

通过例子学习 Eclipse RCP 开发

翻译：孙江湖

2009 年 10 月

ECLIPSE RCP

This article describes how to develop applications using the Eclipse Rich Client Platform (RCP) architecture.

本文讲述如何利用 **Eclipse RCP** 架构来进行应用开发

The article covers among others the creating of RCP applications, using commands, views, editors, dialogs and creating and deploying a Eclipse product.

本文涵盖了创建 **RCP** 应用的各个方面：使用命令、视图、编辑器、对话框及创建与部署 **Eclipse** 产品。

This article is based on Eclipse 3.5 (Eclipse Galileo).

本文基于 **Eclipse 3.5** 讲述（**Eclipse Galileo**）

简介

此文通过若干个例子介绍了使用 **Eclipse** 进行 **RCP** 应用开发的过程。

学习环境：

Eclipse 的不同版本中环境差异较大。本教程是利用 **Eclipse SDK 3.5.0** 编写的。为了保持一致，请使用如下 **Eclipse** 的安装包（小版本也很重要）：

Eclipse SDK: eclipse-sdk-3.5-win32.zip

Eclipse RCP: eclipse-rcp-galileo-win32.zip

代码说明：

本教程中的代码都经过译者的验证，但不保证百分之百正确。屏幕截图是实际运行时的效果。

注：

原文下载地址：<http://www.vogella.de/articles/Eclipse/article.html>。在翻译过程中译者对原文中的有些部分进行了修改、纠正。

欢迎传播，敬请指正。

目 录

Eclipse RCP	2
1. Eclipse RCP Eclipse 富客户端	6
1.1. Overview 概览	6
1.2. Eclipse RCP Architecture - Plugins, Extensions and Extension-Points Eclipse RCP 架构-插件、扩展与扩展点	6
1.3. Main components of an Eclipse RCP application Eclipse RCP 应用的主要组件...	7
1.4. Application versus Product 应用与产品	8
1.5. Important files 重要文件	8
2. Installation 安装	10
3. Create your first RCP application 创建你的第一个 RCP 应用	12
3.1. Create a RCP application 创建 RCP 应用	12
3.2. Start your RCP application 启动你的 RCP 应用	15
4. Startup process of an RCP application RCP 应用的启动过程	17
5. Run configuration 运行配置	18
5.1. Overview 概览	18
5.2. Check your runtime configuration 检查运行时配置	19
5.3. Important Parameters 重要参数	20
6. Commands 命令	22
6.1. Overview 概览	22
6.2. Defining commands 定义命令	22
6.3. Using commands in menus 在菜单中使用命令	26
7. System Tray 系统托盘	31
8. Views 视图	36
8.1. Overview 概览	36

8.2. Create a view 创建视图	36
8.3. Add the view to your perspective. 将视图添加到透视图	39
8.4. Result 结果.....	41
8.5. Add view to perspective via code 利用代码将视图添加到透视图	41
9. Working with Editors, View Interaction and Model Updates 使用编辑器、视图进行交互及模型更新.....	43
9.1. Overview 概览	43
9.2. Create project 创建项目.....	43
9.3. Create and prepare the domain model 创建并准备域模型	44
9.4. Content and Label provider 内容与标签提供者	48
9.5. Use the domain model in the view 在视图使用域模型	51
9.6. Editor area 编辑区	53
9.7. Editor Input 编辑器输入.....	54
9.8. Adding the editor 添加编辑器.....	56
9.9. Creating a command for calling the editor 创建编辑器调用命令	59
10. Dialog 对话框.....	64
10.1. Overview 概览.....	64
10.2. Using standard dialogs 使用标准对话框.....	64
10.3. Selection 选择.....	67
10.4. User defined dialogs 使用自定义对话框.....	69
11. Field Assist 字段助手.....	74
12. Wizards 向导.....	76
12.1. Overview 概览.....	76
12.2. Example 例子	76
13. Adding a status line 添加状态行	84

13.1. Setup Status line 设置状态行	84
13.2. Shared Status Line 共享状态行	85
14. Perspectives 透视图	90
14.1. Adding a perspective to your application 为应用添加透视图	90
14.2. Select the perspective 选择透视图	92
14.2.1. Select perspective via the toolbar / coolbar 通过工具条/快捷键选择透视图	92
14.2.2. Select perspective via the menu 通过菜单选择透视图	94
15. Products and Branding 产品与商标	95
15.1. Product Configuration 产品配置	95
15.2. Create a project 创建项目	95
15.3. Create your product configuration 创建你的产品配置	95
15.4. Maintain the overview tab 概览页维护	96
15.5. Dependencies 依赖性	99
15.6. Launch your product 启动你的应用	100
15.7. Splash Screen 溅射屏幕	101
15.8. Branding your product 制作产品商标	102
15.9. Customizing the start icon and launcher arguments 定制开始图标与启动参数	103
15.10. Done 完成了!	104
16. Deploy your product 部署你的产品	105
17. Tips and Tricks 提示与技巧	108
17.1. Save users layout 保存用户布局	108
17.2. Plugin ID in application 应用中的插件 ID	108
17.3. Finding unused dependencies 查找无用的依赖	109
18. Next steps 下一步	110

1. ECLIPSE RCP **ECLIPSE 富客户端**

1.1. OVERVIEW **概览**

Eclipse RCP allows developers to use the Eclipse architecture to design flexible and extensible stand-alone applications re-using a lot of already existing functionality and coding patterns inherent in Eclipse.

Eclipse RCP 允许开发者利用 **Eclipse** 架构来设计一个灵活的、可扩展的独立应用，此应用可以重用现有功能及继承自 **Eclipse** 的代码

Eclipse is build upon a plugin architecture. This allows Eclipse to get easily extended. See [Eclipse plugin Tutorial](#) for a guide on how-to extend the Eclipse IDE. Eclipse RCP provides the same modular concept for stand-alone applications.

Eclipse 是建立在插件结构上的。这使得其非常容易扩展。有关 **Eclipse** 扩展的内容请参看 **Eclipse** 插件教程中有关如何扩展 **Eclipse IDE** 的部分。对于独立的 **RCP** 应用，**Eclipse RCP** 也提供了同样的模块化概念。

To create an Eclipse RCP application you would create a plugin which contains the application. This application can use the existing hooks of Eclipse (called Extension Points) to define functionality and / or the user interface and / or provide its own extension points to get extended by other plugins.

要创建一个 **Eclipse RCP** 应用，你需要创建一个包含应用的插件。此应用可以利用已有的 **Eclipse** 接口（称之为扩展点）定义功能、接口，或者为其它插件提供其本身的扩展点。

1.2. *ECLIPSE RCP ARCHITECTURE - PLUGINS, EXTENSIONS AND EXTENSION-POINTS* **ECLIPSE RCP 架构-插件、扩展与扩展点**

The most important architectural characteristics of Eclipse is the Plugin architecture. The Eclipse IDE is build as a number of plugins which are dependent on each other.

Eclipse 最重要的结构特点就是其插件化结构。**Eclipse** 本身就是由一组互相独立的插件构建而成的。

Plugins are the smallest deployable and installable software components of Eclipse.

插件就是最基本的可部署、安装的 **Eclipse** 软件单元。

Each plugin can define extension-points which define possibilities for functionality contributions (code and non-code) by other plugins. Non-code functionality contributions are for example the provision of help content.

每个插件都可以定义自己的扩展点，它为其它插件定义了插件的功能贡献（代码或者无代码）。帮助内容就是一个无代码功能贡献的例子。

A plugin can use extensions, e.g. provide functionality to these extension points. In general an extension point can be used several times (either by the same plugin or by other plugins). See [Eclipse Extension Points and Extensions - Tutorial](#) for details.

插件可以利用扩展，即为扩展点提供功能。通常，扩展点可以多次使用（甚至是由同一个插件或者由其它插件）。详细内容请参看 [Eclipse 扩展点与扩展教程](#)。

The basis for this architecture is the runtime environment Equinox of Eclipse which is the reference implementation of OSGI. See [OSGi development - Tutorial](#) for details. The Plugin concept of Eclipse is the same as the bundle concept of OSGI. Generally speaking a OSGI bundle equals a Plugin and vice-versa.

这种结构的基础就是 [Eclipse](#) 的运行时环境设计，这是参考 [OSGi](#) 实现的。详细内容请参看 [OSGi 开发教程](#)。[Eclipse](#) 中插件的概念等同于 [OSGi](#) 中的包概念。也就是说 [Eclipse](#) 中的插件就是 [OSGi](#) 中的包。

The used extensions and the provided extension-points are described in the file plugin.xml. This file is a XML file which can be edited via the PDE (Plugin Development Environment) which provides a nice user interface for editing this file.

所使用的扩展及能提供的扩展点都在 [plugin.xml](#) 文件中进行描述。此文件是一个 XML 文件，可以在 [PDE](#)（插件开发环境）进行编辑。[PDE](#) 为编辑此文件提供了一个相当好的用户界面。

Eclipse RCP provides and uses the same framework as the Eclipse Workbench hence allowing the programmer to divide the application functionality into several plugins, to use existing extension points and to provide additional extension points. This concept of Eclipse allows every programmer to structure his RCP application into several independent components and to easily declare extensions to existing extensions points.

[Eclipse RCP](#) 提供和使用与 [Eclipse Workbench](#) 相同的框架，允许程序员将应用从功能上划分为若干个插件，并利用已有的扩展点及提供新的扩展点。[Eclipse](#) 的这种概念允许每个程序员将其 [RCP](#) 应用构建为若干个独立的组件并且很方便的为已有的扩展点声明扩展。

1.3. MAIN COMPONENTS OF AN ECLIPSE RCP APPLICATION [ECLIPSE RCP 应用的主要组件](#)

An Eclipse RCP application requires:

- Main program - A RCP main application class implements the interface `IApplication`. Eclipse expects that the application class is defined via the extension point `org.eclipse.core.runtime.application`.
- A Perspective - The perspective is extended from `org.eclipse.ui.perspective`

Workbench Advisor- invisible technical component which controls the appearance of the application (menus, toolbars, perspectives, etc)

一个 [Eclipse](#) 应用需要:

- 主程序 - [RCP](#) 主应用类实现 `Iapplication` 接口，[Eclipse](#) 要求是通过 `org.eclipse.core.runtime.application` 扩展点定义这个类。
- 透视图 - 此透视图扩展自 `org.eclipse.ui.perspective`.

工作台向导 - 非可视的技术组件，控制着应用的外观（菜单、工具条、透视图等等）。

All plugins must provide a so-called manifest named "plugin.xml".

所有的插件都必须提供一份名为“**plugin.xml**”的说明文档。

The minimal required plugins to create and run an Eclipse RCP application are the two plugins "org.eclipse.core.runtime" and "org.eclipse.ui"

创建、运行 **Eclipse RCP** 应用至少需要两个插件，即“**org.eclipse.core.runtime**”与“**org.eclipse.ui**”。

1.4. APPLICATION VERSUS PRODUCT 应用与产品

To run an Eclipse RCP program you have to define an application . The application can be seen as the main() method of a standard Java program. If this application shuts down the complete program is terminated.

要运行 **Eclipse RCP** 程序，你必须定义一个应用。此应用可以看成是标准的 **Java** 程序的 **main()** 方法，一旦应用停止整个程序也就停止了。

In Eclipse terms a product is everything that goes with your application, e.g. icons, splash screen, external jars, other plugins, etc.

用 **Eclipse** 的术语来说，产品就是与应用有关的所有东西，例如图标、溅射屏幕、外部 **jar** 包、其它插件等等。

1.5. IMPORTANT FILES 重要文件

The plugin configuration is contained in the two files:

- MANIFEST.MF - the OSGi bundle manifest is stored in MANIFEST.MF
- plugin.xml - additional Eclipse specific (non standard OSGi) configuration, e.g. the extension points

插件的配置包含于如下两个文件中：

- **MANIFEST.MF** – OSGi 包的说明保存在 **MANIFEST.MF** 文件中；
- **Plugin.xml** – 附加的 **Eclipse** 配置文件（非标准 OSGi 要求），例如扩展点。

The PDE provides an editor for editing the "MANIFEST.MF" and "plugin.xml".

