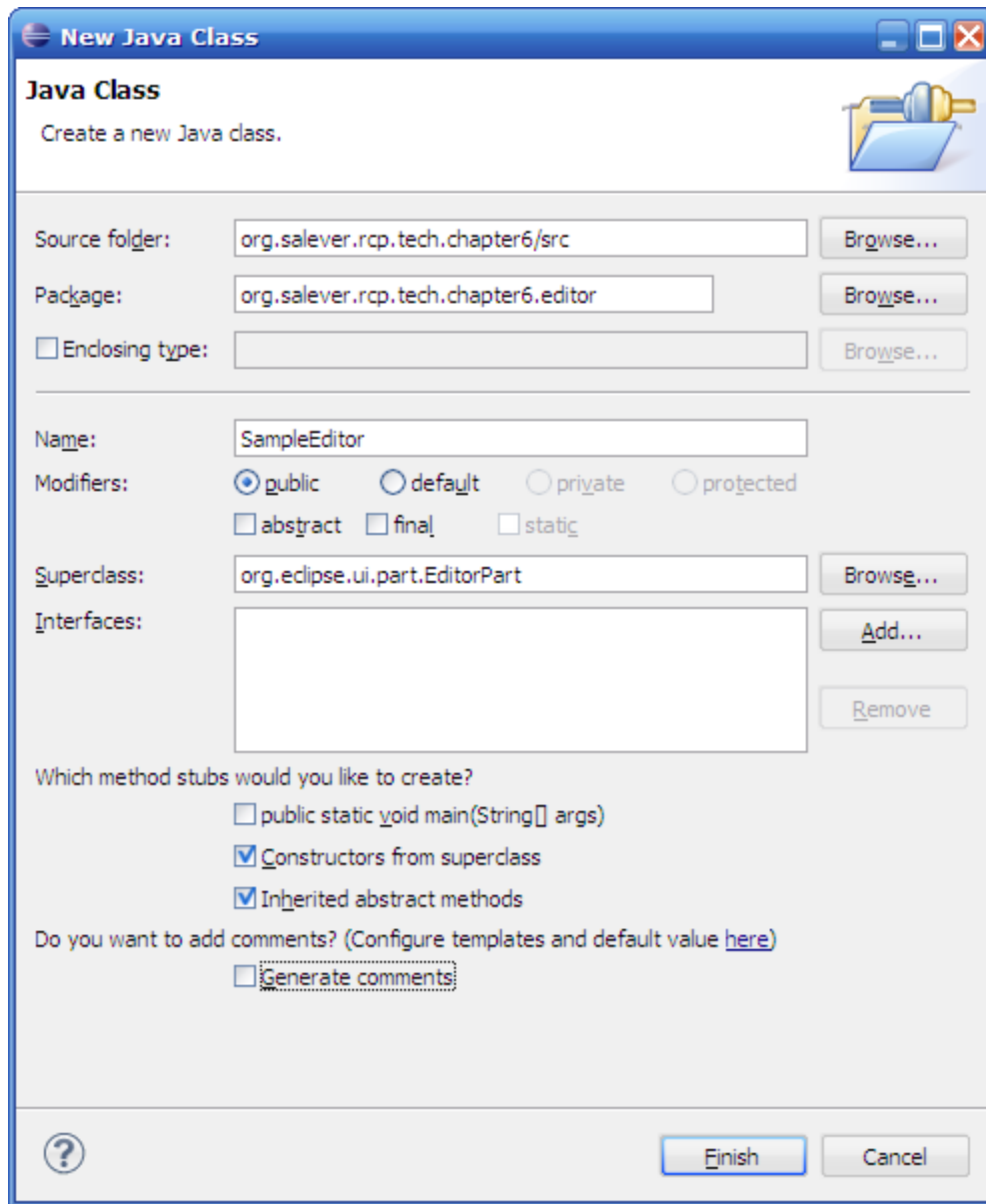
双击 `plugin.xml` 选择 “Extension” 标签。加入扩展 `org.eclipse.ui.editors`。不要使用模板。ID 设置为 “`org.salever.rcp.tech.chapter6.editor1`”，名称设为 “Sample Editor”。确认你选择了一个图标，否则你的编辑器将不会工作。点击 `class` 超连接以创建一个类，这个类就是 `Editor` 的实现类了。



扩展点 “org.eclipase.ui.editors”



扩展“org.eclipase.ui.editors”



新建 Editor 类

先看看 SampleEditor 的实现：

```
package org.salever.rcp.tech.chapter6.editor;
```

```
import org.eclipse.core.runtime.IProgressMonitor;
```

```
public class SampleEditor extends EditorPart {
```

```
public static final String ID = "org.salever.rcp.tech.chapter6.editor1";
```

```
private Text text;
```

```
public SampleEditor() {  
}
```

```
@Override
```

```
public void doSave(IProgressMonitor monitor) {  
  
}
```

```
@Override
```

```
public void doSaveAs() {  
  
}
```

```
@Override
```

```
public void init(IEditorSite site, IEditorInput input)  
    throws PartInitException {  
    //非常重要，必须实现这个方法  
    setSite(site);  
    setInput(input);  
    setPartName(input.getName());  
  
}
```

```
@Override
```

```
public boolean isDirty() {  
    return false;  
}
```

```
@Override
```

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

```
    }

    @Override
    public void createPartControl(Composite parent) {

        Composite composite = new Composite(parent, SWT.NONE);
        composite.setLayout(new GridLayout(2, false));

        Label lblClickAt = new Label(composite, SWT.NONE);
        lblClickAt.setLayoutData(new GridData(SWT.RIGHT, SWT.CENTER, false, false, 1, 1));
        lblClickAt.setText("Click At:");

        text = new Text(composite, SWT.BORDER);
        text.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));

        //设置内容
        text.setText(getEditorInput().getName());
    }

    @Override
    public void setFocus() {
        text.setFocus();
    }
}
```

要打开 Editor 还需要一个 input，这个 input 非常重要，它提供了 Editor 打开的内容。这里我们新建一个类 SampleInput，实现接口 org.eclipse.ui.part.EditorPart。

```
package org.salever.rcp.tech.chapter6.editor;

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



```
/**
 * @author
 *
 */
public class SampleInput implements IEditorInput{

    /**
     * 节点名称
     */
    private String name = "";

    public SampleInput(String name) {
        this.name = name;
    }

    @Override
    public Object getAdapter(Class adapter) {
        return null;
    }

    @Override
    public boolean exists() {
        return false;
    }

    @Override
    public ImageDescriptor getImageDescriptor() {
        return null;
    }

    @Override
    public String getName() {
        return name;
    }
}
```

```
@Override
public IPersistableElement getPersistable() {
    return null;
}

@Override
public String getToolTipText() {
    return name;
}

@Override
public boolean equals(Object obj) {
    if (super.equals(obj)) {
        return true;
    }

    if (obj instanceof SampleInput) {
        return name.equals(((SampleInput) obj).getName());
    }

    return false;
}

@Override
public int hashCode() {
    return name.hashCode();
}
}
```

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。

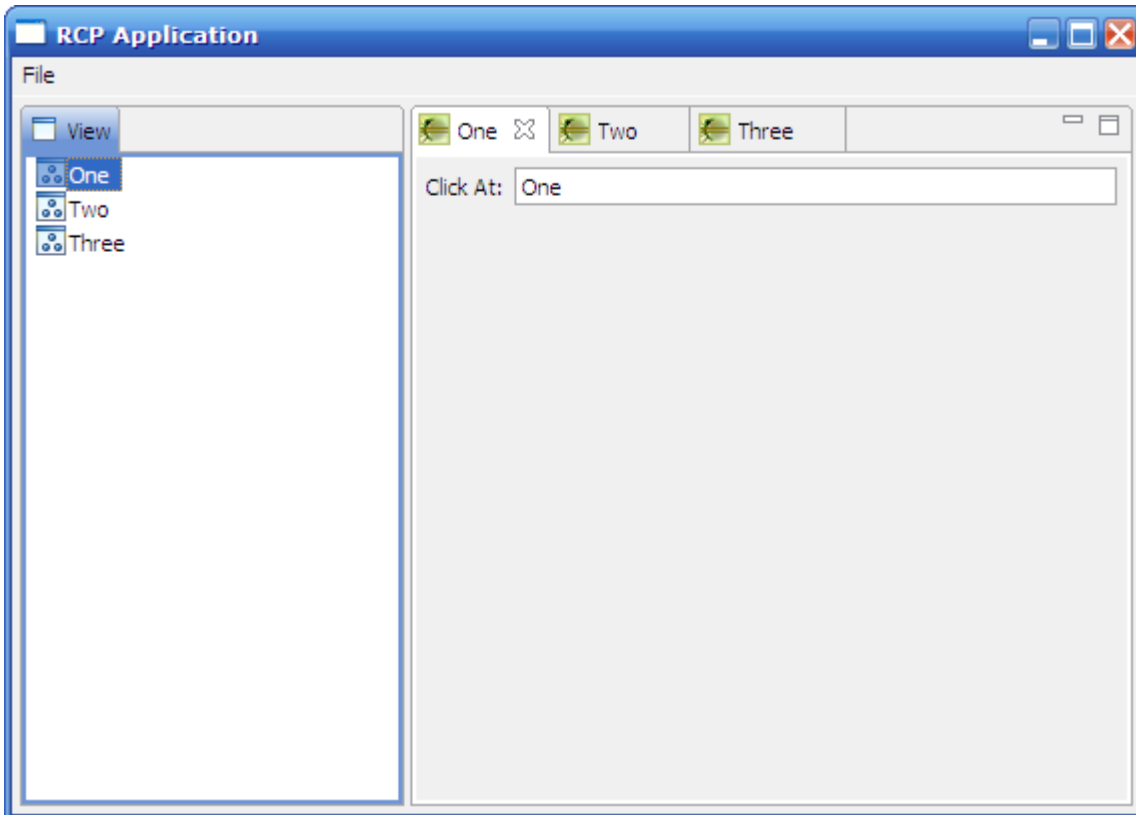
到这里所有的工作都已经准备完毕，接下来看看如何打开编辑器。

6.4 调用编辑器

修改 View 类，给树节点添加双击响应操作，修改 createPartControl()方法：

```
public void createPartControl(Composite parent) {  
viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL  
    | SWT.V_SCROLL);  
viewer.setContentProvider(new ViewContentProvider());  
viewer.setLabelProvider(new ViewLabelProvider());  
// Provide the input to the ContentProvider  
viewer.setInput(new String[] { "One", "Two", "Three" });  
  
viewer.addDoubleClickListener(new IDoubleClickListener() {  
  
    @Override  
    public void doubleClick(DoubleClickEvent event) {  
        ISelection selection = viewer.getSelection();  
        Object obj = ((IStructuredSelection) selection)  
            .getFirstElement();  
  
        String node = obj.toString();  
        SampleInput input = new SampleInput(node);  
        IWorkbenchPage page = PlatformUI.getWorkbench()  
            .getActiveWorkbenchWindow().getActivePage();  
  
        try {  
            page.openEditor(input, SampleEditor.ID);  
        } catch (PartInitException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
});  
}
```

运行程序，结果如图，因为我们重写了 `SampleInput` 的 `equals` 方法，所以每个节点对应的编辑器只能打开一次：

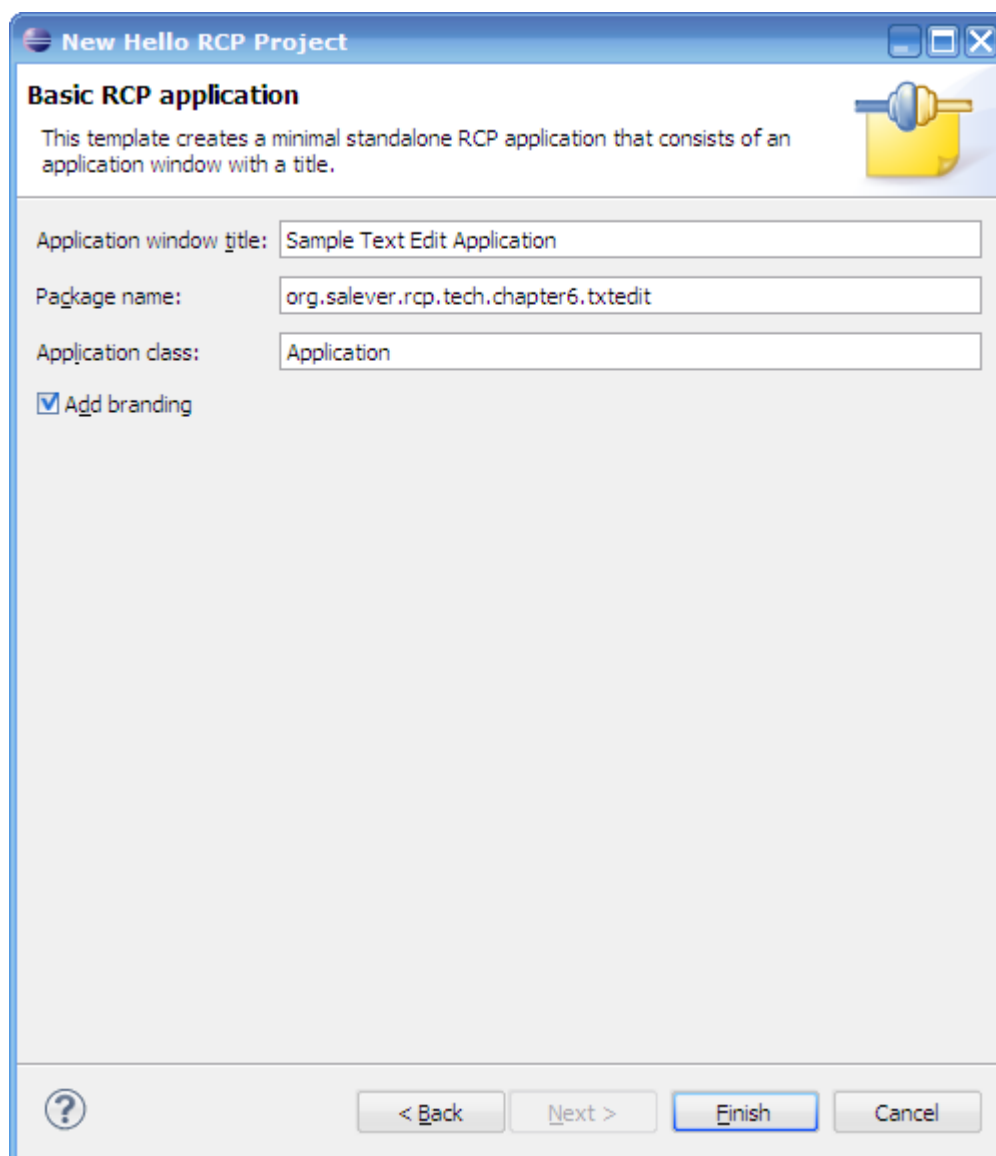


6.5 实例：文本编辑器实现

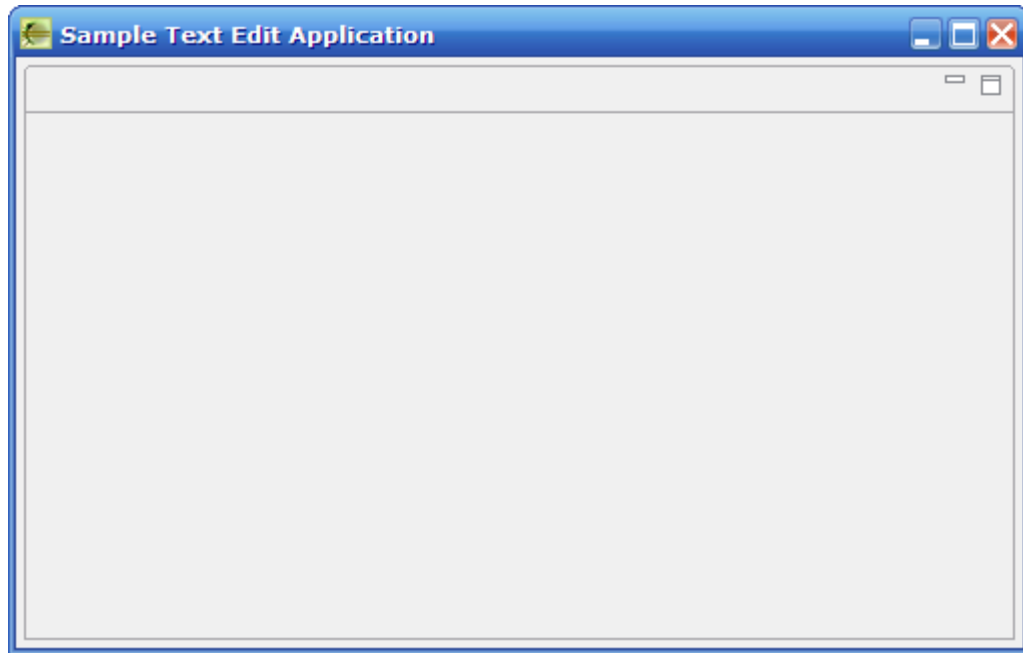
到这里我们就可以完整的实现一个文本编辑器了，这个实例会涉及到文件对话框的打开、菜单栏、编辑器保存等，算是给前面的学习一个总结。

6.5.1 新建工程

新建工程 “`org.salever.rcp.tech.chapter6.txtedit`”，使用 “Hello, world” 模板，在最后一步界面上修改一下标题等信息。



运行工程：



6.5.2 添加菜单栏和工具栏

接下来添加菜单栏和工具栏。

首先新建一个 `OpenAction`，用来浏览打开一个文件，代码如下：

```
package org.salever.rcp.tech.chapter6.txtedit.action;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.ui.IWorkbenchWindow;
import org.salever.rcp.tech.chapter6.txtedit.Activator;

/**
 * @author
 *
 */
public class OpenAction extends Action{

    private static final String ID = "org.salever.rcp.tech.chapter6.txtedit.action.open";
```

```
private IWorkbenchWindow window;  
  
public OpenAction(IWorkbenchWindow window) {  
    this.window = window;  
    setText("&Open File");  
    setId(ID);  
    setImageDescriptor(Activator.getImageDescriptor("icons/open.gif"));  
}  
  
@Override  
public void run() {  
    FileDialog dialog = new FileDialog(window.getShell(), SWT.OPEN);  
    String path = dialog.open();  
    if(path != null){  
        //XXX  
        MessageDialog.openInformation(window.getShell(), "Info", path);  
    }  
}  
}
```

打开文件后的处理部分暂时为空，这里先提示一个信息对话框，表明打开了一个文件。
然后按照第 3 章讲的，将 `OpenAction` 添加到菜单栏和工具栏中，修改文件：

ApplicationActionBarAdvisor.java

```
package org.salever.rcp.tech.chapter6.txtedit;  
  
import org.eclipse.jface.action.Action;  
  
public class ApplicationActionBarAdvisor extends ActionBarAdvisor {  
  
    private IWorkbenchAction iExitAction;  
    private IWorkbenchAction iAboutAction;  
    private IWorkbenchAction iSaveAsAction;  
    private IWorkbenchAction iSaveAction;
```

```
private Action openAction;

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);
    iSaveAsAction = ActionFactory.SAVE_AS.create(window);
    register(iSaveAsAction);

    openAction = new OpenAction(window);
    register(openAction);
}

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(openAction);
    fileMenu.add(new Separator());
    fileMenu.add(iSaveAction);
    fileMenu.add(iSaveAsAction);
    fileMenu.add(new Separator());
    fileMenu.add(iExitAction);
}
```

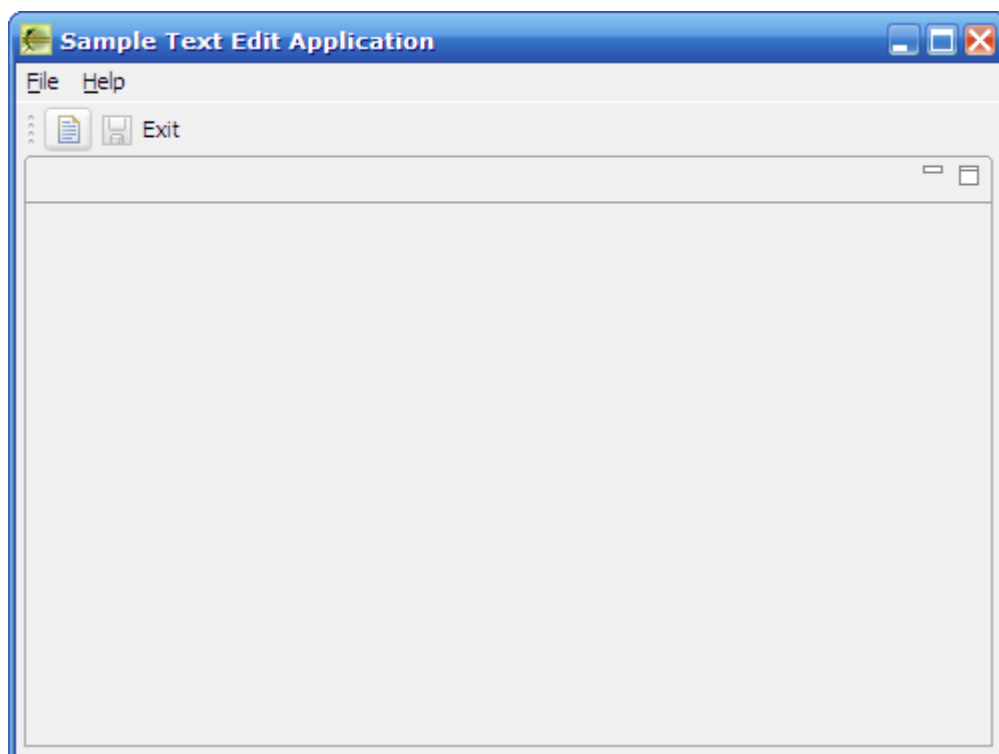


```
// Help Menu
helpMenu.add(iAboutAction);
}

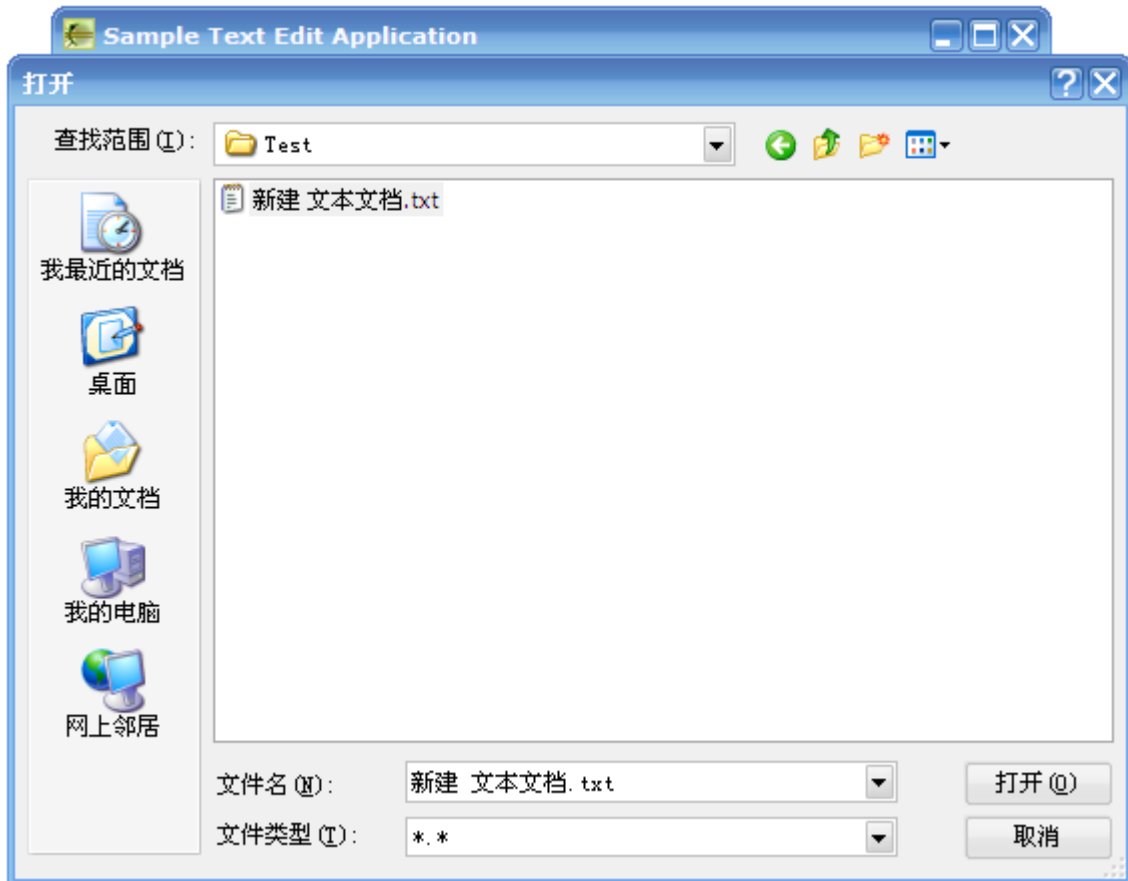
@Override
protected void fillCoolBar(ICoolBarManager coolBar) {
    // This will add a new toolbar to the application
    IToolBarManager toolbar = new ToolBarManager(SWT.FLAT | SWT.RIGHT);
    coolBar.add(new ToolBarContributionItem(toolbar, "main"));
    // Add the entry to the toolbar
    toolbar.add(openAction);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

最后别忘了修改 ApplicationWorkbenchWindowAdvisor.java，显示 coolbar。

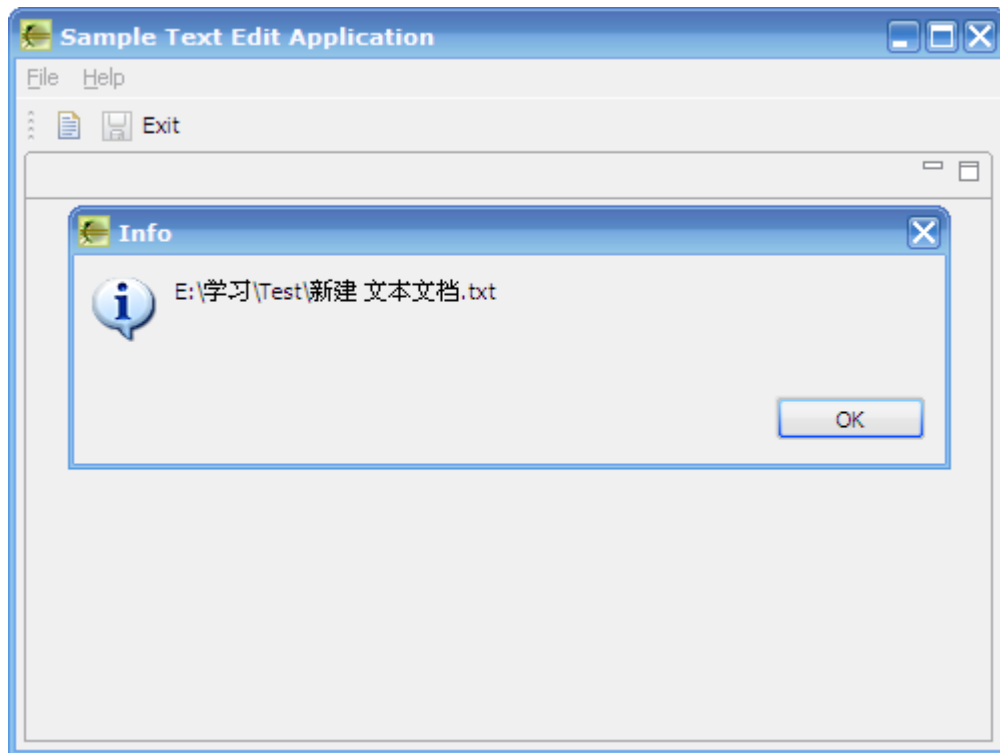
运行效果图：



浏览文件：

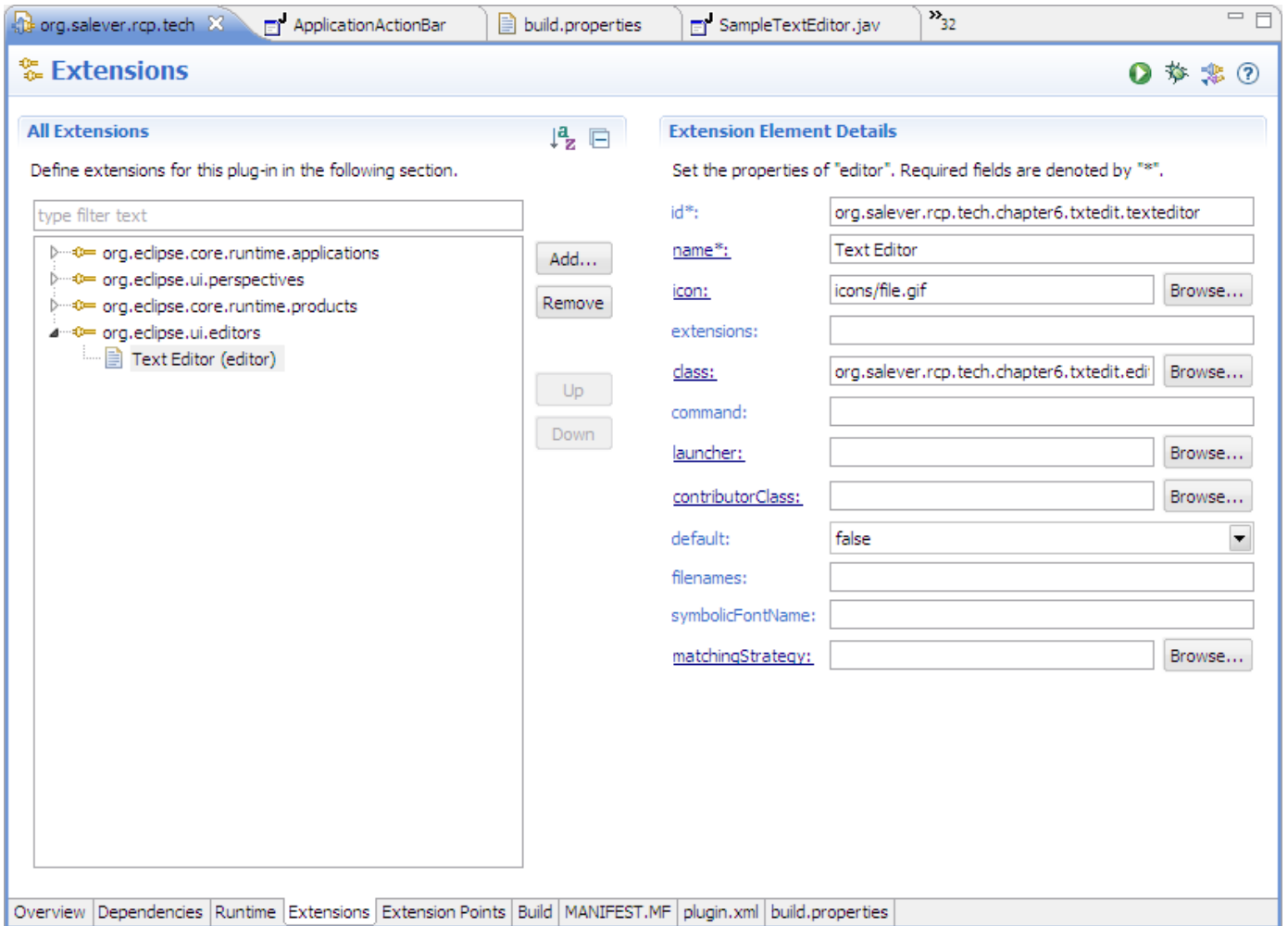


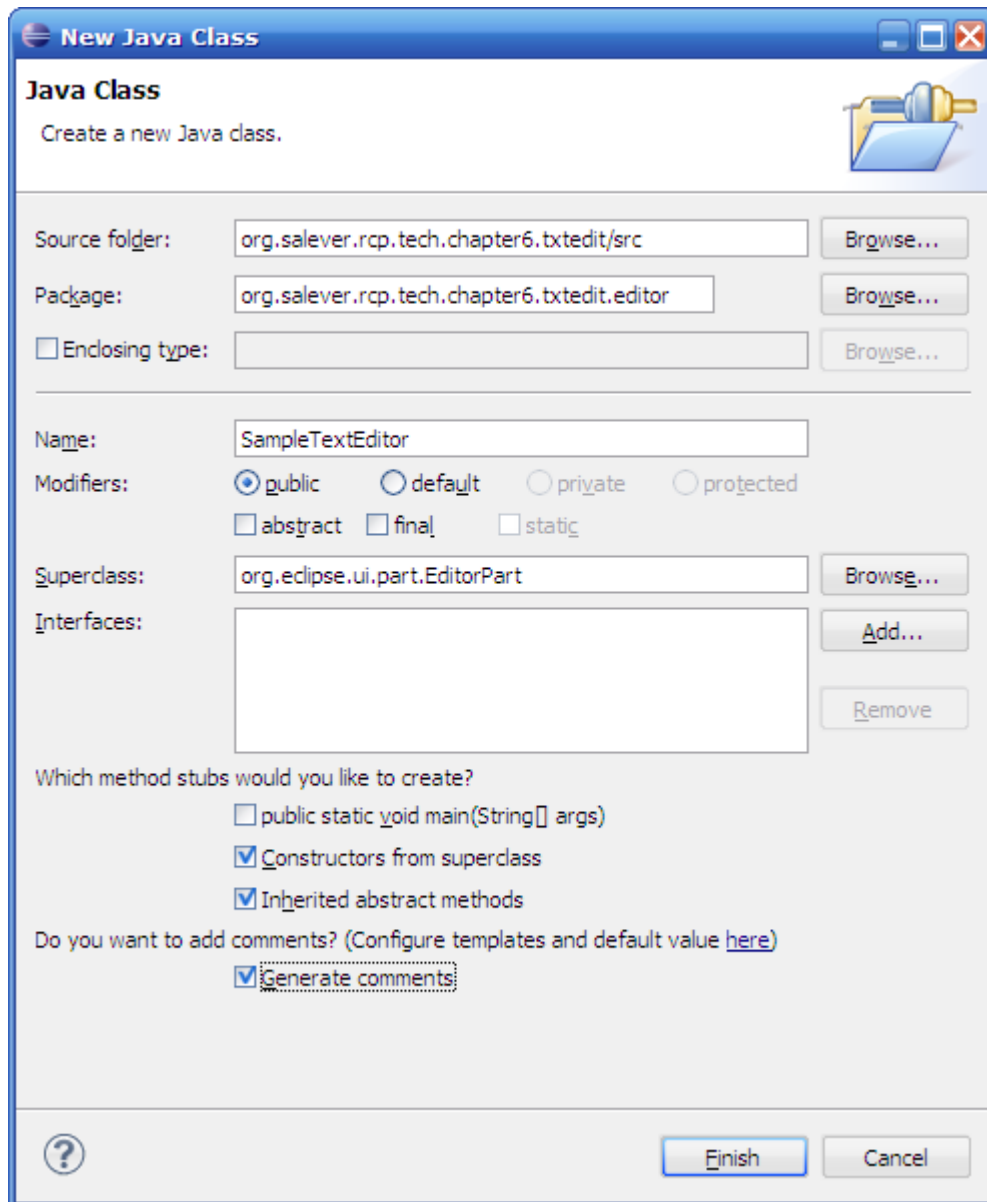
打开文件：



6.5.3 添加编辑器

打开 `plugin.xml`，添加“`org.eclipse.ui.editors`”扩展点，这里不太重复，然后新建对应的类。





首先还是要新建一个 `Input` 类，用来提供给编辑器，这里新建 `FileInput` 类，将需要打开的文件的路径保存并传递给编辑器 `SampleTextEditor`。

```
package org.salever.rcp.tech.chapter6.txtedit.editor;
```

```
import java.io.File;
```

```
import org.eclipse.jface.resource.ImageDescriptor;
```

```
import org.eclipse.ui.IEditorInput;
```

```
import org.eclipse.ui.IPersistableElement;
```

```
/**
 * @author
 *
 */
public class FileInput implements IEditorInput{

    /**
     * 文件路径
     */
    private String path = "";

    public FileInput(String path) {
        this.path = path;
    }

    @Override
    public Object getAdapter(Class adapter) {
        return null;
    }

    @Override
    public boolean exists() {
        return new File(path).exists();
    }

    @Override
    public ImageDescriptor getImageDescriptor() {
        return null;
    }

    @Override
    public String getName() {
        return path;
    }
}
```

```
@Override
public IPersistableElement getPersistable() {
    return null;
}

@Override
public String getToolTipText() {
    return path;
}

@Override
public boolean equals(Object obj) {
    if (super.equals(obj)) {
        return true;
    }

    if (obj instanceof FileInput) {
        return path.equals(((FileInput) obj).getName());
    }

    return false;
}

@Override
public int hashCode() {
    return path.hashCode();
}
}
```

SampleTextEditor 类:

```
package org.salever.rcp.tech.chapter6.txtedit.editor;
```

```
import java.io.BufferedReader;
```

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorSite;
import org.eclipse.ui.ISaveablePart2;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.EditorPart;

/**
 * @author salever@126.com
 *
 */
public class SampleTextEditor extends EditorPart implements ISaveablePart2 {

    public static final String ID = "org.salever.rcp.tech.chapter6.txtedit.texteditor";

    private Text text;

    private boolean dirty;

    private FileInput input;
```



```
/**
 *
 */
public SampleTextEditor() {
}

/**
 * (non-Javadoc)
 *
 * @see
 * org.eclipse.ui.part.WorkbenchPart#createPartControl(org.eclipse.swt.widgets
 * .Composite)
 */
@Override
public void createPartControl(Composite parent) {

    text = new Text(parent, SWT.BORDER | SWT.WRAP | SWT.H_SCROLL
        | SWT.CANCEL | SWT.MULTI);
    loadText();
    text.addModifyListener(new ModifyListener() {

        @Override
        public void modifyText(ModifyEvent e) {
            dirty = true;
            firePropertyChange(ISaveablePart2.PROP_DIRTY);
        }
    });
}

/**
 * (non-Javadoc)
 *
 * @see org.eclipse.ui.part.EditorPart#doSave(org.eclipse.core.runtime.
```

```
    * IProgressMonitor)
    */
    @Override
    public void doSave(IProgressMonitor monitor) {

        saveToLocal(input.getName());
        dirty = false;
        firePropertyChange(ISaveablePart2.PROP_DIRTY);

    }

    /**
     * (non-Javadoc)
     *
     * @see org.eclipse.ui.part.EditorPart#doSaveAs()
     */
    @Override
    public void doSaveAs() {
        FileDialog dialog = new FileDialog(getEditorSite().getShell(), SWT.SAVE);
        String path = dialog.open();
        if (path != null) {
            saveToLocal(path);
        }
    }

    /**
     * (non-Javadoc)
     *
     * @see org.eclipse.ui.part.EditorPart#init(org.eclipse.ui.IEditorSite,
     * org.eclipse.ui.IEditorInput)
     */
    @Override
    public void init(IEditorSite site, IEditorInput input)
        throws PartInitException {
```

```
    setSite(site);
    setInput(input);
    setPartName(input.getName());
    this.input = (FileInput) input;
}

/*
 * (non-Javadoc)
 *
 * @see org.eclipse.ui.part.EditorPart#isDirty()
 */
@Override
public boolean isDirty() {
    return dirty;
}

/*
 * (non-Javadoc)
 *
 * @see org.eclipse.ui.part.EditorPart#isSaveAsAllowed()
 */
@Override
public boolean isSaveAsAllowed() {
    return true;
}

/**
 * 载入文件内容
 */
private void loadText() {
    try {
        BufferedReader reader = new BufferedReader(new InputStreamReader(
            new FileInputStream(input.getName()), "utf-8"));
        StringBuffer sb = new StringBuffer();
        String line = reader.readLine();
```

```
        while (line != null) {
            sb.append(line);
            sb.append("\n");
            line = reader.readLine();
        }
        reader.close();
        text.setText(sb.toString());
    } catch (FileNotFoundException e) {
    } catch (UnsupportedEncodingException e) {
    } catch (IOException e) {
    }
}

@Override
public int promptToSaveOnClose() {
    if(dirty){
        if(MessageDialog.openConfirm(getEditorSite().getShell(),
            "提示", "内容未保存，确认继续关闭操作? ")){
            return ISaveablePart2.NO;
        }else{
            return ISaveablePart2.CANCEL;
        }
    }
    return YES; }

/**
 * 保存文件内容到本地
 * @param path
 */
private void saveToLocal(String path) {
    try {
        OutputStream out = new FileOutputStream(path);
        OutputStreamWriter writer = new OutputStreamWriter(out, "utf-8");
        writer.write(text.getText());
        writer.close();
    }
```

```
        out.close();
    } catch (FileNotFoundException e) {
    } catch (UnsupportedEncodingException e) {
    } catch (IOException e) {
    }
}