PDE 提供了用来编辑 **MANIFEST.MF** 与 **plugin.xml** 文件的编辑器。

In addition to these files you have the .project file. The .project contains the description of your project. It contains a XML tag "natures" where the nature of the project is described. A plugin project has the nature "org.eclipse.pde.PluginNature" and a java project has the nature "org.eclipse.jdt.core.javanature". These tags will also steer some behavior of the development

environment, e.g. a project with PluginNature will update the java class path if you change the dependency information in a plugin project.

除了这些文件，还有.project 文件。 .project 文件包含了项目的有关描述。文件中有 “natures” 标签，描述了项目的特性。插件项目的特性为 “org.eclipse.pde.PluginNature”，而 Java 项目的特性为 “org.eclipse.jdt.core.javanature”，这些标签也会包含一些有关开发环境的信息，例如，如果你在插件项目中修改了依赖性信息，则 PluginNature 也会更新。

Tip

The .* file may be hidden by a filter. In the package explorer select the drop-down menu and then Filter to remove the filter for .* resources.

提示：缺省情况下，.*文件将被过滤而不显示。在包资源管理器中选择下拉菜单，再选择 Filter 可以去掉.* 资源前面的复选勾就可以显示 .*资源文件（如.project 文件）。

2. INSTALLATION 安装

在安装 Eclipse SDK 与 Eclipse RCP for RCP/Plug-in Developers 之前，请先下载安装 JDK 1.6.以上版本。

On the webpage of eclipse (www.eclipse.org), click on DOWNLOADS. Download the package "Eclipse for RCP/Plug-in Developers".

从 Eclipse 的 Web 站点主页(www.eclipse.org)单击 DOWNLOADS，下载 Eclipse for RCP/Plug-in Developers 安装包。

Tip

You might have to click on "More Packages" to see the RCP download package.

提示：你可能需要单击“More Packages”才能看到 RCP 下载页面。

Tip

Don't extract Eclipse into a directory tree that contains special characters. I also recommend to avoid spaces in the directory name.

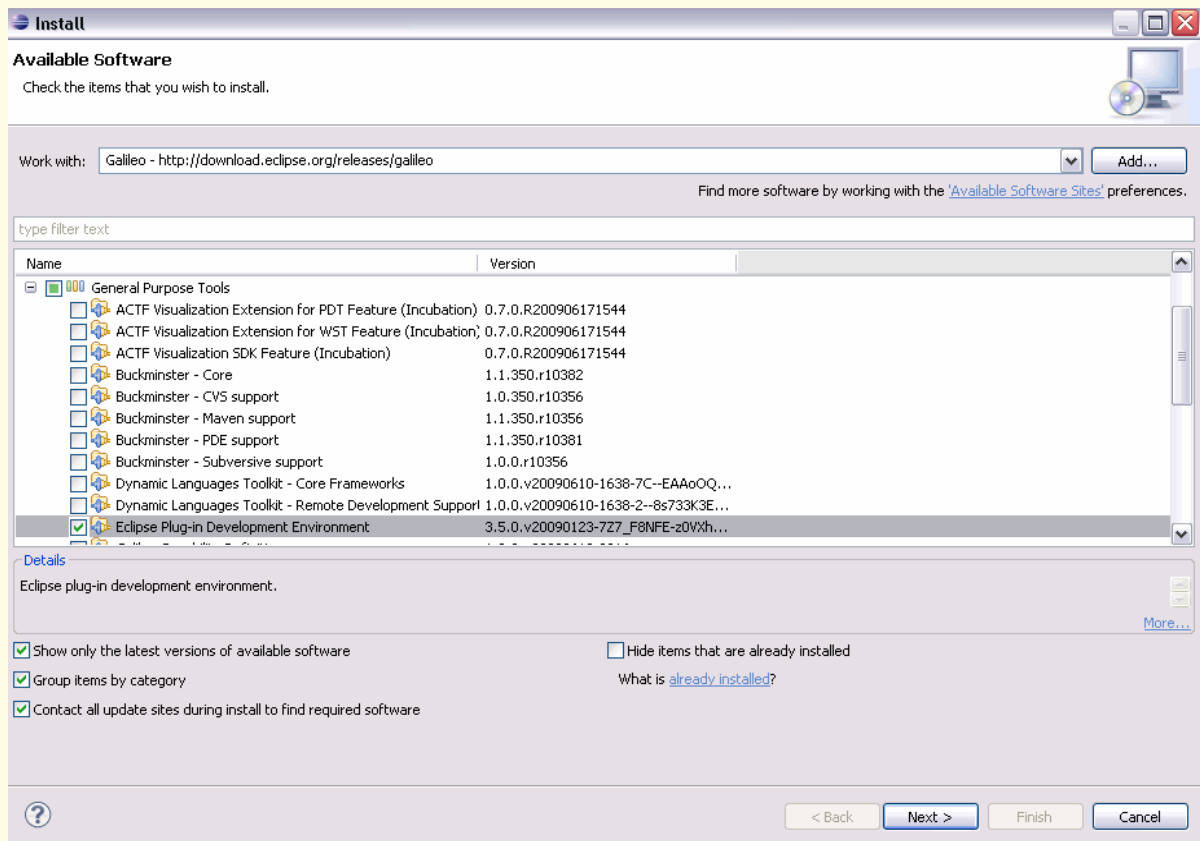
提示：不要将 Eclipse 安装包解压到包含特殊字符的目录树下。我建议目录名中不要包含空格。

In case you have downloaded the Eclipse Java IDE (or any other non RCP flavor) distribution you can use the Eclipse Update Manager to install the plugins required for RCP development. See [Eclipse Update Manager](#) for details on using the update manager.

当你下载了 Eclipse Java IDE（或者其它非 RCP 包）后，你可以利用 Eclipse 更新管理器来安装 RCP 开发所需的插件。关于更新管理器的详尽内容请参看 Eclipse 更新管理器。

Install "General Purpose Tools" -> "Eclipse Plug-in Development Environment" and "Eclipse RCP Plug-in Developer Resources" from the Galileo update site.

从 Galileo 更新站点安装“通用工具”→“Eclipse 插件开发环境”与“Eclipse 插件开发资源”。



RCP	
Name	Version
Eclipse RCP	3.5.0.v20090519-9SA0FwxFv6x089WEf-TWh11
Eclipse RCP Plug-in Developer Resources	3.5.0.v20090519-9SA0FwxFv6x089WEf-TWh11
Eclipse RCP SDK	3.5.0.I20090611-1540

3. CREATE YOUR FIRST RCP APPLICATION 创建你的第一个 RCP 应用

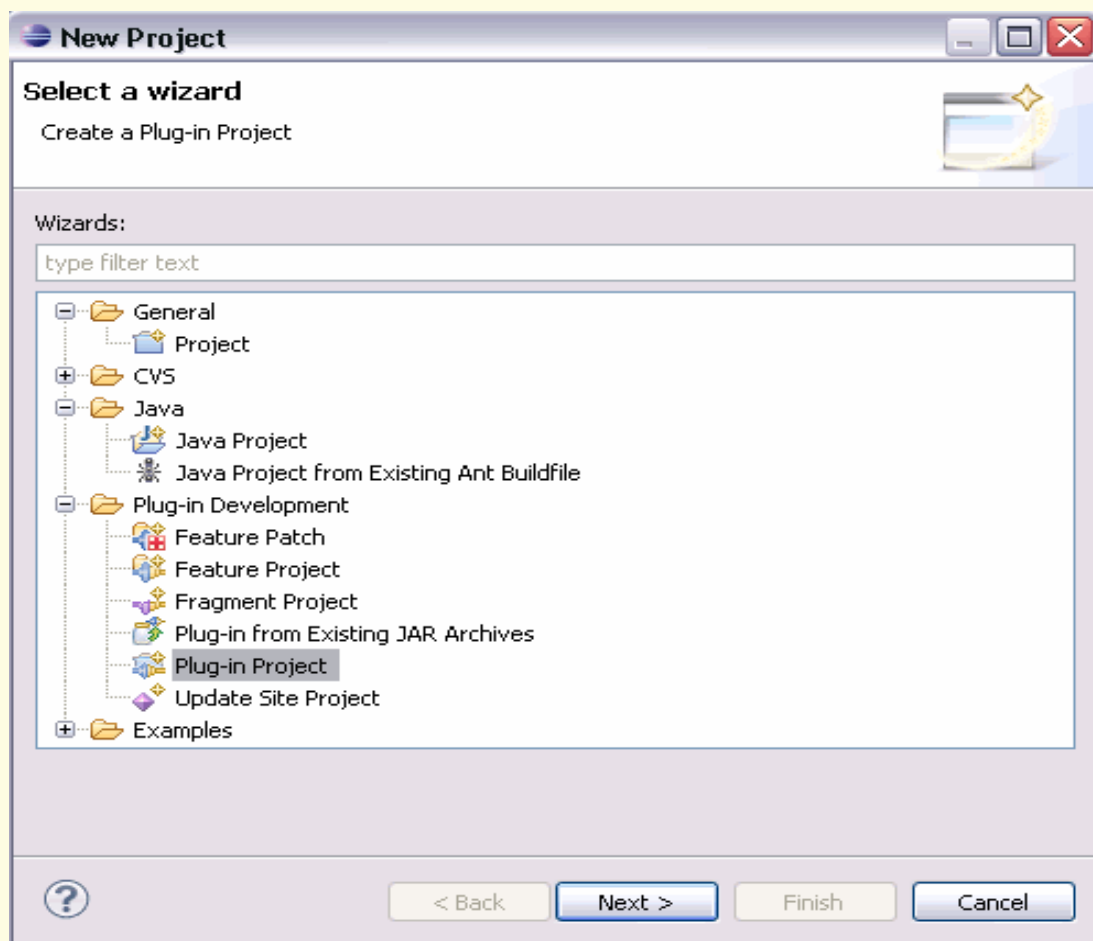
The following gives a quick guide on how to create a simple RCP application.

下面来教你如何快速创建一个简单的 RCP 应用。

3.1. CREATE A RCP APPLICATION 创建 RCP 应用

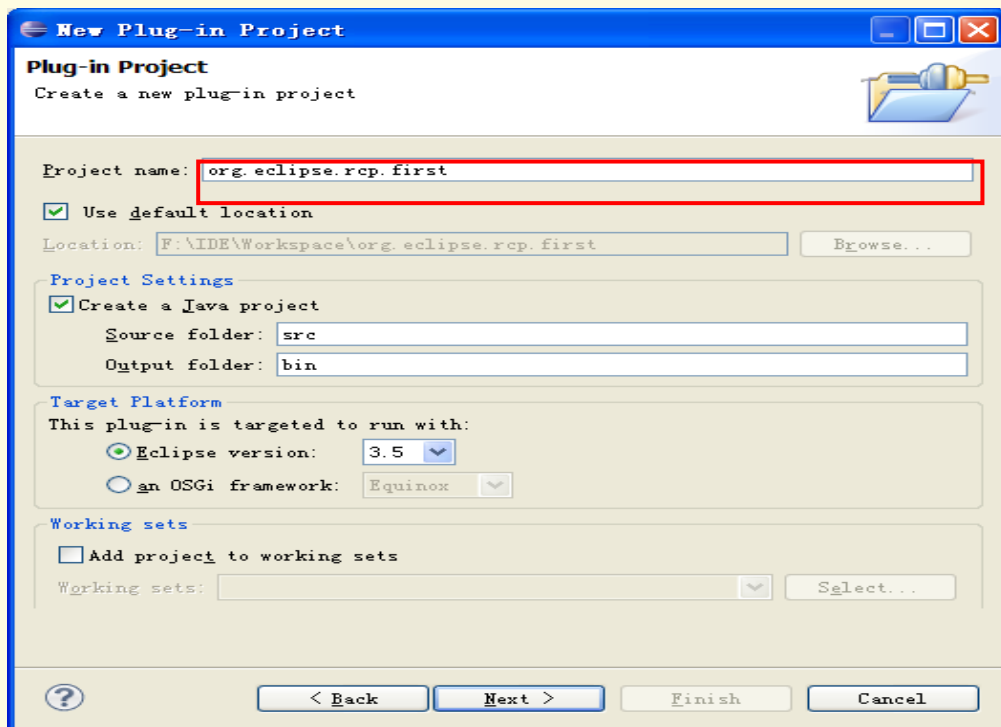
In Eclipse select File-> New Project. From the list select "Plug-In Project".