/*
 * (non-Javadoc)
 *
 * @see org.eclipse.ui.part.WorkbenchPart#setFocus()
 */
@Override
public void setFocus() {
    text.setFocus();
}
}
```

讲解一下几个比较重要的方法：

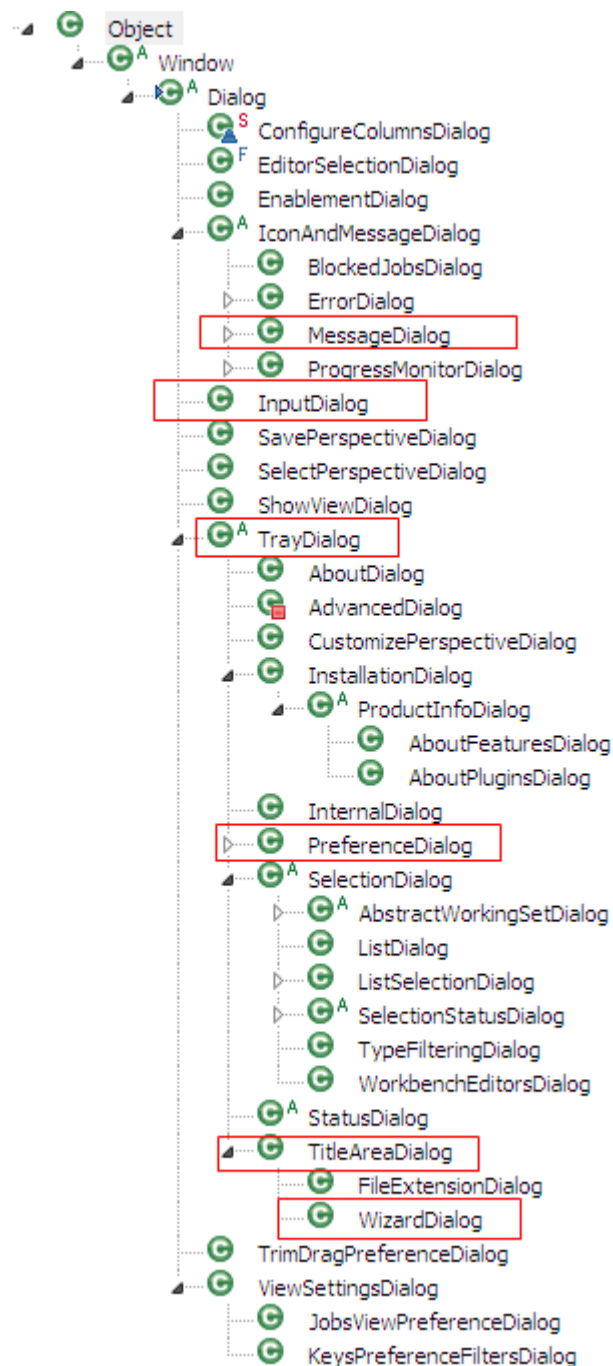
- **isDirty()**：在 **ISaveablePart** 中定义，用来表示编辑器是需要保存，返回 **true** 表明需要保存，否则不需要。与 Eclipse 内置的 **Save** 菜单绑定，当 **isDirty()** 返回 **true** 时候，**Save** 菜单可用，否则不可用。
- **doSave(...)**：在 **ISaveablePart** 中定义，进行编辑器内容保存的方法，点击 **Save** 菜单将触发此方法。
- **doSaveAs()**：在 **ISaveablePart** 中定义，用于另存文件，点击 **Save As** 菜单时触发此方法。
- **isSaveAsAllowed()**：在 **ISaveablePart** 中定义，用于启用或者禁用 **Save As** 菜单。
- **promptToSaveOnClose()**：在 **ISaveablePart2** 定义，用于在关闭编辑器时候提示操作，这里在内容未保存的时候提示操作

完整的实例见附件“org.salever.rcp.tech.chapter6.txtedit”。

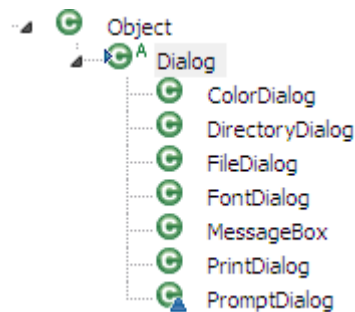
7 对话框

7.1 概述

对话框（Dialog），是 Eclipse 平台提供的最为灵活的组件之一，它无处不在。下面给出 org.eclipse.jface.dialogs.Dialog 的类型层次结构图：



再看看 `org.eclipse.swt.widgets.Dialog` 的类型层次结构：



Eclipse 提供了许多预定义的对话框，常见的有：

- `org.eclipse.swt.widgets.FileDialog`
- `org.eclipse.swt.widgets.ColorDialog`
- `org.eclipse.swt.widgets.DirectoryDialog`
- `org.eclipse.swt.widgets.FontDialog`
- `org.eclipse.swt.widgets.PrintDialog`
- `org.eclipse.jface.dialogs.ErrorDialog`
- `org.eclipse.jface.dialogs.InputDialog`
- `org.eclipse.jface.dialogs.MessageDialog`

Eclipse 同时也支持用户自定义的对话框。通过继承类 `Dialog` 或者其他对话框得到新的对话框。

7.2 预定义的对话框

7.2.1 概述

接下来将描述如何使用预定义的对话框；

7.2.2 创建工程

使用“Hello RCP”模板创建一个新工程 `org.salever.rcp.tech.chapter7`；

7.2.3 添加菜单

添加 `org.eclipse.ui.actionSets` 扩展点，使用“Hello, world” action set 迅速创建一个菜单栏。

7.2.4 调用对话框

修改 `SampleAction.java`，修改 `run` 方法如下：

```
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 ocured");
MessageDialog
    .openInformation(window.getShell(), "Info", "Info for you");
```



```
    MessageDialog.openQuestion(window.getShell(), "Question",  
        "Really, really?");  
    MessageDialog.openWarning(window.getShell(), "Warning", "I warn you");  
}
```

运行程序。如果点击菜单，将依次弹出一些标准对话框，当然你也可以在方法里面添加更多的对话框调用，方法类似。

7.3 用户自定义对话框

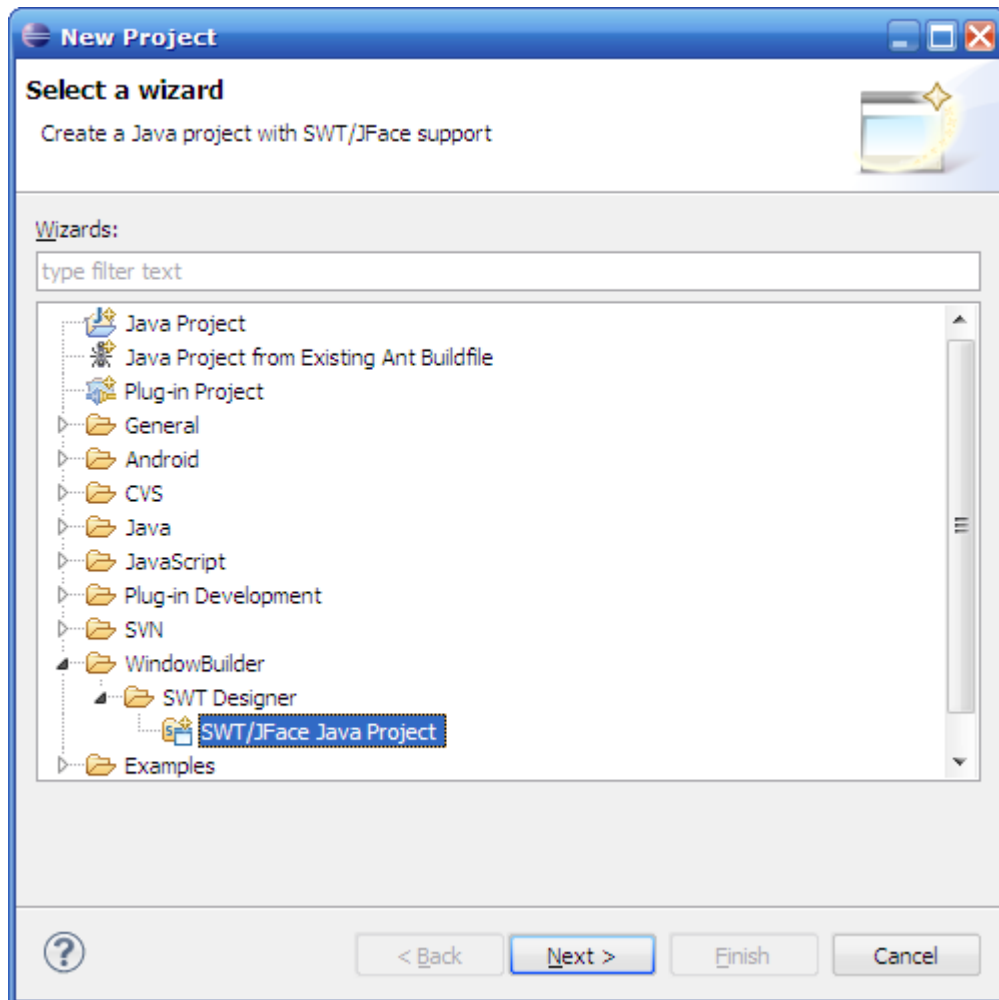
7.3.1 概述

接下来将描述如何自定义对话框，以及如何使用它。

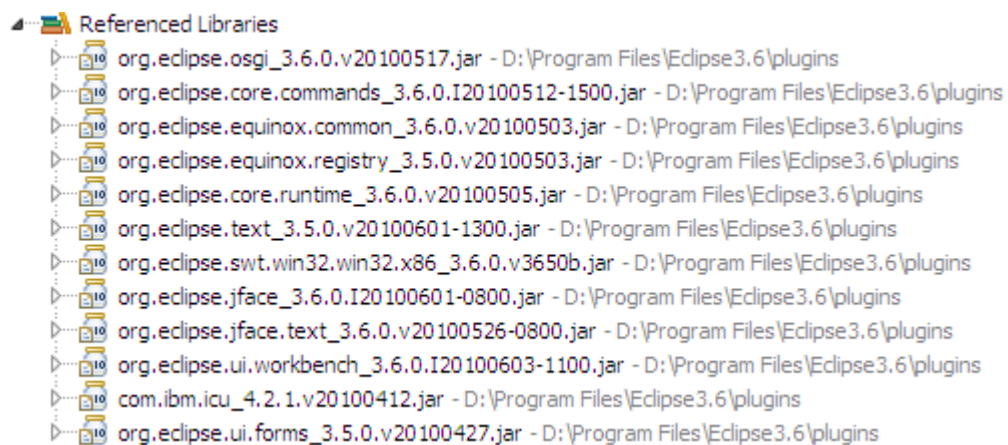
7.3.2 创建 SWT/JFace 工程

至此我们创建的都是 `plug-in` 工程，而且都是 `Application`，其实 `Dialog` 可以脱离于 `RCP` 环境单独运行，只要给它配置好所有依赖的 `jar`，`SWT/Jface` 程序可以单独运行。

这里我使用了 `WindowBuilder` 插件（以前也叫做 `SWT Designer`，现在被 `Google` 收购并且开源了，利用此工具可以大大提高开发 `SWT/JFace` 界面程序的效率），创建一个 `SWT/JFace Java Project`: `org.salever.rcp.tech.chapter7.dialog`，如果没有安装此插件，可以先创建一个 `Java` 工程，然后在 `build path` 中添加一些插件依赖。



一般的一个 SWT/JFace 工程需要的插件为：



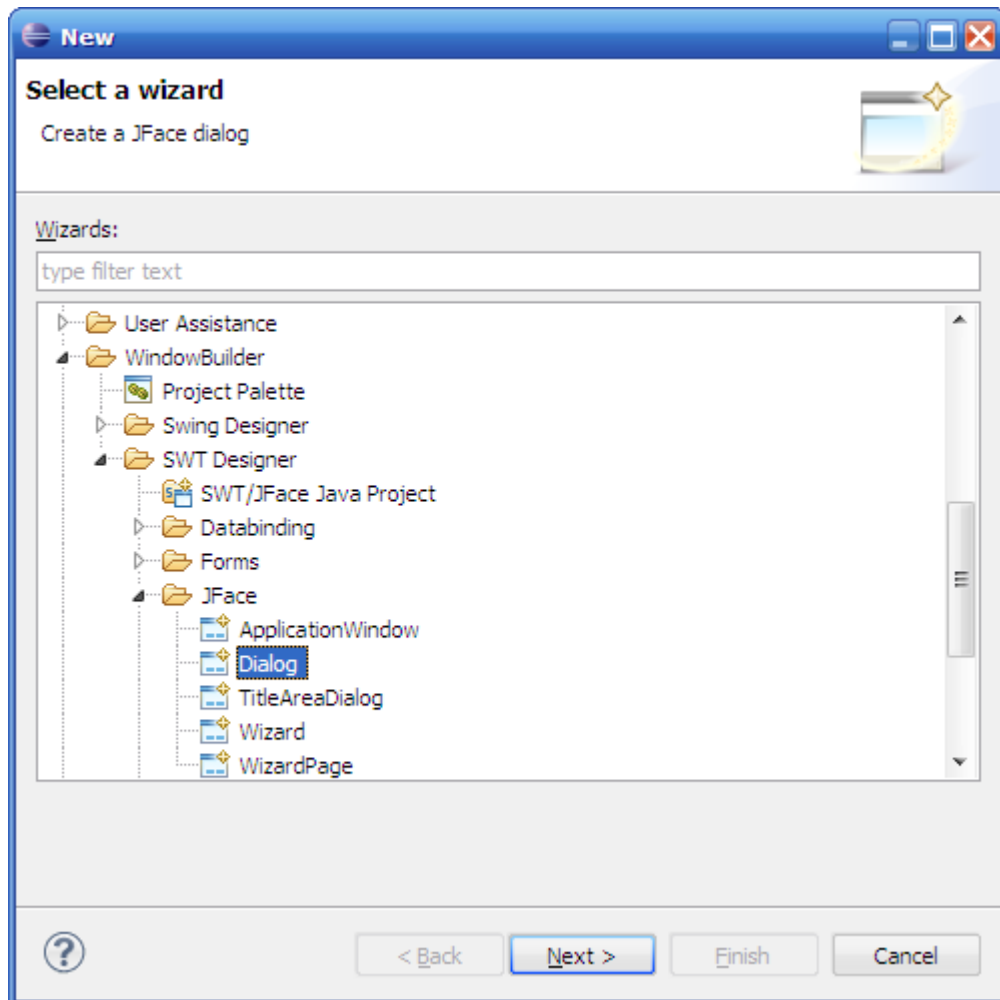
这些插件如果缺失了，可以无法正确运行程序。一般的，它们都在 Eclipse 安装目录的 plugins 找到。注意由于机器上安装的 Eclipse 的位置的不同，附件的源码可以会有错误，你可以打

开.classpath 文件，然后替换一下相关的路径就行了。

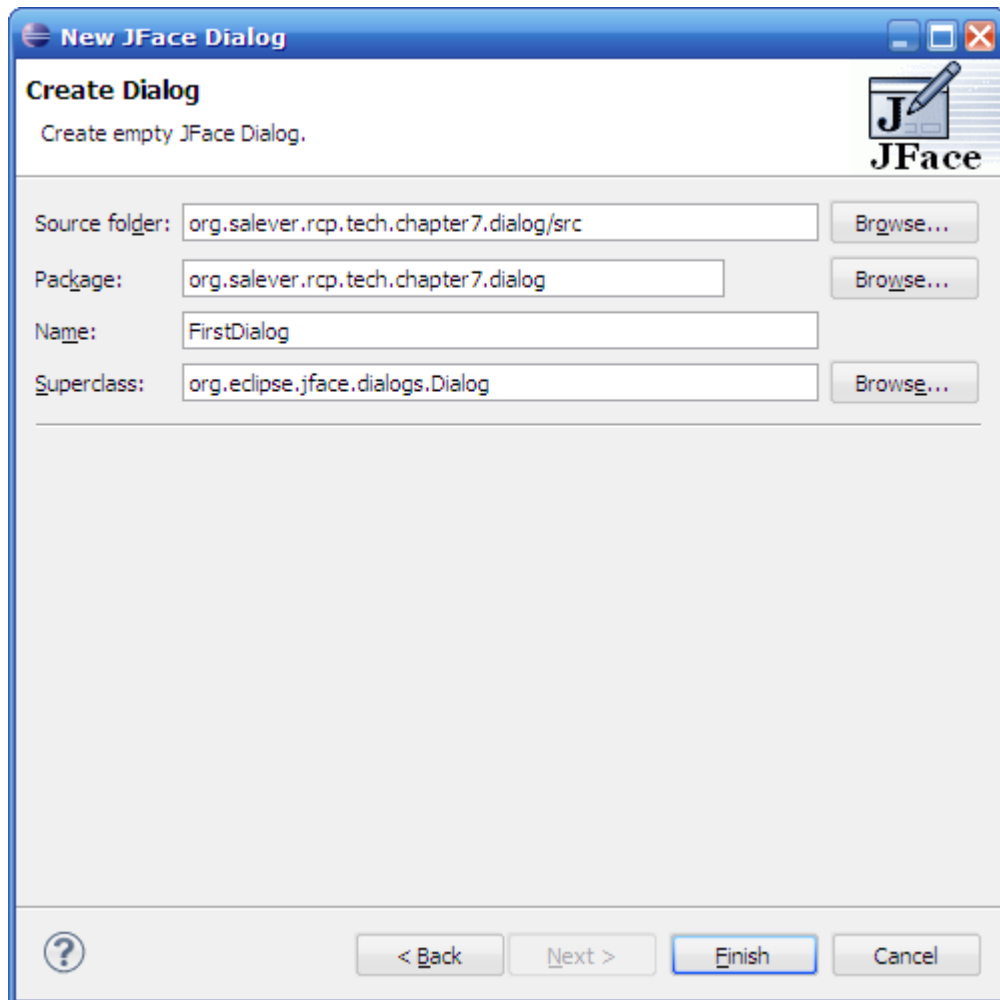
```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
  <classpathentry kind="src" path="src"/>
  <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.la
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.osgi_3.6.0.v20100517.jar" so
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.core.commands_3.6.0.I2010051
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.equinox.common_3.6.0.v201005
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.equinox.registry_3.5.0.v2010
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.core.runtime_3.6.0.v20100505
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.text_3.5.0.v20100601-1300.ja
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.swt.win32.win32.x86_3.6.0.v3
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.jface_3.6.0.I20100601-0800.j
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.jface.text_3.6.0.v20100526-0
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.ui.workbench_3.6.0.I20100603
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/com.ibm.icu_4.2.1.v20100412.jar" sourcecp
  <classpathentry kind="lib" path="D:/Program Files/Eclipse3.6/plugins/org.eclipse.ui.forms_3.5.0.v20100427.jar
  <classpathentry kind="output" path="bin"/>
</classpath>
```

7.3.3 自定义 Dialog

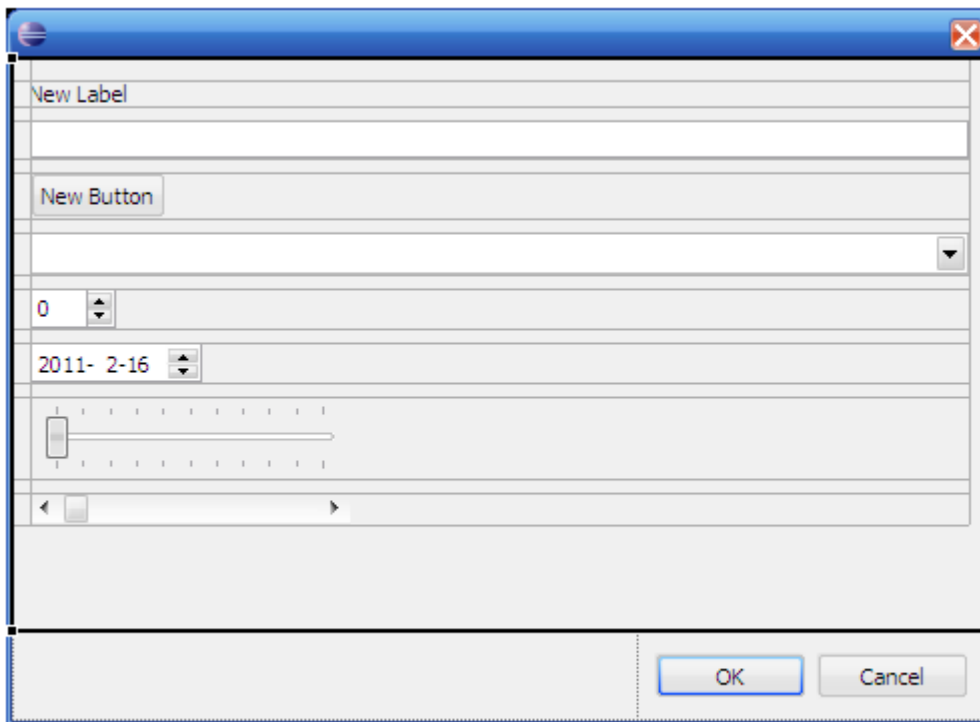
这里我们仍然使用 WindowBuilder，创建自定义 Dialog。



新建一个类 `FirstDialog.java`，位于包 `org.salever.rcp.tech.chapter7.dialog` 下，它继承自 `org.eclipse.jface.dialogs.Dialog` 类。



确定以后，转到打开的 `FirstDialog.java`，点击下方的 **Design** 标签，可以发现一个可视化的编辑界面，这时候就可以轻松定义对话框了。



编辑完毕，在类中添加 `main` 方法，就可以直接运行了。

```
package org.salever.rcp.tech.chapter7.dialog;
```

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

```
public class FirstDialog extends Dialog {  
    private Text text;  
  
    /**  
     * Create the dialog.  
     * @param parentShell  
     */  
    public FirstDialog(Shell parentShell) {  
        super(parentShell);  
    }  
  
    /**  
     * Create contents of the dialog.
```

```
* @param parent
*/
@Override
protected Control createDialogArea(Composite parent) {
    Composite container = (Composite) super.createDialogArea(parent);

    Label label = new Label(container, SWT.NONE);
    label.setText("New Label");

    Text text = new Text(container, SWT.BORDER);
    text.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));

    Button button = new Button(container, SWT.NONE);
    button.setText("New Button");

    Combo combo = new Combo(container, SWT.NONE);
    combo.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));

    Spinner spinner = new Spinner(container, SWT.BORDER);

    DateTime dateTime = new DateTime(container, SWT.BORDER);

    Scale scale = new Scale(container, SWT.NONE);

    Slider slider = new Slider(container, SWT.NONE);

    return container;
}

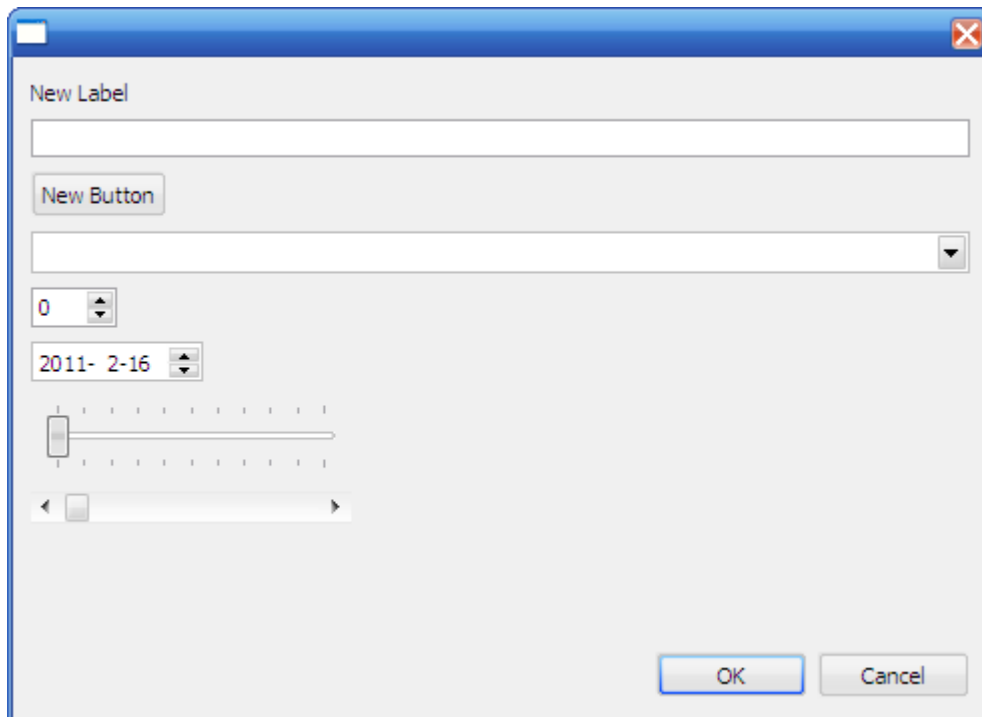
/**
 * Create contents of the button bar.
 * @param parent
 */
@Override
protected void createButtonsForButtonBar(Composite parent) {
```

```
        createButton(parent, IDialogConstants.OK_ID, IDialogConstants.OK_LABEL,
            true);
        createButton(parent, IDialogConstants.CANCEL_ID,
            IDialogConstants.CANCEL_LABEL, false);
    }

    /**
     * Return the initial size of the dialog.
     */
    @Override
    protected Point getInitialSize() {
        return new Point(493, 359);
    }

    public static void main(String[] args){
        new FirstDialog(null).open();
    }
}
```

运行效果为：



7.3.4 说明

使用可视化的编辑工具来进行 **SWT/JFace** 的开发是一个选择，当然你可以使用手工编写的方式开发。不过对于初学者，还是建议先使用工具，了解完毕了所有的控件、布局的概念和用法以后，再脱离组件。这样入手会快一些。

笔者一般先用 **SWT Designer** 进行布局，然后再改写它自动生成的代码，这样可以提高开发效率。

8 向导

8.1 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。

8.2 示例

创建一个新工程 “org.salever.rcp.tech.chapter7”，使用 “Hello RCP” 模板。

添加 org.eclipse.ui.actionSets 扩展点，使用 “Hello, world” action set 迅速创建一个菜单栏。

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。

MyPageOne.java:

```
package org.salever.rcp.tech.chapter8.wizard;
```

```
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.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
// AssertionFailedException
// This method should always return the top widget of the application

public Control getControl() {
    return container;
}

}
```

MyPageTwo.java:

```
package org.salever.rcp.tech.chapter8.wizard;

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.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
// AssertionFailedException
// This method should always return the top widget of the application

public Control getControl() {
    return container;
}
}
```

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

```
package org.salever.rcp.tech.chapter8.wizard;
```

```
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() {
        MessageDialog.openInformation(getShell(), "信息",
            "向导页 1 内填写了" + one.getText1() + "\n 向导页 2 内填写了" + two.getText1());
        return true;
    }
}
```

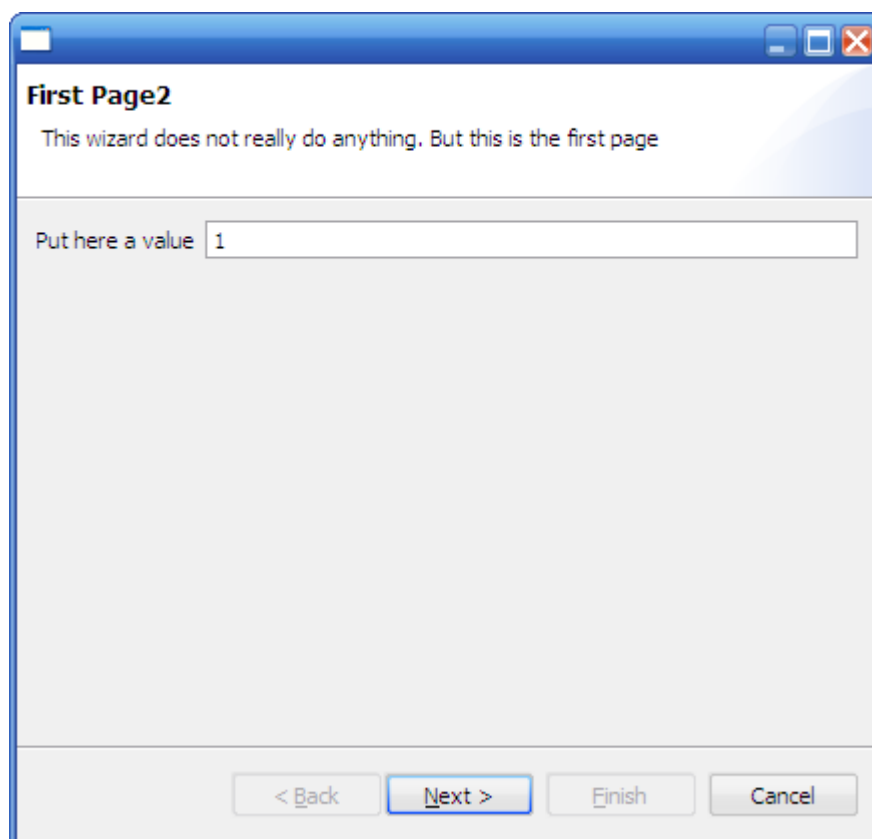
最后调用打开向导，修改 org.salever.rcp.tech.chapter8.actions.SampleAction.java 的 run 方法：

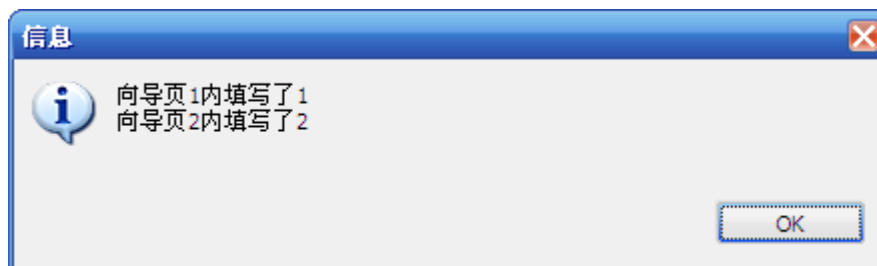
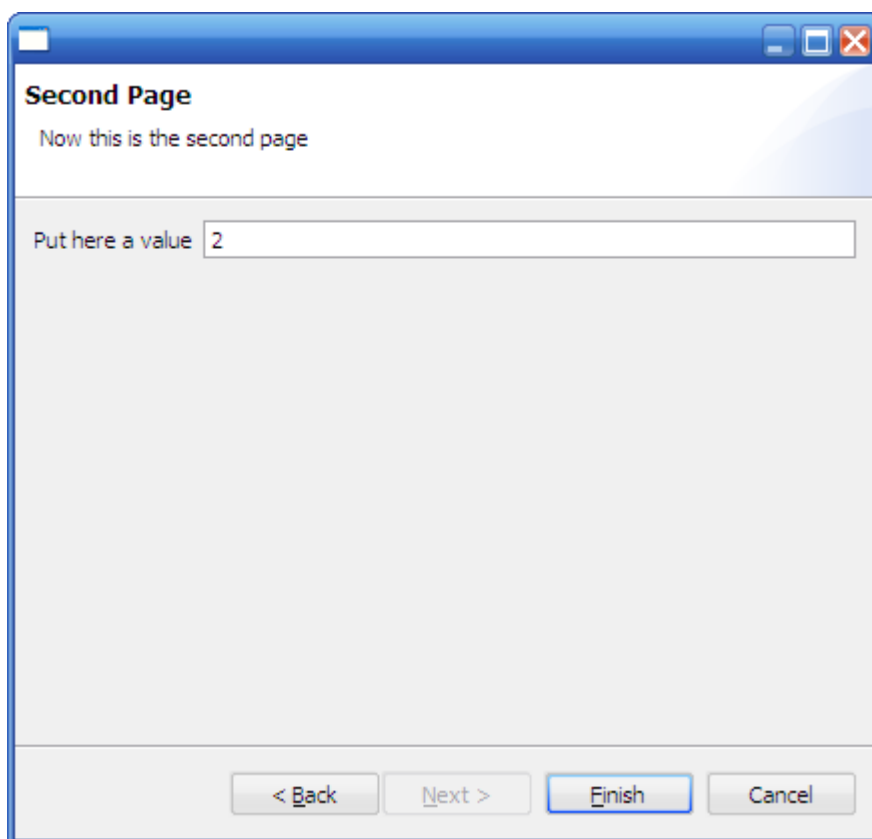
```
public void run(IAction action) {
    MyWizard wizard = new MyWizard();
    WizardDialog dialog = new WizardDialog(window.getShell(), wizard);
```

```
    dialog.open();
```

```
}
```

运行效果为：





9 首选项

9.1 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。

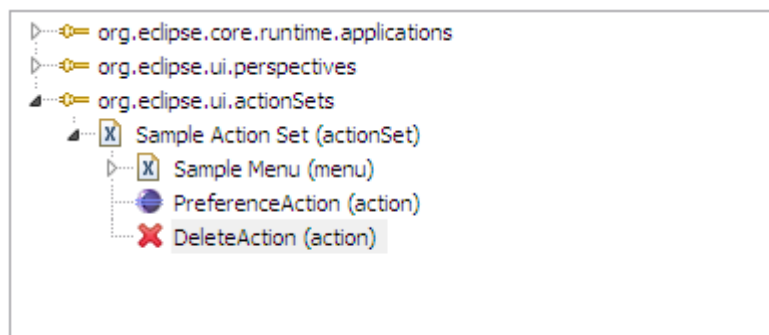
你能使用以下代码存储优先项

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

9.2 使用首选项

使用“Hello RCP”模板，创建一个新工程“`org.salever.rcp.tech.chapter8`”。

添加 `org.eclipse.ui.actionSets` 扩展点，使用“Hello, world” action set 迅速创建一个菜单栏，然后添加两个菜单项，分别命名为 `PreferenceAction` 和 `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，然后删除它们。

PreferenceAction.java:

```
package org.salever.rcp.tech.chapter9.actions;

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;
import org.salever.rcp.tech.chapter9.Activator;

public class PreferenceAction implements IWorkbenchWindowActionDelegate {
    private IWorkbenchWindow window;
    /**
     * The constructor.
     */
    public PreferenceAction() {
    }

    /**
     * The action has been activated. The argument of the
     * method represents the 'real' action sitting
     * in the workbench UI.
     * @see IWorkbenchWindowActionDelegate#run
     */
    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(Activator.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",
    "default"));

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

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

// Forces the application to save the preferences

try {
    preferences.flush();

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

```
    }

    /**
     * Selection in the workbench has been changed. We
     * can change the state of the 'real' action here
     * if we want, but this can only happen after
     * the delegate has been created.
     * @see IWorkbenchWindowActionDelegate#selectionChanged
     */
    public void selectionChanged(IAction action, ISelection selection) {
    }

    /**
     * We can use this method to dispose of any system
     * resources we previously allocated.
     * @see IWorkbenchWindowActionDelegate#dispose
     */
    public void dispose() {
    }

    /**
     * We will cache window object in order to
     * be able to provide parent shell for the message dialog.
     * @see IWorkbenchWindowActionDelegate#init
     */
    public void init(IWorkbenchWindow window) {
        this.window = window;
    }
}
```

DeleteAction.Java:

```
package org.salever.rcp.tech.chapter9.actions;

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;
import org.salever.rcp.tech.chapter9.Activator;

public class DeleteAction implements IWorkbenchWindowActionDelegate {
    private IWorkbenchWindow window;

    @Override
    public void run(IAction action) {
        Preferences preferences = new ConfigurationScope()
            .getNode(Activator.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", "default"));

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

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

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

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

}

@Override
public void selectionChanged(IAction action, ISelection selection) {

}

@Override
public void dispose() {

}

@Override
```

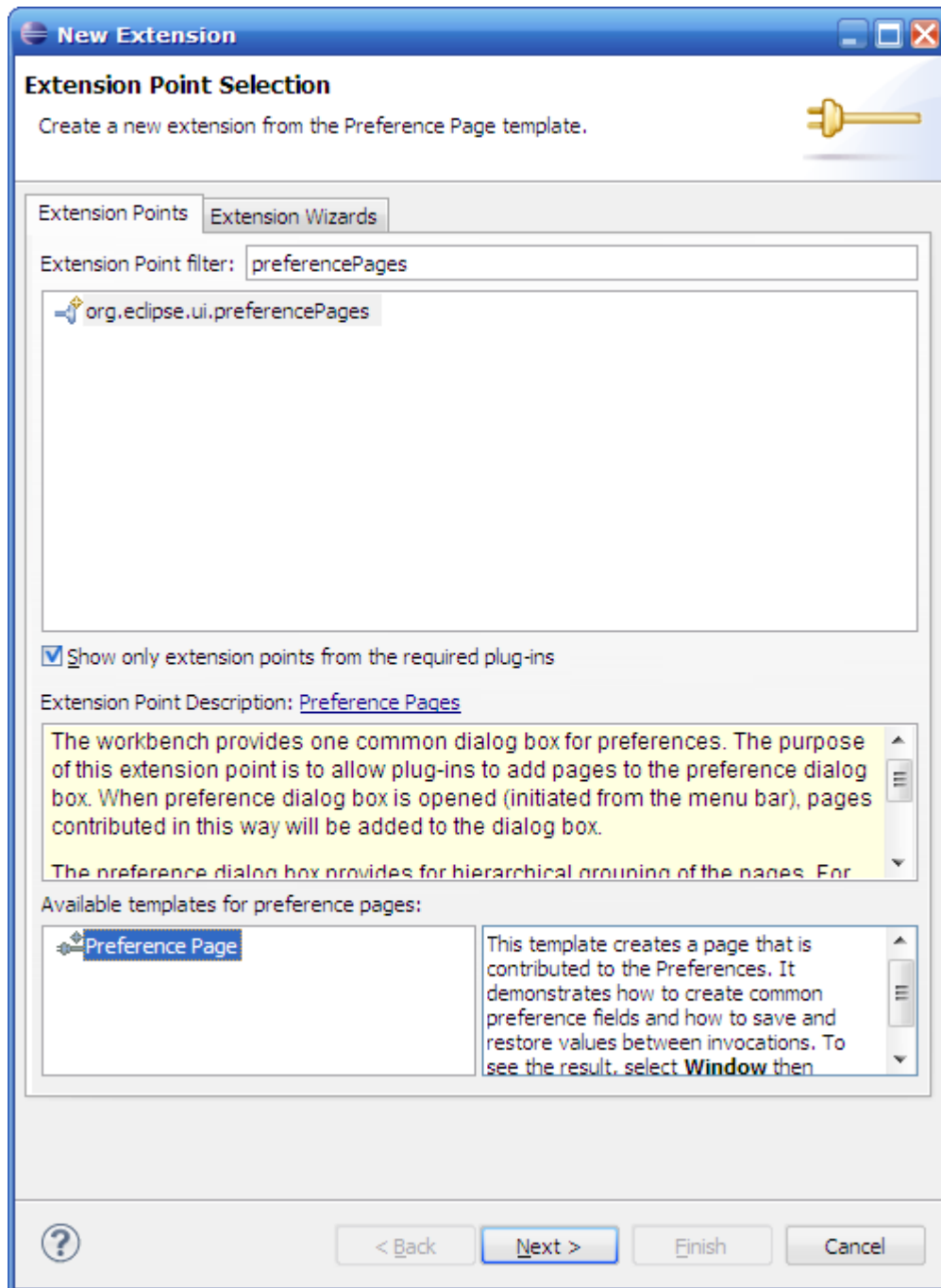
```
public void init(IWorkbenchWindow window) {  
    this.window = window;  
  
}  
  
}
```

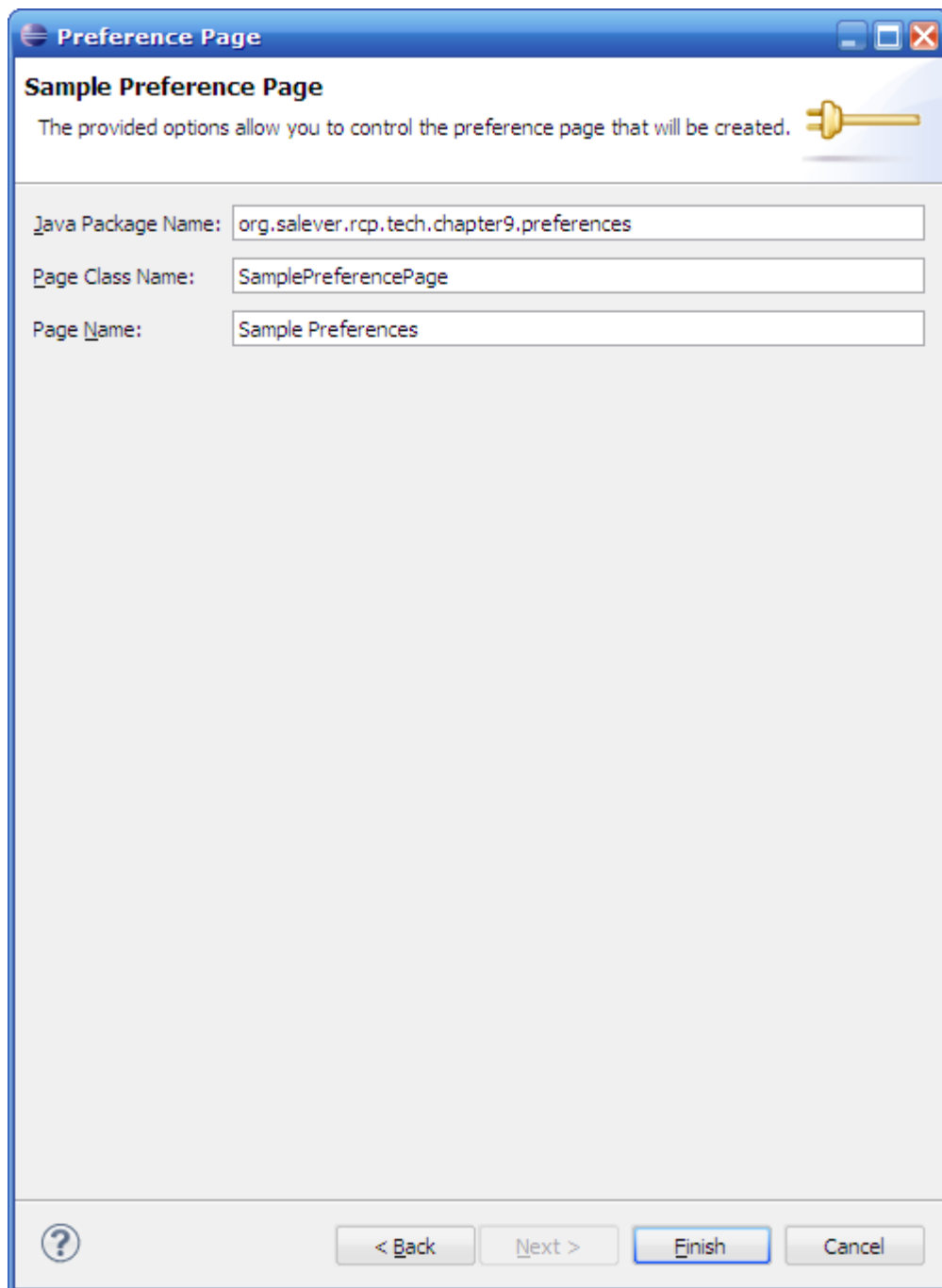
运行一下，可以看到首选项设置和删除的效果。

9.3 首选项页

RCP 开发中更多的是使用首选项页。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`。

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





修改 `ApplicationActionBarAdvisor.java`，向你的菜单添加首选项菜单。

```
package org.salever.rcp.tech.chapter9;
```

```
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;

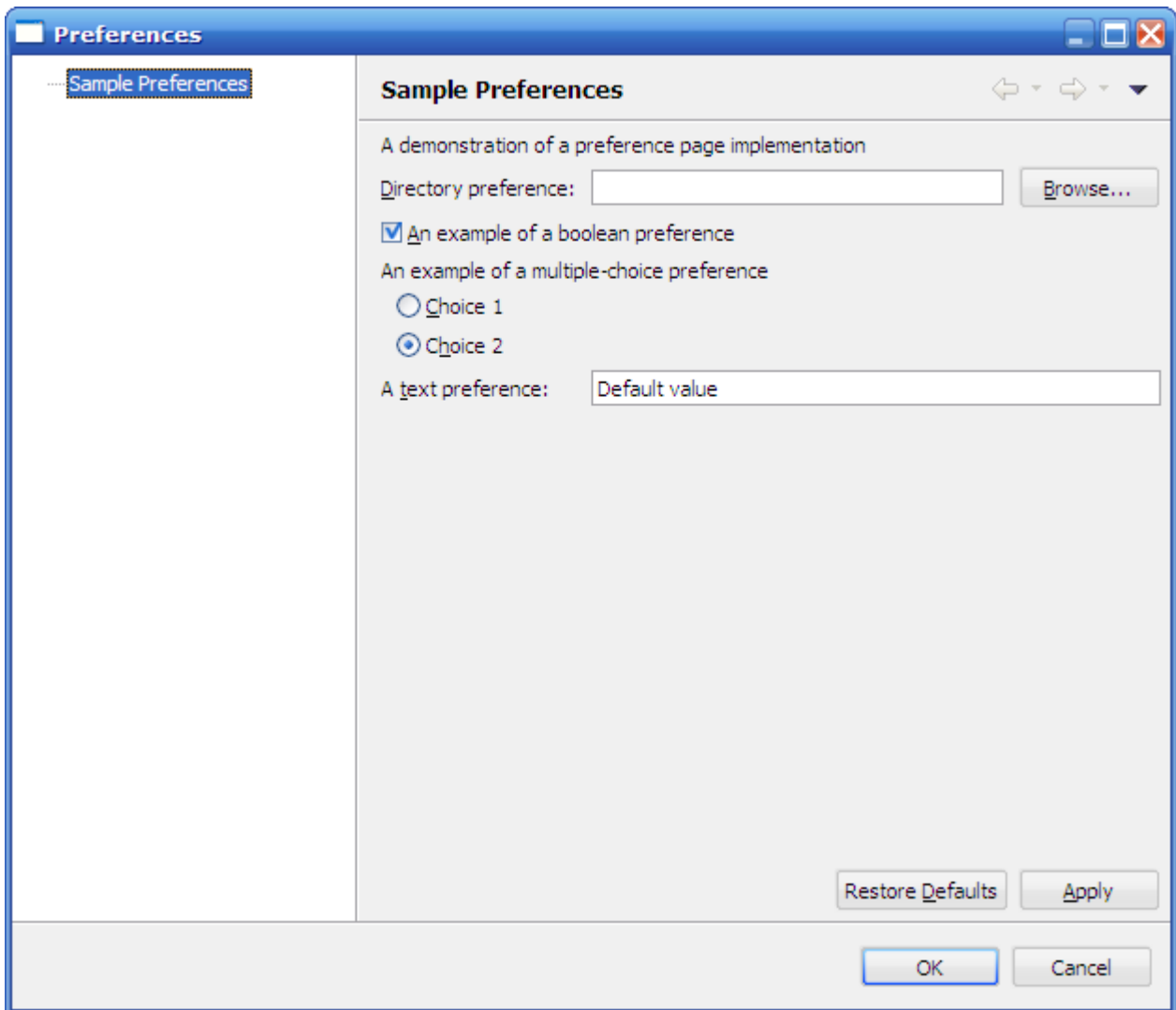
public class ApplicationActionBarAdvisor extends ActionBarAdvisor {
    private IWorkbenchAction preferenceAction;

    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);
    }
}
```

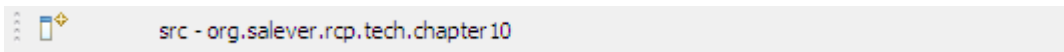
运行程序，你可以通过菜单选择你的首选项。



10 添加状态栏

10.1 简介

状态栏 (StatusLine)，指的是 RCP 主界面下方的信息提示栏，这里可以提示一些正在运行的操作，系统状态等信息。Eclipse 的状态栏如下图：



10.2 安装状态栏

使用 “Hello RCP”，创建一个新工程 “org.salever.rcp.tech.chapter10”。

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

```
package org.salever.rcp.tech.chapter10;

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;

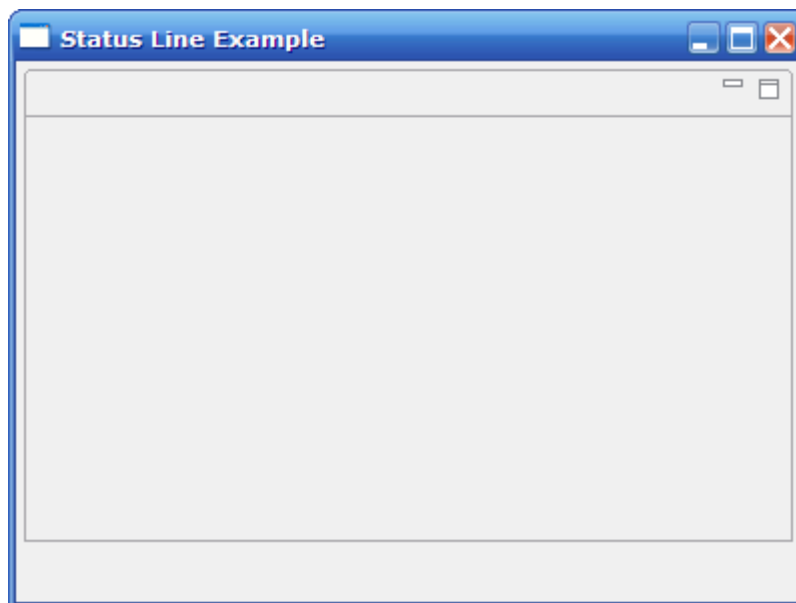
public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    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。



10.3 初始化状态条

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。

```
package org.salever.rcp.tech.chapter10;
```

```
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);
    }

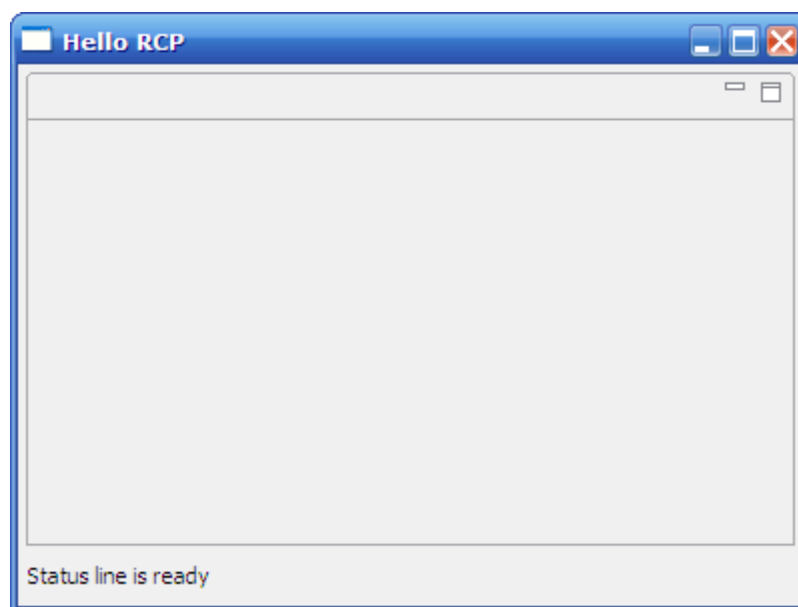
    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowCoolBar(false);
        configurer.setShowStatusLine(true);
        configurer.setTitle("Hello RCP");
    }

    // This is the new method

    public void postWindowOpen() {
```

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

运行结果为：



10.4 控制状态栏

下面考虑通过其他组件修改状态栏信息，使用 `Sample View` 添加一个 `View` 到程序中，修改 `plugin.xml` 中的相关代码使其显示：

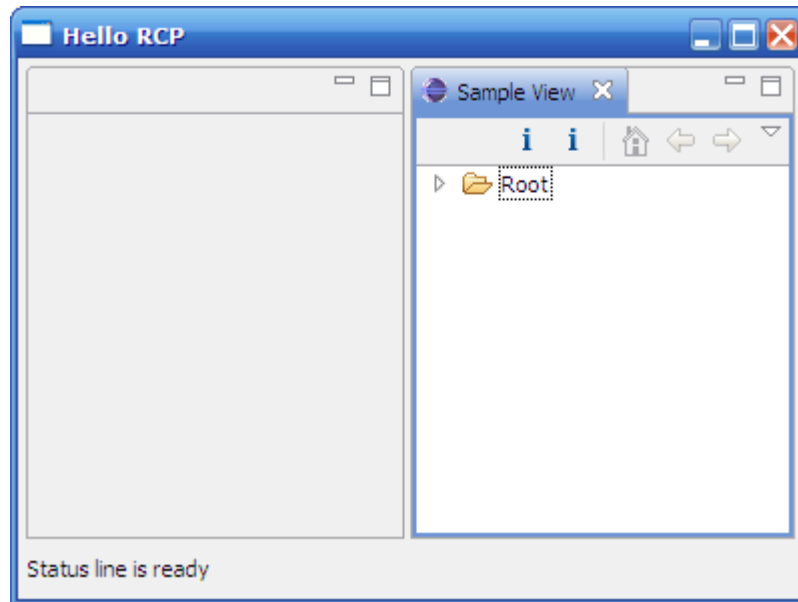
```
<extension
    point="org.eclipse.ui.perspectiveExtensions">
    <perspectiveExtension
        targetID="org.salever.rcp.tech.chapter10.perspective">
        <view
            id="org.salever.rcp.tech.chapter10.views.SampleView"
            ratio="1.0"
            relationship="right"
            relative="org.eclipse.ui.views.TaskList">
        </view>
    </perspectiveExtension>
</extension>
```



```
</perspectiveExtension>
```

```
</extension>
```

运行效果为：



一旦你有 VIEW 后，你就可以进入状态条。如下

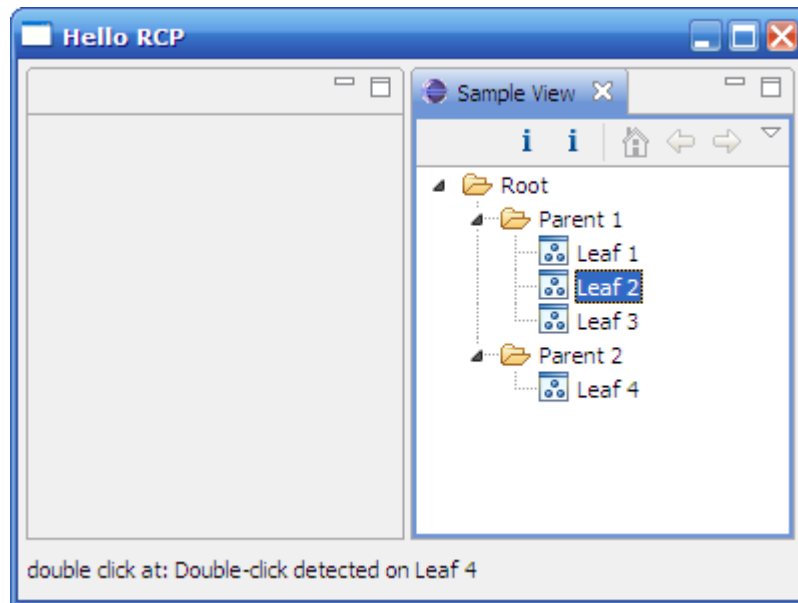
```
IActionBars bars = getViewSite().getActionBars();
```

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

这时候我们修改 SampleView.java 中的双击操作，改为双击修改状态栏信息。

```
private void showMessage(String message) {  
    IActionBars bars = getViewSite().getActionBars();  
    bars.getStatusLineManager().setMessage("double click at: " +message);  
}
```

运行效果为：



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();
```

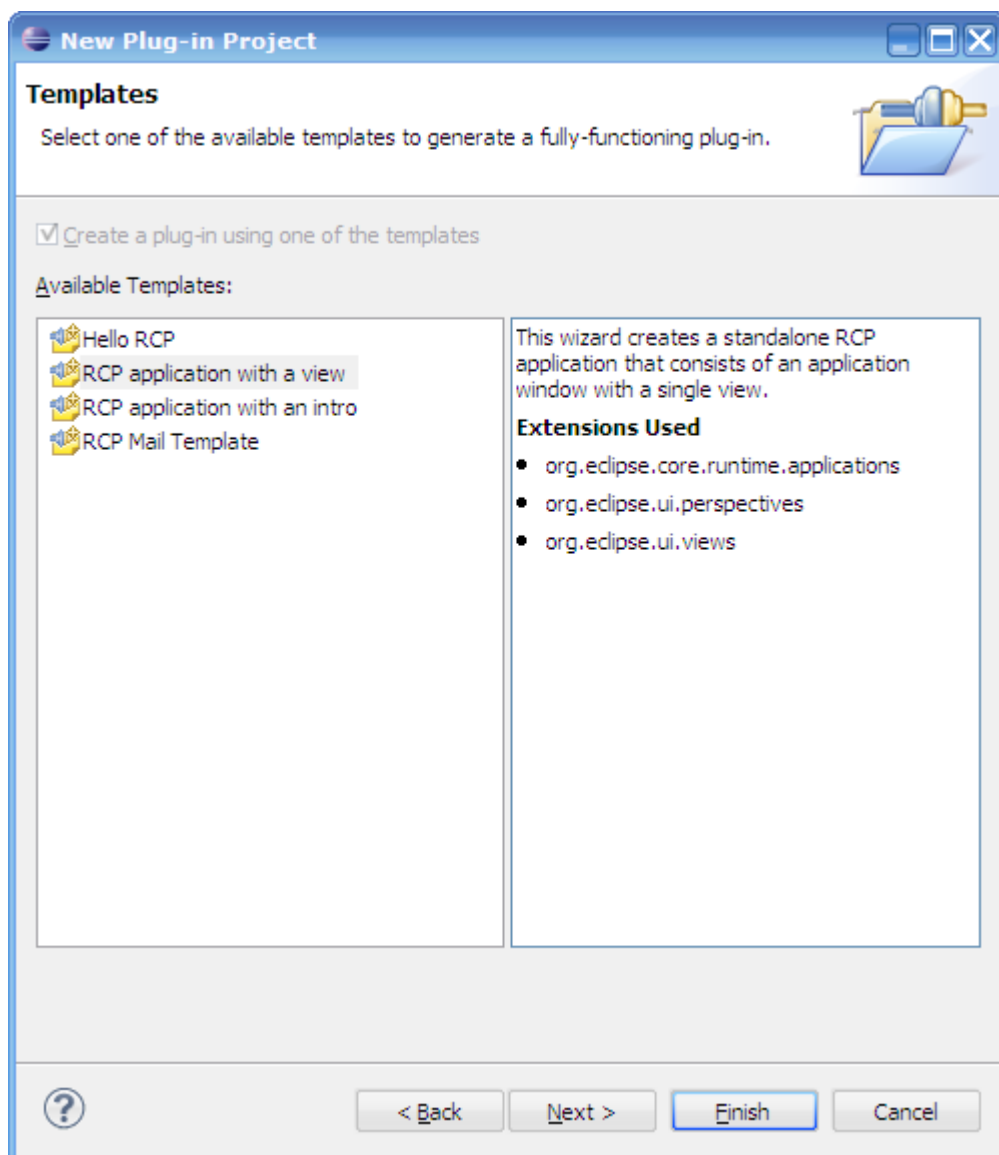
11 透视图

11.1 简介

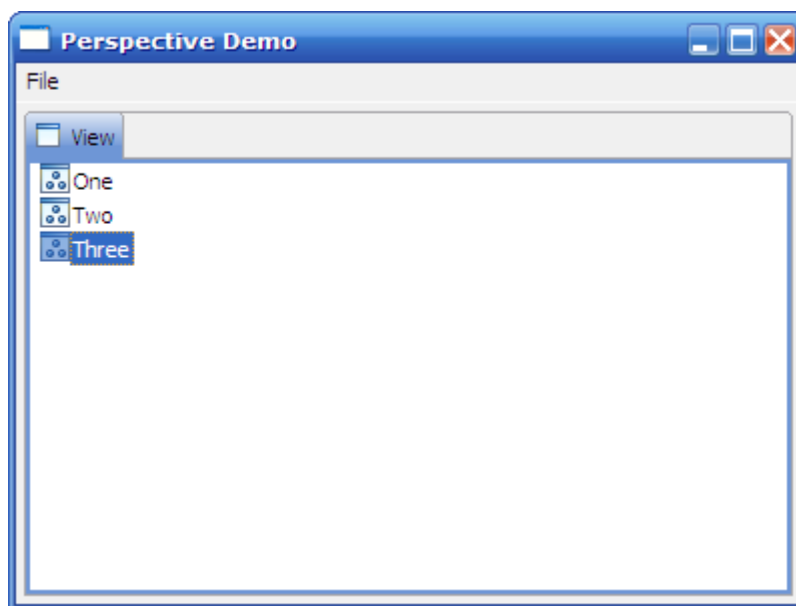
透视图（Perspective）将相关的 UI 元素集合并组织起来，更适合特殊的任务及工作流程。Eclipse RCP 允许你添加透视图到程序中。RCP 程序至少要包含一个透视图，同时 RCP 也提供了一些方法管理和显示更多的透视图。

11.2 添加透视图

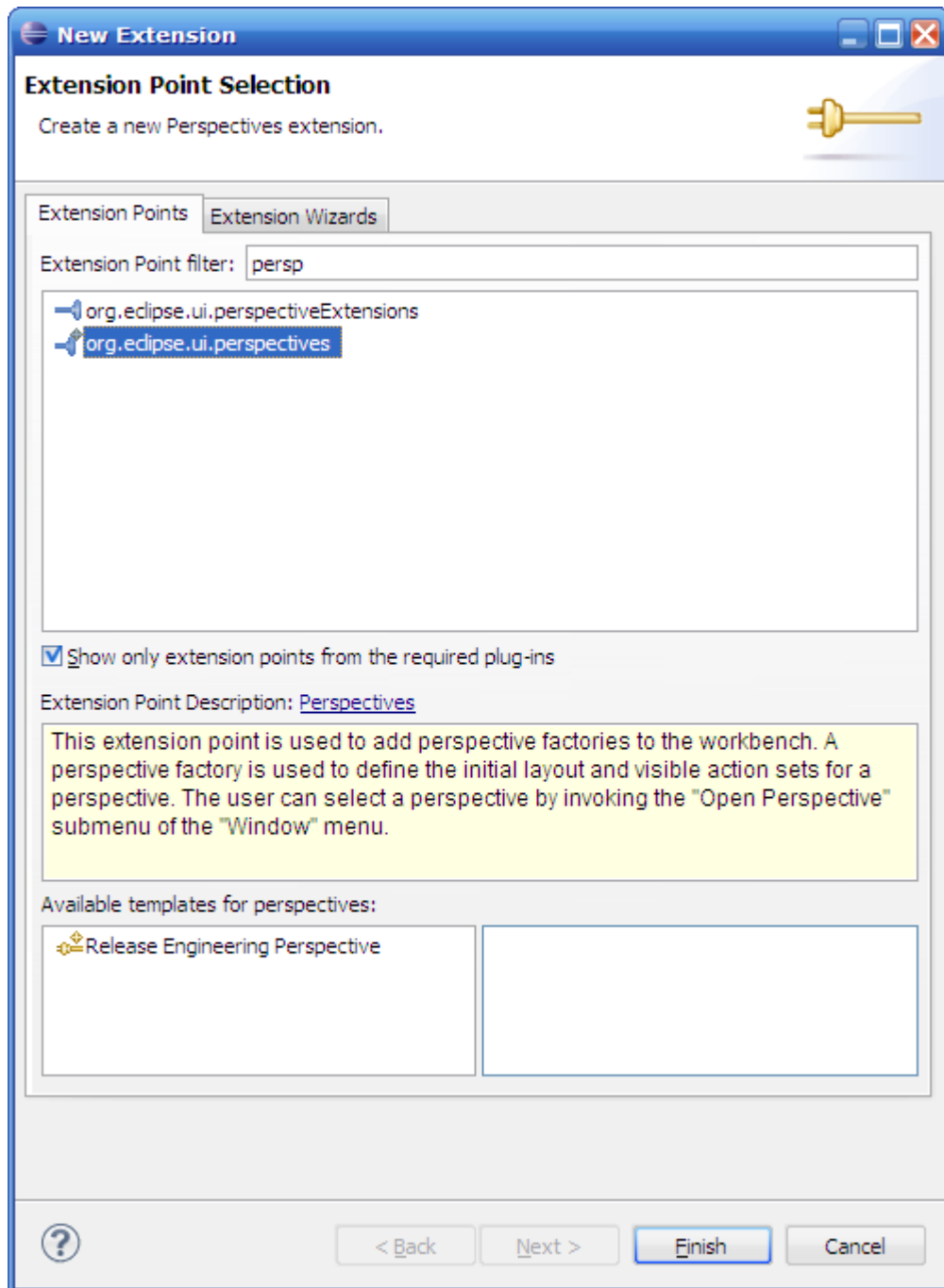
使用“RCP application with a view”模板创建 RCP 工程，命名为“org.salever.rcp.tech.chapter11”。



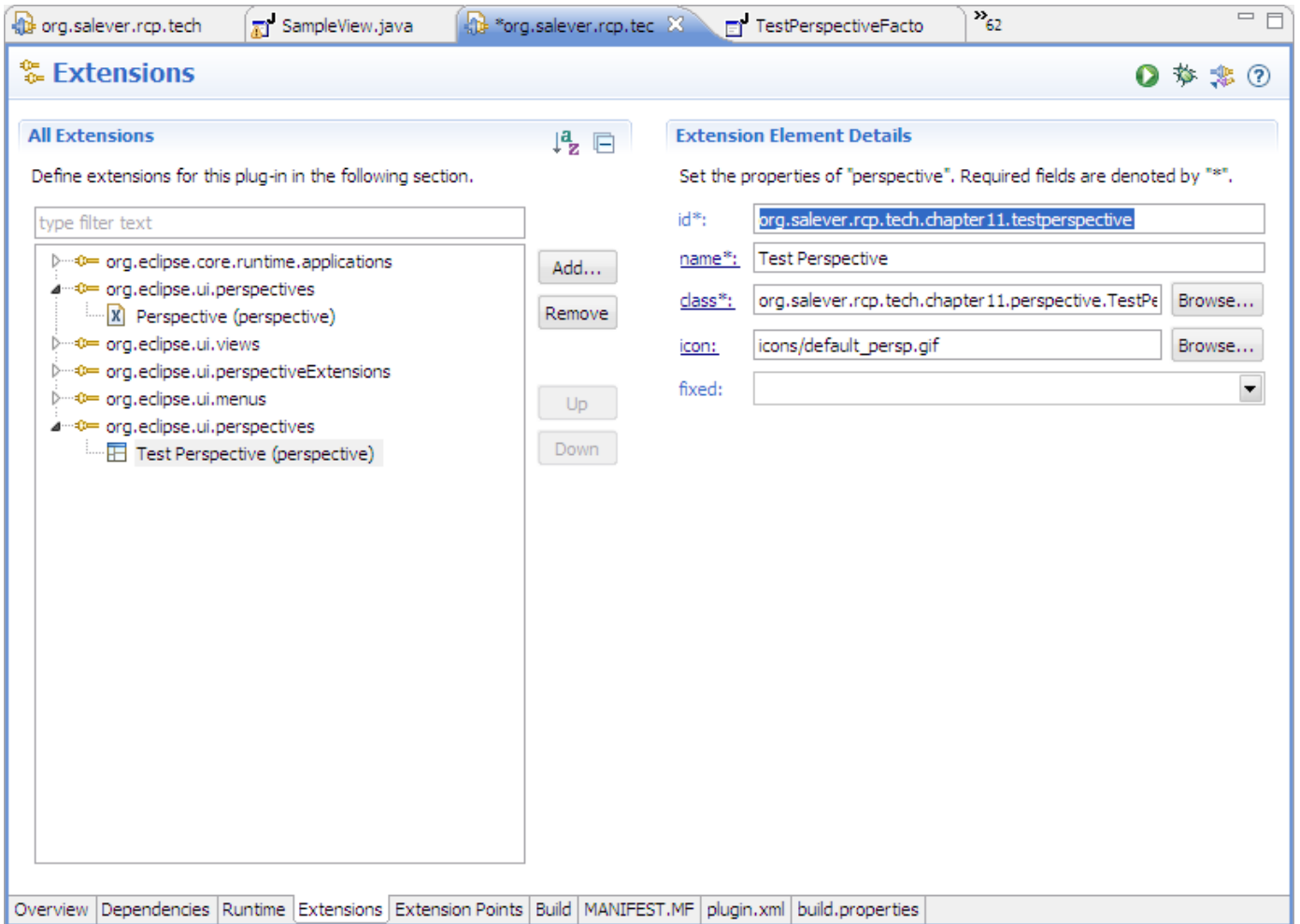
运行，看结果



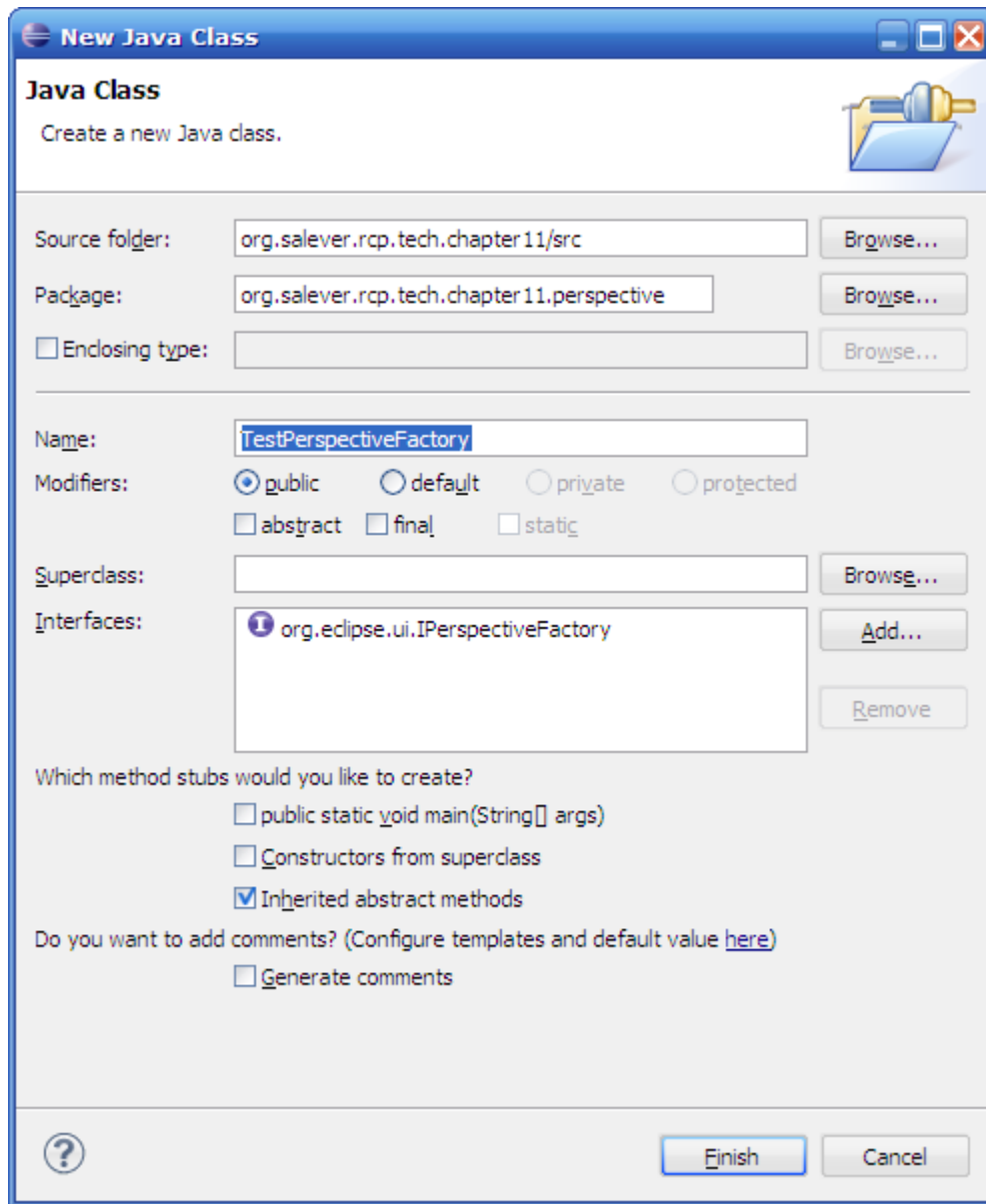
在 `plugin.xml` 里添加一个新的透视图扩展点，



ID 设为“org.salever.rcp.tech.chapter11.testperspective”。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“TestPerspectiveFactory”。



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。

```
package org.salever.rcp.tech.chapter11.perspective;  
import org.eclipse.ui.IPageLayout;  
public class TestPerspectiveFactory implements IPerspectiveFactory {  
  
    @Override  
    public void createInitialLayout(IPageLayout layout) {
```

```
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。

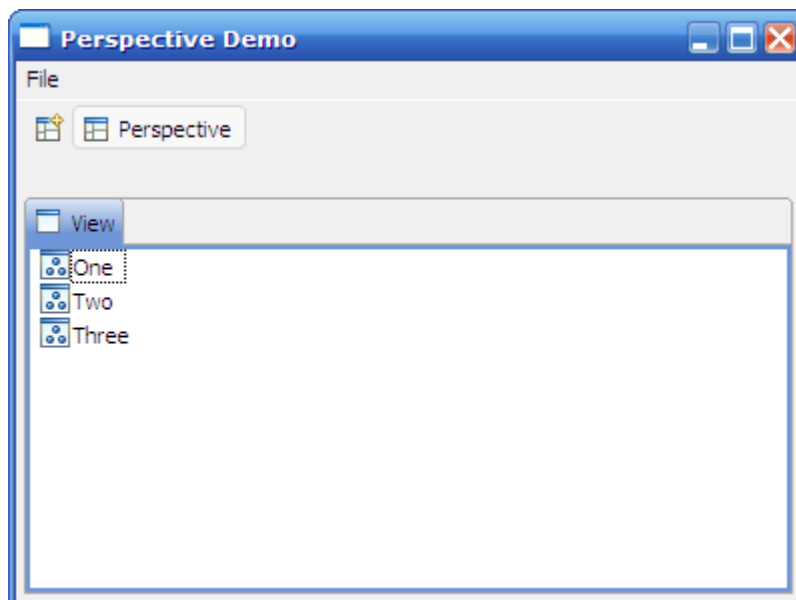
11.3 显示透视图工具栏

修改类 `ApplicationWorkbenchWindowAdvisor`，配置使透视图工具栏菜单可选：

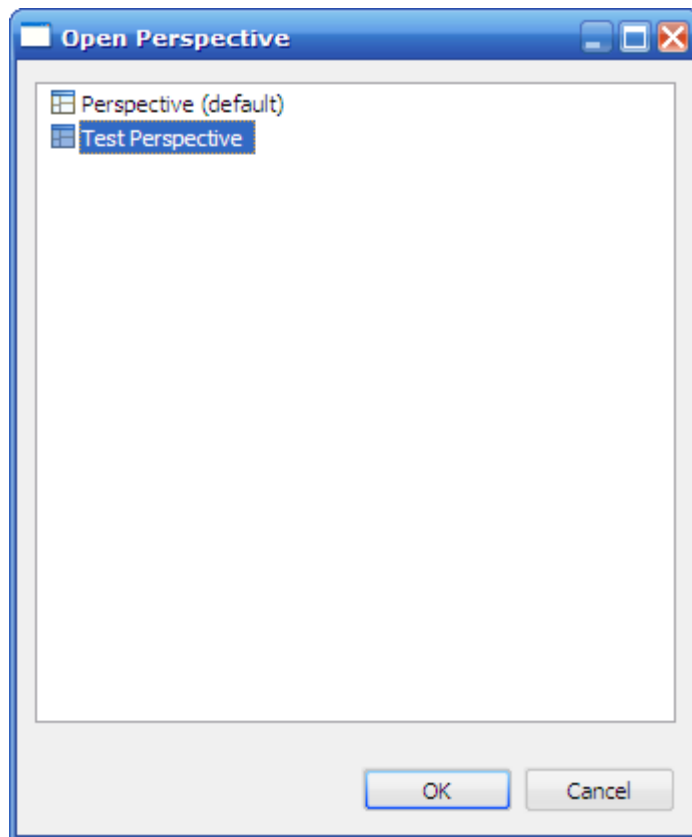
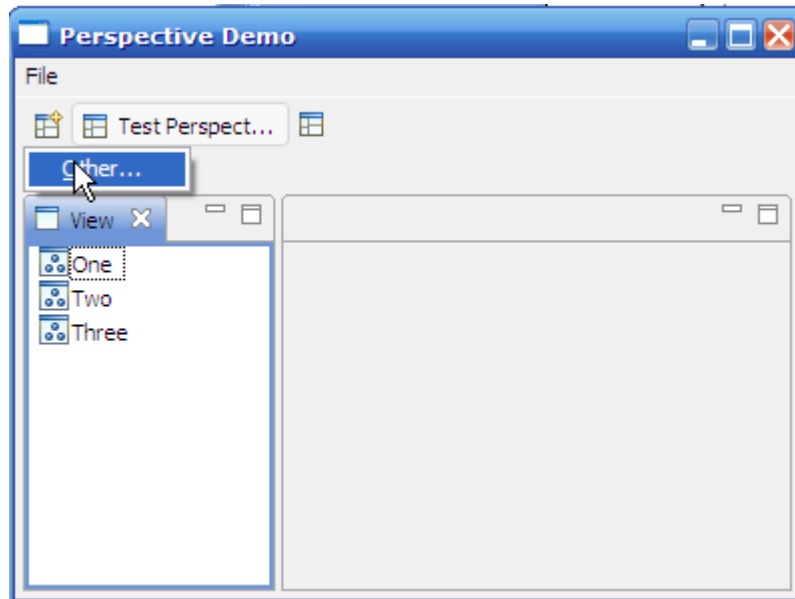
```
public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setInitialSize(new Point(400, 300));
    configurer.setShowCoolBar(false);
    configurer.setShowStatusLine(false);
    configurer.setShowPerspectiveBar(true);
    configurer.setTitle("Perspective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

    // 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");
}
```

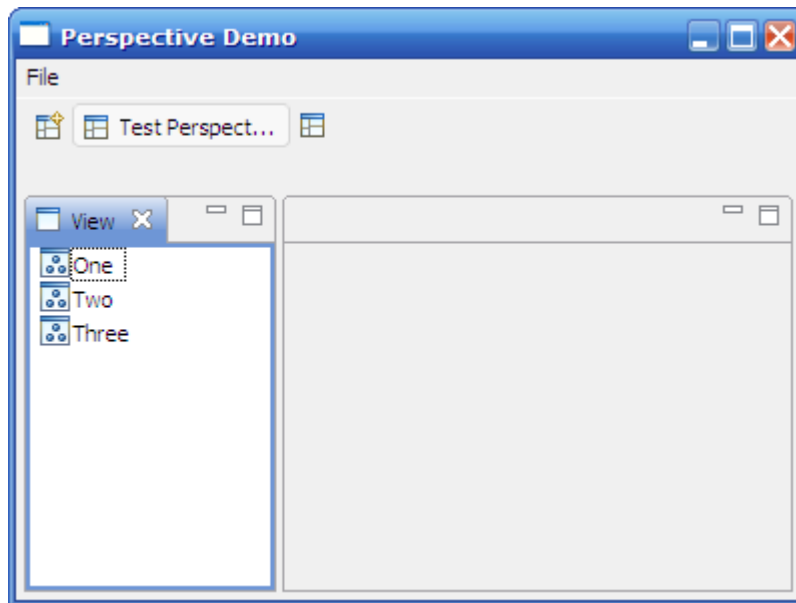
你现在将可以选择你的透视图，运行效果为：



单击透视图工具栏菜单，打开透视图选择对话框：



显示我们自定义的 Test Perspective:



11.4 显示透视图菜单

修改类 `ApplicationActionBarAdvisor`，以显示透视图菜单：

```
package org.salever.rcp.tech.chapter11;  
  
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);

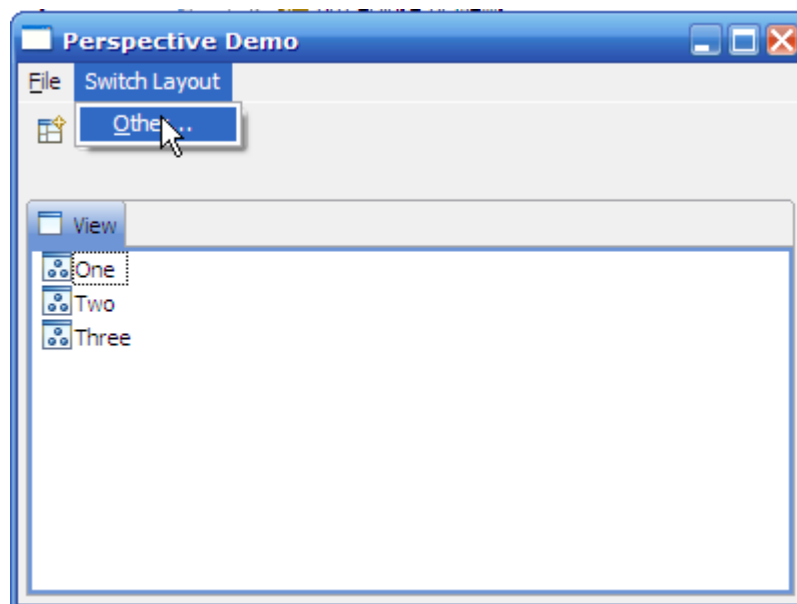
}

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

}

}
```

效果与透视图工具栏一样，运行结果如图：



12 进度条

12.1 简介

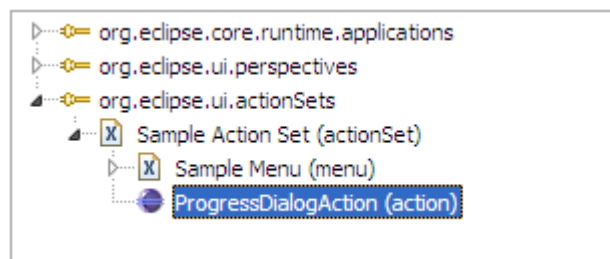
在 Eclipse 中，如何有 long-run 的程序，一般会出现一个进度条对话框，显示当前程序的运行进度，并提供后台运行的功能。

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对话框，或者进度指示器。接下来将展示这两个方法。

12.2 进度条对话框

进度条对话框，ProcessMonitorDialog，用于在长时间运行的程序执行过程中显示进度。

使用“Hello RCP”模板创建一个新的工程“org.salever.rcp.tech.chapter12”。添加 org.eclipse.ui.actionSets 扩展点，使用“Hello, world” action set 迅速创建一个菜单栏，修改 SampleAction 为 ProgressDialogAction。



修改 ProgressDialogAction.java:

```
package org.salever.rcp.tech.chapter12.actions;

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;
```

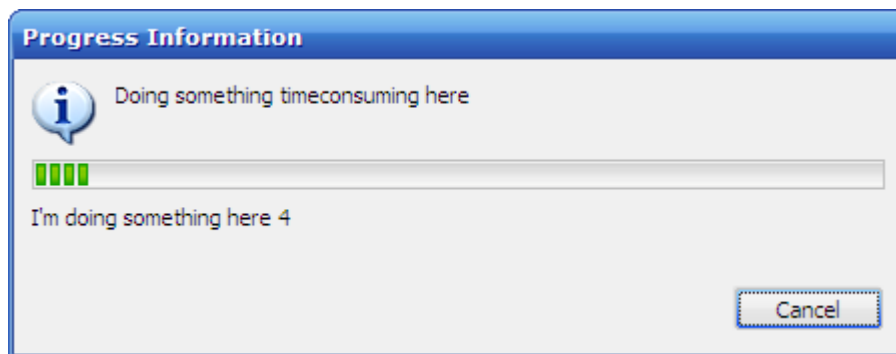
```
public class ProgressDialogAction implements IWorkbenchWindowActionDelegate {  
    private IWorkbenchWindow window;  
  
    public void dispose() {  
        // TODO Auto-generated method stub  
    }  
  
    public void init(IWorkbenchWindow window) {  
        // TODO Auto-generated method stub  
        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
}