从 Eclipse 菜单中选择 File→New Project，从左侧窗口列表中选择“Plug-in Project”



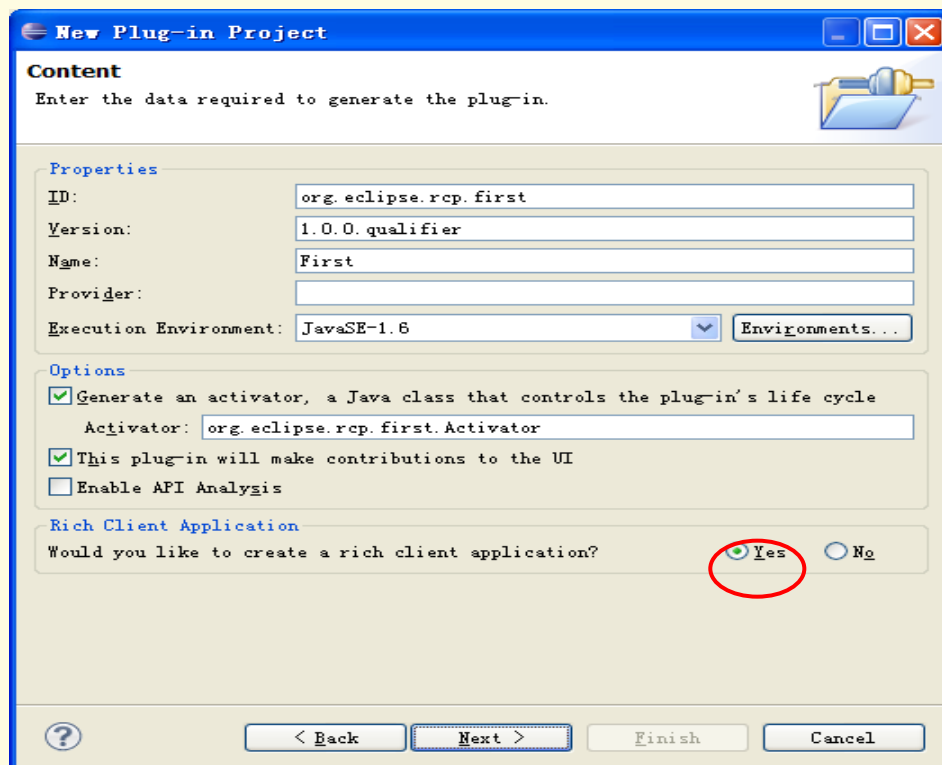
Give your plugin the name " org.eclipse.rcp.first" .

为要开发的插件取名为“org.eclipse.rcp.first”



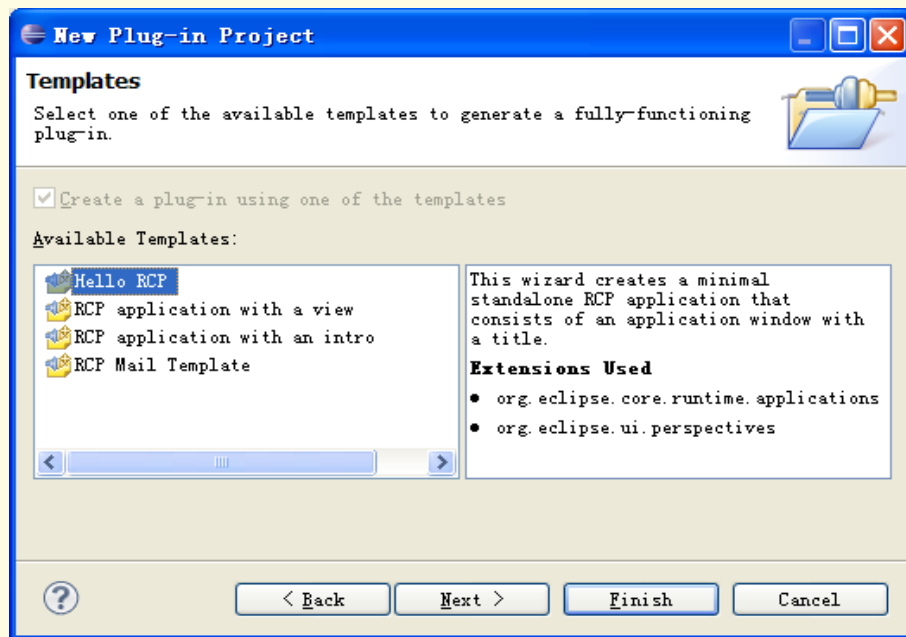
Press "Next" and make the following settings. As we are going to develop a RCP application, select "Yes" at the question "Would you like to create a rich client application".

单击“Next”，进行如下设置。因为我们是要开发一个 RCP 应用，在“Would you like to create a rich client application?”（你要创建一个富客户端应用吗？）后面选择 Yes



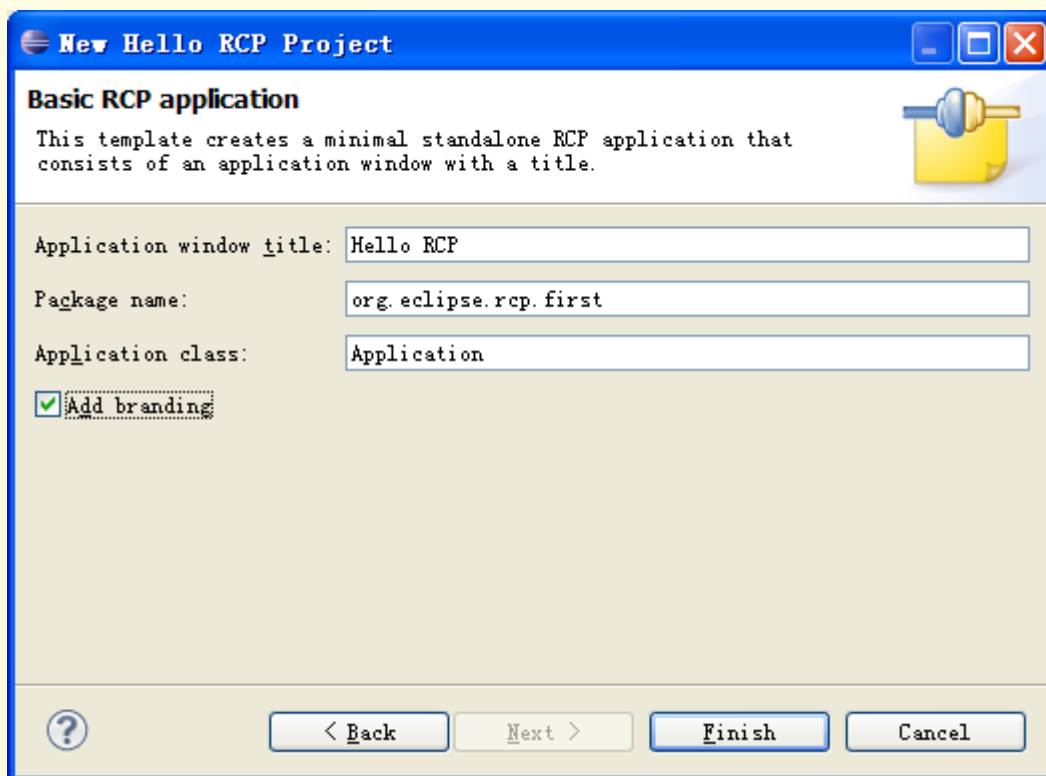
Press next and select the template "Hello RCP".

单击“Next”，选择“Hello RCP”模板，



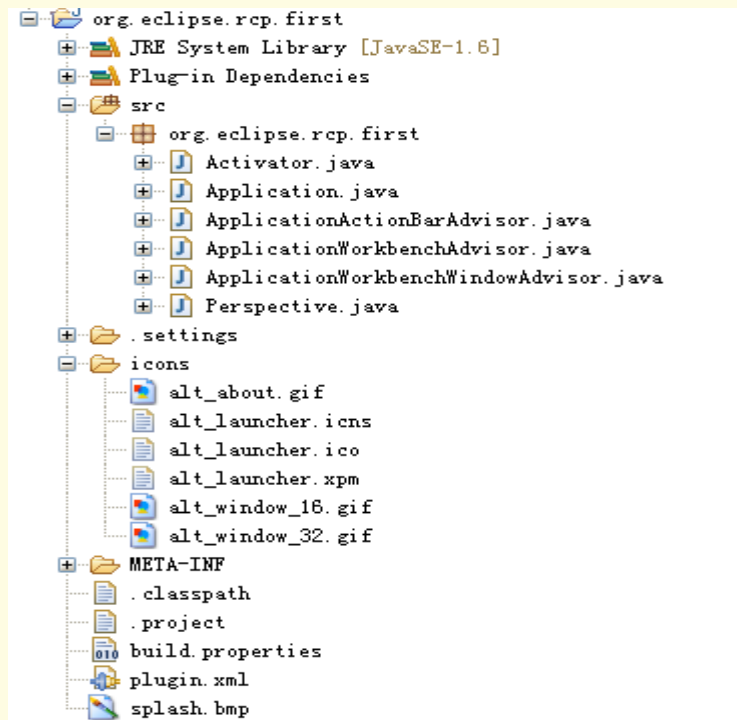
Press next and select "Add branding" and press Finish.

单击“Next”，复选“Add branding”，再单击“Finish”：



As a result a project with the following project structure will be created. Have a look at the different files especially the Java files to get a first feeling about the project structure.

现在一个如下图所示的项目结构已经创建完成。请观察各个文件并体验项目结构：



Tip

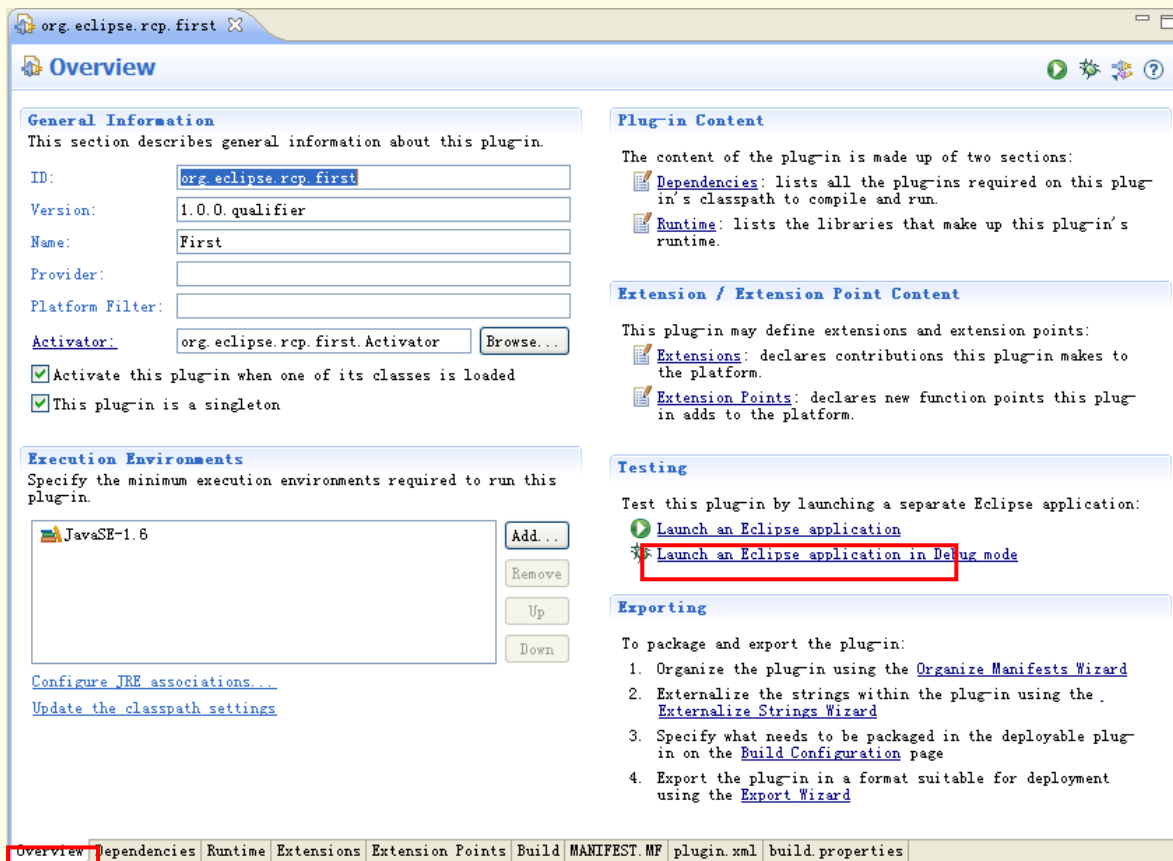
As this tutorial is about RCP development I will use the following interchangeable: "create a new plugin project " or "create a new RCP project". Both mean the same, create a plugin project with the flag "Would you like to create a rich client application" enabled.