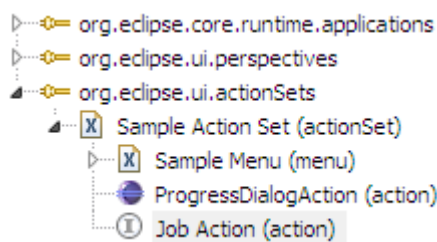
private void sleep(Integer waitTime) {
    try {
        Thread.sleep(waitTime);
    } catch (Throwable t) {
        System.out.println("Wait time interrupted");
    }
}
}
```

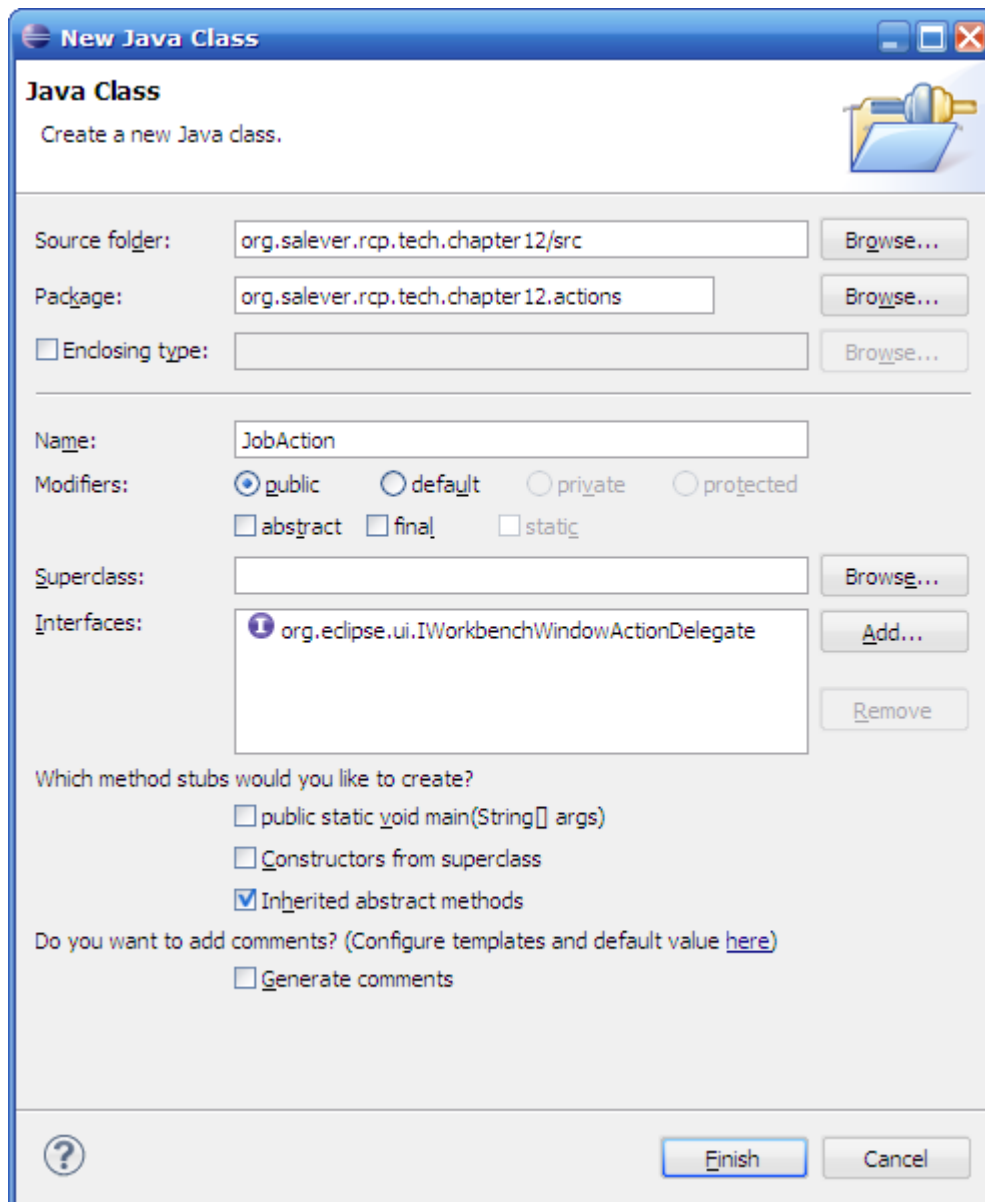
运行工程，点击 **ProgressDialogAction** 菜单，效果如下：



12.3 Job 进度条

Eclipse 中还有一种方式实现进度条执行，那就是 Job。在 Sample Action Sets 下面新建一个 action，命名为 JobAction。





JobAction.java:

```
package org.salever.rcp.tech.chapter12.actions;
```

```
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 JobAction 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

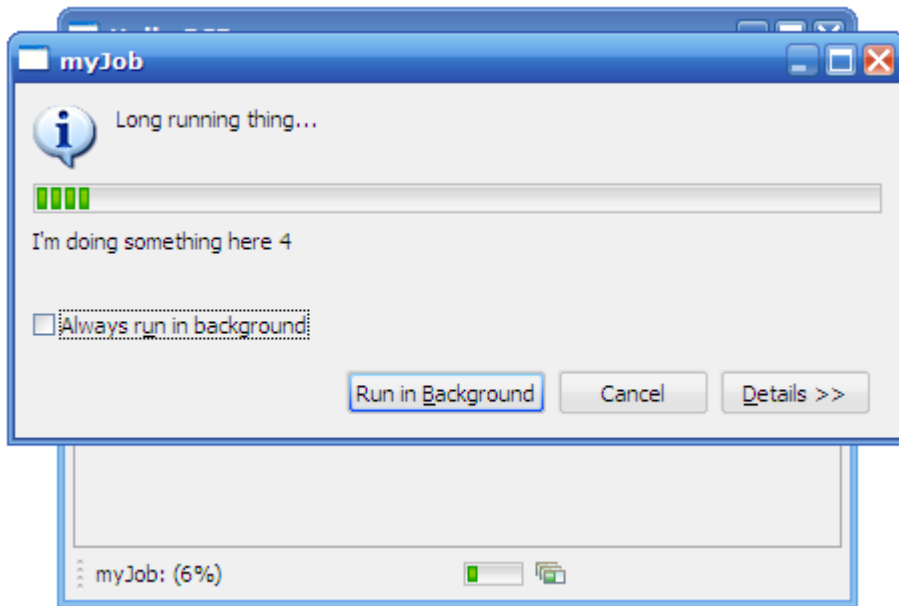
}

}
```

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

```
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:



13 使用第三方 Jar

13.1 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。

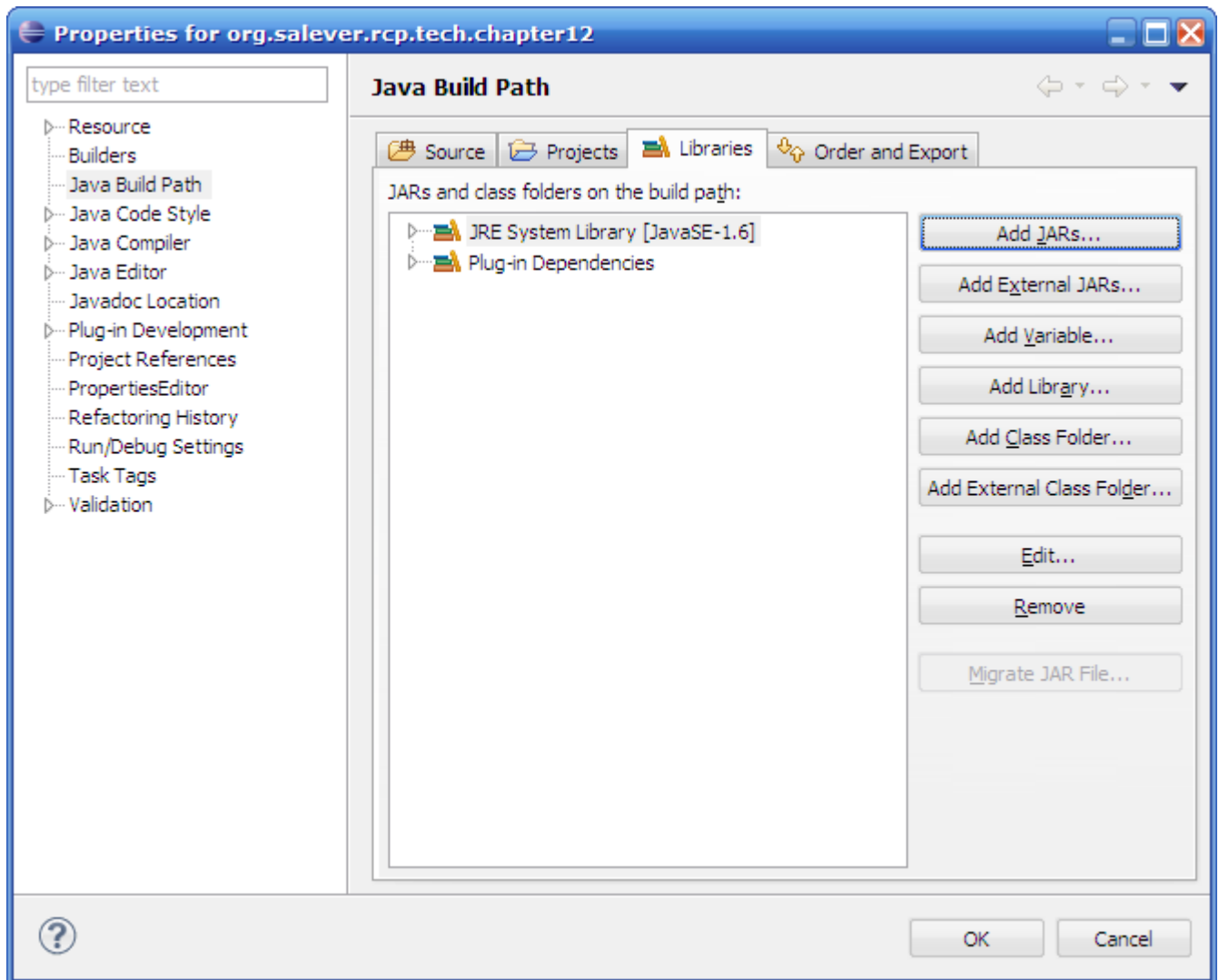
提示：个人习惯在工程里把所有的 jar 放在一个目录里，并命名为/lib，但这仅仅是我个人偏好。你可以将这些 jar 文件放入一个特别的目录里，并且指明他们为外部 jar。

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

13.2 向构建路径中添加 jar

按下面步骤向工程中添加外部 jar

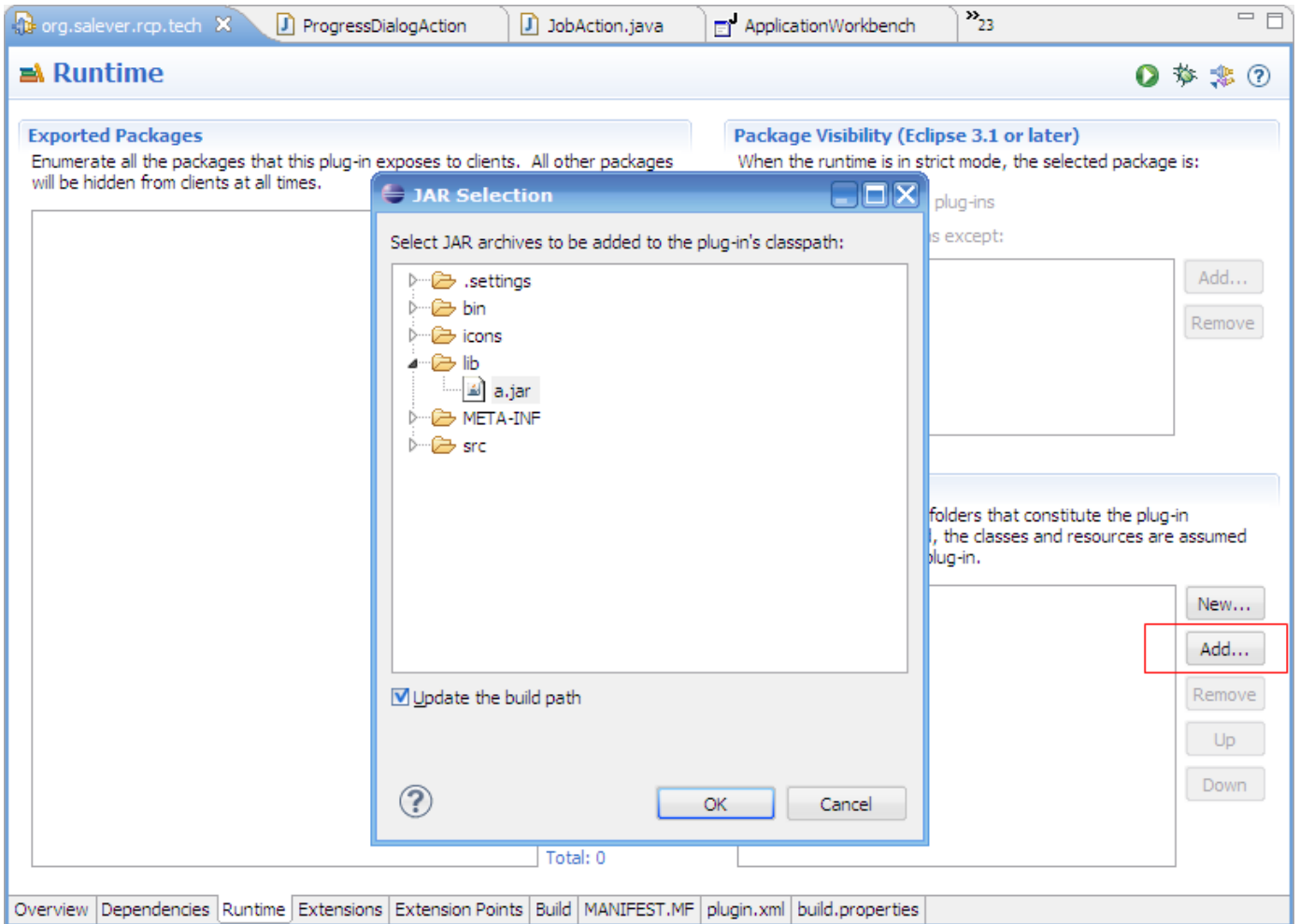
- 创建一个新文件夹，命名为 lib，或者使用已经存在的目录
- 选择 import->file system->import .jar
- 选择你的工程，鼠标右键点击，选择“properties”，在“libraries”里选择“Add JARs”
在“Order and Export”里将你的 jar 文件包含进来，并将他向上移动，避免冲突



13.3 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。

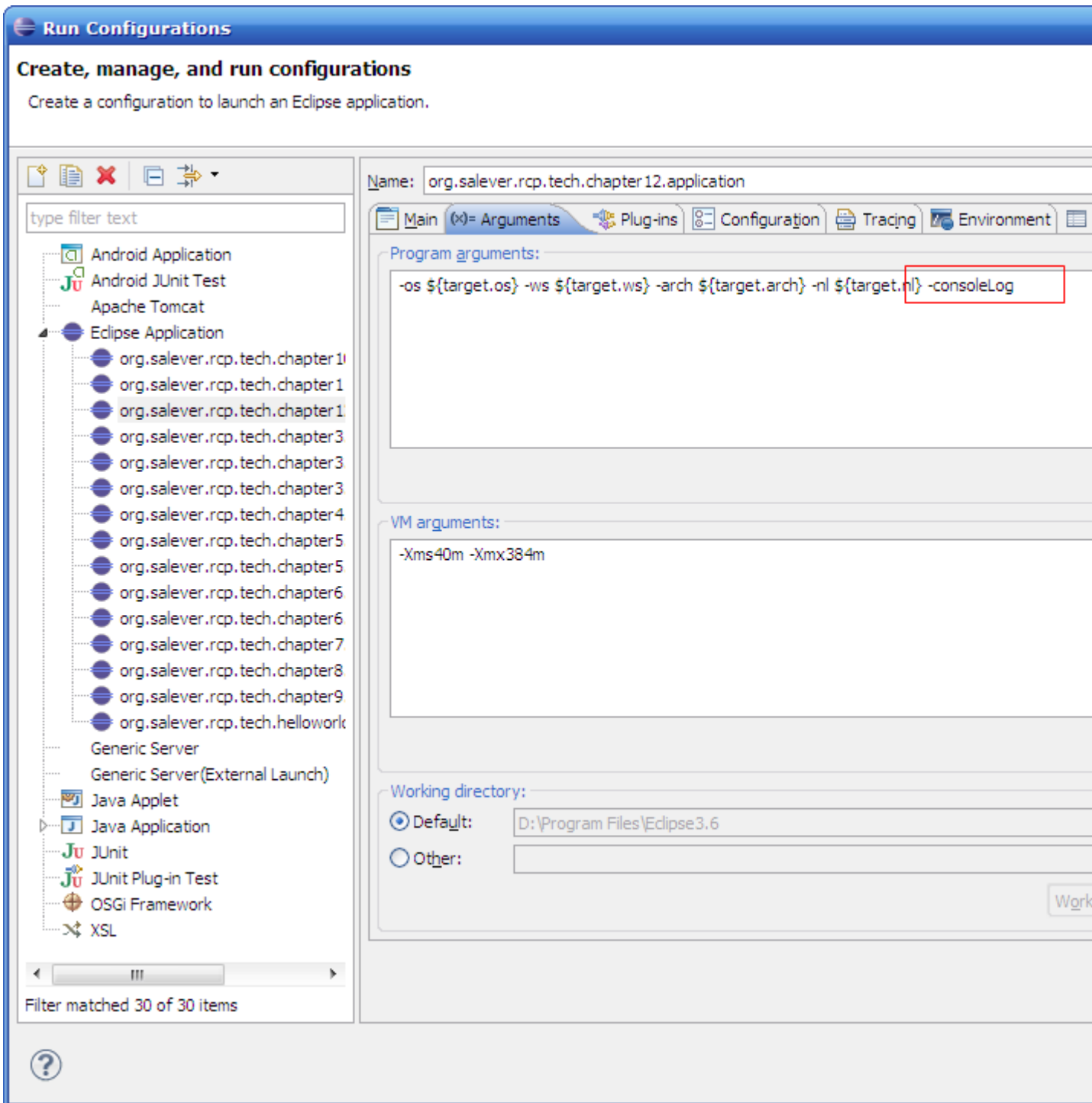
双击 plugin.xml 文件，选择 Runtime 标签，在其中修改就可以了。

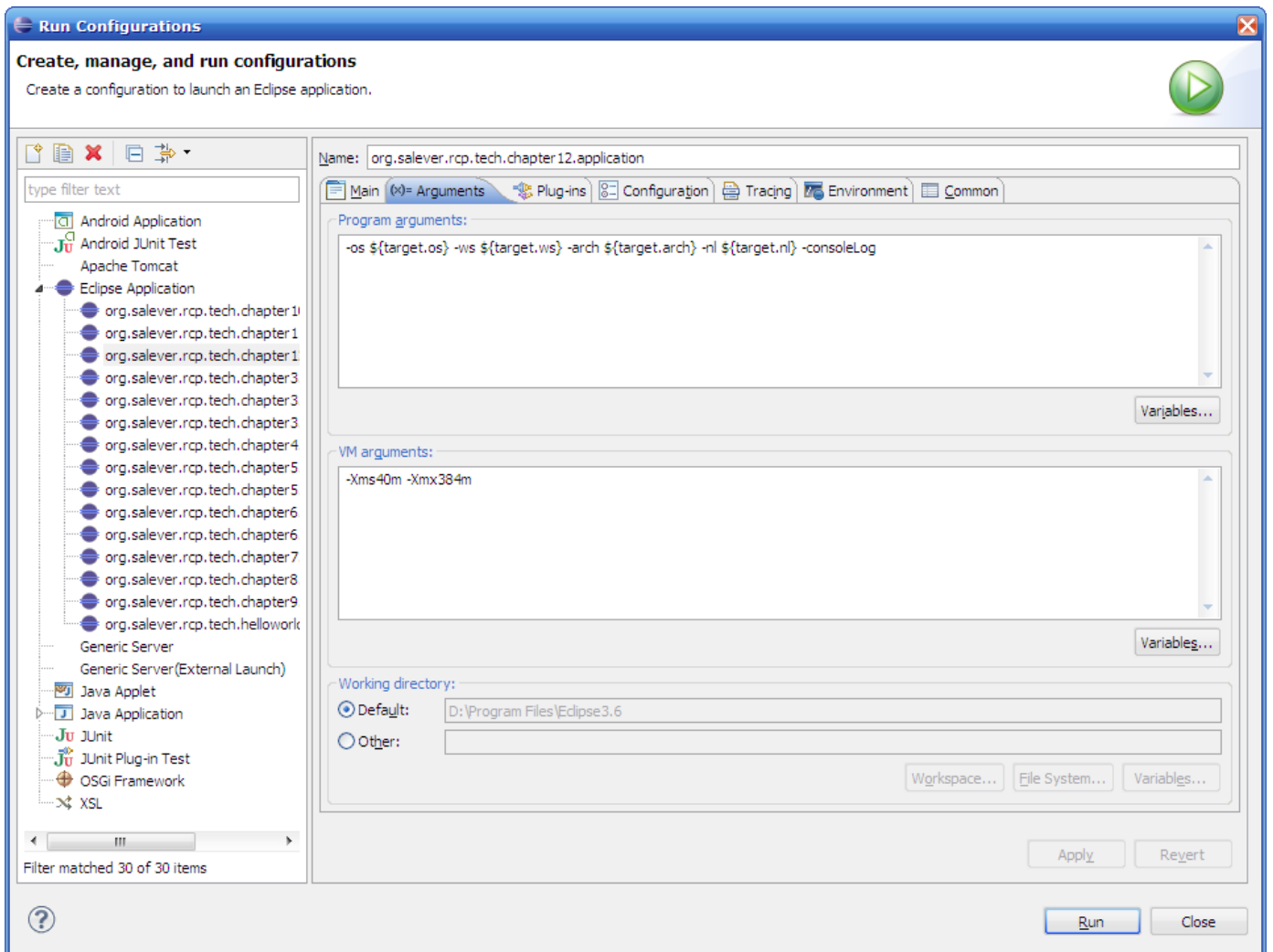


14 提示和策略

14.1 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





14.2 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

```
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)`

14.3 获得 display

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

14.4 使用 eclipse 的“保存”action

见 6.5 文本编辑器的实现。

Eclipse 默认的 Save 菜单，可以通过

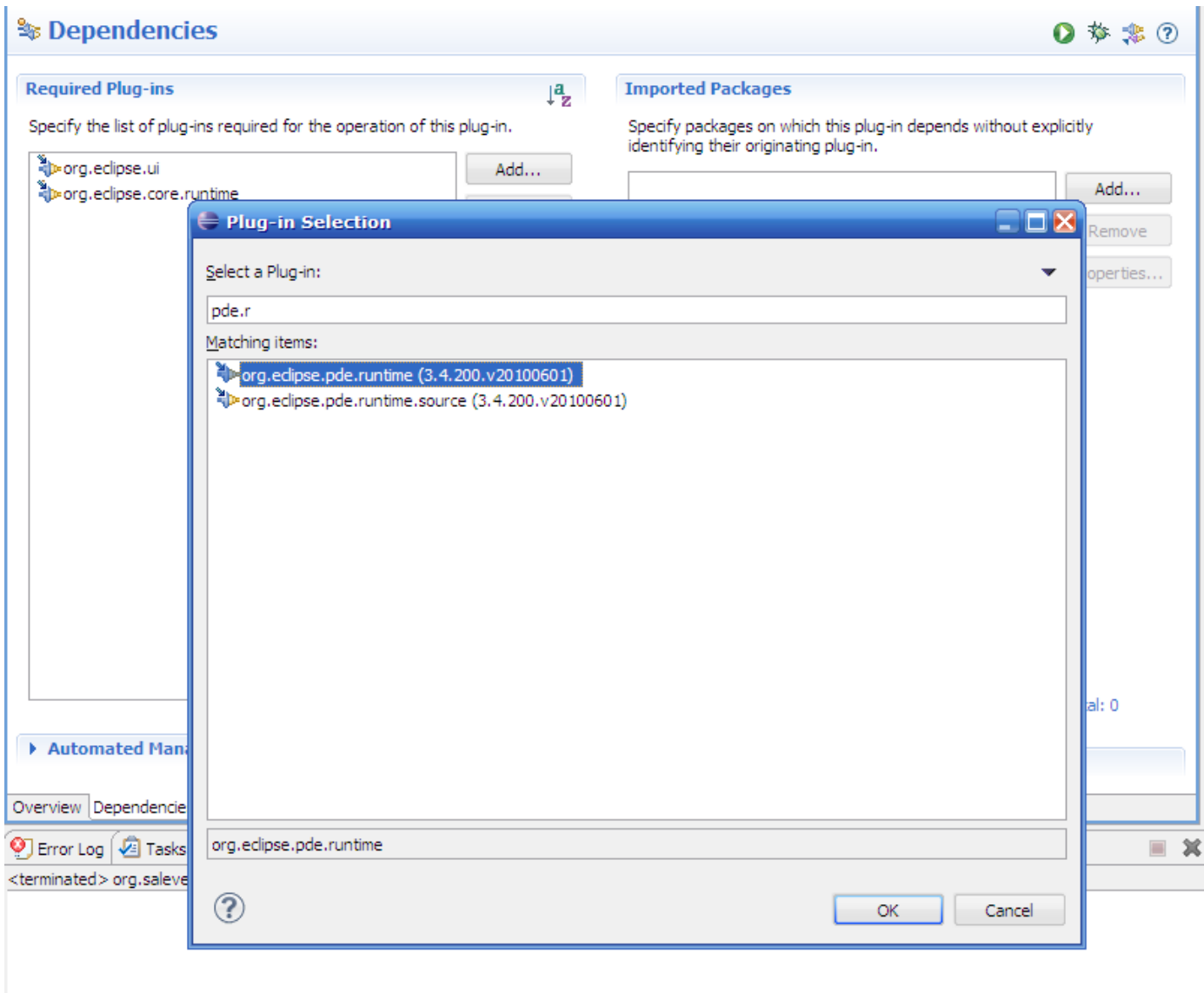
```
saveAction = ActionFactory.SAVE.create(window);
```

获取，它的状态可以用过 `ISaveablePart` 的 `isDirty()` 控制，可以用于 `Editor` 或者 `ViewPart`。

14.5 向你的程序添加错误日志视图

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。

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



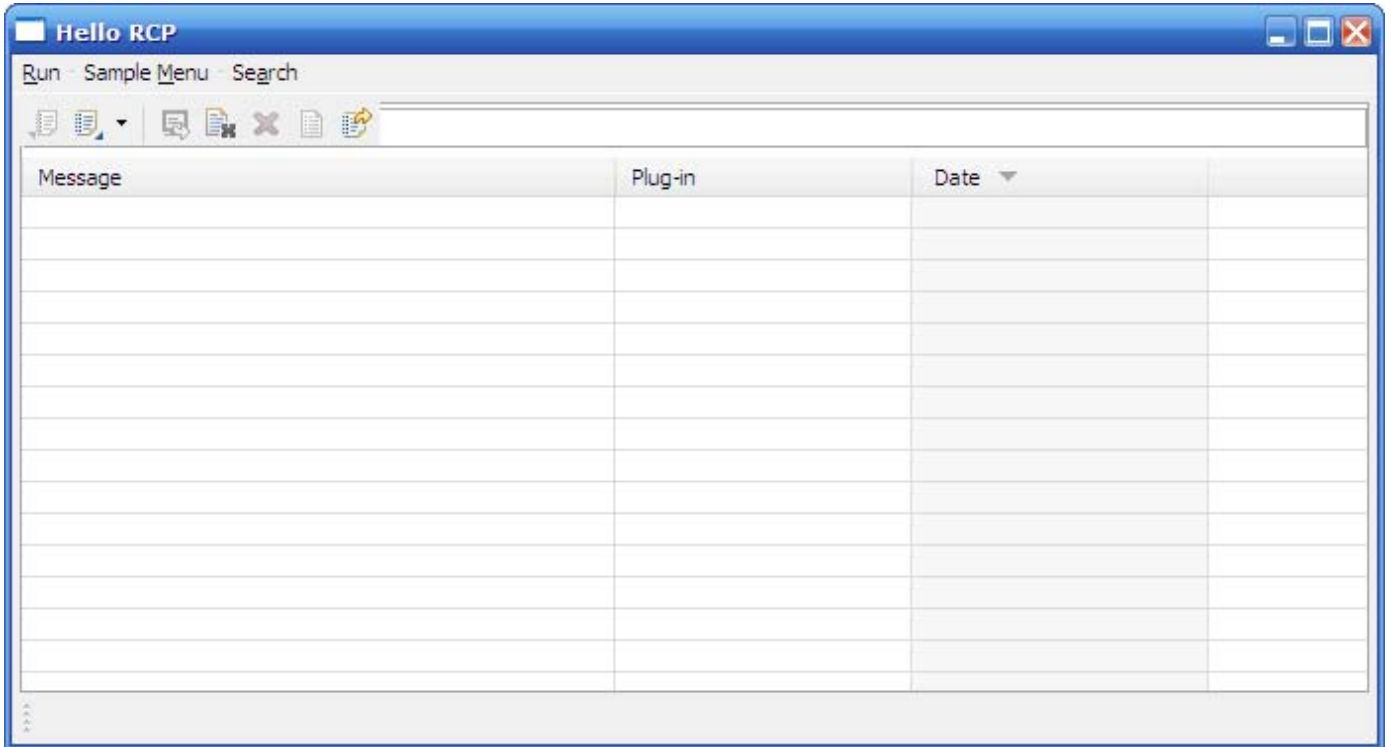
在 Perspective 类中包含如错误日志视口。使用 org.eclipse.pde.runtime.LogView 的 ID，代码如下

```
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：



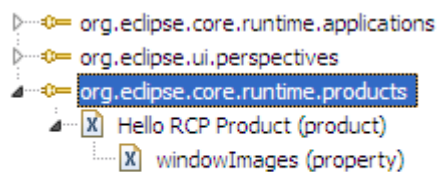
15 产品

15.1 概述

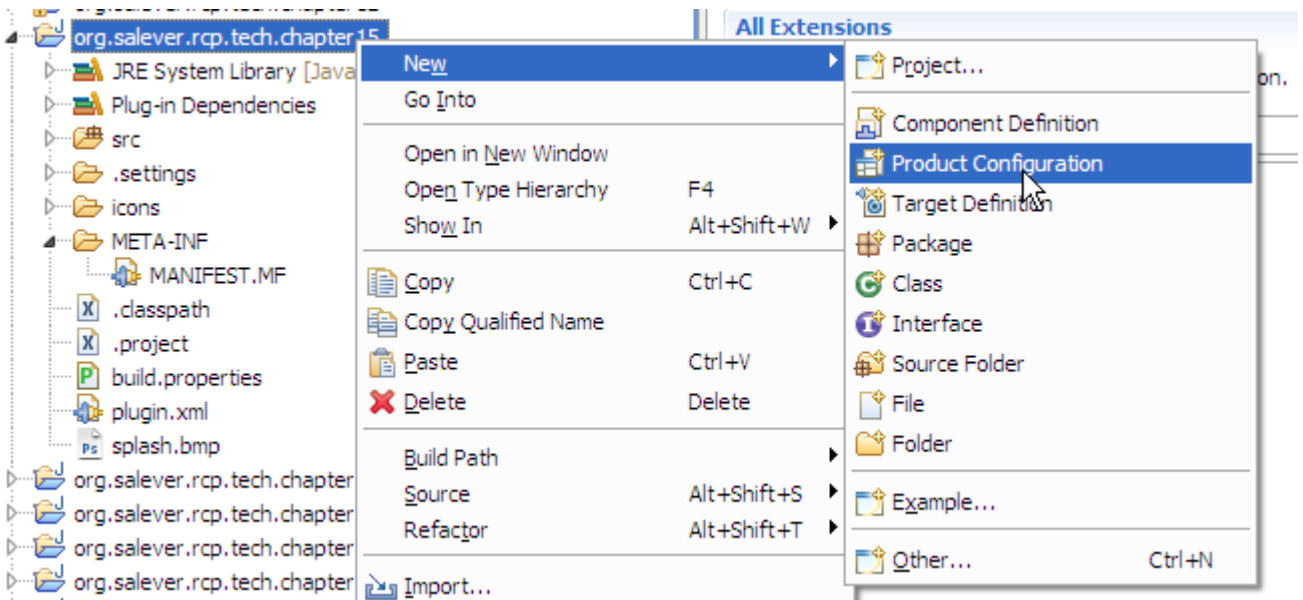
产品（Product），Eclipse plugin 工程是以产品的方式打包和独立运行的，整个 Eclipse 就是一个产品。如果你想发布你的程序，你可以向产品中打包所有需要的包、设置文件等。

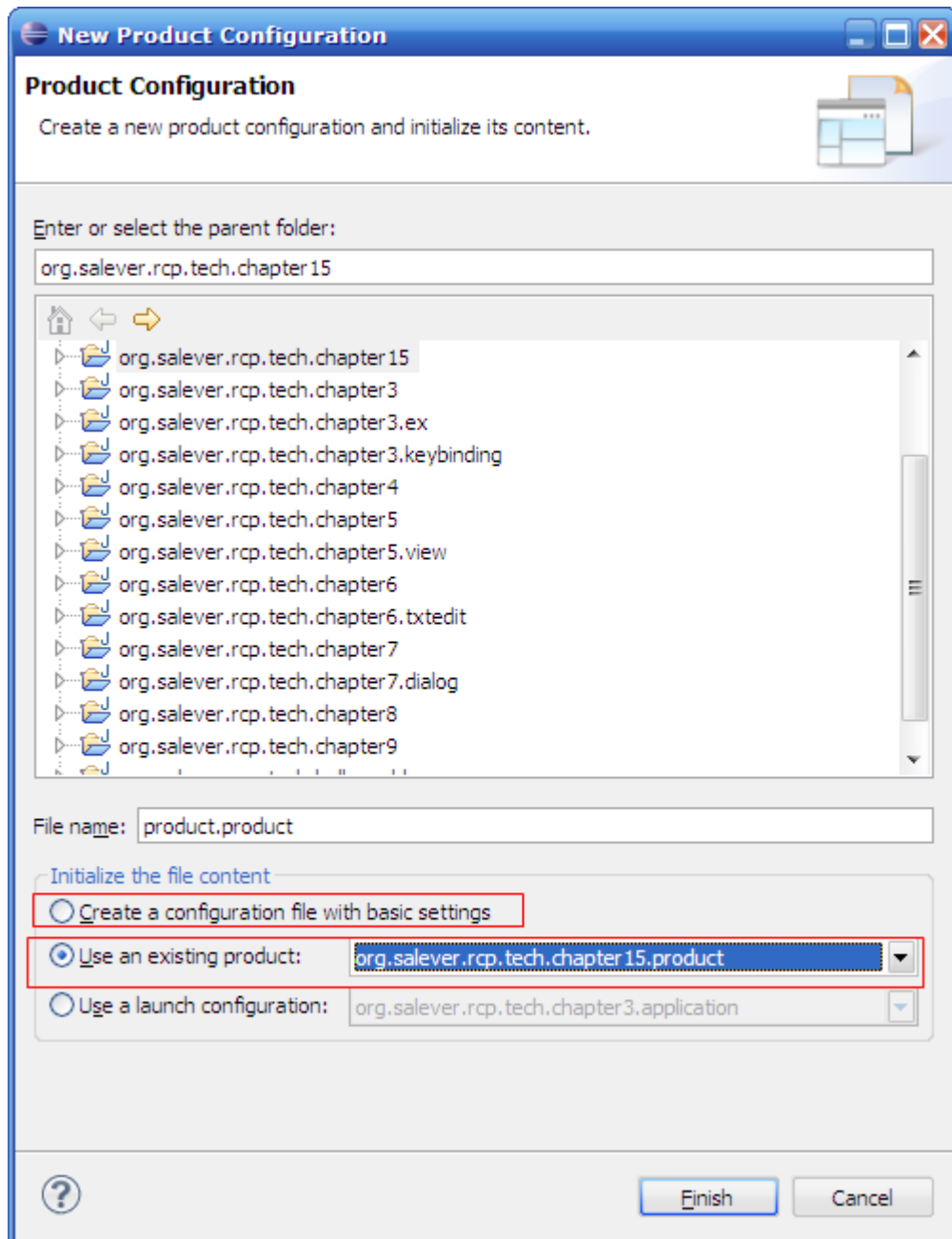
15.2 Product 配置文件

使用“Hello, world”模板新建一个工程“org.salever.rcp.tech.chapter12”，在最后的 finish 页面添加“Add branding”，这时候创建的工程默认已经有一个 product 的扩展了。



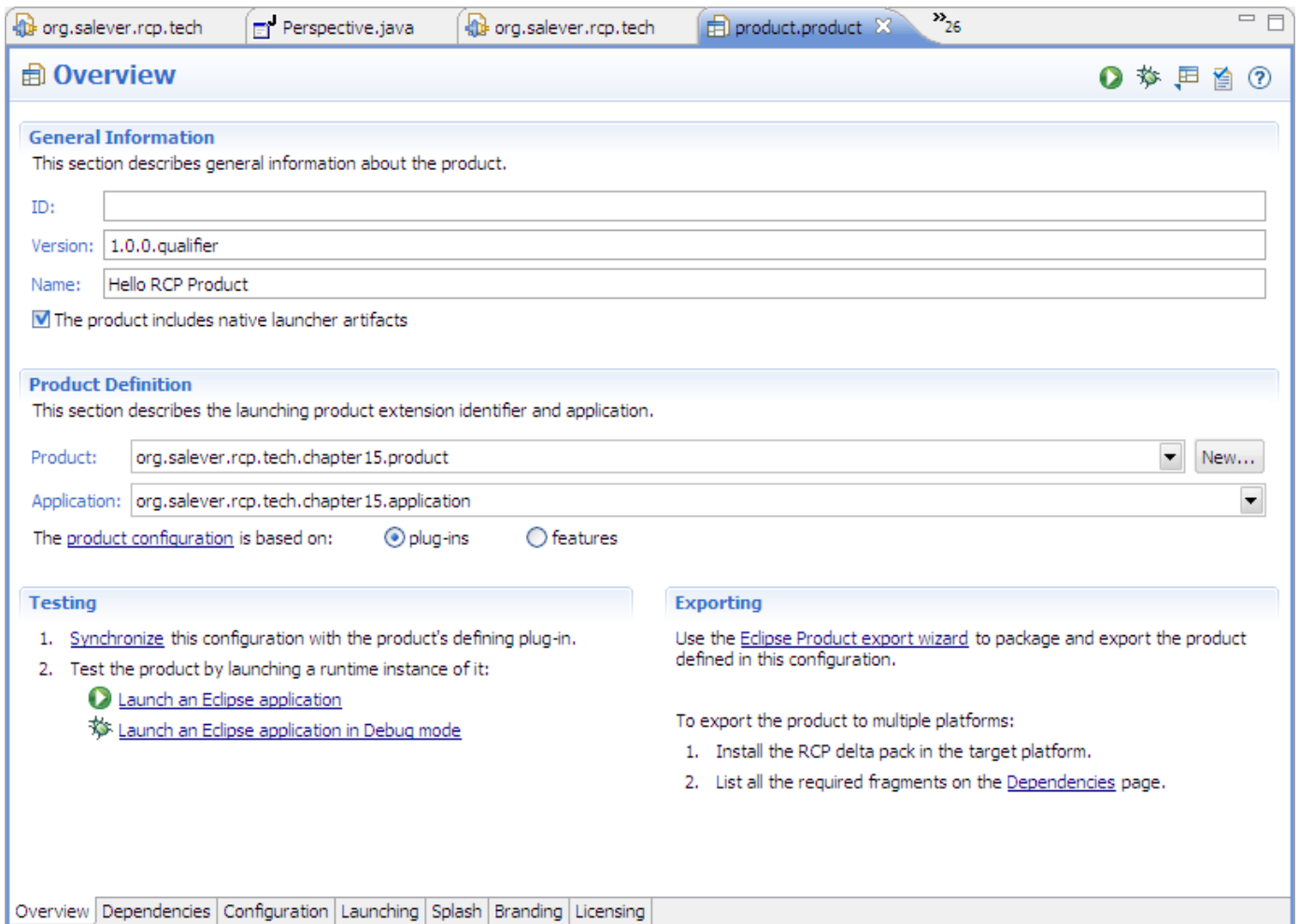
右键点击工程，选择 New->Product Configuration。键入一个名字，点击完成。