提示: 在这个关于 RCP 开发的教程中, 我将交替使用“创建新插件项目”或者“创建一个新 RCP 项目”。两者的含义是一样的。创建一个插件项目就是说“你希望创建一个富客户端应用”。

3.2. START YOUR RCP APPLICATION 启动你的 RCP 应用

Search in the menu path for the file "MANIFEST.MF" and double-click on it. You should see an editor and the tab "Overview" should be selected. Click the link "Launch an Eclipse Application".

从左侧的资源管理器中查找文件“MANIFEST.MF”并双击, 你将看到一个编辑器, 而 Overview (概览) 页已经选择。单击概览页中的“Launch an Eclipse Application”链接。



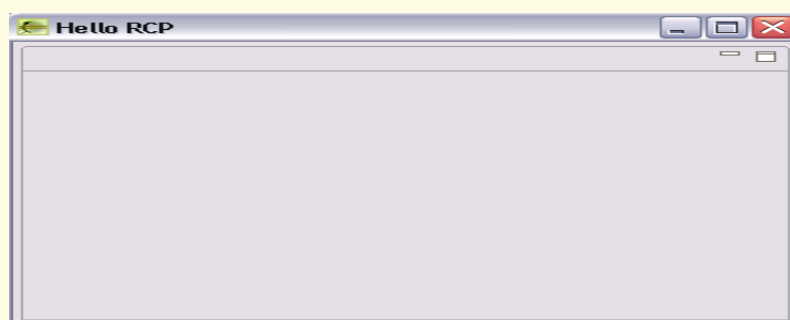
Tip

Alternatively you can run your Eclipse RCP application by selecting the automatically created "plugin.xml", right mouse click and select "Run as" -> "Eclipse Application".

提示：你也可以通过用鼠标右键单击系统自动创建的“plugin.xml”文件，从弹出菜单中选择“Run as→Eclipse Application”来启动 Eclipse RCP 应用。

The result should look like the following:

运行结果如下：



Congratulations, you have created your first RCP application.

恭喜！你已经创建了你的第一个 RCP 应用！

4. STARTUP PROCESS OF AN RCP APPLICATION RCP 应用的启动过程

During the startup of an Eclipse RCP application the Eclipse runtime will evaluate which class is defined via the "org.eclipse.core.runtime.application" extension point.

在 Eclipse RCP 应用启动期间，Eclipse 运行时将寻找通过扩展点 “org.eclipse.core.runtime.application” 所定义的类。

This class will then be loaded. This class creates and runs a Workbench. The Workbench is configured via a WorkbenchAdvisor. The Workbench will start a WorkbenchWindow which is configured via a WorkbenchWindowAdvisor. This WorkbenchWindow will create the toolbar of the application which can get configured at startup via the ActionBarAdvisor.

然后加载这个类。此类会创建并运行工作台。工作台通过 **WorkbenchAdvisor** 进行配置，工作台将启动一个 **WorkbenchWindow**，此窗口通过 **WorkbenchWindowAdvisor** 配置。

WorkbenchWindow 将创建工具条，此工具条可以通过 **ActionBarAdvisor** 在启动时配置。

Each adviser allow to configure certain behavior of the application, e.g. the WorkbenchAdvisor allows to perform certain actions at startup or shutdown by overriding the methods `preStartup()` and `preShutdown()`.

每个 **Advisor**（顾问）都负责应用的一个配置方面，例如，**WorkbenchAdvisor** 可以通过重载 `preStartup()` 方法与 `preShutdown()` 方法而允许在启动或关闭前执行某些动作。

5. RUN CONFIGURATION 运行配置

5.1. OVERVIEW 概览

A run configuration in Eclipse defines the environment under which your application will be started, e.g. compiler flag, plugin (classpath) dependencies etc. Sometime a run configuration is called referred to as "launch configuration".

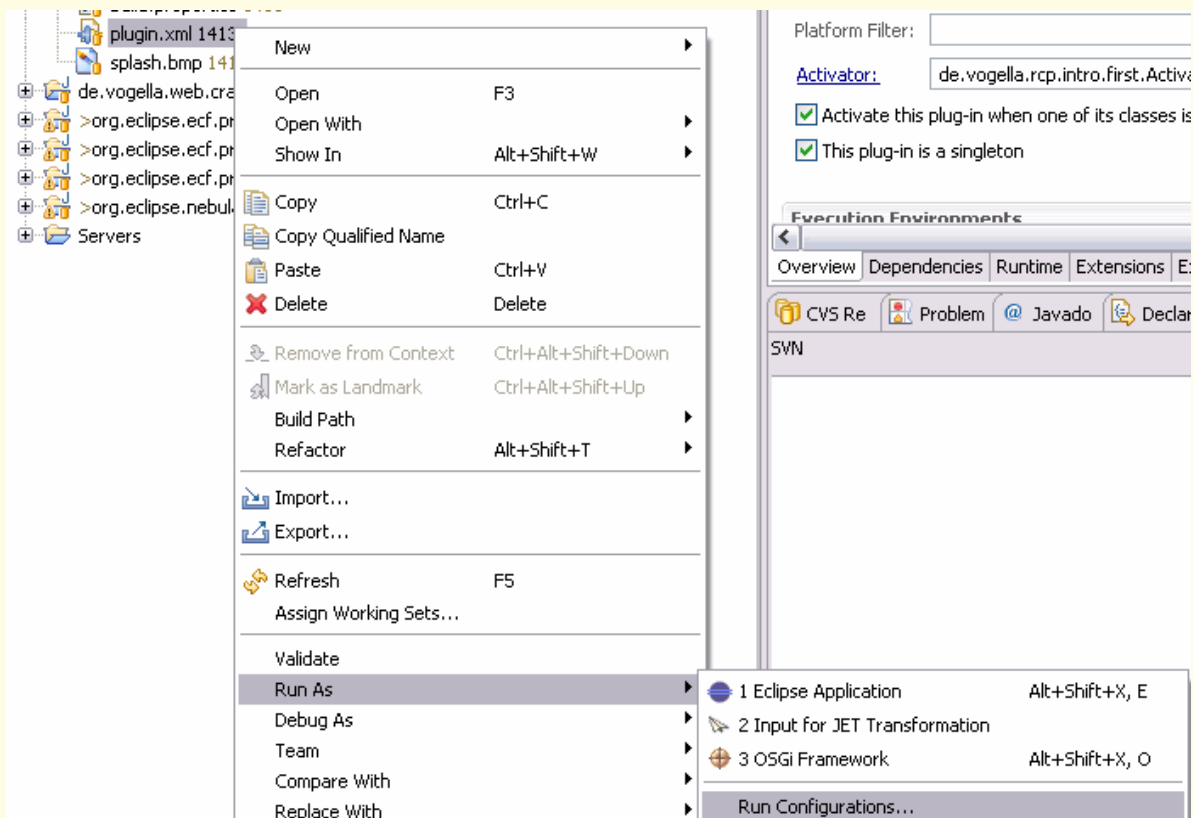
在 Eclipse 中的运行配置定义了用户应用启动时的环境配置，例如编译器标识、插件的依赖性（类路径）等等。有时也将运行配置看成是“启动配置”。

If you start your application a default run configuration will be automatically created for you.

在你运行你的应用时，系统会为你自动创建一个缺省的运行配置。

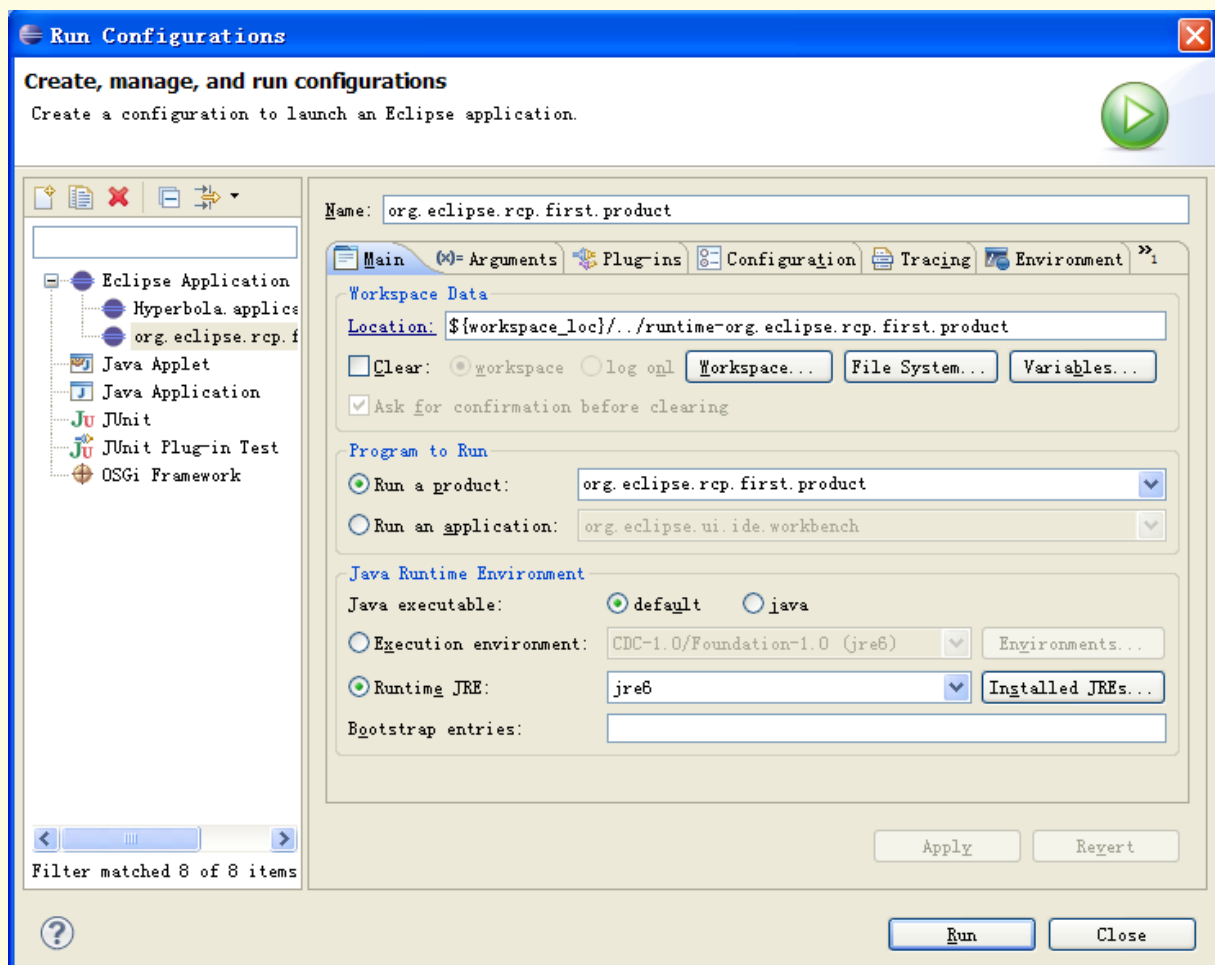
To see and edit your run configuration select your plugin.xml -> Run As -> Run Configurations

要查看或修改你的运行配置，选择 plugin.xml → Run As → Run Configuration



In the field "location" you specify their the files will be created which are necessary to run your RCP application.

在“Location”一栏指定为运行 RCP 应用所必需的文件。



5.2. CHECK YOUR RUNTIME CONFIGURATION 检查运行时配置

On the Plug-ins Tab select "Validate plug-ins automatically prior to launching". This will check if you have all required plugins in your launch configuration.