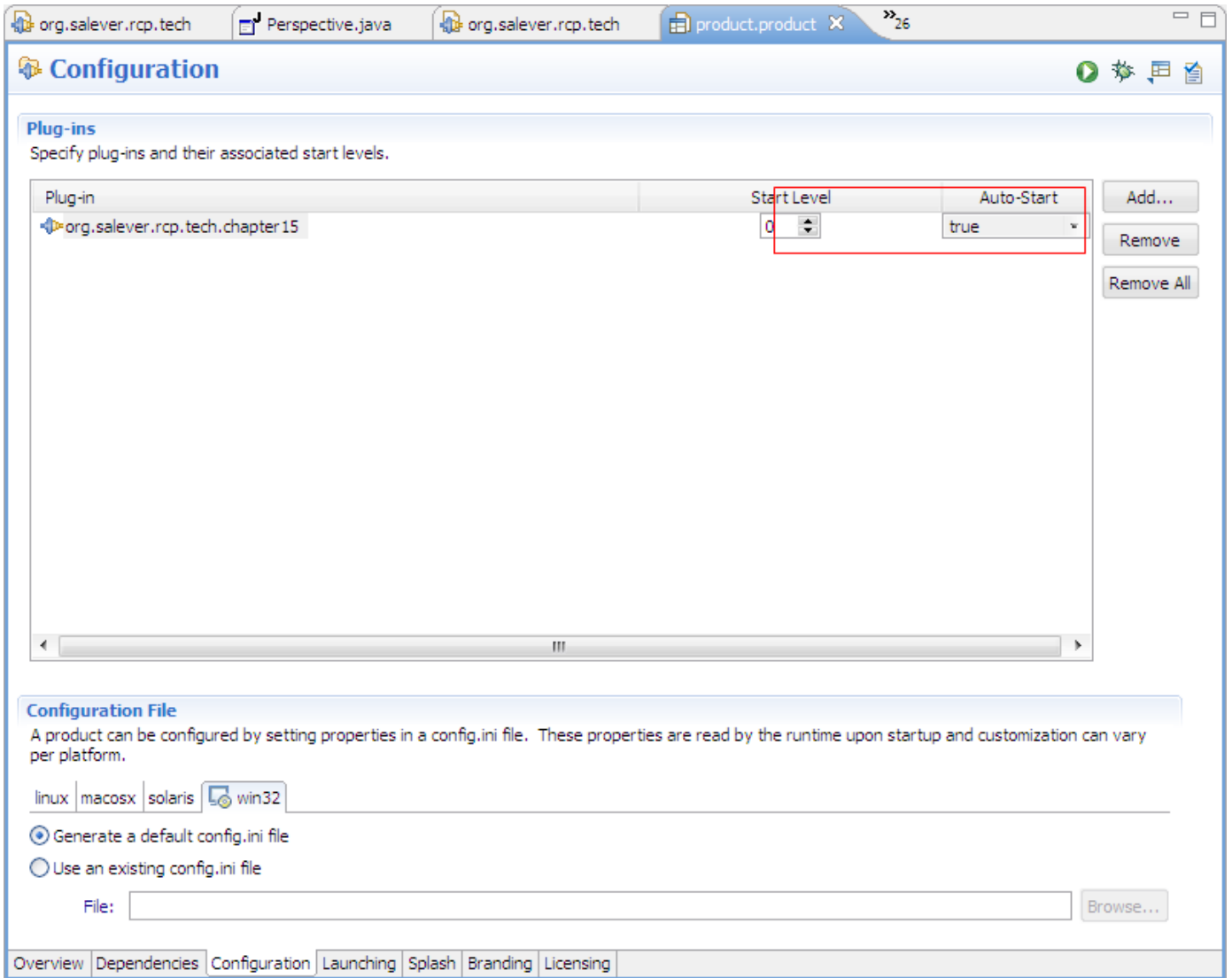


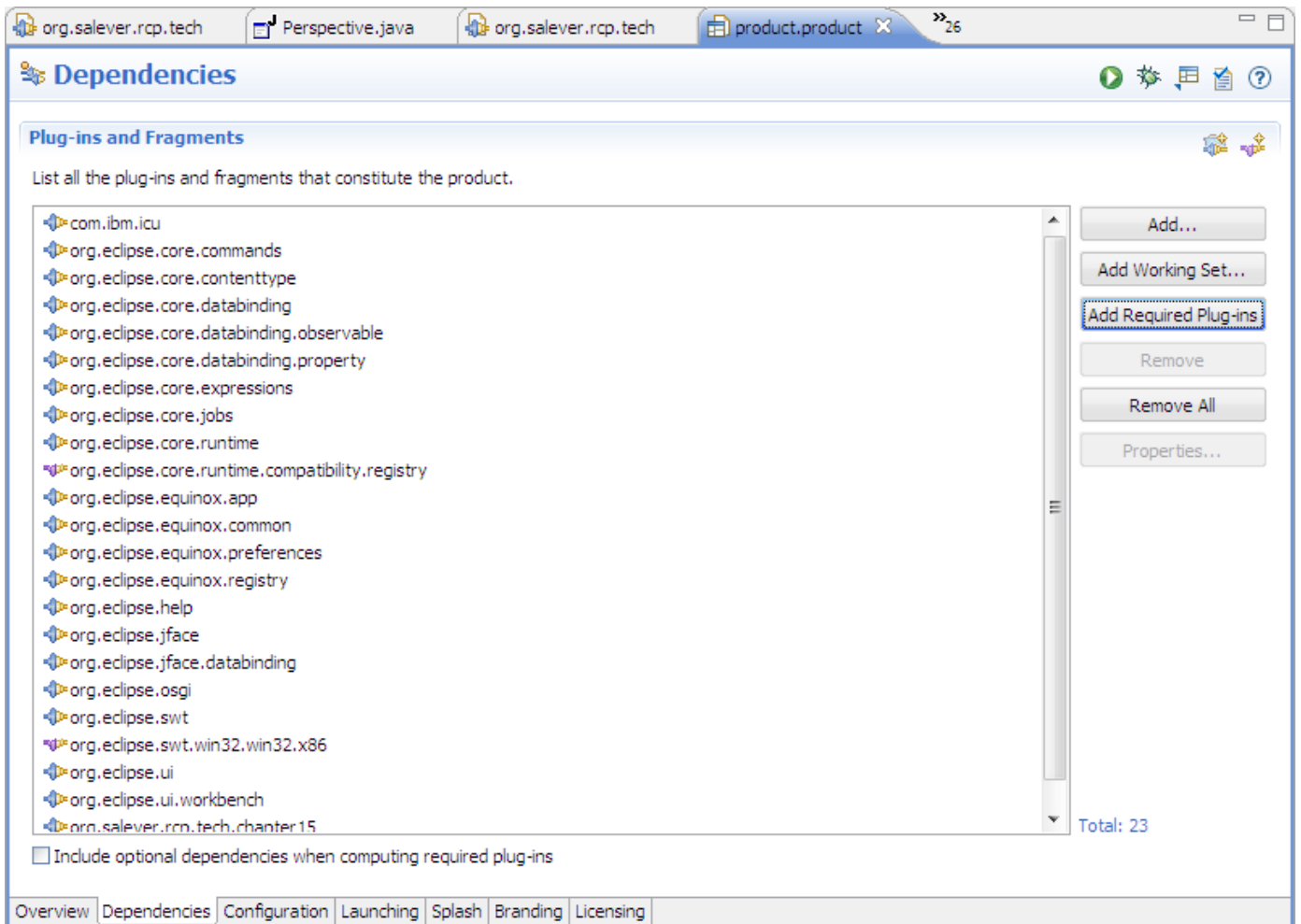
如果 `plugin.xml` 中没有扩展 `org.eclipse.core.runtime.products`，那么可以选择 `Create a configuration file basic settings` 或者 `Use a launch configuration`，这里由于我们新建工程的时候勾选了 `Add branding`，Eclipse 已经为我们新建了一个 `product` 扩展，直接使用即可。

打开 `product.product` 文件：



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，这里为“org.salever.rcp.tech.chapter12”），并且点击“Add Required Plugins”。保存之后在返回到“overview”画面。这里可以选择不添加任何插件，但是如果添加了插件，需要把主插件的 Auto-start 设置 true。





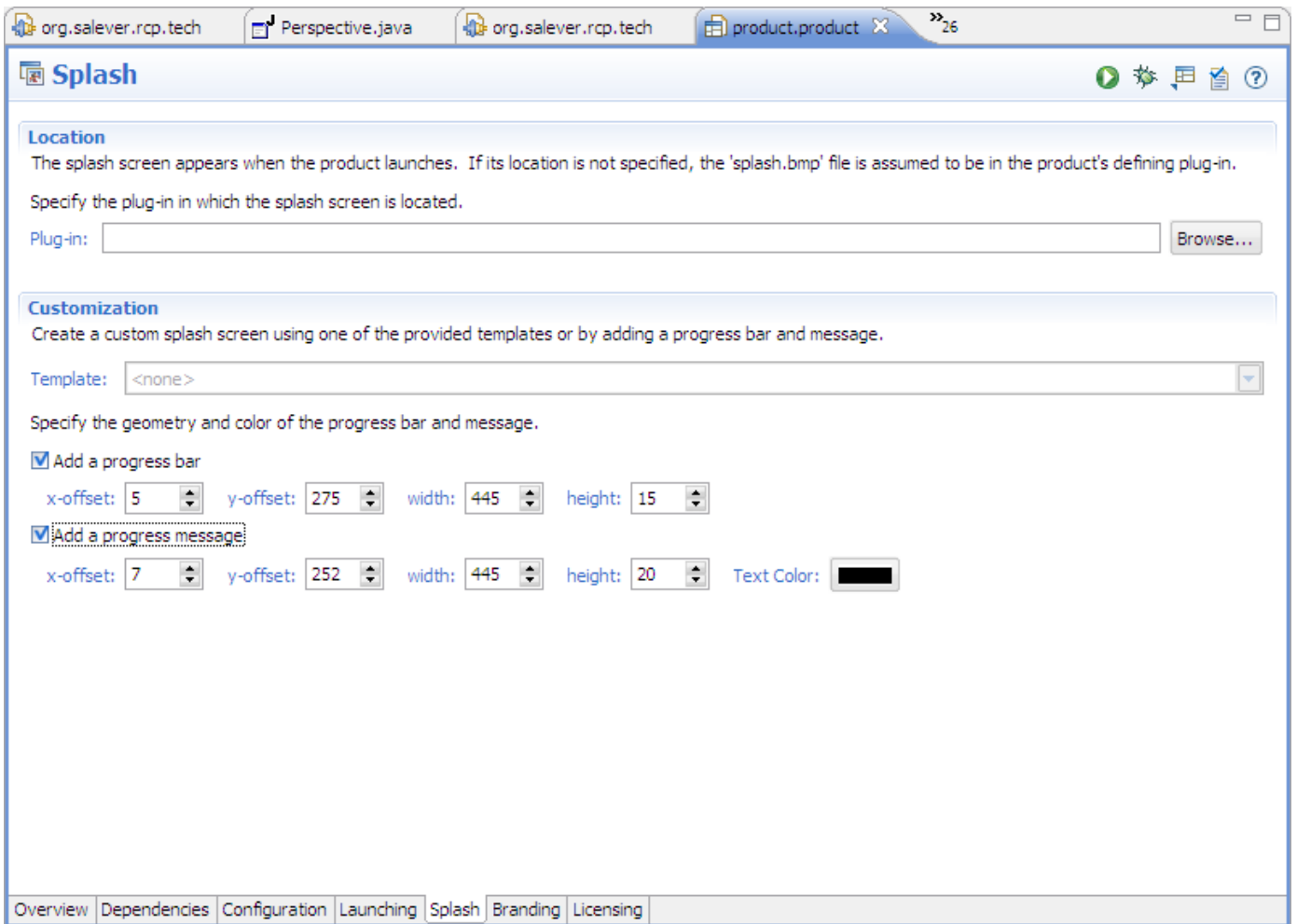
15.3 测试你的产品

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

15.4 欢迎页面

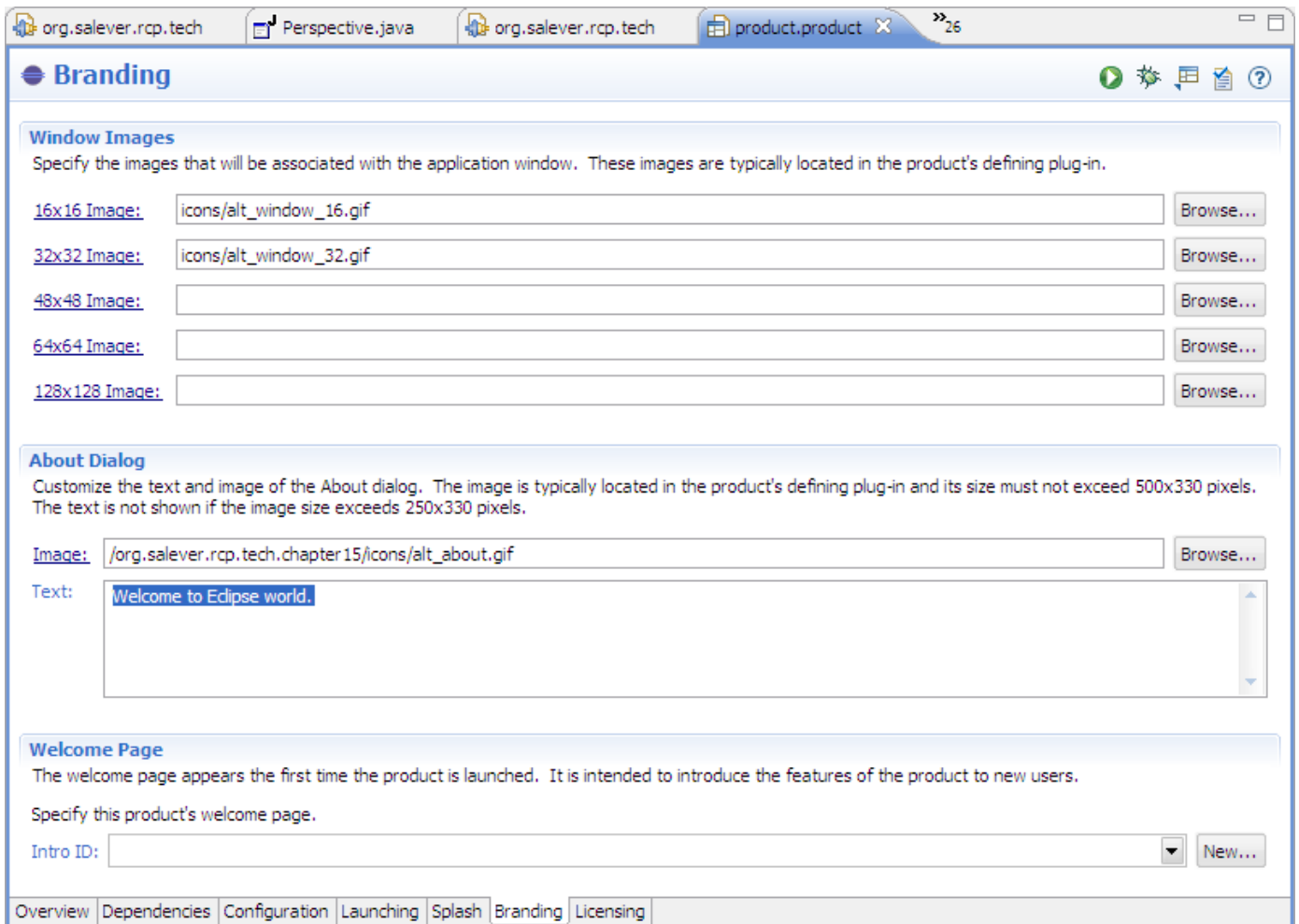
打开产品文件*.product，切换到“Splash”标签页，在这里你可以指定一个欢迎页面。你的应用程序将使用一个必须放在在目录下的 splash.bmp 文件。这个文件必须是 BMP 类型的。并且，必须命名为 splash.bmp

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。



15.5 商标

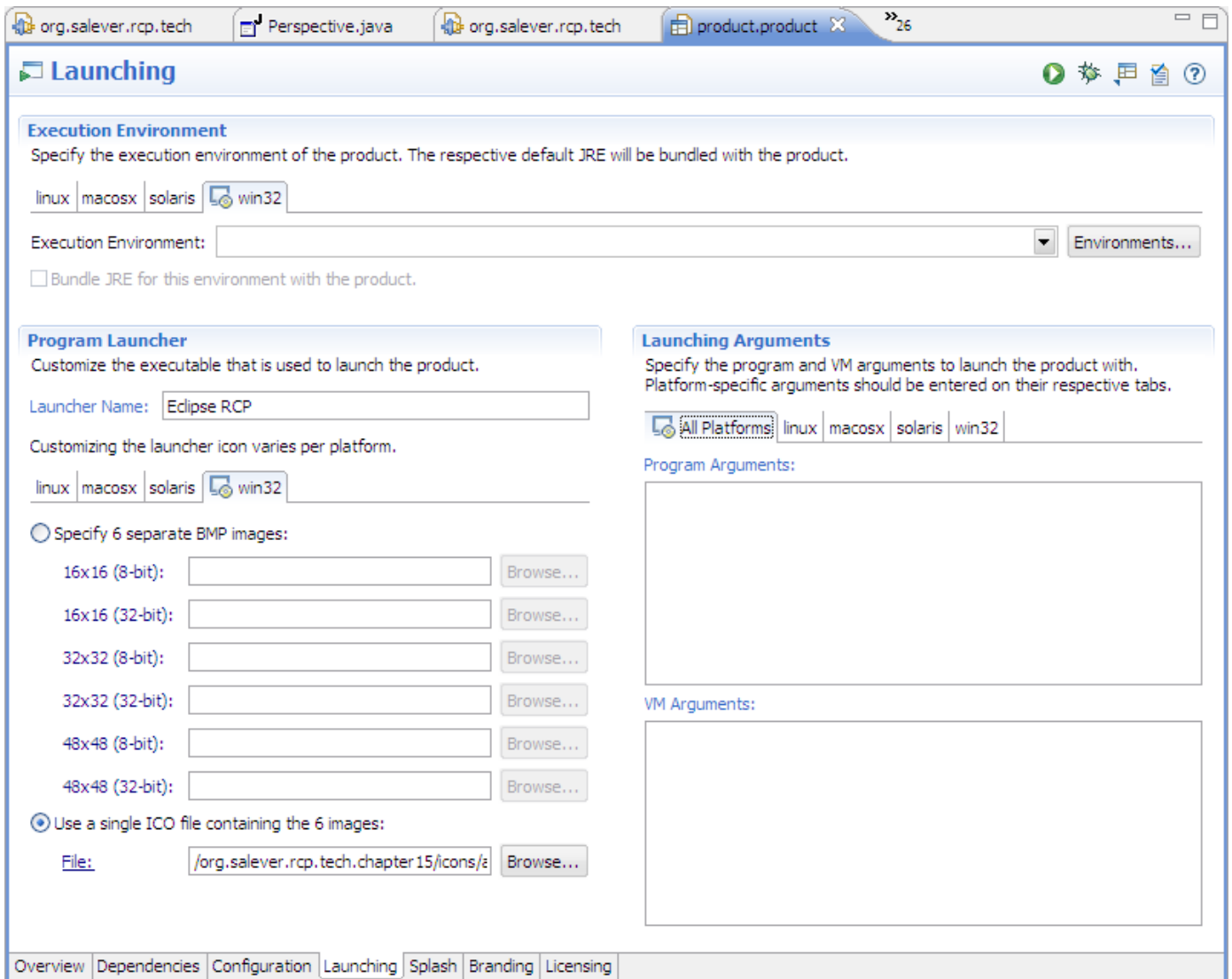
标准 eclipse 的“About Dialog”可以被注明商标。你可以向你的程序添加一个图标或者文字。



15.6 风格化 Launching

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。确认图标大小被修改正确，否则这些图标将不被 eclipse 使用



15.7 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。

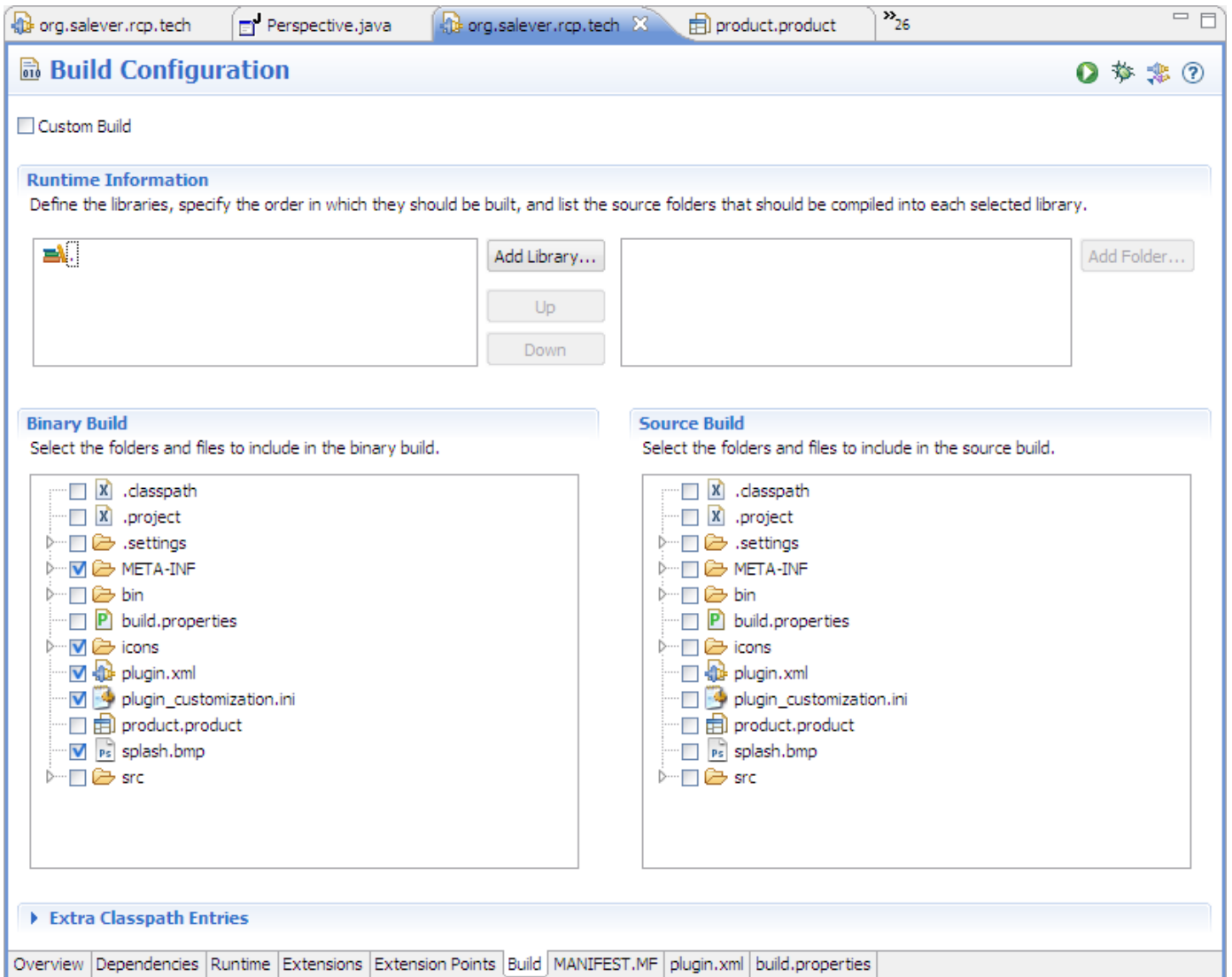
下面假定你没有使用外部 jar。如果你想在你的发布产品中添加第三方库（jars），必须将他们捆绑进插件。

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。

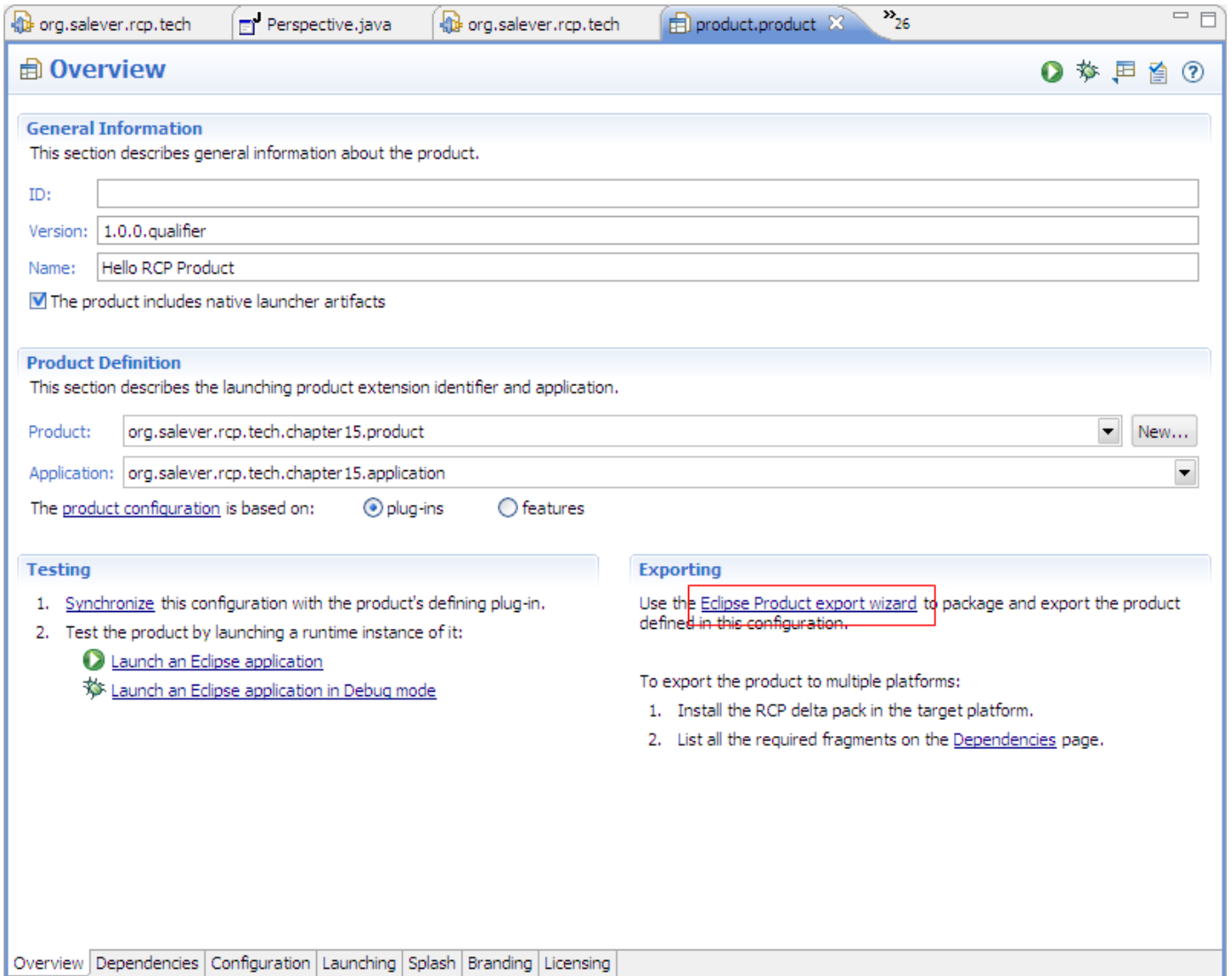
如果你使用了图标或者欢迎页面，也必须将他们假如构建输出中

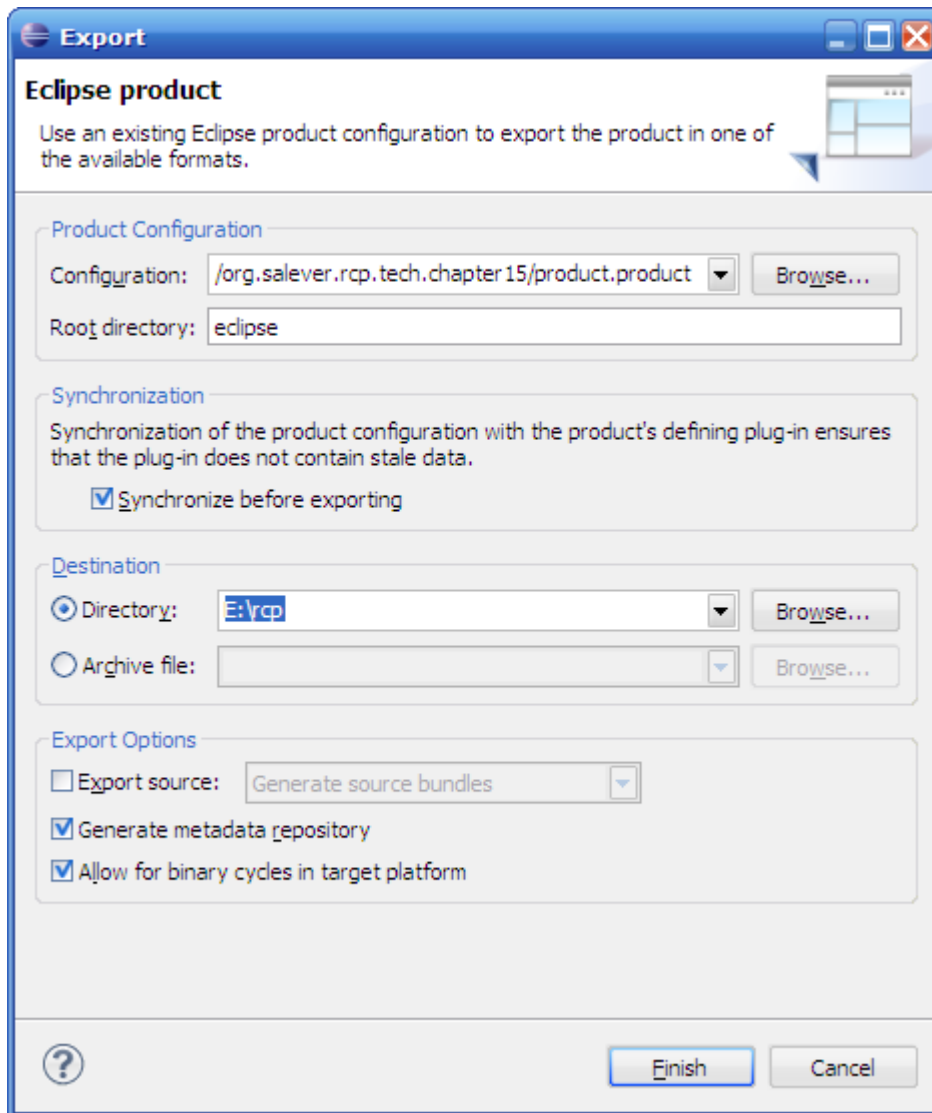
选择你的 plugin.xml 并且选择“build”标签。确认 META-INF 目录被选。选择你的图标目录或者

其他可被包含进输出的图形文件。



点击*.product 的 Overview 标签页中的“Eclipse Product export wizard”导出你的产品：

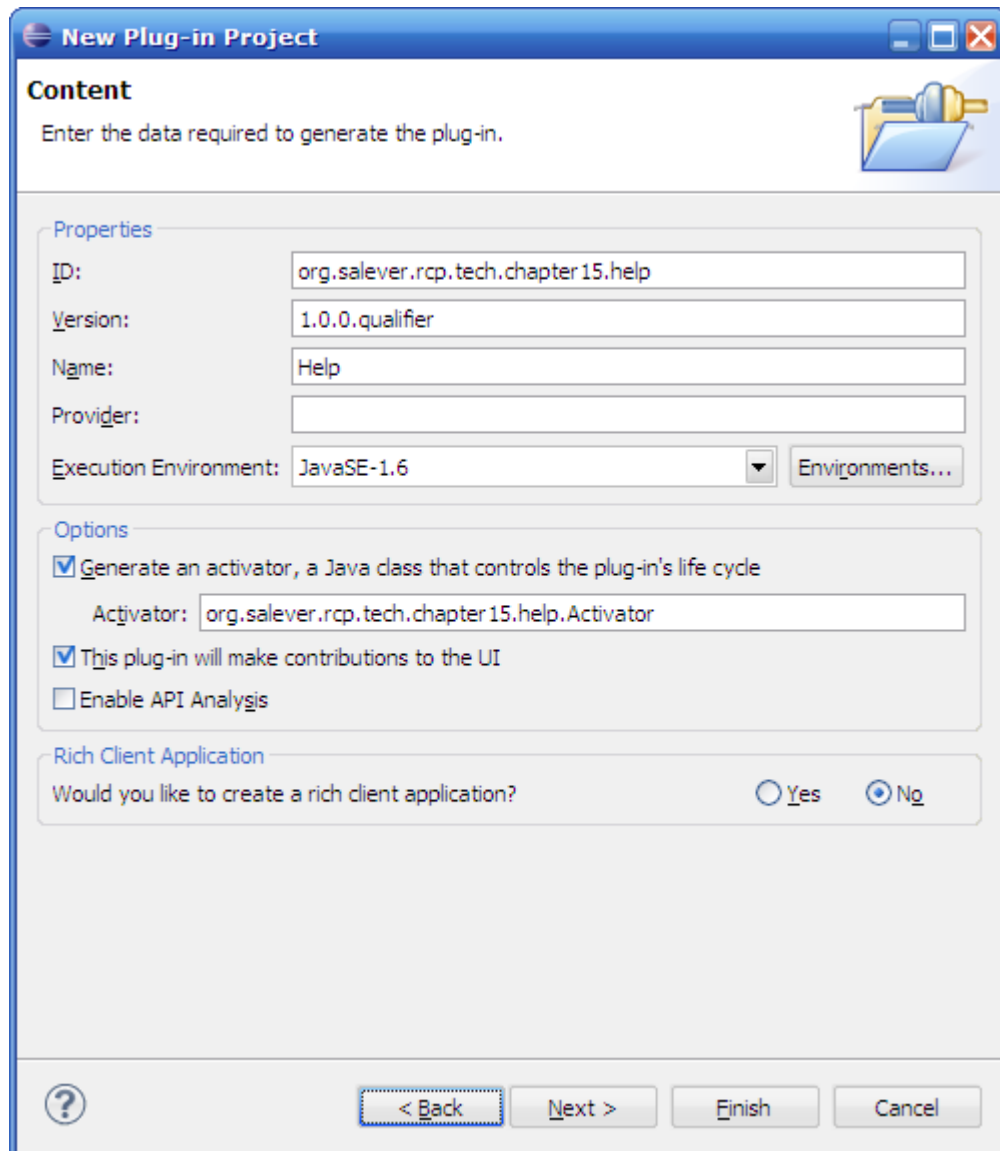




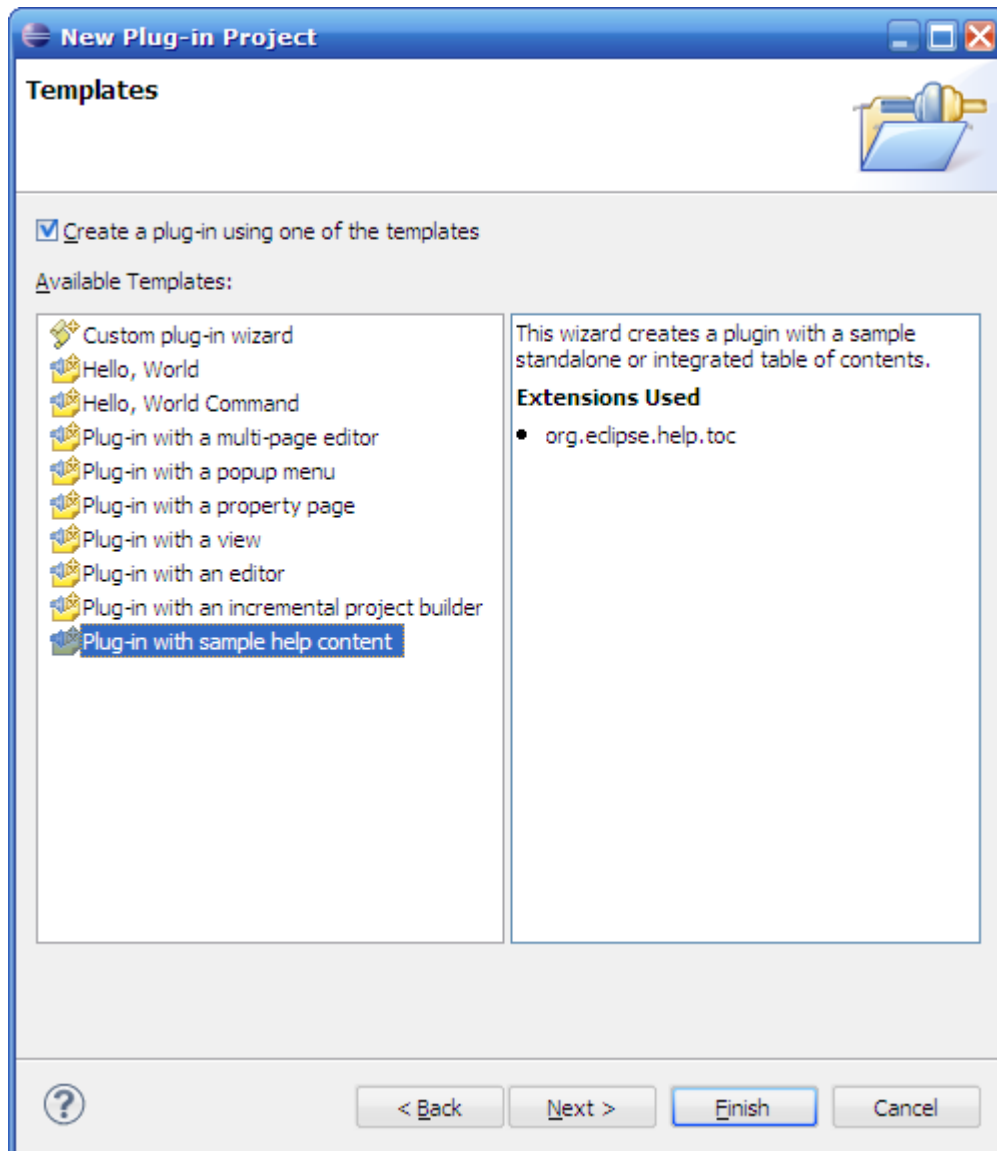
这时，在你设定好的目录里会出现一系列文件及文件夹，其中有一个“eclipse.exe”，双击它，可以用来启动你的程序。（如果在“launching”里更改过名称之后，可执行文件叫不在被命名为“eclipse.exe”）

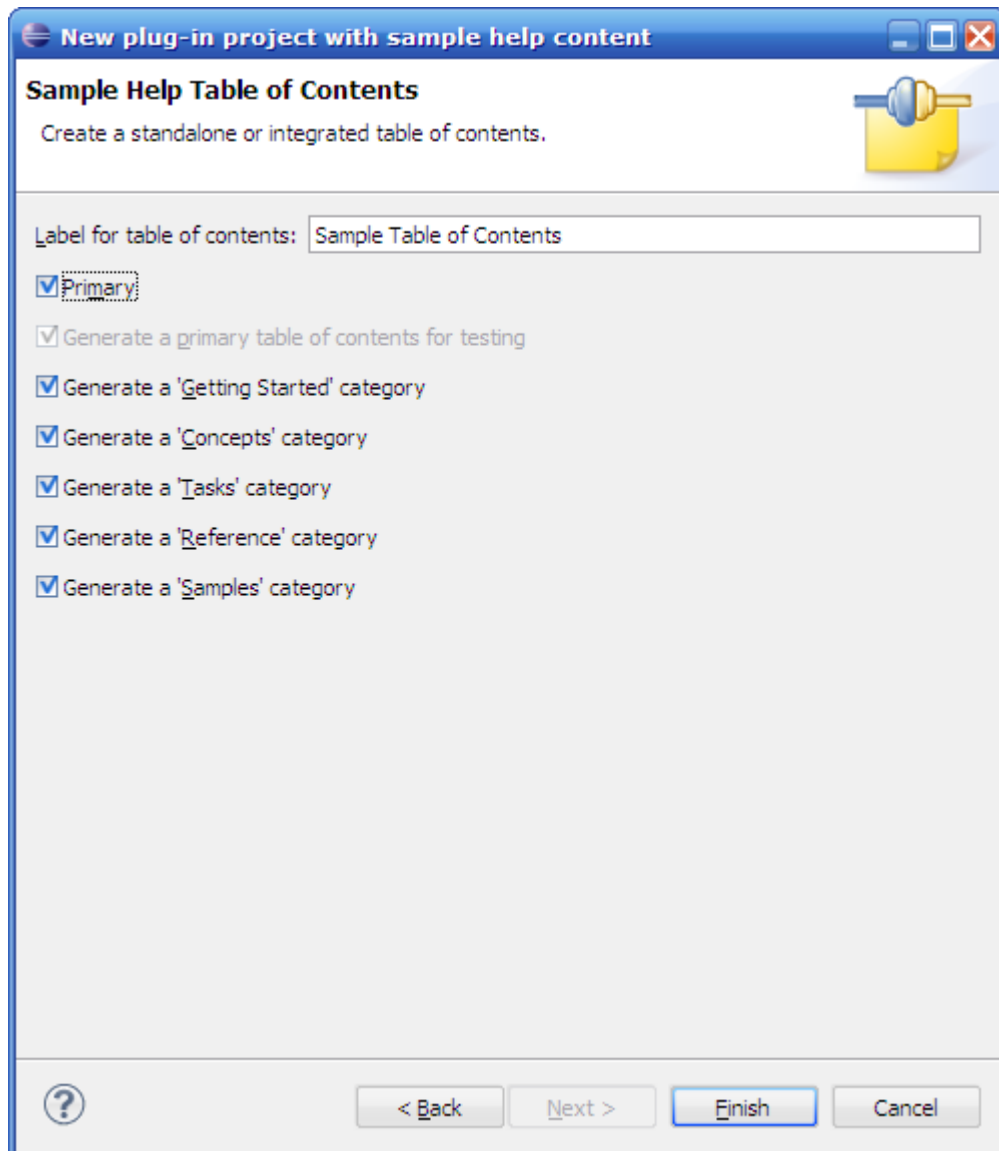
15.8 创建一个帮助插件工程

新建一个插件工程，“org.salever.rcp.tech.chapter15.help”，在 Rich Client Application 栏中选择 No。