在 Plug-ins 标签页，复选“Validate plug-ins automatically prior to launch”（在启动前自动检查插件）项，系统将检查在启动配置中的全部所需插件。

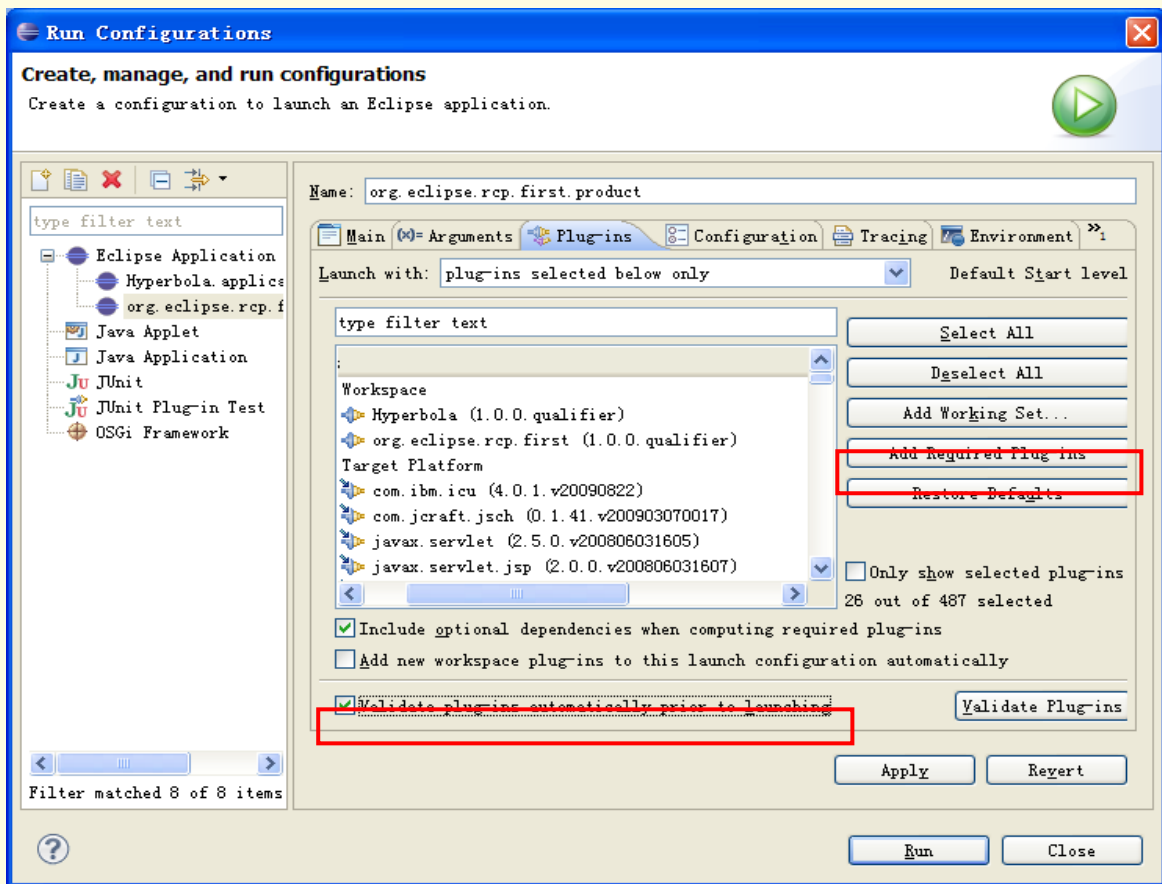
Tip

In my opinion this option should be flagged per default. I submitted a patch for this. Please share you opinion in [Bug: Validate should be flagged as default](#)

提示：就我看来此选项应该是个缺省选项。我建议有个补丁来修改此选项。

If this check reports that some plugins are missing, try clicking the "Add Required-Plug-Ins" button.

如果此检查报告了某些插件缺失，可以尝试点击“Add Required-Plug-ins”按钮（添加所需插件）。



Tip

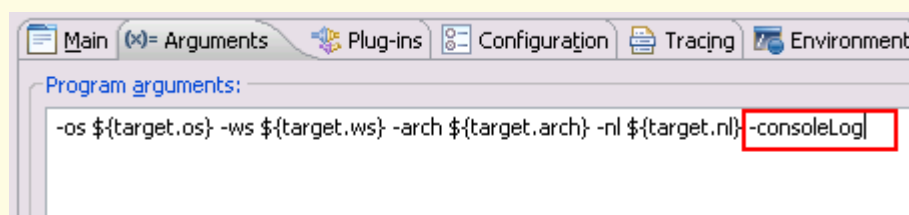
This may solve errors like "One or more bundles are not resolved because the following root constraints are not resolved" or "java.lang.RuntimeException: No application id has been found."

提示：这样可以解决一些类似于 “One or more bundles are not resolved because the following root constraints are not resolved” 或者 “java.lang.RuntimeException: No application id has been found.” 之类的错误。

5.3. IMPORTANT PARAMETERS 重要参数

On the tab Arguments you should add the parameter `-consoleLog`. This will send error message of your Eclipse RCP application to your Eclipse development IDE. Otherwise you will not necessary know that your Eclipse RCP application has a problem.

你需要添加的一个参数为 “`-consoleLog`”。此参数会将用户 Eclipse RCP 应用的错误消息发送到 Eclipse IDE 开发环境。除非你不需要了解你的 ECP 应用出现了什么问题，否则请使用此参数。



Tip

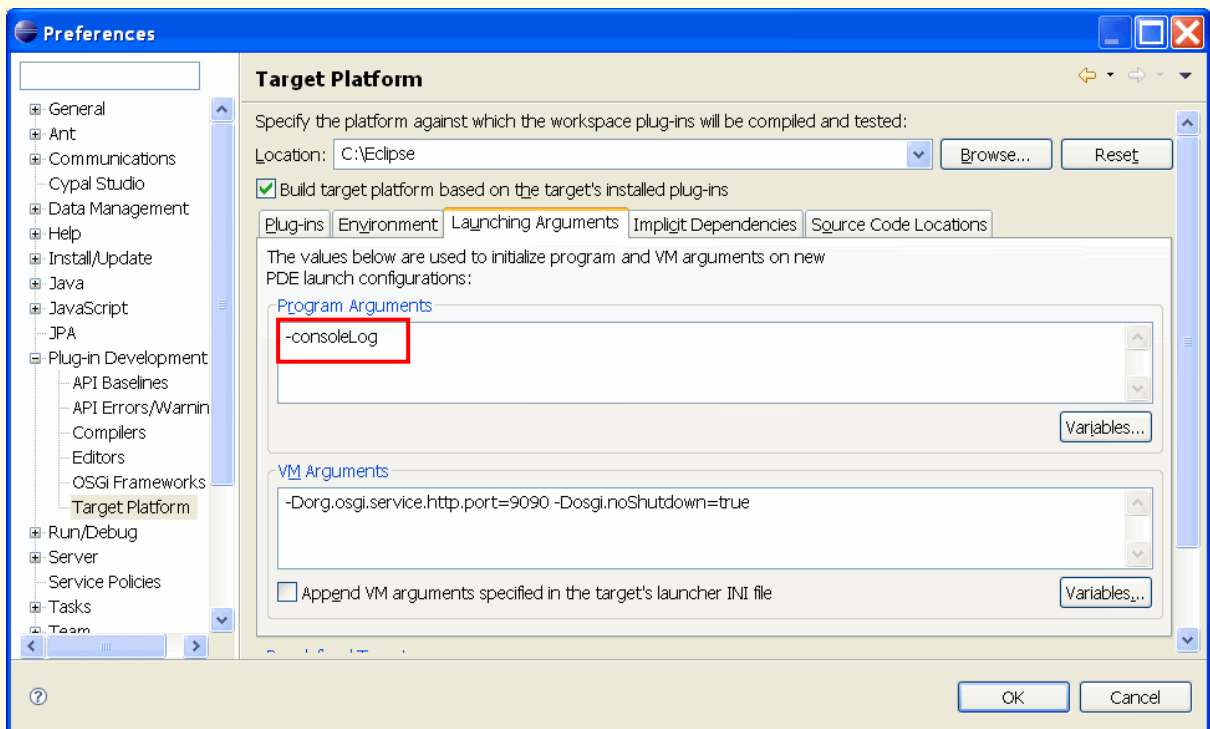
In my opinion this option should be flagged per default. I submitted a patch for this. Please share you opinion in [Bug: -consoleLog should be default](#)

提示：就我看来此选项应该是个缺省选项。我建议有个补丁来修改此选项。

Tip

Under Windows -> Preference -> Plug-in Development -> Target Platform it is possible to maintain -consoleLog in the tab launching Arguments -> Program Arguments. Then the parameter will always be set for your RCP runtime configurations.

提示：从菜单中选择 Windows → Preference → Plug-in Development → Target Platform，再从 Arguments → Program Arguments 也可以维护 -consoleLog 参数。这样设置后此参数即会作为 RCP 运行时的配置参数而一直有效。



Tip

Other nice parameters are -console (which will give you a OSGI console where you can check the status of your application) and -noExit (which will keep the OSGI console open even if the application ends / crashes).

提示：其它的一些有益参数包括：-console（将给出一个 OSGI 控制台，你可以在此控制台检查你的应用状态），-noExit（即使应用结束或者崩溃，OSGI 控制台也将保持打开）。

6. COMMANDS 命令

6.1. OVERVIEW 概览

A command is a declarative description of a component and is independent from the implementation details. A command can be categorized and a hot key (key binding) can be assigned to the command. Commands can be used in menus, toolbars and / or context menus.

命令是组件的说明性描述，它独立于其实现细节。可以对命令进行分类并为命令分配一个热键。命令可以用于菜单、工具条或者弹出菜单中。

The following will focus on a simple definition and usage of commands. For details on the usage of commands please see [Eclipse Commands - Tutorial](#)

下面我们主要讨论命令的简单定义及用法。有关命令的详细内容请参看 [Eclipse 命令教程](#)。

6.2. DEFINING COMMANDS 定义命令

Tip

Commands are superior to Actions. In my opinion Actions should be marked both is confusing for new Eclipse platform developers. Please share your opinion in [Bug for deprecating Actions](#)

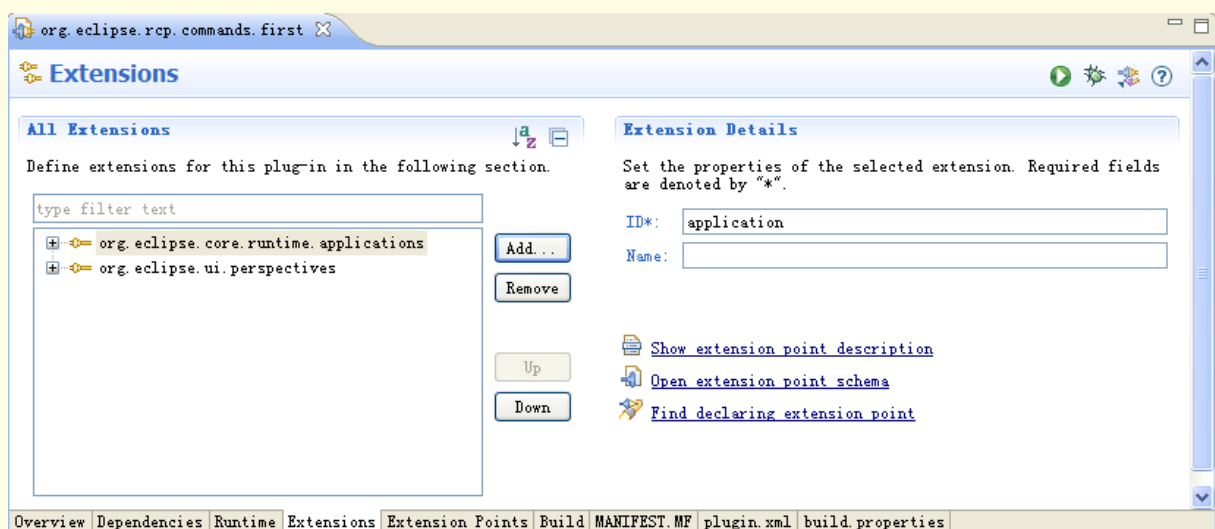
提示：命令是一个动作（操作）的前奏。就我个人的观点来看，动作这个称呼对于新的 Eclipse 平台开发者而言其含义不清。

Our first example will be a command which will exit the application. Create a new RCP project "intro.commands" and use the "Hello RCP" Template.

我们的下一个例子是创建一个用来退出应用的命令。利用“Hello RCP”模板创建一个名为“intro.commands”的 RCP 项目。

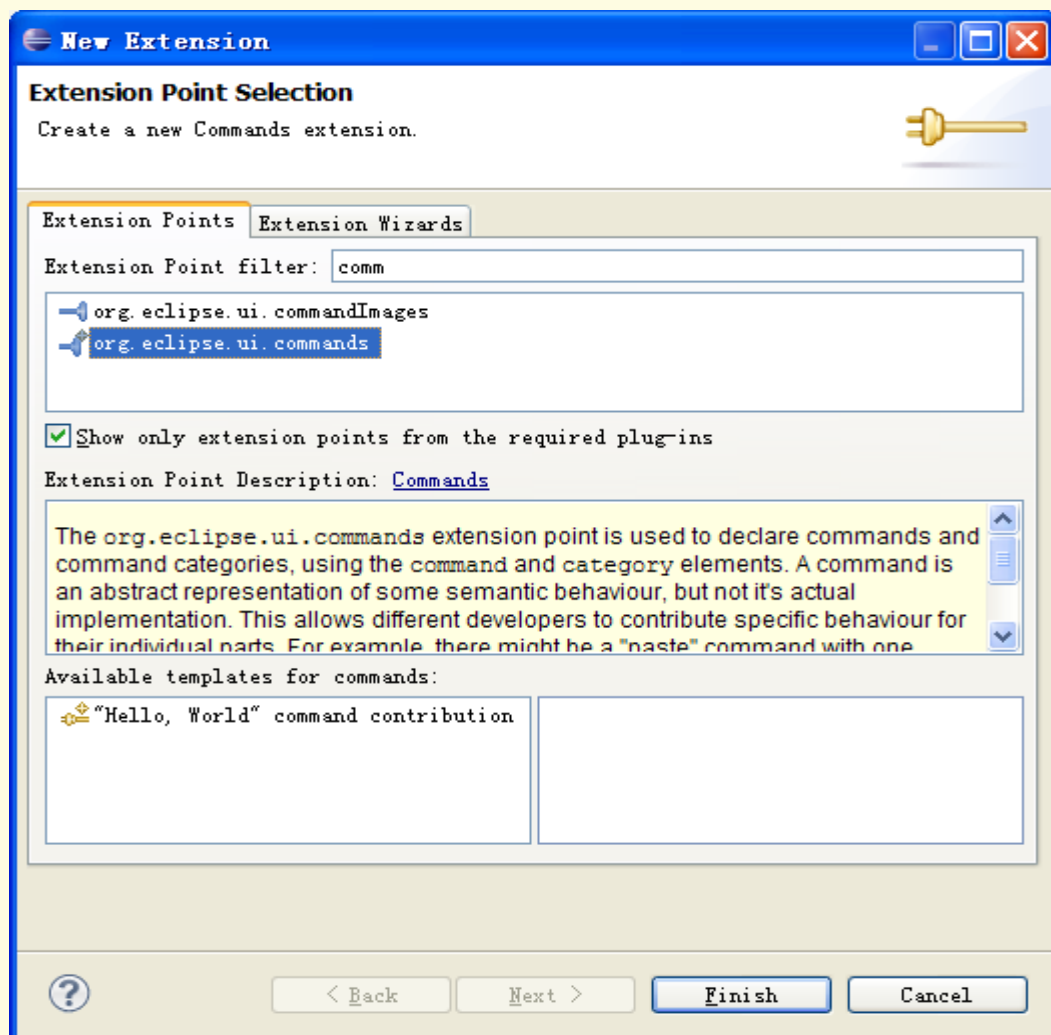
Click on the plugin.xml and select the Extensions tab.

双击 plugin.xml，并选择 Extensions 标签页：



Press the add button and search for the extension `org.eclipse.ui.commands`. Select it and press finish.

单击 **Add** 按钮并查找 `org.eclipse.ui.commands` 扩展，选择并单击 **Finish** 按钮。



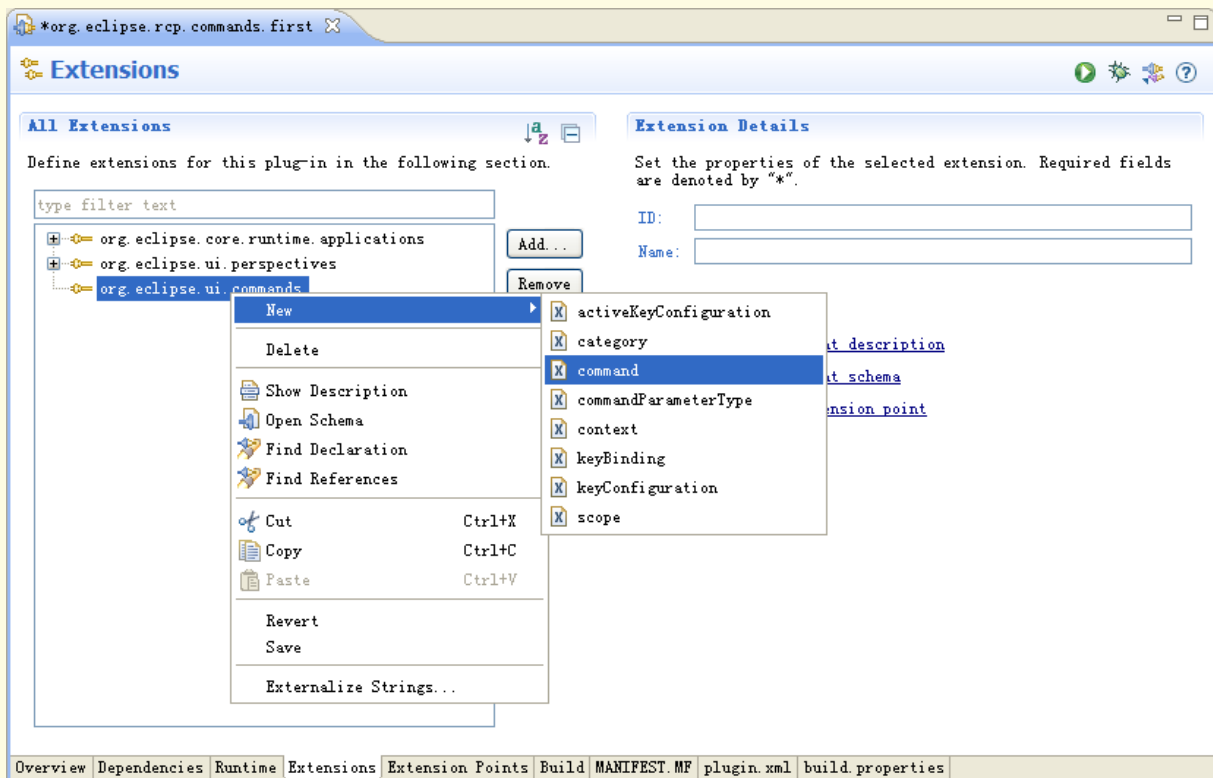
Create a new command by right-clicking New -> command.

在 `org.eclipse.ui.commands` 扩展上单击右键，从弹出菜单中选择 **New**→**Command** 创建一个新命令。

Tip

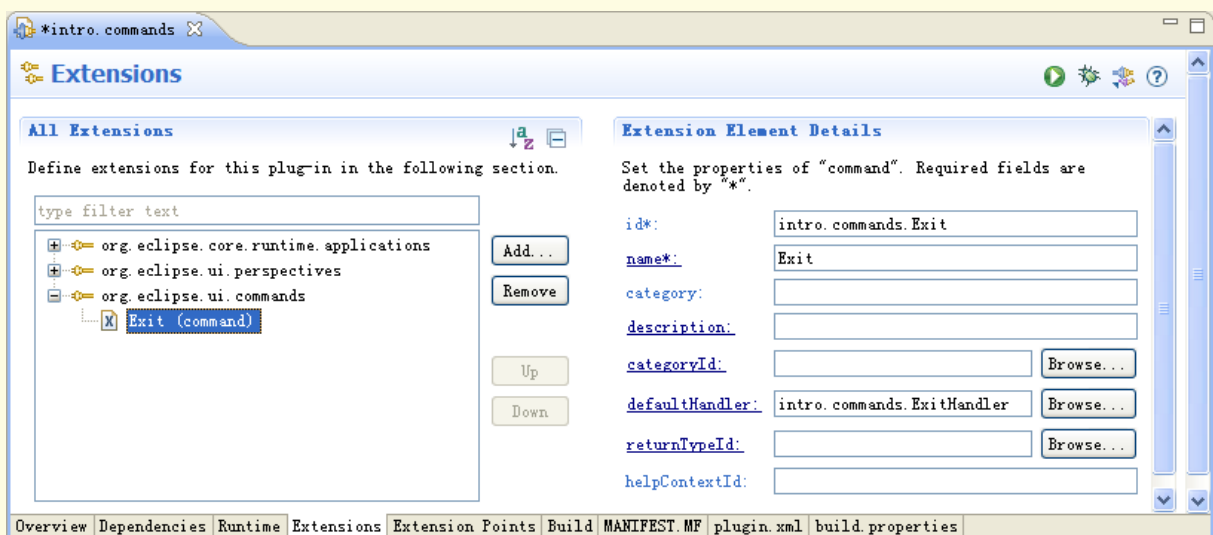
Lots of people report that if they try this they can only select a "Generic" entry. The common source of this problem seems to be that you did not download the package "Eclipse for RCP/Plug-in Developers". Please see [Eclipse Installation](#).

提示：许多人报告说如果他们按照上面所说方法进行时只有一个“Generic”选项。通常出现此问题的原因可能是你没有下载“Eclipse for RCP/Plug-in Developers”包。请参考 [Eclipse 安装一节](#)。



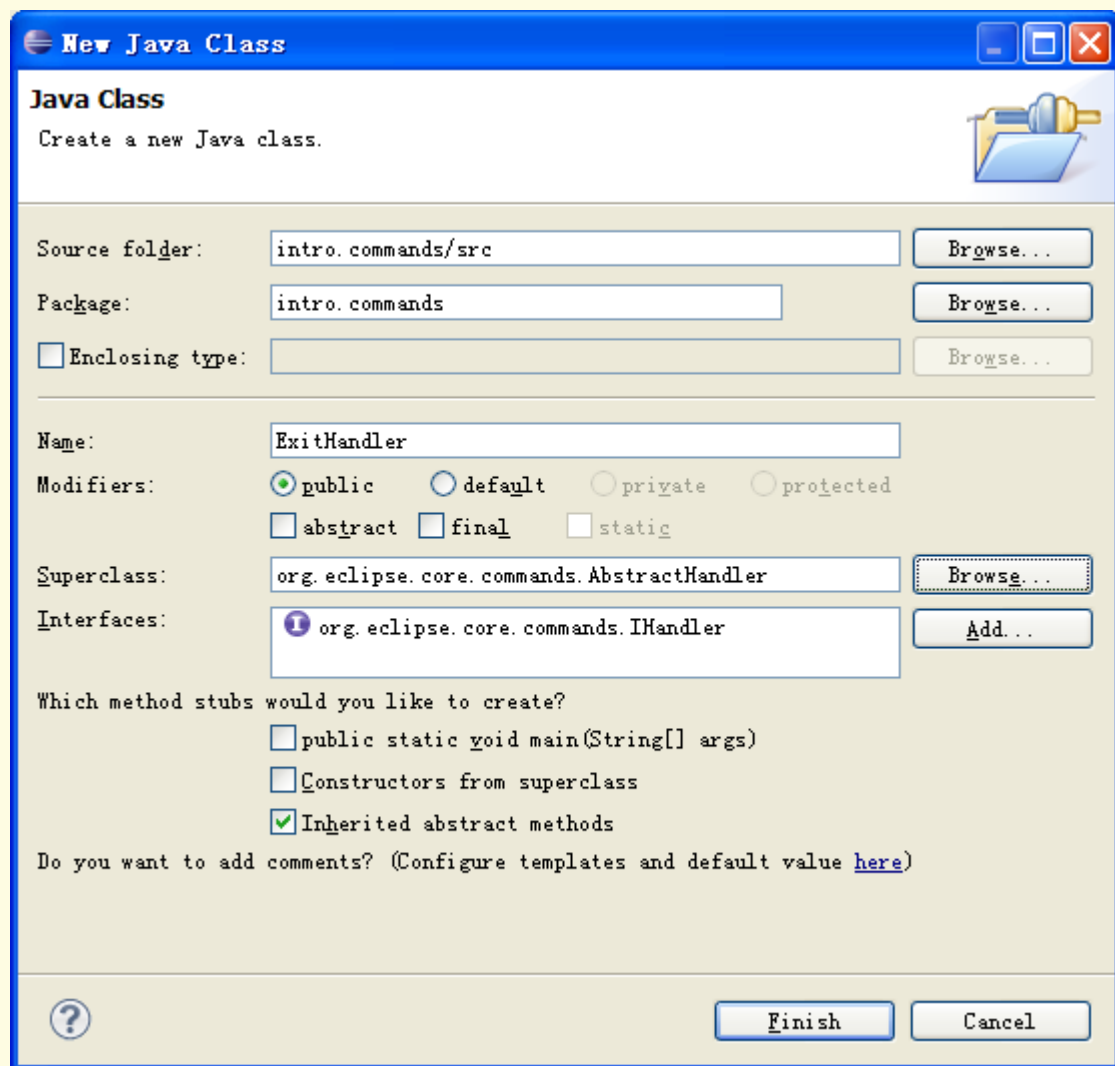
Set the ID to "org.eclipse.rcp.first.commands.Exit" and the name to "Exit". Enter the class "first.commands.ExitHandler" as defaultHandler.