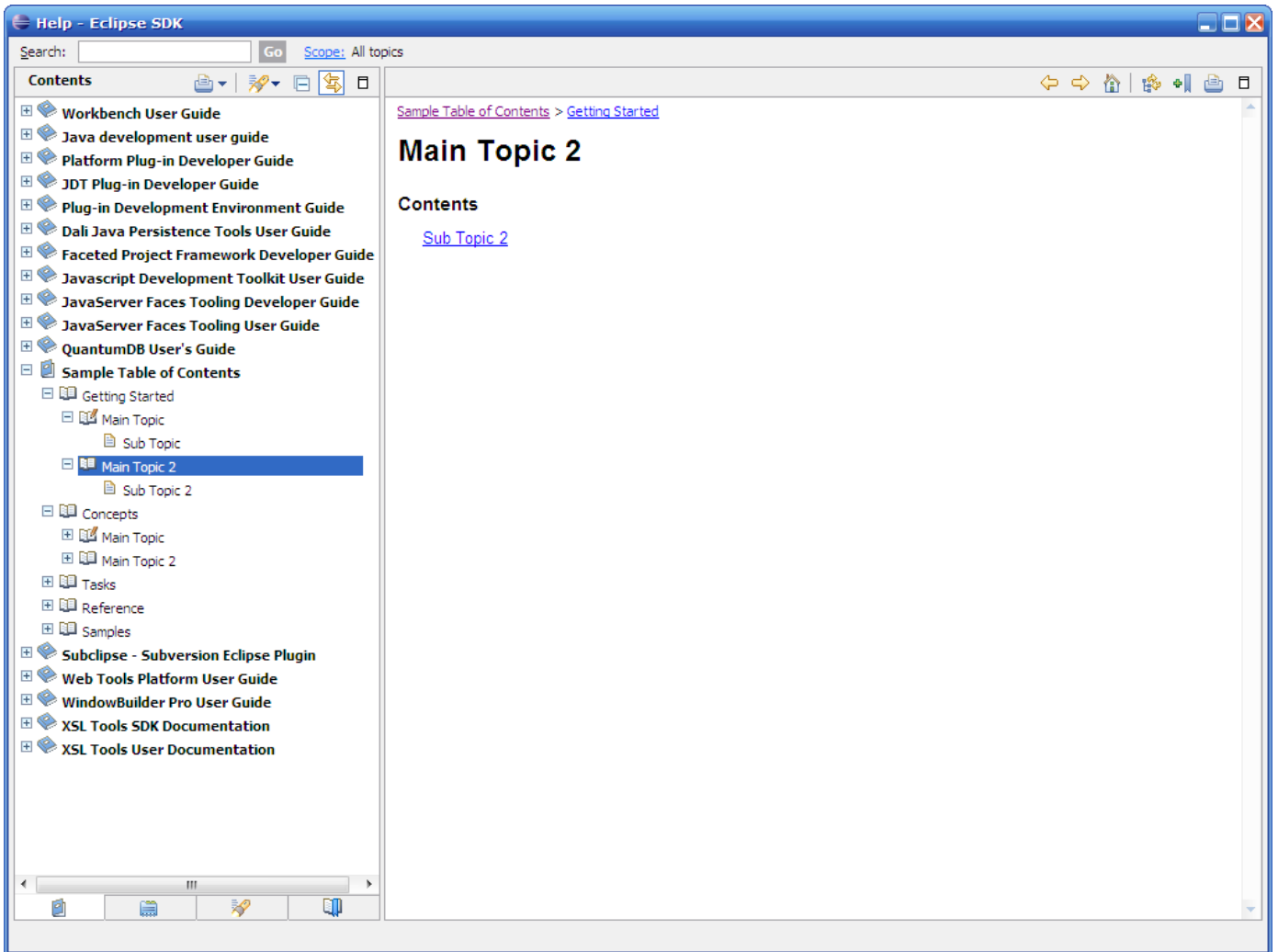


接下来选一个模板创建，





右键工程运行 Eclipse Application，然后在打开的 Eclipse 中，点击“help”-“help content”，出现如下界面，Sample Table of Content 即我们创建的帮助文档：



16 专题一 Eclipse 的版本和发行包

很多刚接触 Eclipse 的朋友在选择 Eclipse 时候会遇到这个问题，这里简单的讲解一下。这里区分一下 Eclipse 的版本（Version）和发行包（Edition）的概念。

16.1 版本 Version

16.1.1 版本的理解

版本，Version，基于同一产品、经过局部调整而形成不同款式：如软件的不同版本（如 1.0 和 2.0）之间主体功能是不变的，所变化的可能是代码进行了优化、除错，或者增加了个别新功能。从版本控制的理论上讲，同一个文档或程序，只要经过修改（哪怕是一个字符的改动）和保存，都会被视作新版本（如 1.0 到 1.01 或 1.1 之类），虽然对象的内容有些许的变化，但核心或主体是一样的。对于 Eclipse 来说，从最初的 1.0.0 到现在 3.7.0，这些都属于版本的更替。

16.1.2 Eclipse 的版本

Eclipse 最初是由 IBM 公司开发的替代商业软件 Visual Age for Java 的下一代 IDE 开发环境，2001 年 11 月贡献给开源社区，现在它由非营利软件供应商联盟 Eclipse 基金会（Eclipse Foundation）管理。2003 年，Eclipse 3.0 选择 OSGi 服务平台规范为运行时架构。2007 年 6 月，稳定版 3.3 发布。2008 年 6 月发布代号为 Ganymede 的 3.4 版。2009 年 7 月发布代号为 GALILEO 的 3.5 版。2010 年 6 月发布代号为 Helios 的 3.6 版。目前 3.7 版本也进入了 M7 阶段，代号为 Indigo。

16.1.3 版本的选择和下载

首先，对于 eclipse 的版本问题，官方建议的都是最新版的，至于用哪一个就完全看你个人喜好和公司的需要的，我一般用稍微旧一点的，新版本的太新，可能还不太稳定，等到完全发布了再更新不迟。比如 Eclipse Classic 3.7.0 M3，是最新的版本，但是还出在 Mile Stone 阶段，没到 release 阶段，可能还在测试与稳定期，所以建议你使用 Eclipse Classic 3.6.1，这个是稳定版本

再说一下 eclipse 项目发布的整个流程，这里以 GEF 为例，下面是它的发布计划：

M1	08/19/2009
M2	09/30/2009
M3	11/11/2009
M4	12/16/2009

M5	02/03/2010	
M6	03/17/2010	API Freeze
M7	05/05/2010	Feature Freeze
RC1	05/19/2010	
RC2	05/26/2010	
RC3	06/02/2010	
RC4	06/09/2010	
Final	06/16/2010	
Helios	06/23/2010	

可以看到有 M、RC、Final 之分：

M 指的就是 MileStone（里程碑）版本，这时候可能功能、API 都在完善和添加，这时候出的版本都会在名称中添加 M，比如 3.7.0 M3 或者 3.7M3a2 等

RC 指的是 Release（发布）版本，这时候功能和 API 都已经稳定，进入最后的测试和 bug 修复阶段，这时候的版本名称会有 RC 或 R 标记，比如 3.6.0R3

最后 Final 才是最终稳定版本，也就是发行的正式版本了，这时候的名称就不再有 M 和 R 了。

另外，大家可能有时候会下载一些特定的小版本，就会遇到 Stream Nightly Builds、Stream Integration Builds、Stream Stable Builds、Latest Releases 等，也简单解释一下，Nightly 版本指的是每天都在更新的版本，版本号上会添加 N 标识，而 Integration 版本指的是新集成的版本，这个版本号中有 I，有时候下载的 eclipse 中就会有这个版本的插件，而 Stable 指的是比较稳定的版本，版本号中一般有 M（里程碑）标记。

比如这个页面<http://download.eclipse.org/eclipse/downloads/>：

Build Type	Build Name	Build Status	Build Date
Latest Release	3.6.1	J U	Thu, 9 Sep 2010 -- 08:00 (-0400)
3.6.2 and 3.7 Stream Stable Build	3.6.2RC4	J U	Thu, 10 Feb 2011 -- 12:00 (-0500)
3.7 Stream Integration Build	I20110215-0800	J U	Tue, 15 Feb 2011 -- 08:00 (-0500)
3.7 Stream Nightly Build	N20110216-2000	J U	Wed, 16 Feb 2011 -- 20:00 (-0500)
3.6.x Stream Build	M20110210-1200	J U	Thu, 10 Feb 2011 -- 12:00 (-0500)
Language Pack			

Latest Releases

Build Name	Build Status	Build Date
3.6.1	J U	Thu, 9 Sep 2010 -- 08:00 (-0400)
3.6	J U	Tue, 8 Jun 2010 -- 09:11 (-0400)

3.6.2 and 3.7 Stream Stable Builds

Build Name	Build Status	Build Date
3.7M5	J U	Thu, 27 Jan 2011 -- 20:34 (-0500)
3.7M4	J U	Wed, 8 Dec 2010 -- 13:00 (-0500)
3.7M3	J U	Thu, 28 Oct 2010 -- 14:41 (-0400)
3.7M2a	J U	Tue, 21 Sep 2010 -- 10:24 (-0400)
3.7M1	J U	Thu, 5 Aug 2010 -- 17:00 (-0400)
3.6.2RC4	J U	Thu, 10 Feb 2011 -- 12:00 (-0500)
3.6.2RC3	J U	Fri, 4 Feb 2011 -- 10:30 (-0500)
3.6.2RC2	J U	Wed, 19 Jan 2011 -- 08:34 (-0500)
3.6.2RC1	J U	Wed, 12 Jan 2011 -- 08:00 (-0500)

里面详细列出了当前处于活跃状态的 Eclipse 的各个版本的进度状态，从这里可以直接进入下载。

16.2 发行包 Edition

16.2.1 发行包的理解

发行包，Edition，其实也可以成为版本，只不过这里用来与 Version 区别。它表示版本时侧重“形式、类别、时间上”的大同小异，比如：

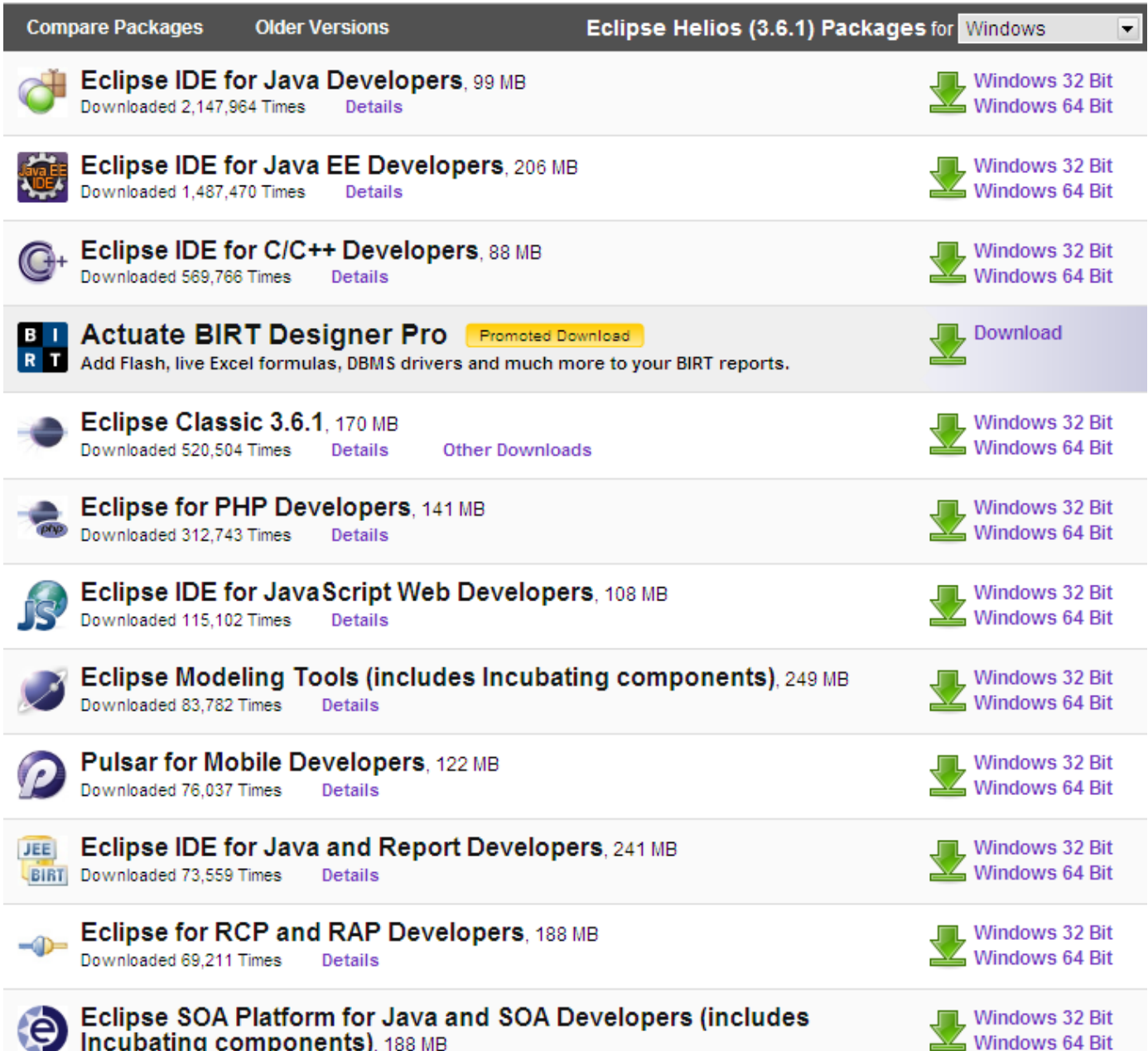
- (1) 包装/出版形式上的大同小异：如书籍、报刊、杂志等的平装版、精装版、软皮版、硬皮版等；
- (2) 分发/发布形式上的大同小异：如书籍、报刊、杂志等的纸面版、电子版、年终特别版等；
- (3) 时间先后上的大同小异：同一报刊、杂志等的第几期、第几辑等；
- (4) 用户群体上的大同小异：软件的网络版、桌面版，c 语言版、java 语言版、php 语言版，windows 版、linux 版、Mac 版等；

16.2.2 Eclipse 的发行包

Eclipse 实际上是一个平台，它最初是用来进行 Java 开发的，可是后来也融入了其他语言和技

术，所以才有了 Plug-in 开发这一行业。

打开Eclipse的官方下载页面，<http://www.eclipse.org/downloads/>，这里提供的是最新版的Eclipse的各个发行包。



Package Name	Size	Download Count	Download Link
Eclipse IDE for Java Developers	99 MB	2,147,964 Times	Windows 32 Bit Windows 64 Bit
Eclipse IDE for Java EE Developers	206 MB	1,487,470 Times	Windows 32 Bit Windows 64 Bit
Eclipse IDE for C/C++ Developers	88 MB	569,766 Times	Windows 32 Bit Windows 64 Bit
Actuate BIRT Designer Pro			Download
Eclipse Classic 3.6.1	170 MB	520,504 Times	Windows 32 Bit Windows 64 Bit
Eclipse for PHP Developers	141 MB	312,743 Times	Windows 32 Bit Windows 64 Bit
Eclipse IDE for JavaScript Web Developers	108 MB	115,102 Times	Windows 32 Bit Windows 64 Bit
Eclipse Modeling Tools (includes Incubating components)	249 MB	83,782 Times	Windows 32 Bit Windows 64 Bit
Pulsar for Mobile Developers	122 MB	76,037 Times	Windows 32 Bit Windows 64 Bit
Eclipse IDE for Java and Report Developers	241 MB	73,559 Times	Windows 32 Bit Windows 64 Bit
Eclipse for RCP and RAP Developers	188 MB	69,211 Times	Windows 32 Bit Windows 64 Bit
Eclipse SOA Platform for Java and SOA Developers (includes Incubating components)	188 MB		Windows 32 Bit Windows 64 Bit

可以看到对于当前的稳定版本 3.6.1，就有这么多的发行包，这么多的版本，到底应该下载哪一个呢？简单介绍一下：

- Eclipse IDE for Java Developers 是为 java 开发的

- Eclipse IDE for Java EE Developers 是为 J2EE 开发的
- Eclipse for RCP/Plug-in Developers 是为 RCP 和插件开发的
- Eclipse IDE for C/C++ Developers 是为 C/C++开发的
- Eclipse Classic 3.6.1 是它的经典版本

笔者一般选择 Classic 版本，它可以支持绝大部分的 Java 开发，包括 J2SE、RCP，甚至 J2EE 开发，不过建议 J2EE 的开发者还是选择 Eclipse IDE for Java EE Developers，它集成的插件和工具更丰富一下。

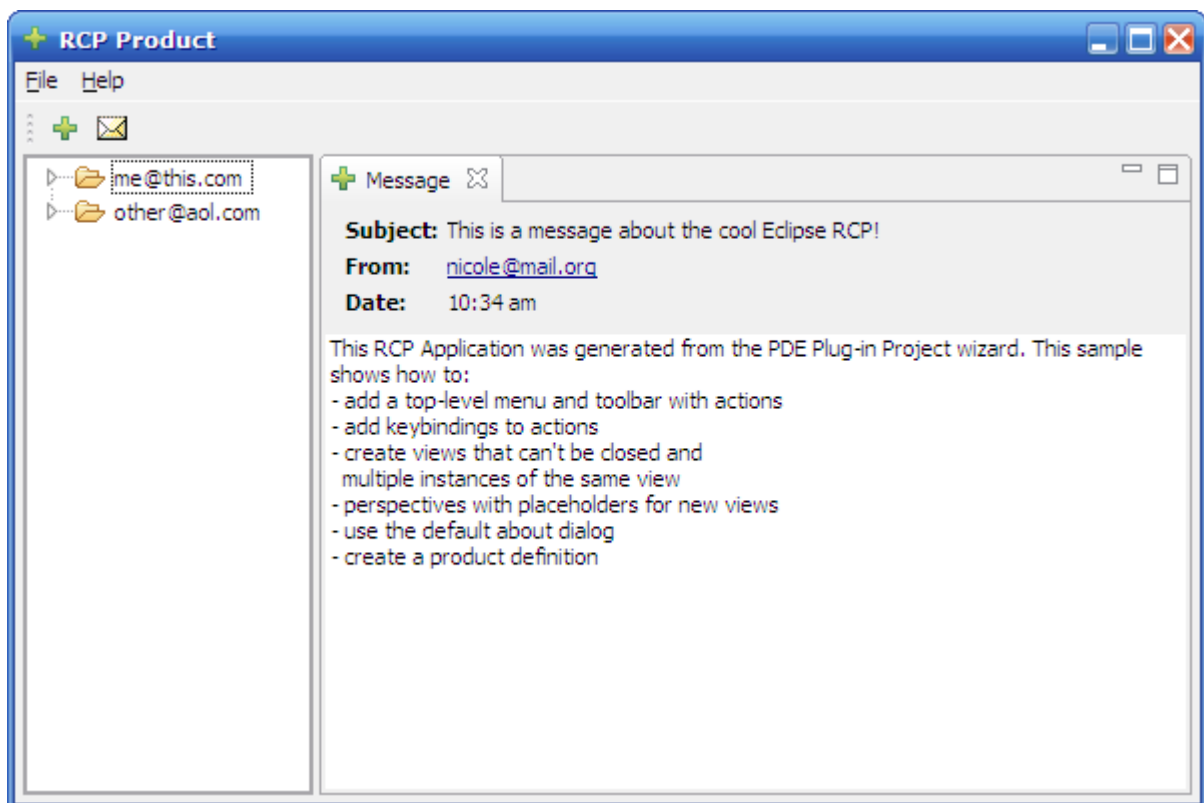
17 专题二 Eclipse 的国际化与语言包

Eclipse 默认的是英文环境，考虑到英文环境以外的开发者，它也提供了许多其他的语言包，这里简单讲讲 Eclipse 的国际化，以及如何使用它的语言包。

17.1 国际化

先讲解一下如何将一个 plug-in 国际化。

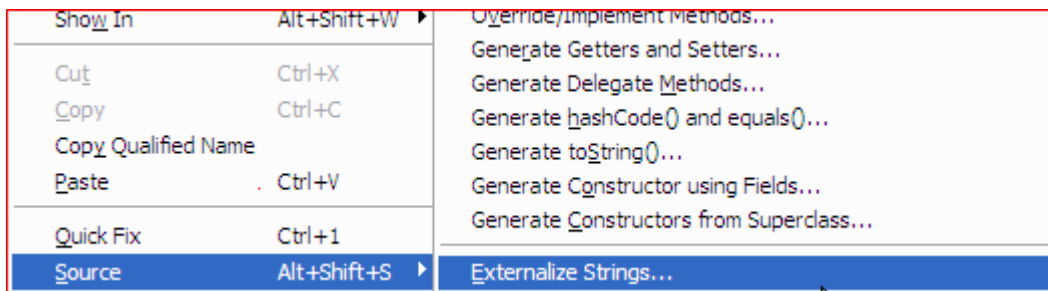
使用“RCP Mail”模板创建一个新的工程“org.salever.rcp.tech.chapter17”。我们用这个示例程序演示如何国际化插件。运行效果如下：



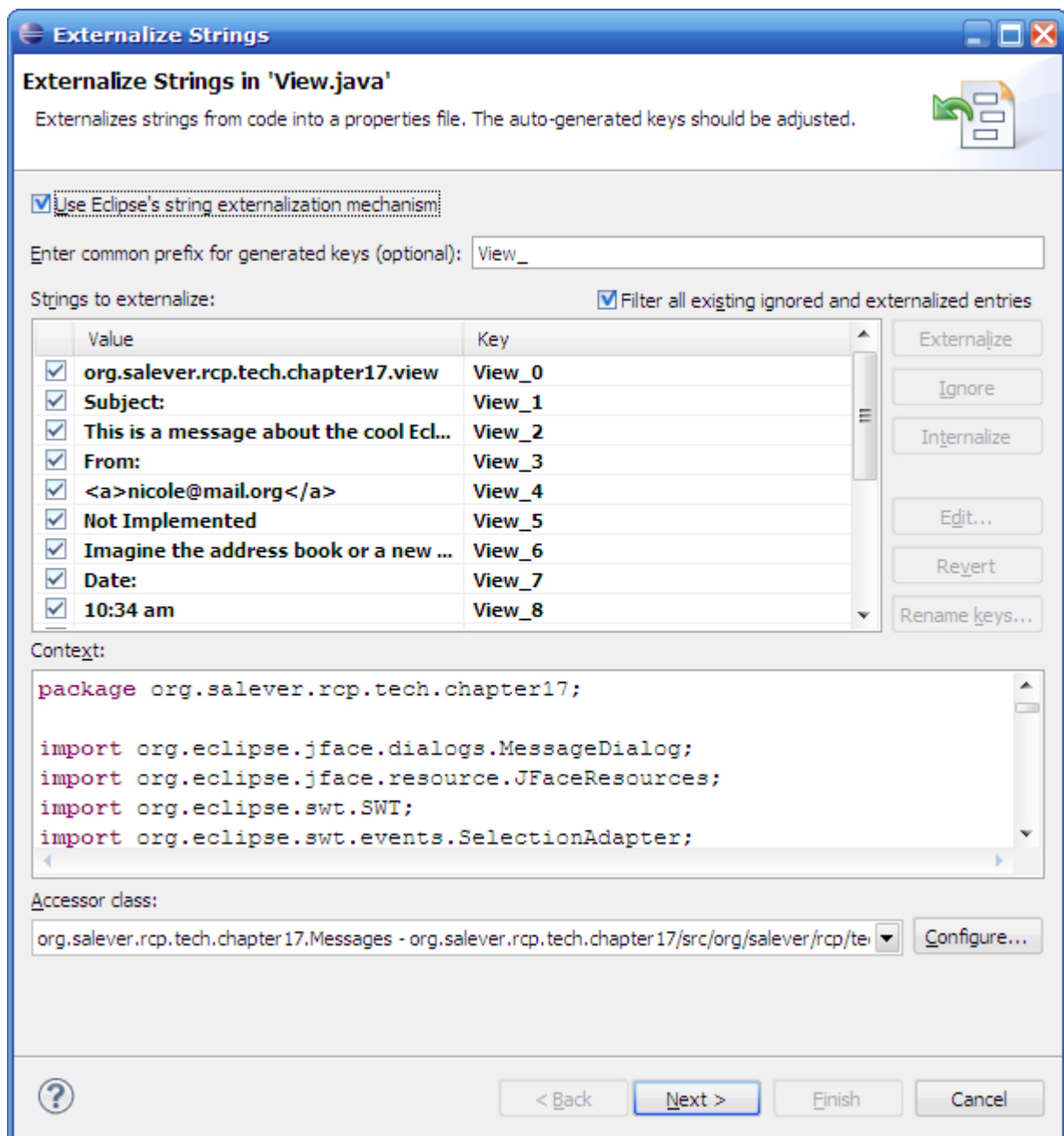
这时候我们看到菜单以及视图标题都是英文的，如果我们的客户有的要求中文界面，有的要求英文界面怎么办，这时候很明显这些描述文字不能简单的使用 Hard Code 写死。

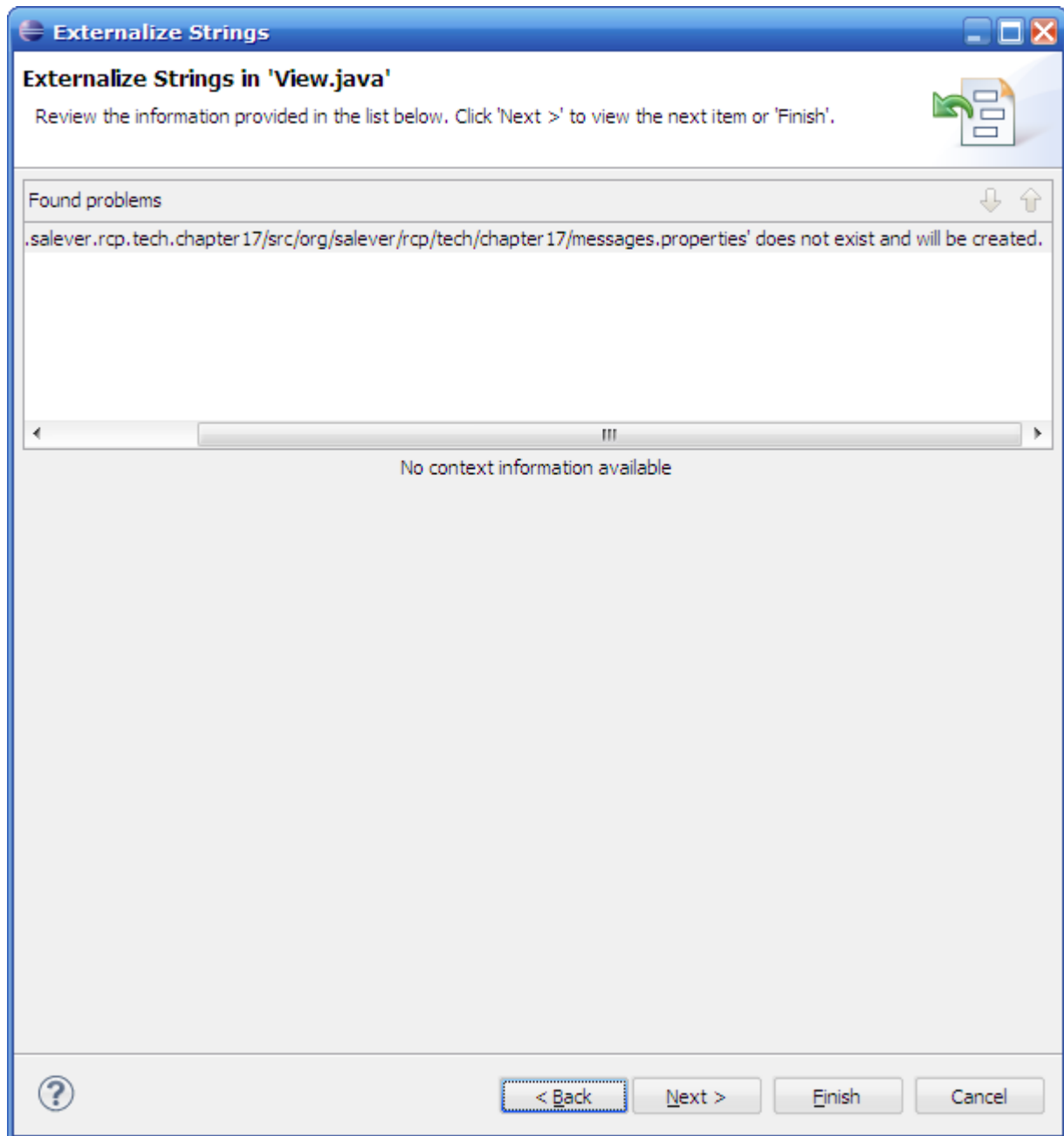
17.1.1 Externalize Strings

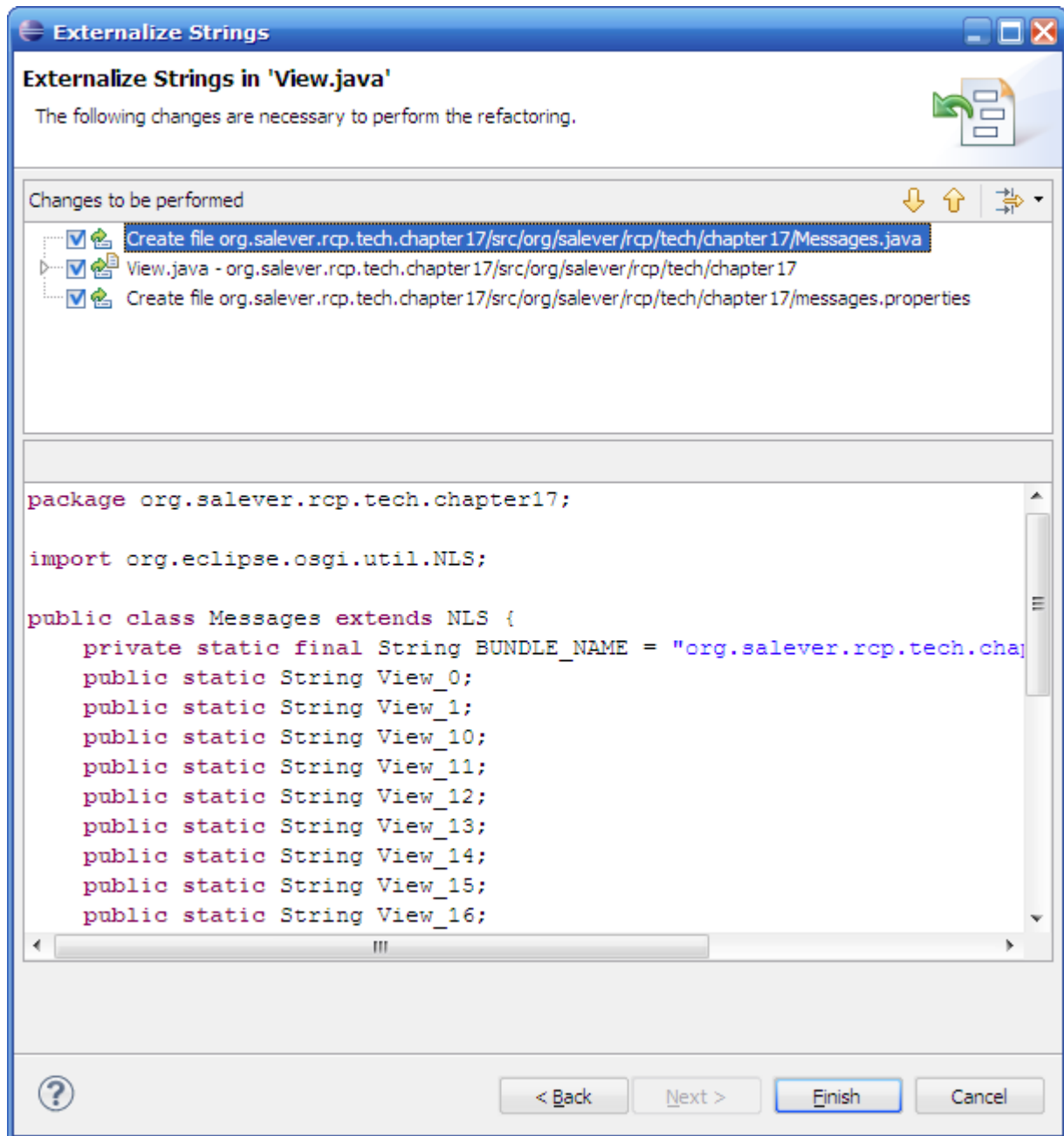
使用 Eclipse 的 Externalize String 工具，我们可以很方便的国际化我们的工程，选中一个 Java 文件，右键菜单“Source” → “Externalize String”，这里我们尝试对 View.java 国际化：



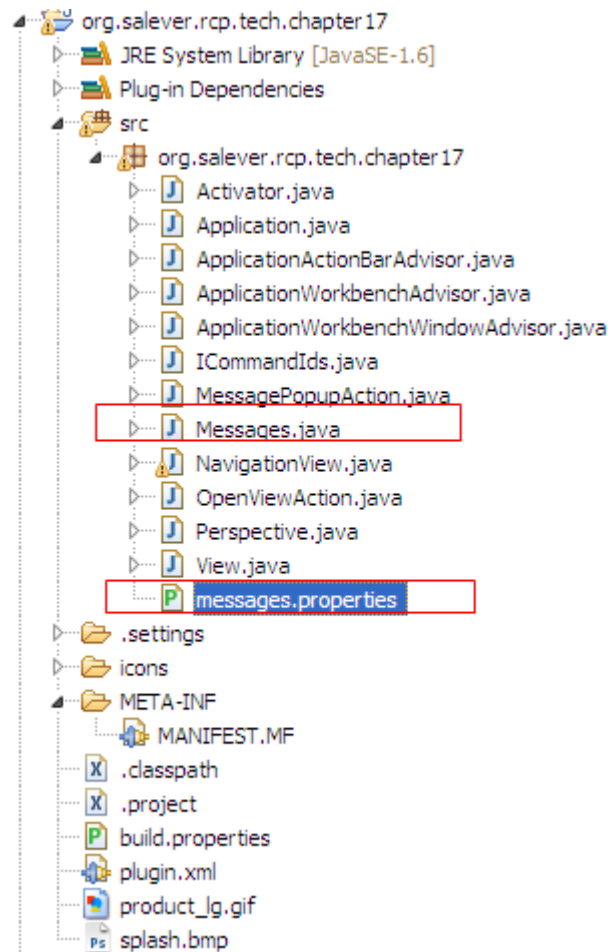
按照提示一步一步进行，







确定以后，发现 View.java 代码中，字符串都被 Messages.View_10 代替了，而工程下也出现了两个新的文件，Messages.java，messages.properties。



打开 Messages.java:

```
package org.salever.rcp.tech.chapter17;
```

```
import org.eclipse.osgi.util.NLS;
```

```
public class Messages extends NLS {
```

```
    private static final String BUNDLE_NAME = "org.salever.rcp.tech.chapter17.messages"; //$NON-NLS-1$
```

```
    public static String View_0;
```

```
    public static String View_1;
```

```
    public static String View_10;
```

```
    public static String View_11;
```

```
    public static String View_12;
```

```
    public static String View_13;
```

```
    public static String View_14;
```

```
    public static String View_15;
```

```
public static String View_16;
public static String View_2;
public static String View_3;
public static String View_4;
public static String View_5;
public static String View_6;
public static String View_7;
public static String View_8;
public static String View_9;
static {
    // initialize resource bundle
    NLS.initializeMessages(BUNDLE_NAME, Messages.class);
}

private Messages() {
}
}
messages.properties:
View_0=org.salever.rcp.tech.chapter17.view
View_1=Subject:
View_10=- add a top-level menu and toolbar with actions\n
View_11=- add keybindings to actions\n
View_12=- create views that can't be closed and\n
View_13=\\ \ multiple instances of the same view\n
View_14=- perspectives with placeholders for new views\n
View_15=- use the default about dialog\n
View_16=- create a product definition\n
View_2=This is a message about the cool Eclipse RCP!\n
View_3=From:
View_4=<a>nicole@mail.org</a>
View_5=Not Implemented
View_6=Imagine the address book or a new message being created now.
View_7=Date:
View_8=10:34 am
View_9=This RCP Application was generated from the PDE Plug-in Project wizard. This sample shows how to:\n
```

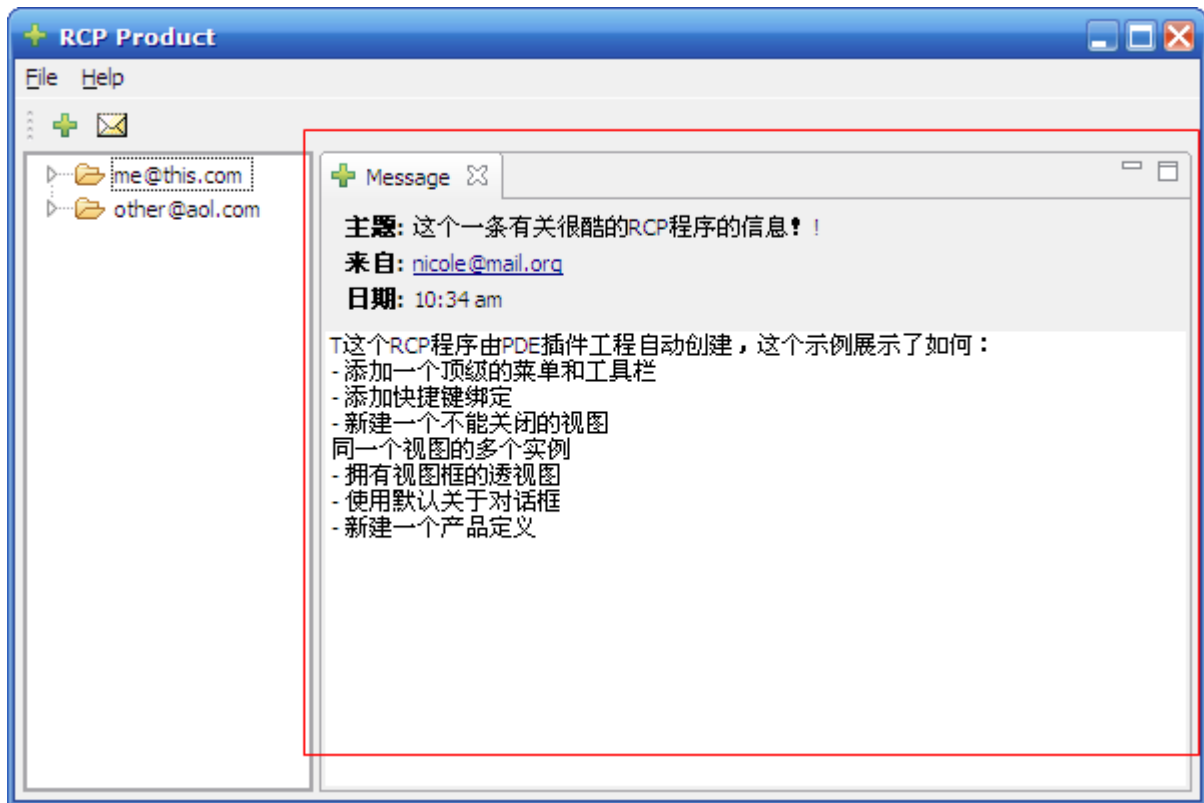
这里我们就对这个文件实现了国际化。但是怎么显示中文呢？

17.1.2 中文属性文件

在 `messages.properties` 同目录下新建 `messages_zh.properties` 文件，注意名字不能错，修改内容如下：

```
View_0=org.salever.rcp.tech.chapter17.view
View_1=\u4e3b\u9898:
View_10=- \u6dfb\u52a0\u4e00\u4e2a\u9876\u7ea7\u7684\u83dc\u5355\u548c\u5de5\u5177\u680f\n
View_11=- \u6dfb\u52a0\u5feb\u6377\u952e\u7ed1\u5b9a\n
View_12=- \u65b0\u5efa\u4e00\u4e2a\u4e0d\u80fd\u5173\u95ed\u7684\u89c6\u56fe\n
View_13=\u540c\u4e00\u4e2a\u89c6\u56fe\u7684\u591a\u4e2a\u5b9e\u4f8b\n
View_14=- \u62e5\u6709\u89c6\u56fe\u6846\u7684\u900f\u89c6\u56fe\n
View_15=- \u4f7f\u7528\u9ed8\u8ba4\u5173\u4e8e\u5bf9\u8bdd\u6846\n
View_16=- \u65b0\u5efa\u4e00\u4e2a\u4ea7\u54c1\u5b9a\u4e49\n
View_2=\u8fd9\u4e2a\u4e00\u6761\u6709\u5173\u5f88\u9177\u7684RCP\u7a0b\u5e8f\u7684\u4fe1\u606f\u01!
View_3=\u6765\u81ea:
View_4=<a>nicole@mail.org</a>
View_5=\u672a\u5b9e\u73b0
View_6=\u60f3\u8c61\u4e00\u4e0b\u5730\u5740\u8584\u6216\u8005\u4e00\u4e2a\u65b0\u7684\u6d88\u606f\u5df
2\u7ecf\u88ab\u521b\u5efa
View_7=\u65e5\u671f:
View_8=10:34 am
View_9=T\u8fd9\u4e2aRCP\u7a0b\u5e8f\u7531PDE\u63d2\u4ef6\u5de5\u7a0b\u81ea\u52a8\u521b\u5efa\u0c\u8f
d9\u4e2a\u793a\u4f8b\u5c55\u793a\u4e86\u5982\u4f55\u01a\n
```

再次运行 RCP 程序，效果为（红色部分已经变成中文）：



17.1.3 国际化文件

Eclipse RCP 程序正是通过这些 properties 文件进行国际化的，对于如下两个文件：

- messages.properties
- messages_zh.properties

默认情况下会选择 messages.properties，在中文环境下，则会自动选择 messages_zh.properties。

Eclipse 正是通过 zh 识别中文的，相似的还有 zh_CN（繁体中文）、zh_TW（繁体中文）、en（英文）、en_US（美式英文）、en_UK（英式英文）等。