将命令的 ID 设置为 org.eclipse.rcp.first.commands.Exit，其名称为 Exit。在 defaultHandler 栏中输入类 intro.commands.ExitHadler。



Press the hyperlink "defaultHandler" to create this class. Choose org.eclipse.core.commands.AbstractHandler as Superclass.

单击 defaultHandler 链接创建这个类。并选择 org.eclipse.core.commands.AbstractHandler 作为其 Superclass:



Implement the following coding

完成如下代码：

```
package intro.commands;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.ui.handlers.HandlerUtil;

public class ExitHandler extends AbstractHandler {

    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {
        HandlerUtil.getActiveWorkbenchWindow(event).close();
        return null;
    }
}
```

```
}  
  
}
```

You have correctly implemented the command. This command can now be used in various places in your application.

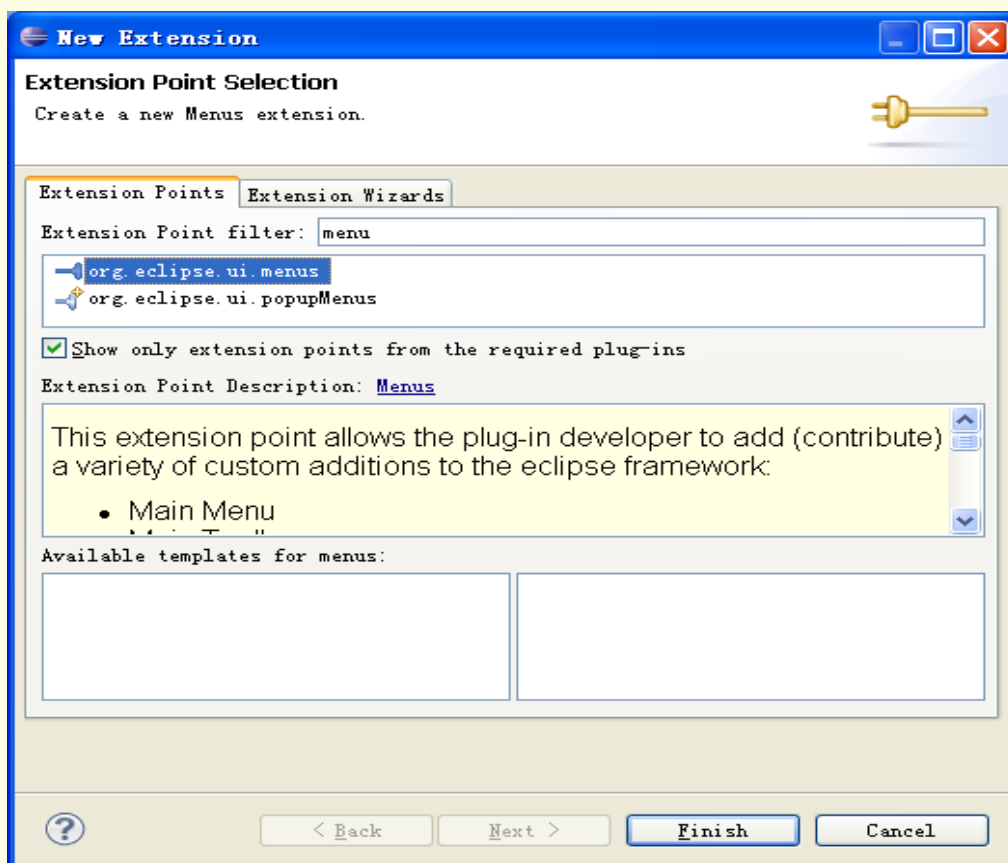
你已经正确的实现了此命令。现在你可以在你的应用中使用此命令了。

6.3. USING COMMANDS IN MENUS 在菜单中使用命令

The command which we just defined should now be used in a new menu.

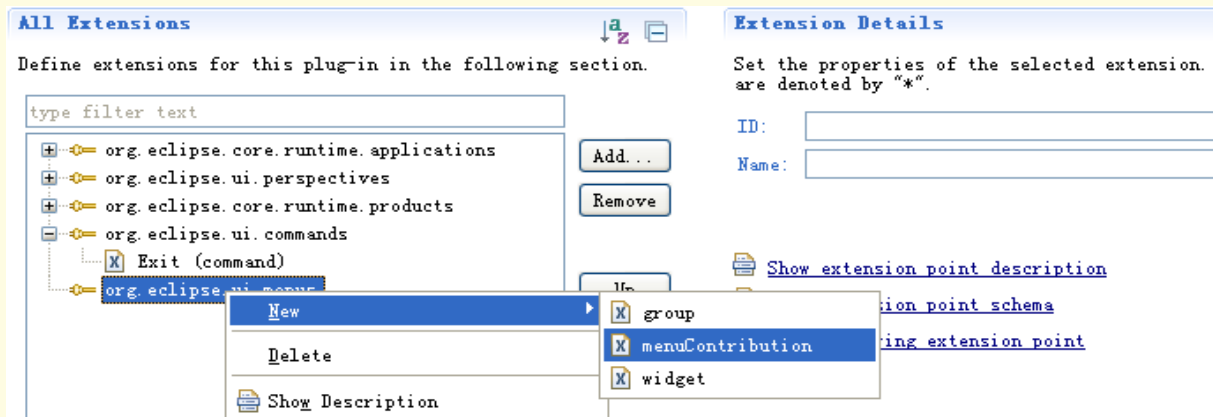
Add the extension point "org.eclipse.ui.menu" to your application. Select therefore again plugin.xml and the tab "Extensions". Press Add, select the extension point "org.eclipse.ui.menu" and press Finish.

我们刚才定义的命令可以在菜单中使用。现在我们再次打开文件 **plugin.xml**，选择 **Extensions** 标签，来为应用添加扩展点 “**org.eclipse.ui.menu**”。点击 **Add**，选择 **org.eclipse.ui.menu** 扩展点，再单击 **Finish** 按钮。



Right click on the extension point and select new -> menuContribution.

在新增的扩展点上单击右键，选择 **New→menuContribution**



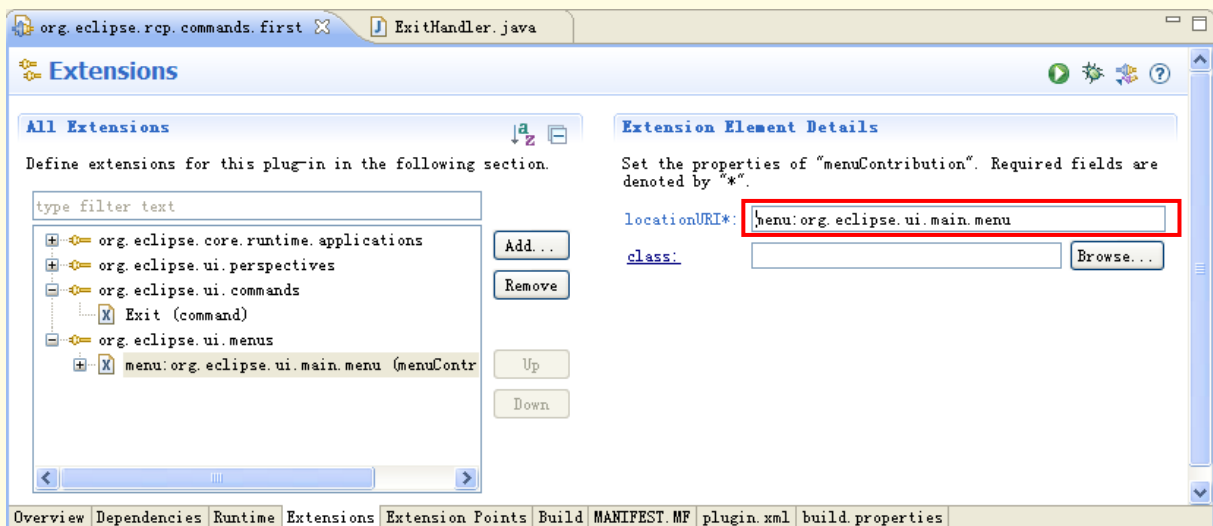
Create a new menu contribution with the location URI "menu:org.eclipse.ui.main.menu".

创建一个菜单贡献，其 locationURI（定位 URI）为 “menu:org.eclipse.ui.main.menu”。

Tip

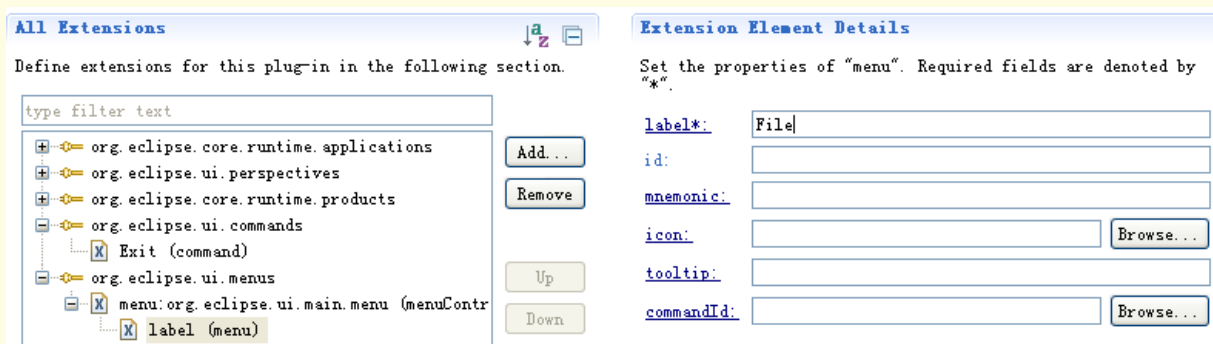
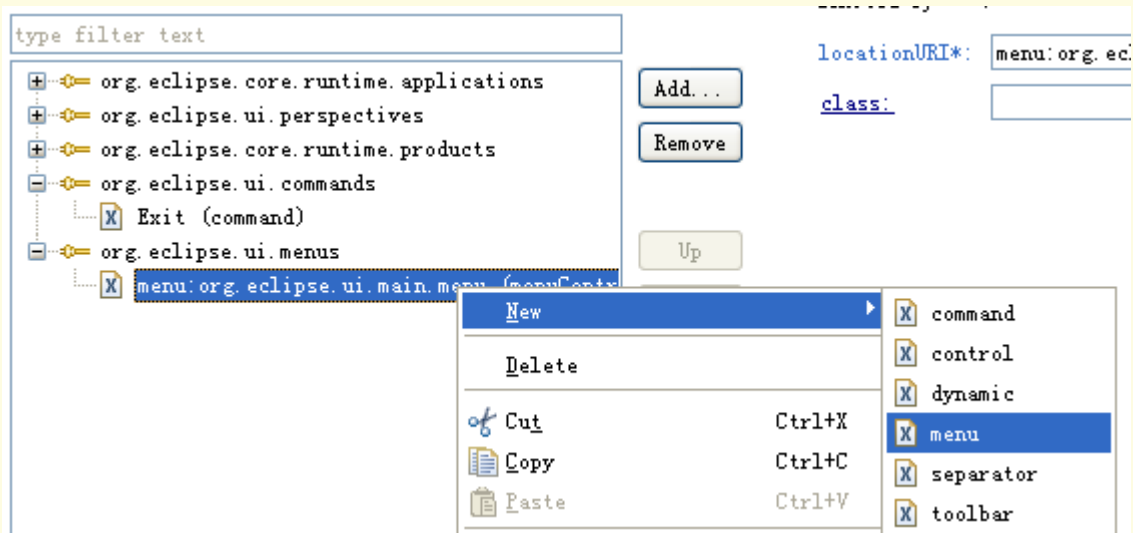
Make sure the URI must is correct, otherwise the menu will not get displayed.