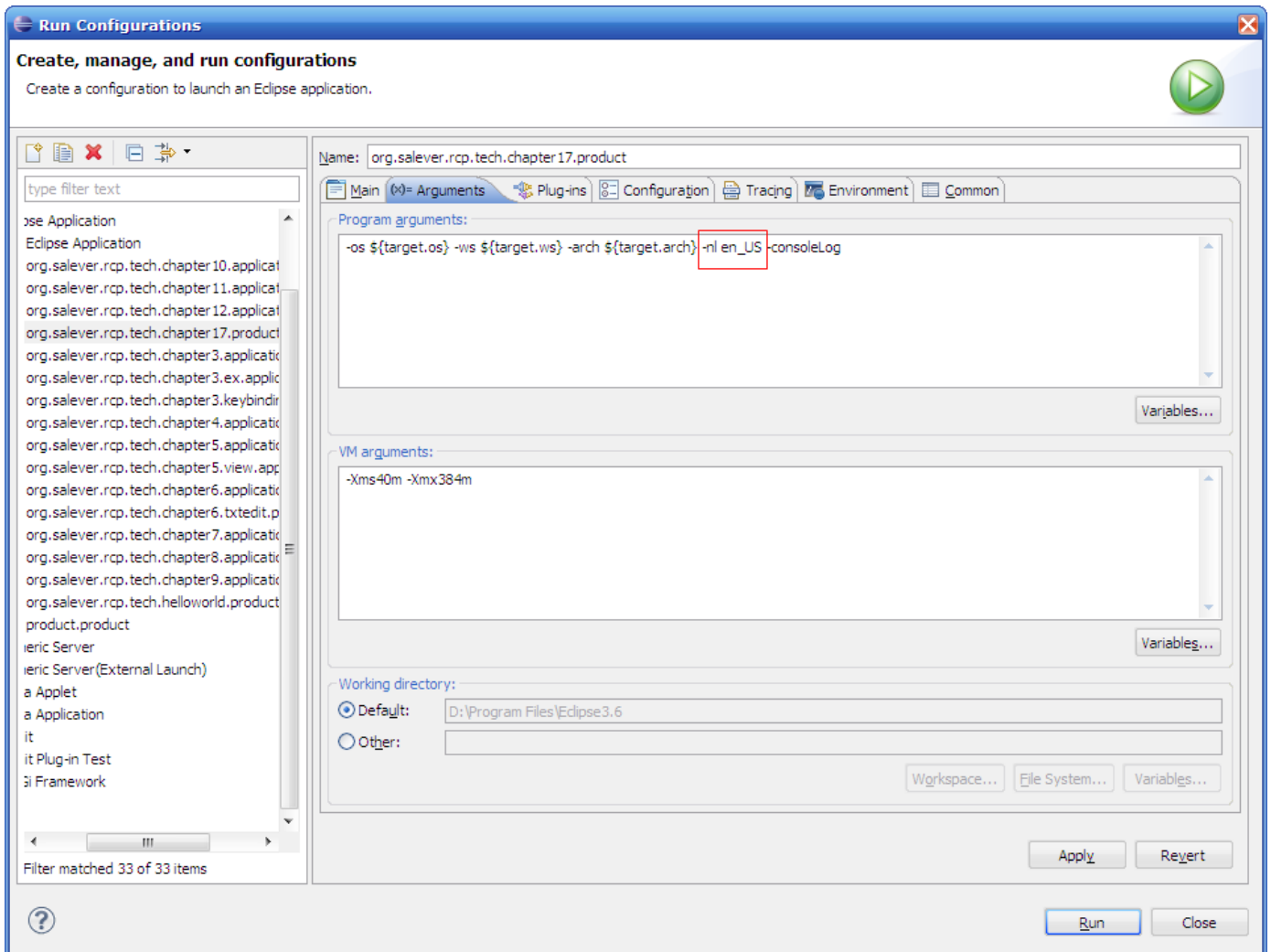
对于中文属性文件，内容必须是 unicode 的，不然会出现乱码。这里介绍一下如何将中文文字转换为 Unicode 编码格式。JDK 的 bin 目录下有一个 native2ascii 程序，使用它可以很轻松的将中文转为 Unicode，格式为：

```
Native2ascii messages_zh.properties.src messages_zh.properties
messages_zh.properties.src 为中文源文件。
```

17.1.4 指定语言环境

RCP 程序国际化以后，可以通过指定语言环境的方式，强制显示某一个语言，比如中文环境

下，在运行的时候附上“-nl en_US”参数，即使提供了中文语言包，RCP 程序也会显示为英文。
类似的，测试的时候，在 Run Configuration 的配置中，修改 Arguments，也会有此效果。



17.1.5 Propedit 工具

这里提供一个属性文件编辑工具，中文下编辑，然后直接将中文保存为 Unicode。

地址：http://propedit.sourceforge.jp/index_en.html

在线更新地址：<http://propedit.sourceforge.jp/eclipse/updates/>


17.2 语言包

17.2.1 Babel 小组


Eclipse 的语言国际化现在由 babel 小组负责，他们仅提供和维护大版本的 Eclipse 的语言包，比如 3.6，3.6.1 之类的版本仍然可以选择 3.6 的，影响不大。


打开babel的官方地址，<http://www.eclipse.org/babel/downloads.php>,


Babel Language Packs - 0.8.1

 Babel Language Pack Zips
[Helios](#) | [Galileo](#) | [Ganymede](#) | [Europa](#)
[Installation instructions](#)

Babel Update Sites - 0.8.1

 Babel Language Pack Update Site for Helios
<http://download.eclipse.org/technology/babel/update-site/R0.8.1/helios>

 Babel Language Pack Update Site for Galileo
<http://download.eclipse.org/technology/babel/update-site/R0.8.1/galileo>

 Babel Language Pack Update Site for Ganymede
<http://download.eclipse.org/technology/babel/update-site/R0.8.1/ganymede>

可以选择在线更新或者下载离线更新包，网络环境不理想的情况下，推荐下载离线更新方式。

17.2.2 中文语言包的下载

这里来以Eclipse 3.6.1 为例，讲解如何下载对应的中文语言包。点击图中的Helios链接，进入地址http://download.eclipse.org/technology/babel/babel_language_packs/R0.8.1/helios.php,

Language: Chinese (Simplified)

- BabelLanguagePack-birt-zh_3.6.0.v20101211043401.zip (94.91%)
- BabelLanguagePack-datatools-zh_3.6.0.v20101211043401.zip (84.64%)
- BabelLanguagePack-dsdp.mtj-zh_3.6.0.v20101211043401.zip (15.75%)
- BabelLanguagePack-dsdp.sequoyah-zh_3.6.0.v20101211043401.zip (14.62%)
- BabelLanguagePack-dsdp.tm-zh_3.6.0.v20101211043401.zip (24.77%)
- BabelLanguagePack-eclipse-zh_3.6.0.v20101211043401.zip (92.91%)
- BabelLanguagePack-modeling.emf.validation-zh_3.6.0.v20101211043401.zip (67.37%)
- BabelLanguagePack-modeling.emft.eef-zh_3.6.0.v20101211043401.zip (39.89%)
- BabelLanguagePack-modeling.gmp.gmf-notation-zh_3.6.0.v20101211043401.zip (41.15%)
- BabelLanguagePack-modeling.gmp.gmf-runtime-zh_3.6.0.v20101211043401.zip (35.73%)
- BabelLanguagePack-modeling.gmp.gmf-tooling-zh_3.6.0.v20101211043401.zip (31.98%)
- BabelLanguagePack-modeling.mdt.ocl-zh_3.6.0.v20101211043401.zip (40.51%)
- BabelLanguagePack-rt.equinox-zh_3.6.0.v20101211043401.zip (100%)
- BabelLanguagePack-rt.equinox.p2-zh_3.6.0.v20101211043401.zip (24.77%)
- BabelLanguagePack-rt.rap-zh_3.6.0.v20101211043401.zip (76.7%)
- BabelLanguagePack-technology.actf-zh_3.6.0.v20101211043401.zip (7.05%)
- BabelLanguagePack-technology.egit-zh_3.6.0.v20101211043401.zip (10.53%)
- BabelLanguagePack-technology.jgit-zh_3.6.0.v20101211043401.zip (0.53%)
- BabelLanguagePack-technology.linux-distros-zh_3.6.0.v20101211043401.zip (28.37%)
- BabelLanguagePack-technology.packaging.mpc-zh_3.6.0.v20101211043401.zip (10.48%)
- BabelLanguagePack-tools.cdt-zh_3.6.0.v20101211043401.zip (72.44%)
- BabelLanguagePack-tools.gef-zh_3.6.0.v20101211043401.zip (69.9%)
- BabelLanguagePack-tools.mat-zh_3.6.0.v20101211043401.zip (3.84%)
- BabelLanguagePack-tools.mylyn-zh_3.6.0.v20101211043401.zip (50.8%)
- BabelLanguagePack-tools.ptp-zh_3.6.0.v20101211043401.zip (17.54%)
- BabelLanguagePack-tools.ptp.photran-zh_3.6.0.v20101211043401.zip (28.68%)
- BabelLanguagePack-webtools.common-zh_3.6.0.v20101211043401.zip (57.76%)
- BabelLanguagePack-webtools.dali-zh_3.6.0.v20101211043401.zip (24.78%)
- BabelLanguagePack-webtools.ejbtools-zh_3.6.0.v20101211043401.zip (95.14%)
- BabelLanguagePack-webtools.jeetools-zh_3.6.0.v20101211043401.zip (100%)
- BabelLanguagePack-webtools.jsdt-zh_3.6.0.v20101211043401.zip (80.47%)
- BabelLanguagePack-webtools.jsf-zh_3.6.0.v20101211043401.zip (39.73%)
- BabelLanguagePack-webtools.servertools-zh_3.6.0.v20101211043401.zip (88.38%)
- BabelLanguagePack-webtools.sourceediting-zh_3.6.0.v20101211043401.zip (75.61%)
- BabelLanguagePack-webtools.webservices-zh_3.6.0.v20101211043401.zip (75.69%)

找到Chinese (Simplified)，下面对应的就是简体中文语言包了，后面的比例为汉化的进度。比如想汉化 Eclipse UI，选择下载 [BabelLanguagePack-eclipse-zh_TW_3.6.0.v20101211043401.zip \(89.61%\)](#)，其他的类推。

至于下载下来的压缩包怎么用的问题，解压完了直接复制到 eclipse\dropins 目录里面就行了，不建议直接放到 plugins 目录下，注意保持文件架构完整，比如解压后得到 BabelLanguagePack-eclipse-zh_3.6.0.v20100814043401，将其复制到 eclipse\dropins 目录里面就行

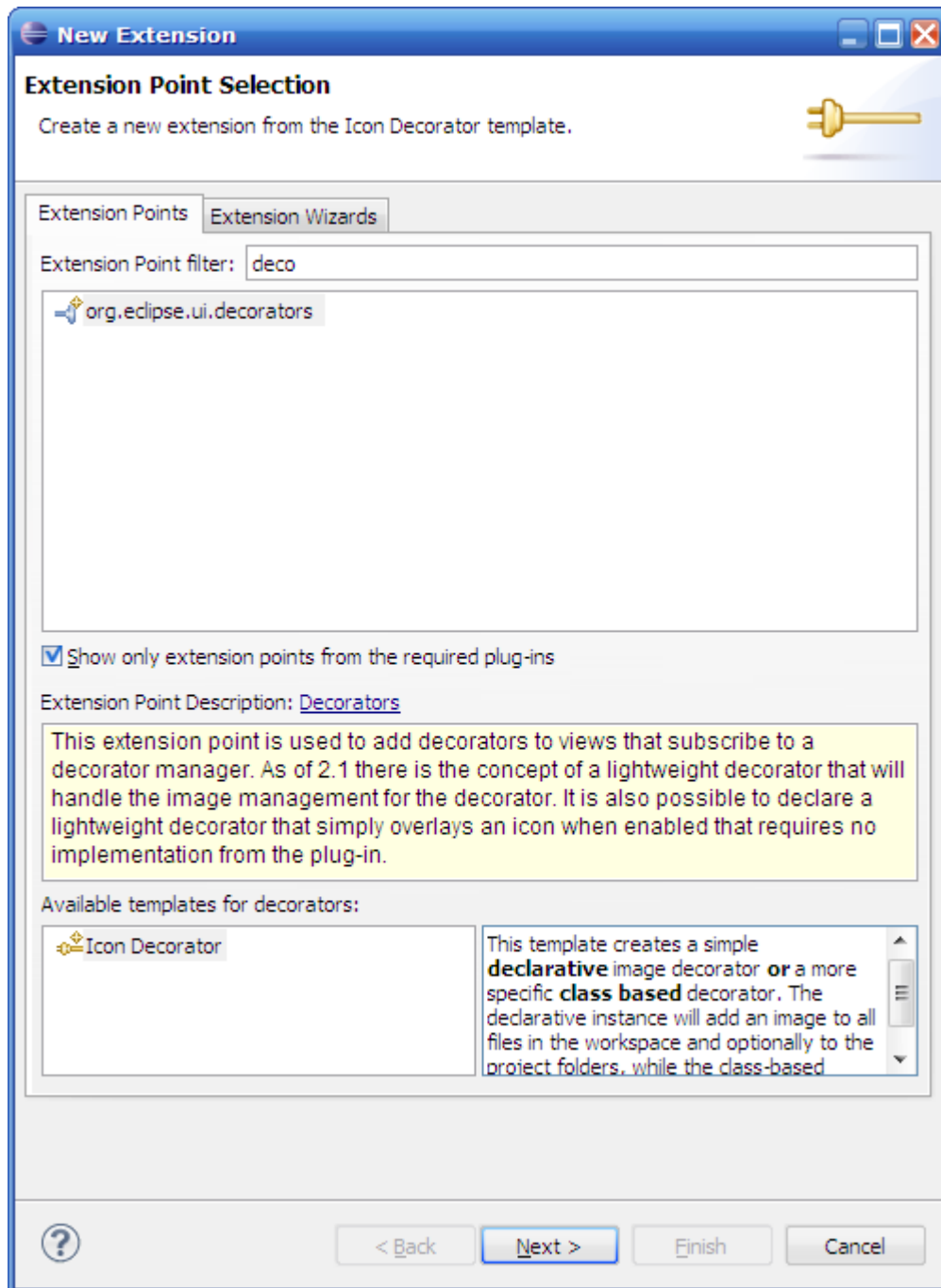
18 专题三 Decorator 与 Marker 的使用

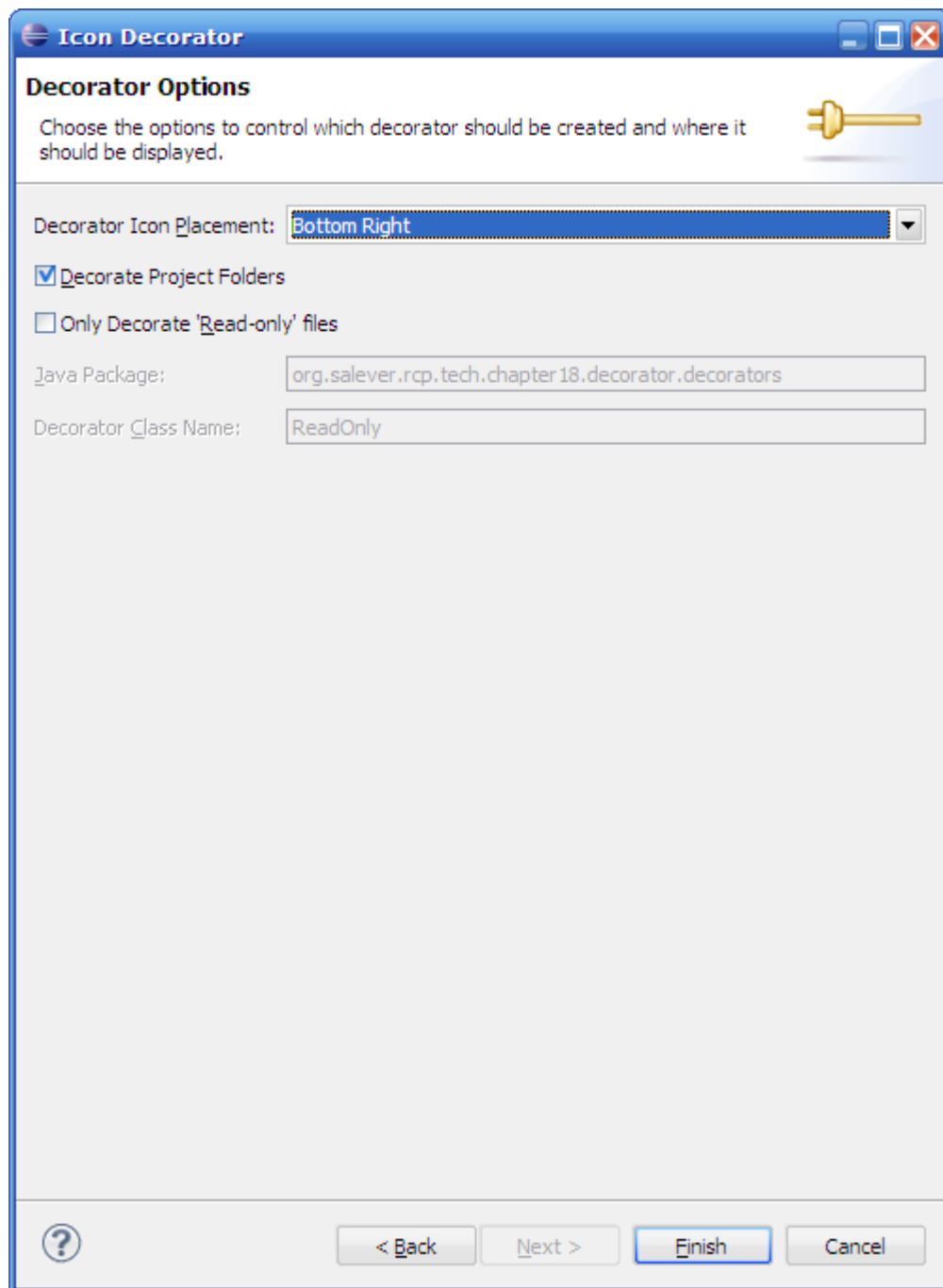
18.1 简介

Decorator 和 Marker 都是 Eclipse 中用来进行标记和修饰的，decorator 用来进行图标修饰，常见的应用为错误图标修饰、SVN 等的修饰图标，而 Marker 用于标记，常见应用为 TODO 标记、错误标记等。

18.2 扩展 Decorator

新建一个 plug-in 工程 “org.salever.rcp.tech.chapter18.decorator”，不使用任何模板，打开 MANIFEST.MF 文件，添加扩展点 org.eclipse.ui.decorators，使用 Icon Decorator 模板。



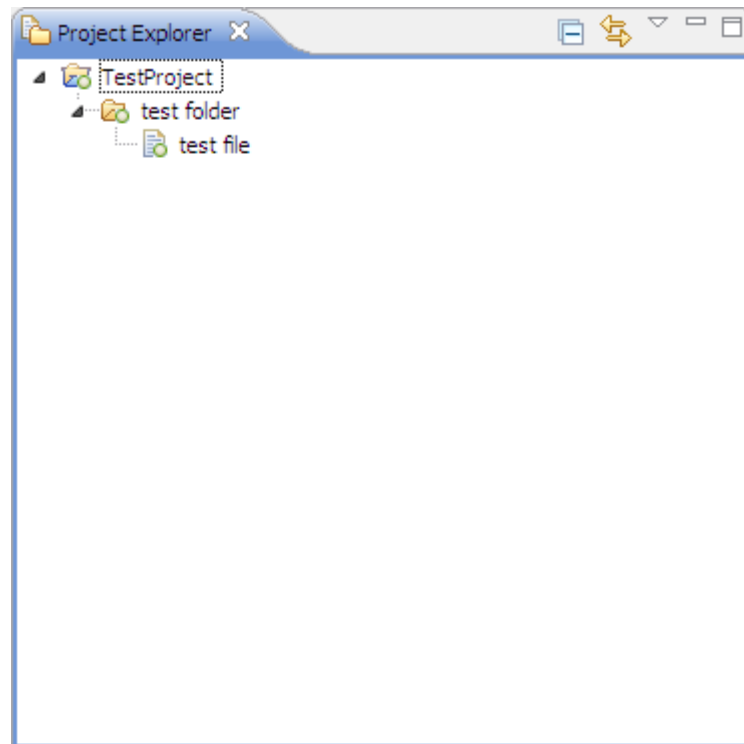


添加的代码为：

```
<extension
    point="org.eclipse.ui.decorators">
<decorator
    adaptable="true"
    icon="icons/sample_decorator.gif"
```

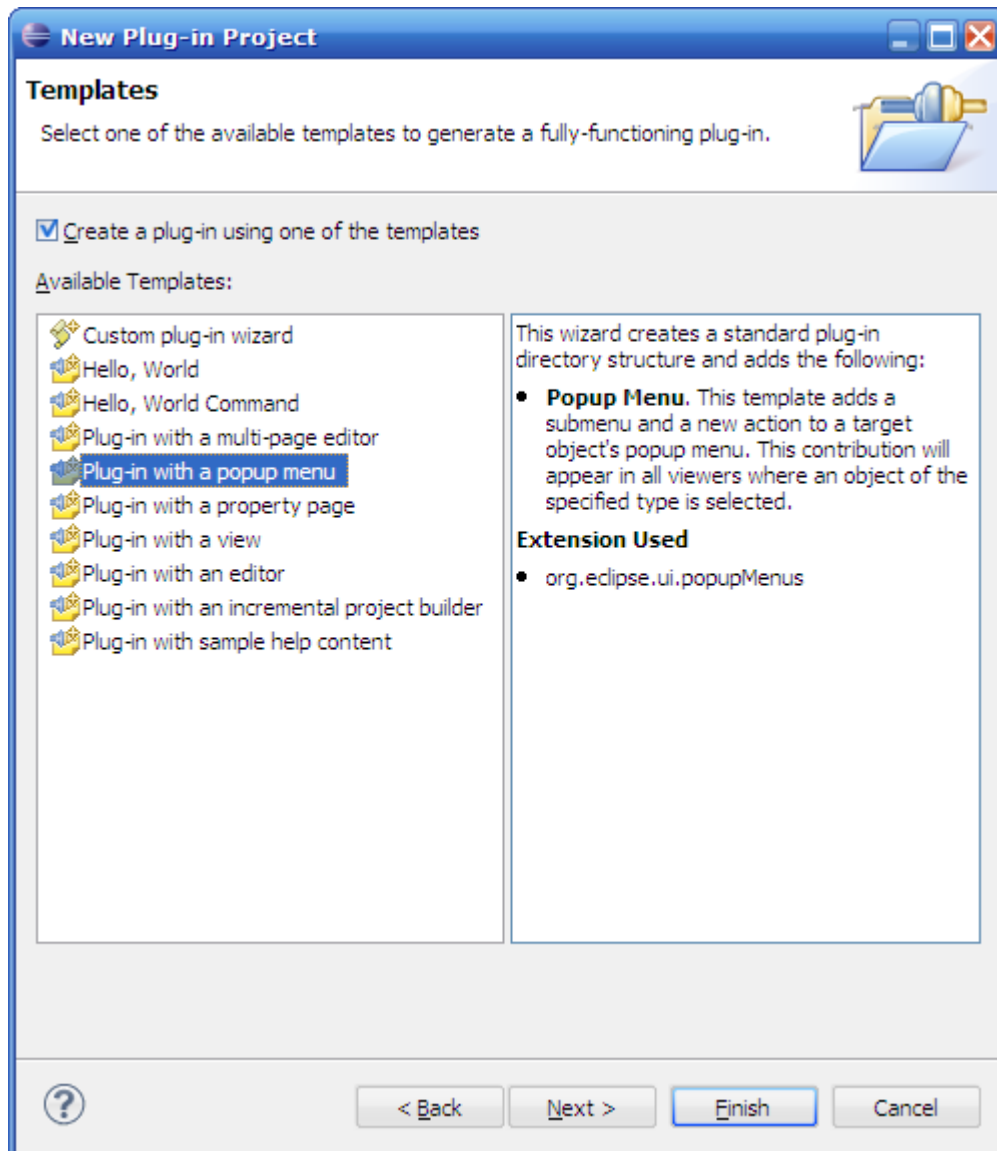
```
        id="org.salever.rcp.tech.chapter18.decorator"
        label="Resource Decorator"
        lightweight="true"
        location="BOTTOM_RIGHT"
        state="true">
    <enablement>
        <and>
            <objectClass
                name="org.eclipse.core.resources.IResource">
            </objectClass>
            <or>
                <objectClass
                    name="org.eclipse.core.resources.IProject">
                </objectClass>
                <objectClass
                    name="org.eclipse.core.resources.IFile">
                </objectClass>
            </or>
        </and>
    </enablement>
</decorator>
</extension>
```

作用为给所有工程视图中的所有资源（工程、文件夹、文件）图标右下角添加一个修饰图标，右键工程，Run As—> Eclipse Application，在新打开的 Eclipse 中，切换到 Resource 透视图，然后依次新建一个工程“TestProject”、文件夹“Test folder”和文件“Test file”，在 Project Explorer 里面查看 decorator 效果：



18.3 扩展 Marker

新建 plug-in 工程，使用“Plug-in with a popup menu”模板。



修改 plugin.xml, 将 Action 的名称改为 Test Maker Action,

```
<extension
```

```
    point="org.eclipse.ui.popupMenus">
```

```
<objectContribution
```

```
    objectClass="org.eclipse.core.resources.IFile"
```

```
    id="org.salever.rcp.tech.chapter18.marker.contribution1">
```

```
<menu
```

```
    label="Test Marker Group"
```

```
    path="additions"
```

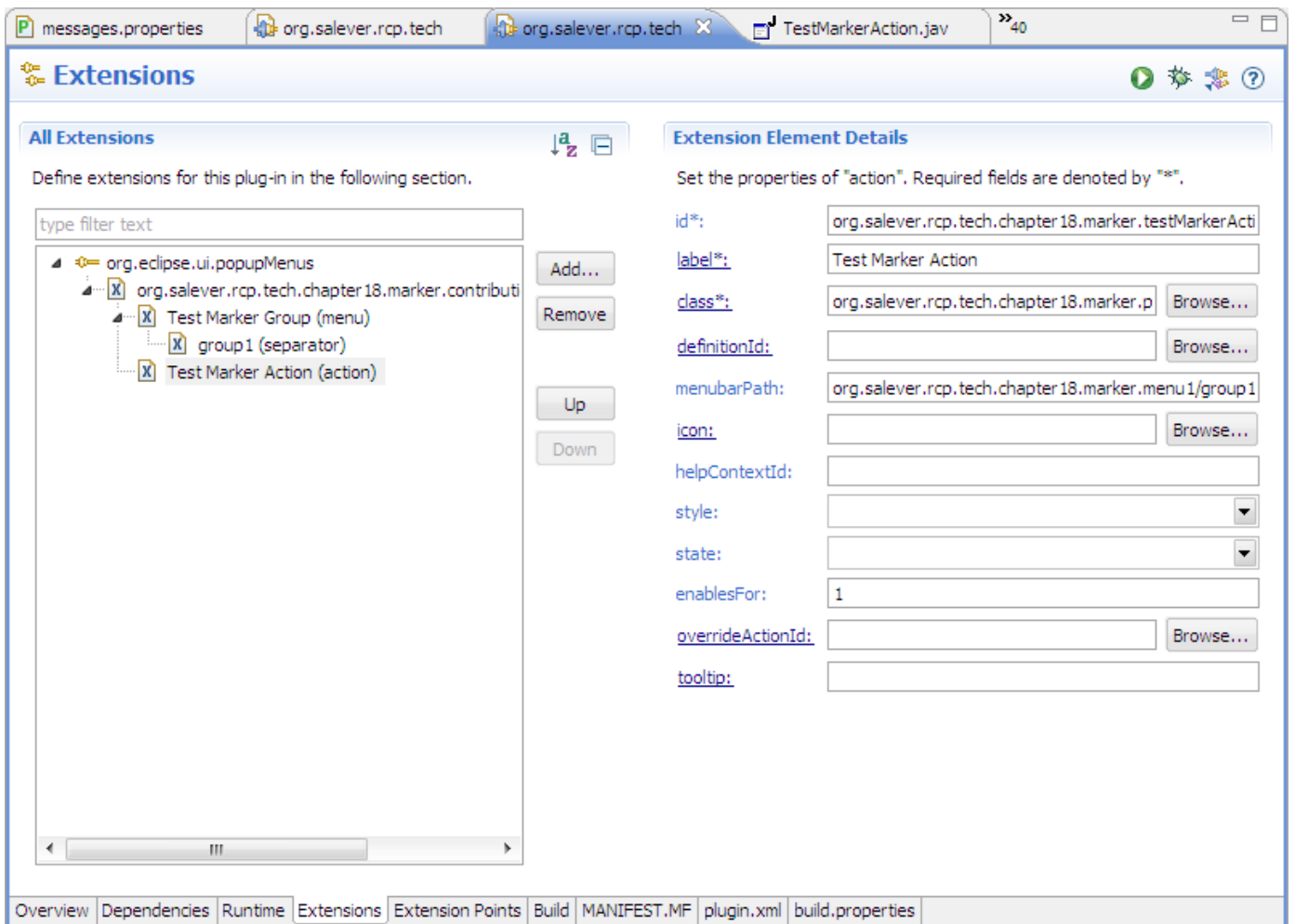
```
    id="org.salever.rcp.tech.chapter18.marker.menu1">
```

```
<separator
```

```

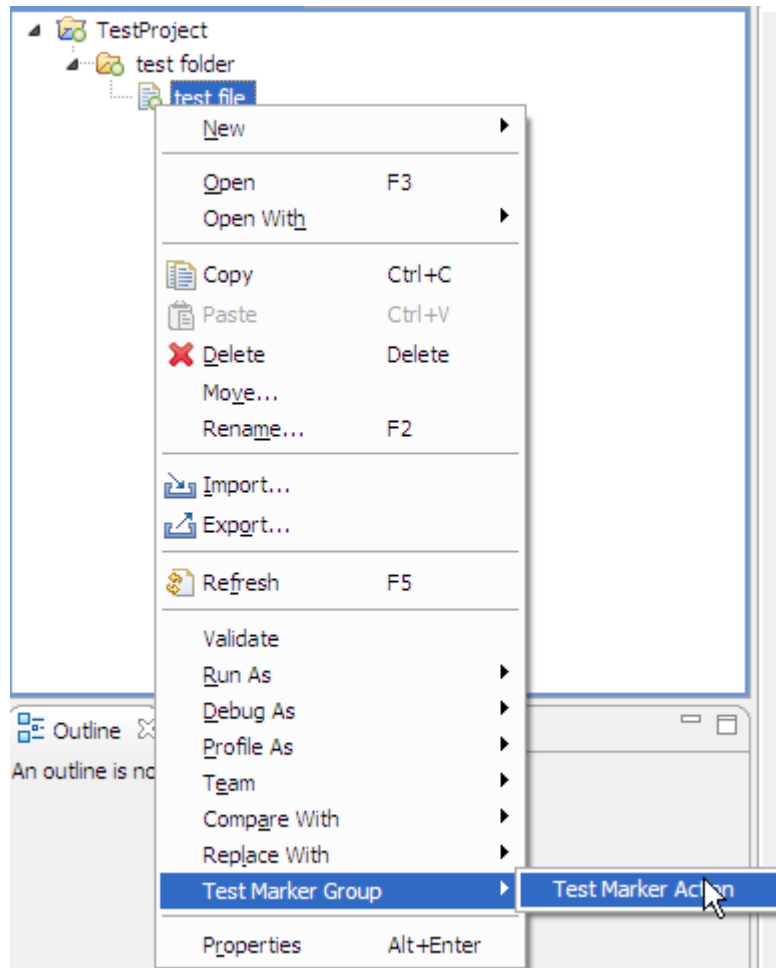
        name="group1">
    </separator>
</menu>
<action
    label="Test Marker Action"
    class="org.salever.rcp.tech.chapter18.marker.popup.actions.TestMarkerAction"
    menubarPath="org.salever.rcp.tech.chapter18.marker.menu1/group1"
    enablesFor="1"
    id="org.salever.rcp.tech.chapter18.marker.testMarkerAction">
</action>
</objectContribution>
</extension>

```



右键工程，Run As → Eclipse Application，在我们刚刚创建的 TestProject 中的 Test file 文件右键

菜单，看到新建的 Test Marker Group→Test Marker Action



接下来我们要添加 Marker 扩展了，新建 org.eclipse.core.resources.markers 扩展：

```
<extension
    id="org.salever.rcp.tech.chapter18.testMarker"
    name="Test Folder Marker"
    point="org.eclipse.core.resources.markers">
<persistent
    value="true">
</persistent>
<super
    type="org.eclipse.core.resources.bookmark">
</super>
</extension>
```

然后修改 TestMarkerAction.java，使其操作 marker：

```
package org.salever.rcp.tech.chapter18.marker.popup.actions;

import org.eclipse.core.resources.IMarker;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.StructuredSelection;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.IActionDelegate;
import org.eclipse.ui.IObjectActionDelegate;
import org.eclipse.ui.IWorkbenchPart;

public class TestMarkerAction implements IObjectActionDelegate {

    private Shell shell;

    private IResource resource;

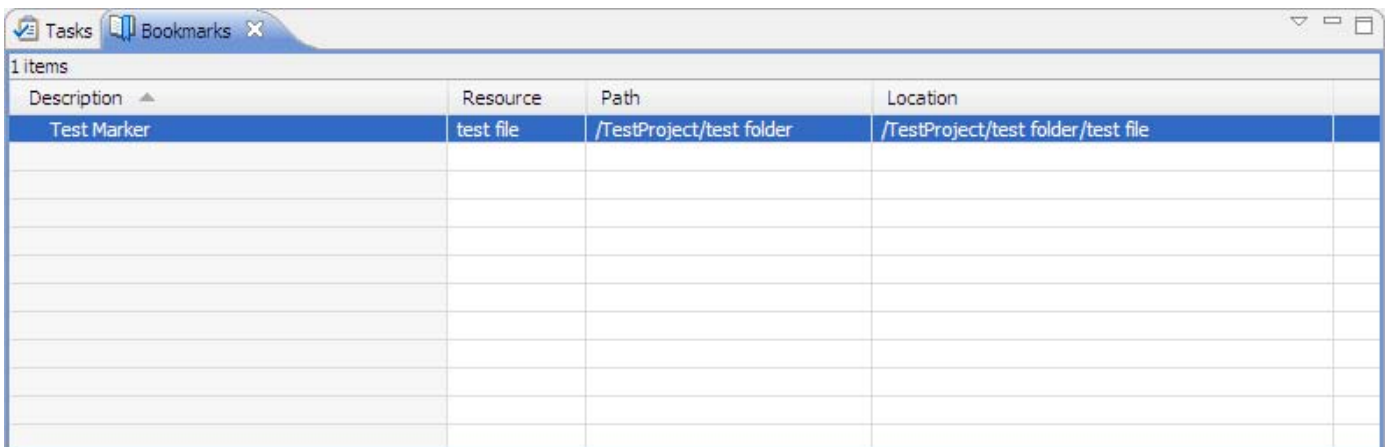
    /**
     * Constructor for Action1.
     */
    public TestMarkerAction() {
        super();
    }

    /**
     * @see IObjectActionDelegate#setActivePart(IAction, IWorkbenchPart)
     */
    public void setActivePart(IAction action, IWorkbenchPart targetPart) {
        shell = targetPart.getSite().getShell();
    }

    /**
     * @see IActionDelegate#run(IAction)
     */
}
```

```
*/  
public void run(IAction action) {  
  
    try {  
        IMarker marker;  
        marker = resource  
            .createMarker("org.salever.rcp.tech.chapter18.testMarker");  
        marker.setAttribute(IMarker.MESSAGE, "Test Marker");  
        marker.setAttribute(IMarker.LOCATION, resource.getFullPath()  
            .toString());  
    } catch (CoreException e) {  
        e.printStackTrace();  
    }  
}  
/**  
 * @see IActionDelegate#selectionChanged(IAction, ISelection)  
 */  
public void selectionChanged(IAction action, ISelection selection) {  
    StructuredSelection sselection = (StructuredSelection) selection;  
    resource = (IResource) sselection.getFirstElement();  
}  
}
```

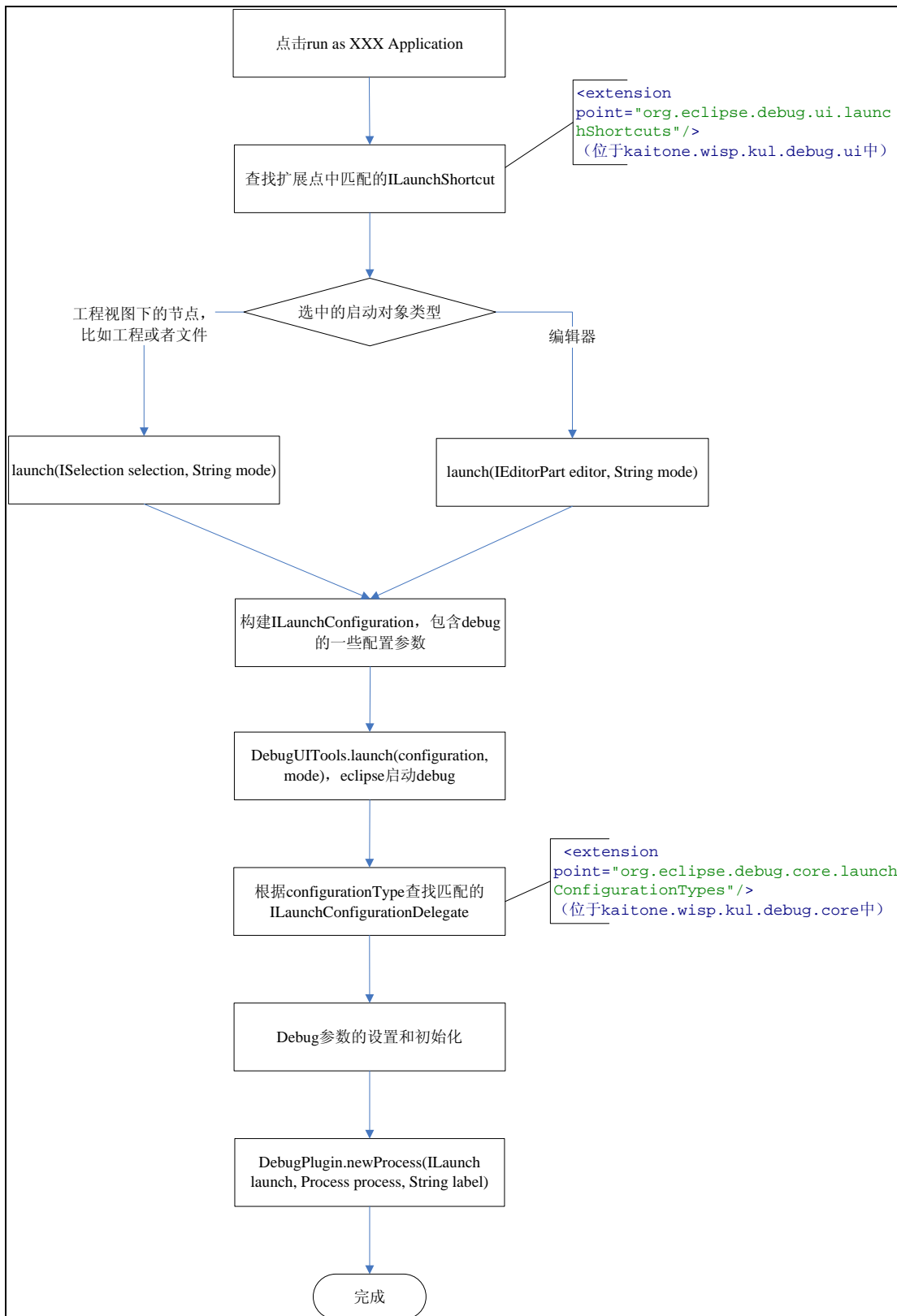
运行 plugin，然后点击 Test Marker Action 菜单，这时候打开 bookmark 视图，会发现里面添加了一条记录：



Description	Resource	Path	Location
Test Marker	test file	/TestProject/test folder	/TestProject/test folder/test file

19 专题四 Run/Debug Launcher 实现

19.1 Eclipse Run/Debug 实现流程



19.2 扩展 configurationType

添加扩展点 org.eclipse.debug.core.launchConfigurationTypes:

```
<extension
    point="org.eclipse.debug.core.launchConfigurationTypes">
    <launchConfigurationType
        delegate="org.salever.rcp.examples.debug.launch.XMLLaunchConfigurationDelegate"
        id="org.salever.rcp.debug.demo.xmlLaunchConfigurationType"
        modes="run, debug"
        name="Launch XML debug">
    </launchConfigurationType>
</extension>
```

类 org.salever.rcp.tech.chapter19.launch.XMLLaunchConfigurationDelegate 负责定义 Launch Configuration:

```
package org.salever.rcp.tech.chapter19.launch;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.debug.core.DebugPlugin;
import org.eclipse.debug.core.ILaunch;
import org.eclipse.debug.core.ILaunchConfiguration;
import org.eclipse.debug.core.model.ILaunchConfigurationDelegate;
import org.eclipse.debug.core.model.IProcess;
import org.salever.rcp.tech.chapter19.DebugConstants;

/**
 * Using "Run"-->"Run Configurations"--> "New Configuration" --> "Run" will lead here.
 * @author salever
 *
 */
```



```
*/
public class XMLLaunchConfigurationDelegate implements
    ILaunchConfigurationDelegate {

    /* (non-Javadoc)
     *
     * @see
     org.eclipse.debug.core.model.ILaunchConfigurationDelegate#launch(org.eclipse.debug.core.ILaunchConfiguration,
     java.lang.String, org.eclipse.debug.core.ILaunch, org.eclipse.core.runtime.IProgressMonitor)
     */
    @Override
    public void launch(ILaunchConfiguration configuration, String mode,
        ILaunch launch, IProgressMonitor monitor) throws CoreException {

        //Using configuration to build command line
        List<String> cmdLine = new ArrayList<String>();
        cmdLine.add("C:\\windows\\NOTEPAD.EXE");//Application path should be stored in preference.
        String file = configuration.getAttribute(DebugConstants.XML_FILE, "");
        String filePath = ResourcesPlugin.getWorkspace().getRoot().findMember(file).getLocation().toOSString();
        cmdLine.add(filePath);//path is relative, so can not found it.
        String[] cmds = {};
        cmds = cmdLine.toArray(cmds);
        try {
            IProcess p = DebugPlugin.newProcess(
                launch,
                //Launch a process to debug.eg, launch NOTEPAD.EXE to show the XML file we choose.
                Runtime.getRuntime().exec(cmds, null),
                "Tomcat debug Process");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

19.3 扩展 ILaunchShortcut

添加扩展点 org.eclipse.debug.ui.launchShortcuts:

```
<extension
  point= "org.eclipse.debug.ui.launchShortcuts" >
  <shortcut
    class= "org.salever.rcp.examples.debug.launch.XmlLaunchShortcut"
    icon= "icons/sample.gif"
    id= "org.salever.rcp.debug.demo.xmlShortcut"
    label= "Launch XML debug"
    modes= "run, debug" >
  <configurationType
    id= "org.salever.rcp.debug.demo.xmlLaunchConfigurationType" ></configurationType>
  <contextualLaunch>
    <enablement>
      <with
        variable= "selection" >
        <count
          value= "1" >
        </count>
        <iterate>
          <or>
            <test
              property= "org.eclipse.debug.ui.matchesPattern"
              value= "*.xml" >
            </test>
          </or>
        </iterate>
      </with>
    </enablement>
  </contextualLaunch>
</shortcut>
</extension>
```

然后还需要创建类 org.salever.rcp.tech.chapter19.launch.XmlLaunchShortcut, 具体内容为:

```
package org.salever.rcp.tech.chapter19.launch;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.debug.core.DebugPlugin;
import org.eclipse.debug.core.ILaunchConfiguration;
import org.eclipse.debug.core.ILaunchConfigurationType;
import org.eclipse.debug.core.ILaunchConfigurationWorkingCopy;
import org.eclipse.debug.core.ILaunchManager;
import org.eclipse.debug.ui.DebugUITools;
import org.eclipse.debug.ui.ILaunchShortcut;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.IEditorPart;
import org.salever.rcp.tech.chapter19.DebugConstants;

/**
 * Using "Run As" --> "Launch XML debug" will lead here.
 * @author salever
 *
 */
public class XmlLaunchShortcut implements ILaunchShortcut {

    /* (non-Javadoc)
     * @see org.eclipse.debug.ui.ILaunchShortcut#launch(org.eclipse.jface.viewers.ISelection, java.lang.String)
     */
    @Override
    public void launch(ISelection selection, String mode) {
        Object selectObj = ((IStructuredSelection) selection).getFirstElement();
        if (selectObj instanceof IFile) {
            launchFile((IFile) selectObj, mode);
        }
    }
}
```

```
else if(selectObj instanceof IProject){
    MessageDialog.openWarning(null, "Warning", "Not implemeneted yet!");
}

}

/* (non-Javadoc)
 * @see org.eclipse.debug.ui.ILaunchShortcut#launch(org.eclipse.ui.IEditorPart, java.lang.String)
 */
@Override
public void launch(IEditorPart editor, String mode) {
    //empty
}

/**
 * Launch an XML file,using the file information, which means using default launch configurations.
 *
 * @param file
 * @param mode
 */
private void launchFile(IFile file, String mode) {

    // check for an existing launch config for the js file
    String path = file.getFullPath().toString();
    ILaunchManager launchManager = DebugPlugin.getDefault()
        .getLaunchManager();
    ILaunchConfigurationType type = launchManager
        .getLaunchConfigurationType(DebugConstants.LAUNCH_CONFIGURATION_TYPE);

    try {
        ILaunchConfiguration configuration = createLaunchConfiguration(
            type, path, file);
        DebugUITools.launch(configuration, mode);
    } catch (CoreException e1) {
    }
}
```

```
    }

    /**
     * Create a new configuration and set useful data.
     *
     * @param type
     * @param path
     * @param file
     * @return
     * @throws CoreException
     */
    private ILaunchConfiguration createLaunchConfiguration(
        ILaunchConfigurationType type, String path, IFile file)
        throws CoreException {
        // create a new configuration for the js file
        ILaunchConfigurationWorkingCopy workingCopy = type.newInstance(null,
            file.getName());
        workingCopy.setAttribute(DebugConstants.XML_FILE, path);
        workingCopy.setAttribute(DebugConstants._TYPE,
            "");
        workingCopy.setMappedResources(new IResource[] { file });
        return workingCopy.doSave();
    }
}
```

19.4 创建 Run/Debug Configuration 界面

添加扩展点 org.eclipse.debug.ui.launchConfigurationTabGroups:

```
<extension
    point= "org.eclipse.debug.ui.launchConfigurationTabGroups" >
    <launchConfigurationTabGroup
        class= "org.salever.rcp.examples.debug.launch.XmlLaunchConfigurationTabGroup"
        id= "org.salever.rcp.debug.demo.xmllaunchConfigurationTabGroup"
        type= "org.salever.rcp.debug.demo.xmlLaunchConfigurationType" >
    </launchConfigurationTabGroup>
</extension>
```

界面需要用代码定义，类 `org.salever.rcp.tech.chapter19.launch.XmlLaunchConfigurationTabGroup` 负责实现 `Run Configuration` 对话框的 UI:

```
package org.salever.rcp.tech.chapter19.launch;

import org.eclipse.debug.ui.AbstractLaunchConfigurationTabGroup;
import org.eclipse.debug.ui.CommonTab;
import org.eclipse.debug.ui.ILaunchConfigurationDialog;
import org.eclipse.debug.ui.ILaunchConfigurationTab;

/**
 * @author salever
 *
 */
public class XmlLaunchConfigurationTabGroup extends
    AbstractLaunchConfigurationTabGroup {

    /**
     *
     */
    public XmlLaunchConfigurationTabGroup() {

    }

    @Override
    public void createTabs(ILaunchConfigurationDialog dialog, String mode) {
        ILaunchConfigurationTab[] tabs = new ILaunchConfigurationTab[] {
            new XMLMainTab(),
            new CommonTab()
        };
        setTabs(tabs);
    }
}
```

`XMLMainTab.java` 内容为:

```
package org.salever.rcp.tech.chapter19.launch;
```

```
import org.eclipse.core.resources.IFile;
```

```
public class XMLMainTab extends AbstractLaunchConfigurationTab {
```

```
    private Text fProgramText;
```

```
    private Button fProgramButton;
```

```
    private Combo typeCombo;
```

```
    /* (non-Javadoc)
```

```
    * @see org.eclipse.debug.ui.ILaunchConfigurationTab#createControl(org.eclipse.swt.widgets.Composite)
```

```
    */
```

```
    public void createControl(Composite parent) {
```

```
        Font font = parent.getFont();
```

```
        Composite comp = createComposite(parent, font, 1, 1, GridData.FILL_BOTH);
```

```
        createKulGroup(comp);
```

```
        createVerticalSpacer(comp, 1);
```

```
        createEmulatorGroup(comp);
```

```
        setControl(comp);
```

```
    }
```

```
    private void createEmulatorGroup(Composite parent) {
```

```
        Group group = new Group(parent, SWT.NONE);
```

```
        group.setText("&Type");
```

```
        GridData gd = new GridData(GridData.FILL_HORIZONTAL);
```

```
        group.setLayoutData(gd);
```

```
        GridLayout layout = new GridLayout();
```

```
        layout.numColumns = 1;
```

```
        group.setLayout(layout);
```

```
        group.setFont(parent.getFont());
```

```
        typeCombo = new Combo(group, SWT.READ_ONLY);
```

```
        typeCombo.add("6.0.0");
```

```
gd = new GridData(GridData.FILL_HORIZONTAL);
typeCombo.setLayoutData(gd);
typeCombo.setFont(parent.getFont());
}

private void createKulGroup(Composite parent) {
    Group kulGroup = new Group(parent, SWT.NONE);
    kulGroup.setText("&XML File");
    GridData gd = new GridData(GridData.FILL_HORIZONTAL);
    kulGroup.setLayoutData(gd);
    GridLayout layout = new GridLayout();
    layout.numColumns = 2;
    kulGroup.setLayout(layout);
    kulGroup.setFont(parent.getFont());

    fProgramText = new Text(kulGroup, SWT.SINGLE | SWT.BORDER);
    gd = new GridData(GridData.FILL_HORIZONTAL);
    fProgramText.setLayoutData(gd);
    fProgramText.setFont(parent.getFont());
    fProgramText.addModifyListener(new ModifyListener() {
        public void modifyText(ModifyEvent e) {
            updateLaunchConfigurationDialog();
        }
    });

    fProgramButton = createPushButton(kulGroup, "&Browse...", null); //$NON-NLS-1$
    gd = new GridData(GridData.FILL_HORIZONTAL);
    fProgramButton.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent e) {
            browseXMLFiles();
        }
    });
}

/**
```



```

    * Open a resource chooser to select a PDA program
    */

    protected void browseXMLFiles() {
        ResourceListSelectionDialog dialog = new ResourceListSelectionDialog(getShell(),
ResourcesPlugin.getWorkspace().getRoot(), IResource.FILE);
        dialog.setTitle("Xml Program");
        dialog.setMessage("Select XML Program");
        if (dialog.open() == Window.OK) {
            Object[] files = dialog.getResult();
            IFile file = (IFile) files[0];
            fProgramText.setText(file.getFullPath().toString());
        }

    }

    /* (non-Javadoc)
    *
    * @see
    org.eclipse.debug.ui.ILaunchConfigurationTab#setDefaults(org.eclipse.debug.core.ILaunchConfigurationWorkingCopy)
    */
    public void setDefaults(ILaunchConfigurationWorkingCopy configuration) {
    }

    /* (non-Javadoc)
    *
    * @see
    org.eclipse.debug.ui.ILaunchConfigurationTab#initializeFrom(org.eclipse.debug.core.ILaunchConfiguration)
    */
    public void initializeFrom(ILaunchConfiguration configuration) {

        try {
            String program = null;
            program = configuration.getAttribute("file", (String)null);
            String emulatorName = configuration.getAttribute("type", (String)null);
            if (program != null) {
                fProgramText.setText(program);
            }
            if (emulatorName == null) {
                typeCombo.select(0);
            }
        }
    }

```



```
    IPath path = new Path(program);
    IResource res = ResourcesPlugin.getWorkspace().getRoot().findMember(path);
    if (res != null) {
        resources = new IResource[]{res};
    }
}
configuration.setMappedResources(resources);
}

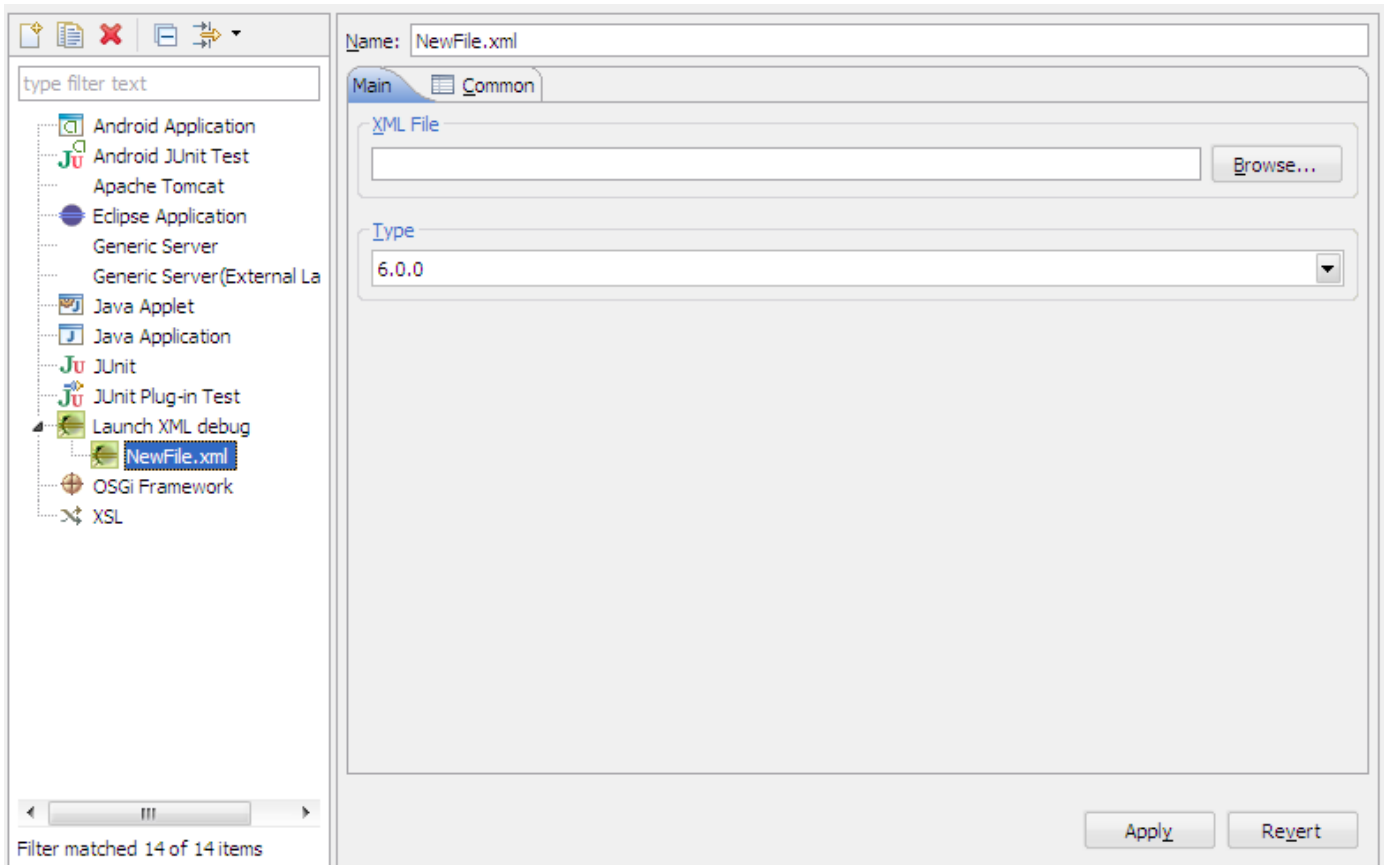
/* (non-Javadoc)
 * @see org.eclipse.debug.ui.ILaunchConfigurationTab#getName()
 */
public String getName() {
    return "Main";
}

/* (non-Javadoc)
 * @see org.eclipse.debug.ui.ILaunchConfigurationTab#isValid(org.eclipse.debug.core.ILaunchConfiguration)
 */
public boolean isValid(ILaunchConfiguration launchConfig) {
    setErrorMessage(null);
    setMessage(null);
    String text = fProgramText.getText();

    if (text.length() > 0) {
        IPath path = new Path(text);
        if (ResourcesPlugin.getWorkspace().getRoot().findMember(path) == null) {
            setErrorMessage("Specified xml file does not exist");
            return false;
        }
    } else {
        setMessage("Specify an xml file");
    }
    return true;
}
```

```
public Composite createComposite(Composite parent, Font font, int columns, int hspan, int fill) {  
    Composite g = new Composite(parent, SWT.NONE);  
    g.setLayout(new GridLayout(columns, false));  
    g.setFont(font);  
    GridData gd = new GridData(fill);  
        gd.horizontalSpan = hspan;  
    g.setLayoutData(gd);  
    return g;  
}  
}
```

界面效果为：



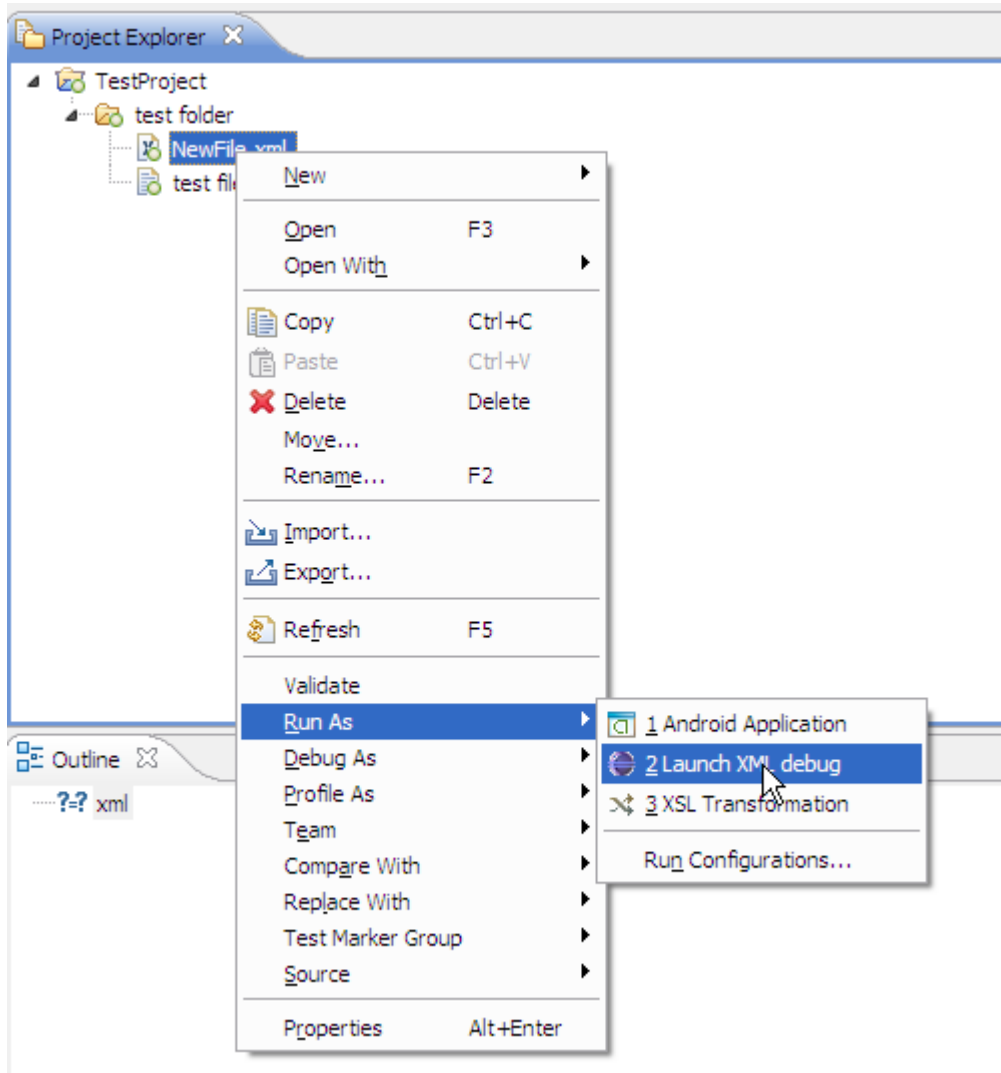
19.5 指定 Run/Debug 显示图片

添加扩展点 `org.eclipse.debug.ui.launchConfigurationTypeImages`：

```
<extension
  point= "org.eclipse.debug.ui.launchConfigurationTypeImages" >
  <launchConfigurationTypeImage
    configTypeID= "org.salever.rcp.debug.demo.xmlLaunchConfigurationType"
    icon= "icons/alt_window_16.gif"
    id= "org.salever.rcp.debug.demo.launchConfigurationTypeImage1" >
  </launchConfigurationTypeImage>
</extension>
```

19.6 说明

附件“org.salever.rcp.tech.chapter19”为一个Eclipse Launcher示例，点击*.XML文件右键Run As会调用 Windows Notepad 程序打开此XML文件。另见：<http://www.ceclipse.org/read-cec-tid-27916.html>



20 专题五 Equinox P2 方式实现 RCP 自动更新

20.1 概述

Eclipse 的软件管理很方便，尤其是在 E3.4 以及以后的版本中使用了 Equinox P2 框架以后，本文将如何使用 Equinox P2 框架实现 RCP 程序的软件安装、更新、管理等进行介绍。

使用旧的 UpdateManagerUI 实现更新 RCP 程序(E3.4 以前)请见：

<http://www.ibm.com/developerworks/cn/opensource/os-ecl-rcpum/>

Equinox P2 方式进行 Eclipse 插件安装介绍请见：

<http://www.ibm.com/developerworks/cn/opensource/os-eclipse-equinox-p2/index.html>

Equinox P2 进行 Update 另见：

http://www.vogella.de/articles/EclipseP2Update/article.html#firstfeature_category

http://wiki.eclipse.org/Equinox/p2/Adding_Self-Update_to_an_RCP_Application

http://www.ralfebert.de/blog/eclipsercp/p2_updates_tutorial/

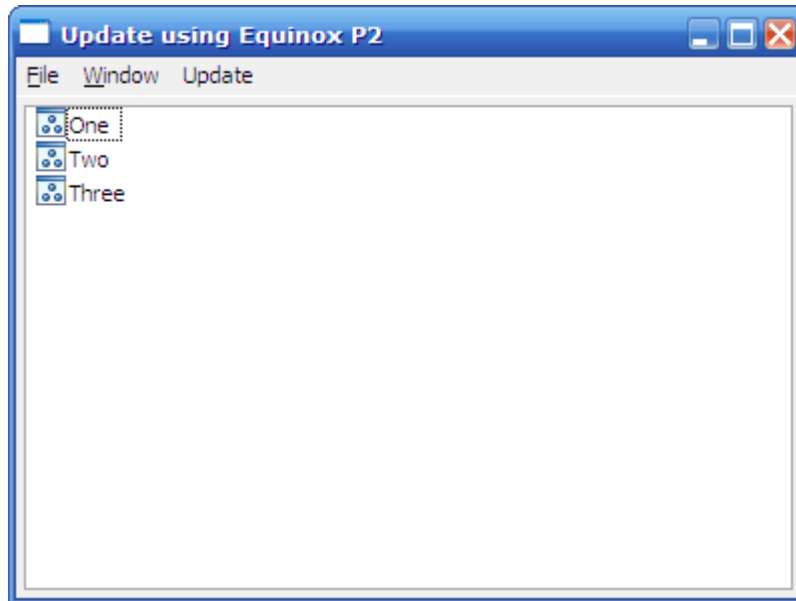
使用 Equinox P2 框架实现 RCP 更新，首先需要获得 P2 相关的插件。Eclipse SDK 版本都会包含这些插件，如果您的版本中没有，那么请到 eclipse.org 上自行下载，或者使用 Eclipse 的 Installer New Software 功能在线更新。主要用到的插件为 org.eclipse.equinox.p2.ui 以及其依赖的其他插件。

20.2 示例

20.2.1 Feature 概念

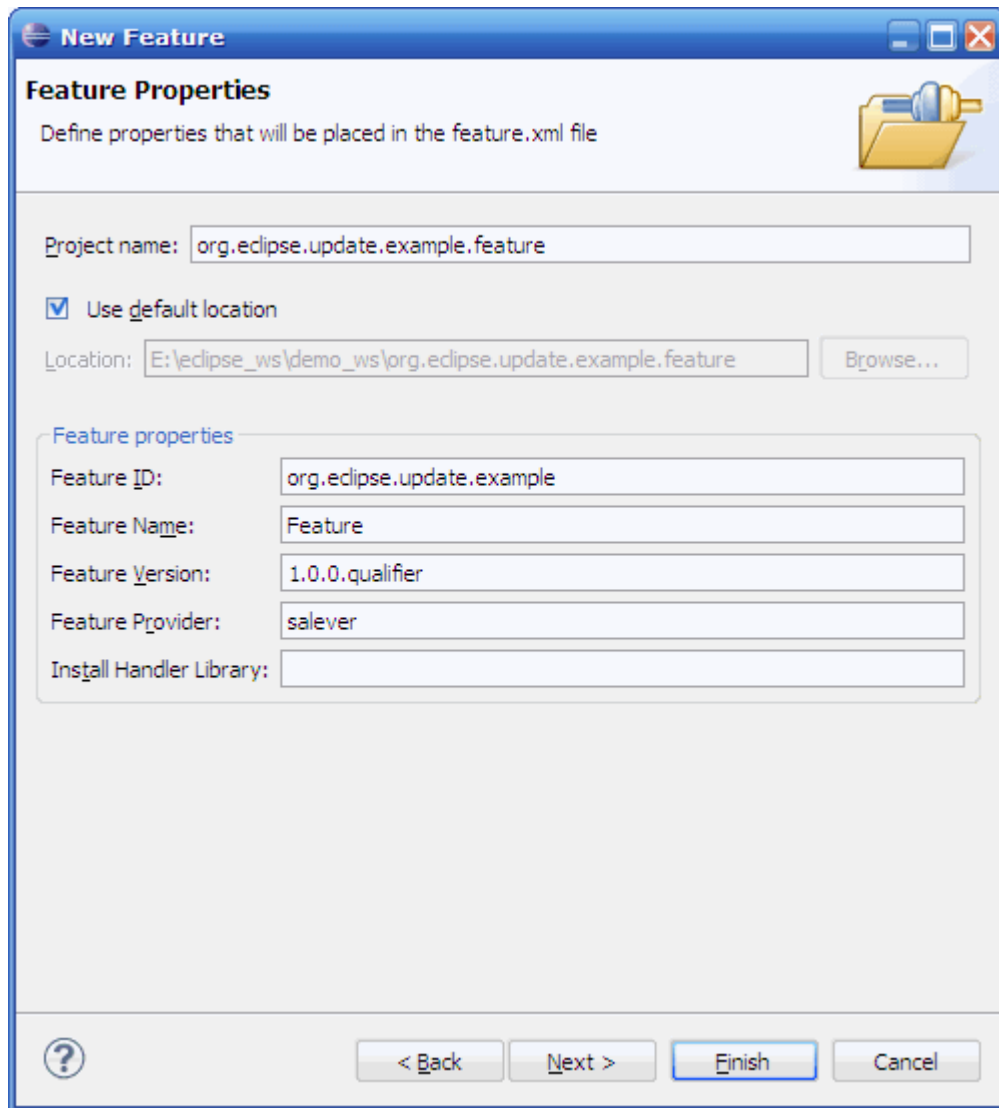
Feature 用来组织插件，更新时需要使用 Feature。RCP 程序都会包含一个基础 Feature——org.eclipse.rcp, 它包含了 RCP 程序需要的基础插件，内容为见 eclipse 目录下 features/org.eclipse.rcp_3.6.0.v20100519-9OArFKvFtsd7WLUKh-DcYTS。

先创建一个用于更新的 RCP 程序，org.eclipse.update.example。具体过程略。运行效果为：

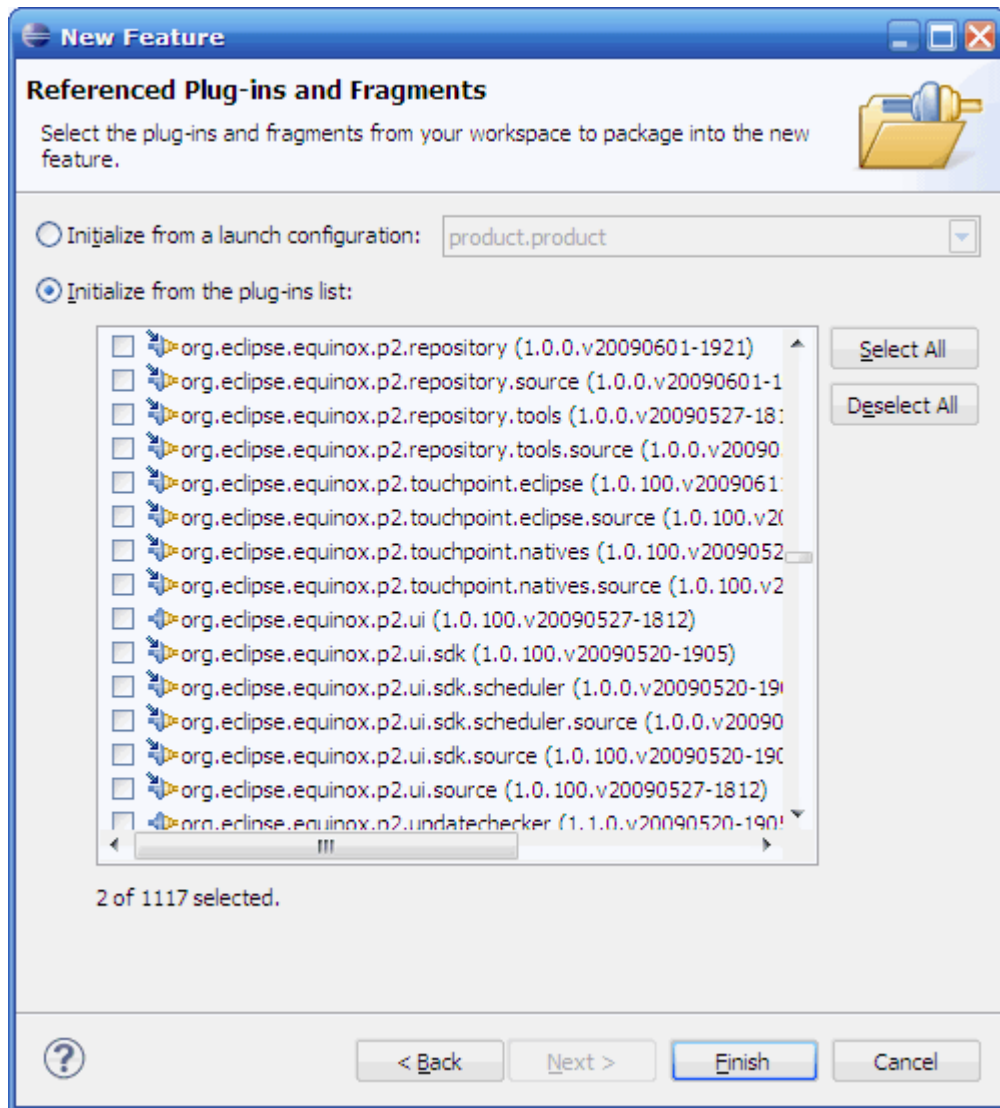


下面来创建一个 Feature Project。

File——New——Project——Plug-in Development——Feature Project, Feature 工程一般命名为 XXXX.feateu, 主要目的是为了与 Plug-in 工程区别开, 但是 Feature 的 ID 却不一定以 feature 结尾。



Feature Initialize Plug-in 设置，这里设置为 `org.eclipse.update.example` 和 `org.eclipse.ui.forms`。



20.2.2 配置 Product

先给 `org.eclipse.update.example` 添加一个 `product` configuration，添加完毕以后，进行 `product` 配置。在 `product` 中有很多依赖插件，没有这些插件，RCP 程序将无法启动。默认情况下 `product` 是基于 `plug-in` 的，在这里需要修改为基于 `feature`。修改位置：

General Information

This section describes general information about the product.

ID:

Version:

Name:

The product includes native launcher artifacts

Product Definition

This section describes the launching product extension identifier and application.

Product: New...

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 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 [Dependencies](#) page.

这时候以 feature 为依赖项的 product 就配置好了。但是运行提示失败，因为还没有给它添加以来的 feature，在 product 的 Dependencies 中添加我们刚刚创建 feature“org.eclipse.update.example”。具体原因和步骤可以参考“Equinox P2 方式进行 Eclipse 插件”。

现在我们让插件依赖于 feature，所以在对应的 feature 中必须定义包含的插件，不然 RCP 会因为找不到依赖项而无法启动。

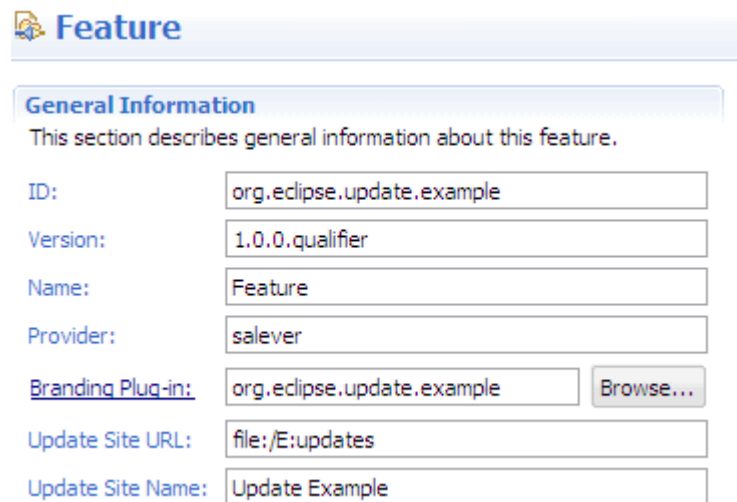
20.2.3 配置 Feature

feature 的配置包括以下几个方面：

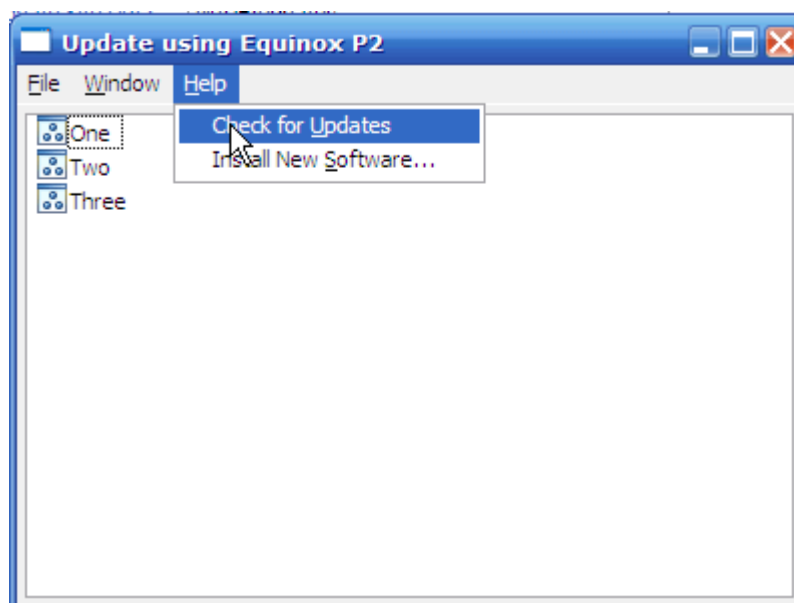
- 包含插件：这些插件就是上面 product 的依赖项
- 包含 feature：如果使用了 eclipse 定义的基本 feature，那么使用它们会减少工作量。主要是向 feature 中添加 plug-in 的工作量
- Update site：更新源 URL，可以是一个网址，也可以使本地文件，这里使用本地文件测试。首先在 feature.xml 的 Included Feature 中添加 org.eclispe.rcp, org.eclipse.equinox.p2.user.ui,

org.eclipse.help，这样它们所包含的 插件就无需再添加到包含插件中了。接着在 feature.xml 的 Plug-in 中添加 org.eclipse.update.example。更新源 URL 的设置比较简单，在 feature.xml 的 Overview 中，进行设置。注意 URL 的格式，这里使用本地文件 E:/updates。

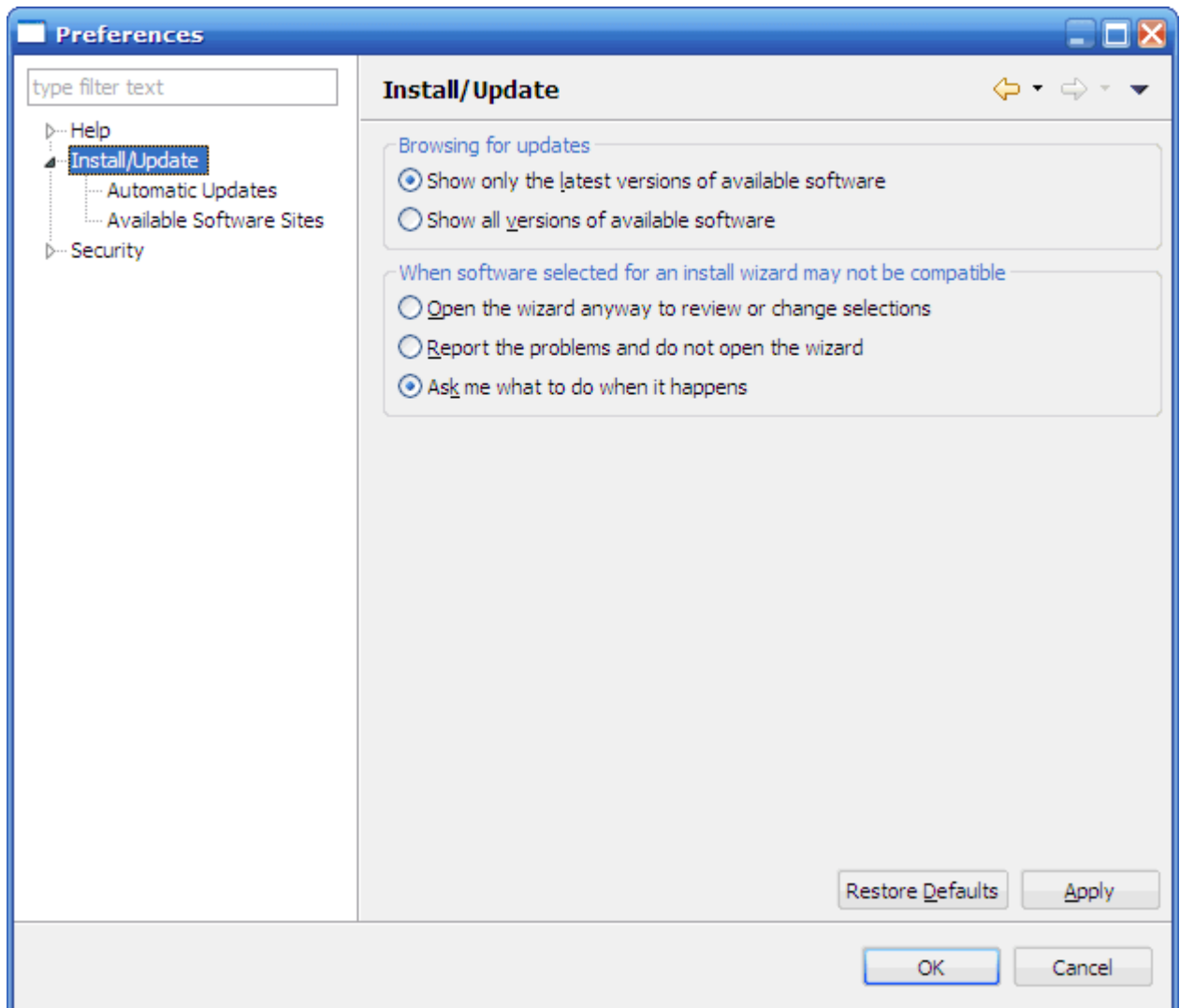
4, Dependencies: Feature 自己也有依赖项。这里添加 org.eclipse.ui, org.eclipse.core.runtime, org.eclipse.equinox.p2.ui。



如果配置正确，这时候再启动 org.eclipse.update.example（使用 product），就会发现多出一些菜单项和首选项了。



更新相关首选项：



20.2.4 product 导出

注意这时候 Export Product 时一定要记得选中 Generate Metadata Repository 选项，否则 Update 功能不能使用。大家可以发现选中与不选中时，产生的 config.ini 文件是不一样的。

注意 Eclipse3.6 下使用 Equinox p2 方式更新 Feature 时有 2 种情况，一种是针对新安装的插件的更新，另一种是针对打包时就有的插件的更新，两种方式下更新源文件的制作方式不同。

- 新安装的插件的更新：给插件对应的 Feature 建立 category.xml，然后使用这个 XML 导出 metadata repository，就可以作为更新源，也可以使用上面的 Update Site Project 方法。
- 打包时就有的 feature 则麻烦一些，必须使用每次打包时生成的 repository 文件夹下的内容作为更新源，单独导出 feature 是无效的。

20.2.5 配置 Equinox P2

P2 提供了自定义 Update UI 的功能，你可以通过扩展 `org.eclipse.equinox.internal.provisional.p2.ui.policy.Policy` 来实现 UI 的定制。

比如 UpdatePolicy:

```
package org.eclipse.update.example.policy;

import org.eclipse.equinox.internal.provisional.p2.ui.policy.IUViewQueryContext;
import org.eclipse.equinox.internal.provisional.p2.ui.policy.Policy;

/**
 * @author salever
 *
 */
public class UpdatePolicy extends Policy{

    public UpdatePolicy() {
        // Disable the ability to manipulate repositories.
        setRepositoryManipulator(null);
        // View everything in the repository.
        IUViewQueryContext context =
            new IUViewQueryContext(IUViewQueryContext.AVAILABLE_VIEW_FLAT);
        context.setVisibleAvailableIUProperty(null);
        setQueryContext(context);
    }
}
```

然后在 Activator 中使用这个 policy:

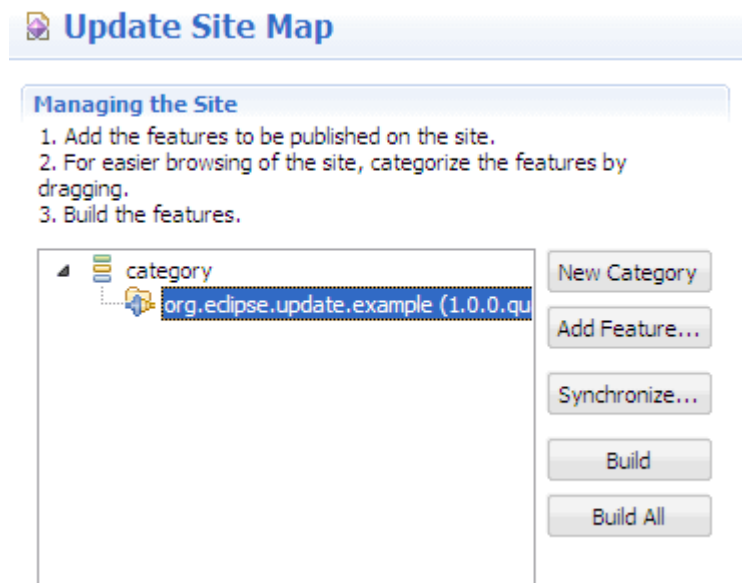
```
public void start(BundleContext context) throws Exception {
    plugin = this;
    registration = context.registerService(Policy.class.getName(),
        new UpdatePolicy(), null);
}

/*
```

```
* (non-Javadoc)
* @see org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.framework.BundleContext)
*/
public void stop(BundleContext context) throws Exception {
    context.ungetService(registration.getReference());
    registration = null;
}
```

20.2.6 配置 Update Site

New——Plug-in Development——Update Site Project，然后添加 Category 和要更新的 Feature。



点击上面的 **Build** 按钮，在工程目录下会生成更新源文件，将这些文件复制到 update site 目录中。

另见：<http://salever.javaeye.com/blog/712772>

21 专题六 Common Navigator Framework 初探

见<http://www.ceclipse.org/read-cec-tid-27476.html>