提示：请注意确认此 URI 必须正确，否则菜单无法显示。



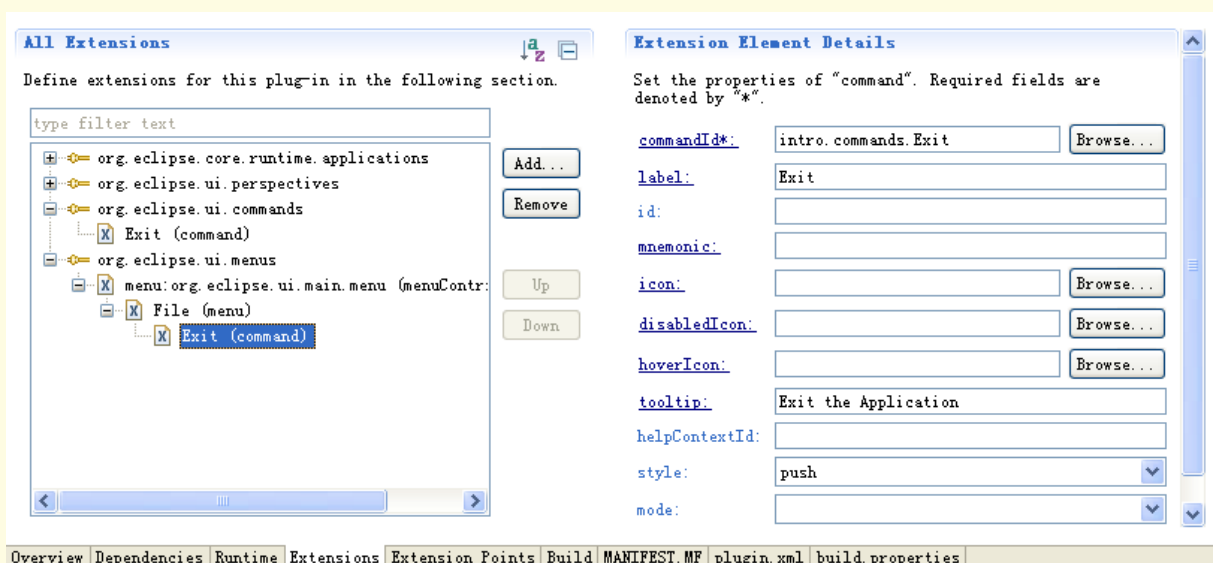
Right click your menuContribution and select New -> Menu. Add a menu with the label "File" and the id "fileMenu".

用鼠标右键单击新建的菜单贡献，从弹出花间中选择 New→menu 添加一个菜单，在其 Label 栏输入 File:



Now select your menu, right-click on it and select New-> Command. Maintain the commandID you used earlier. Set the label to "Exit" and the tooltip to "Exit the Application".

现在右键点击新建的菜单，从弹出菜单中选择 New→Command，修改 commandID 栏为前面所创建的命令 ID（intro.commands.Exit），设置 label 栏为 Exit，提示栏为“Exit the Application”



Your work should result in a plugin.xml file which looks like the following.

你刚才所完成的工作将会对 plugin.xml 文件进行修改，其内容现在大概如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>

    <extension
        id="application"
        point="org.eclipse.core.runtime.applications">
        <application>
            <run
                class="intro.commands.Application">
            </run>
        </application>
    </extension>

    <extension
        point="org.eclipse.ui.perspectives">
        <perspective
            name="RCP Perspective"
            class="intro.commands.Perspective"
            id="intro.commands.perspective">
        </perspective>
    </extension>

    <extension
        point="org.eclipse.ui.commands">
        <command
            defaultHandler="intro.commands.ExitHandler"
            id="intro.commands.Exit"
            name="Exit">
        </command>
    </extension>

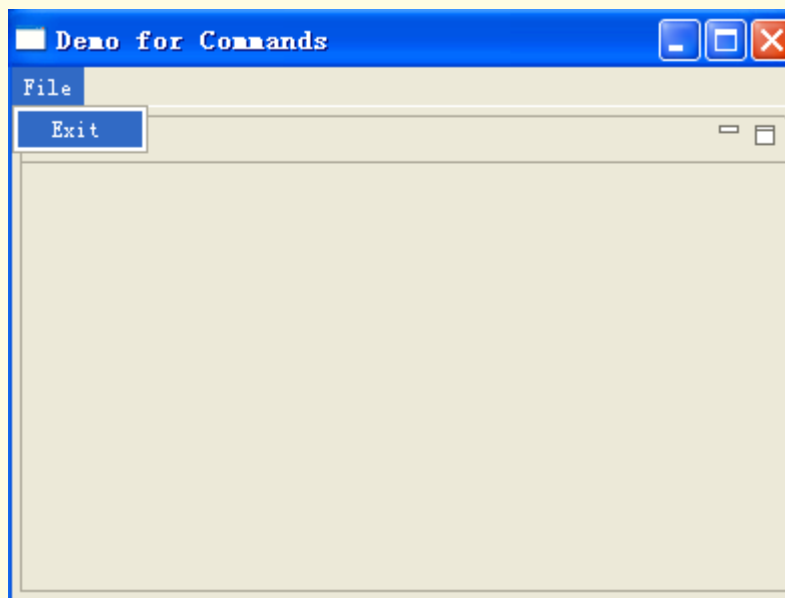
    <extension
        point="org.eclipse.ui.menus">

```

```
<menuContribution
    locationURI="menu:org.eclipse.ui.main.menu">
    <menu
        label="File">
        <command
            commandId="intro.commands.Exit"
            label="Exit"
            style="push"
            tooltip="Exit the Application">
        </command>
    </menu>
</menuContribution>
</extension>
</plugin>
```

If you run the example you should see menu with the file. Selecting Exit should end your application.

如果现在运行该应用，你可以看到在 **File** 菜单下出现了一个子菜单项 **Exit**。单击 **Exit** 会结束应用。



7. SYSTEM TRAY 系统托盘

In the following we will add a system tray icon, add the functionality that if the window is minimized then the program is not visible in the taskpane (only via the tray icon) and we will add a menu with some entries into the system tray icon.

下面我们将为应用添加系统托盘图标，为应用添加这样一个功能：当应用最小化后，其在任务栏中不再可见（只能通过托盘上的图标）。我们还将添加一个包含最小化到系统托盘项的菜单。

Maintain the following code in class `ApplicationWorkbenchWindowAdvisor`.

修改 `ApplicationWorkbenchWindowAdvisor` 类，添加如下代码：

Tip

To get a tray icon you need a `Display`. A display is an SWT object that represents the underlying graphics system. This object is available as of method `postWindowOpen()` in `ApplicationWorkbenchWindowAdvisor`.

提示：要添加托盘图标，你需要一个 `Display`。`Display` 是一个表述底层图像系统的 SWT 对象。这个对象有一个 `postWindowOpen()` 方法可以在 `ApplicationWorkbenchWindowAdvisor` 中使用。

```
package intro.commands;

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.MenuItem;
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.handlers.IHandlerService;
import org.eclipse.ui.plugin.AbstractUIPlugin;

public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {
    private IWorkbenchWindow window;
    private TrayItem trayItem;
    private Image trayImage;
    private final static String COMMAND_ID = "intro.commands.Exit";

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

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

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

    @Override
    public void postWindowOpen() {
        super.postWindowOpen();
        window = getWindowConfigurer().getWindow();
    }
}

```



```

    trayItem = initTaskItem(window);
    if (trayItem != null) {
        createMinimize();
        // Create exit and about action on the icon
        hookPopupMenu();
    }
}

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 void hookPopupMenu() {
    trayItem.addListener(SWT.MenuDetect, new Listener() {
        public void handleEvent(Event event) {
            Menu menu = new Menu(window.getShell(), SWT.POP_UP);
            // Creates a new menu item that terminates the program
            // when selected

```

```

MenuItem exit = new MenuItem(menu, SWT.NONE);
exit.setText("Goodbye!");
exit.addListener(SWT.Selection, new Listener() {
    public void handleEvent(Event event) {
        // Lets call our command
        IHandlerService handlerService = (IHandlerService) window
            .getService(IHandlerService.class);
        try {
            handlerService.executeCommand(COMMAND_ID, null);
        } catch (Exception ex) {
            throw new RuntimeException(COMMAND_ID);
        }
    }
});
// We need to make the menu visible
menu.setVisible(true);
}
});
}

```

```

private TrayItem initTaskItem(IWorkbenchWindow window) {
    final Tray tray = window.getShell().getDisplay().getSystemTray();
    TrayItem trayItem = new TrayItem(tray, SWT.NONE);
    trayImage = AbstractUIPlugin.imageDescriptorFromPlugin(
        "intro.commands", "/icons/alt_window_16.gif").createImage();
    trayItem.setImage(trayImage);
    trayItem.setToolTipText("TrayItem");
    return trayItem;
}

```

```

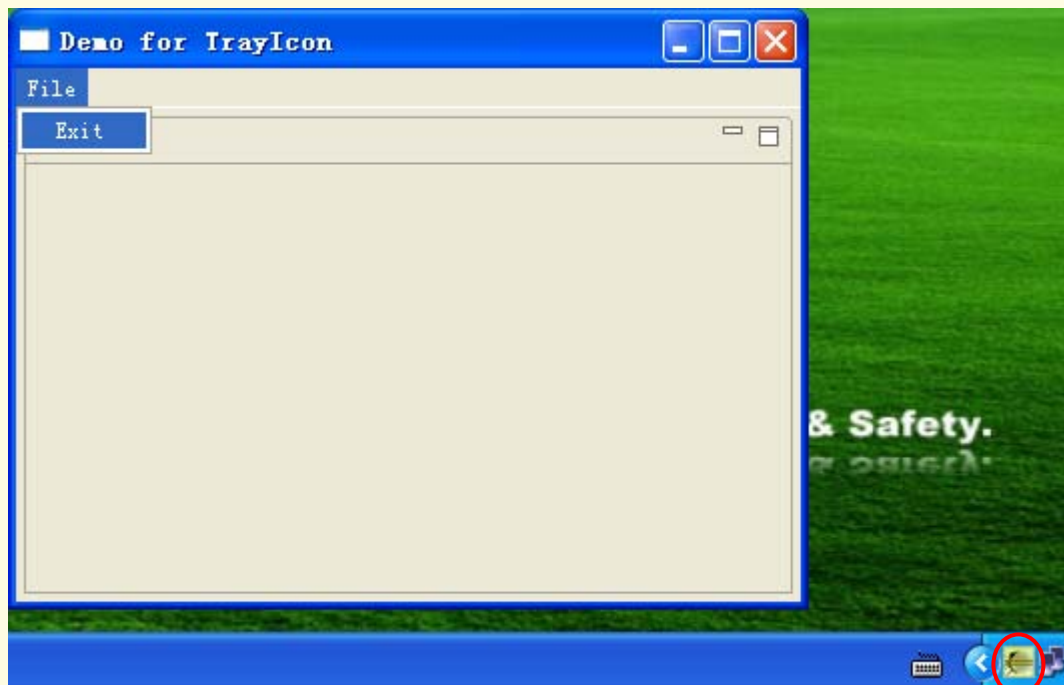
public void dispose() {

```

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

Run your application and see that you have a system tray icon. Test the menu and the minimized behavior.

运行程序，可以看到在任务条上出现了一个应用图标，试一试最小化应用，哈，在任务托盘出现了一个应用图标！你甚至在此图标上点击右键，并退出应用。



8. VIEWS 视图

8.1. OVERVIEW 概览

Views provide information for a given task. A view is typically used to navigate a hierarchy of information, open an editor, or display properties for the active editor.

视图提供了给定任务的信息。视图通常用于进行层次化的信息导航、打开编辑器或者显示当前编辑器的信息。

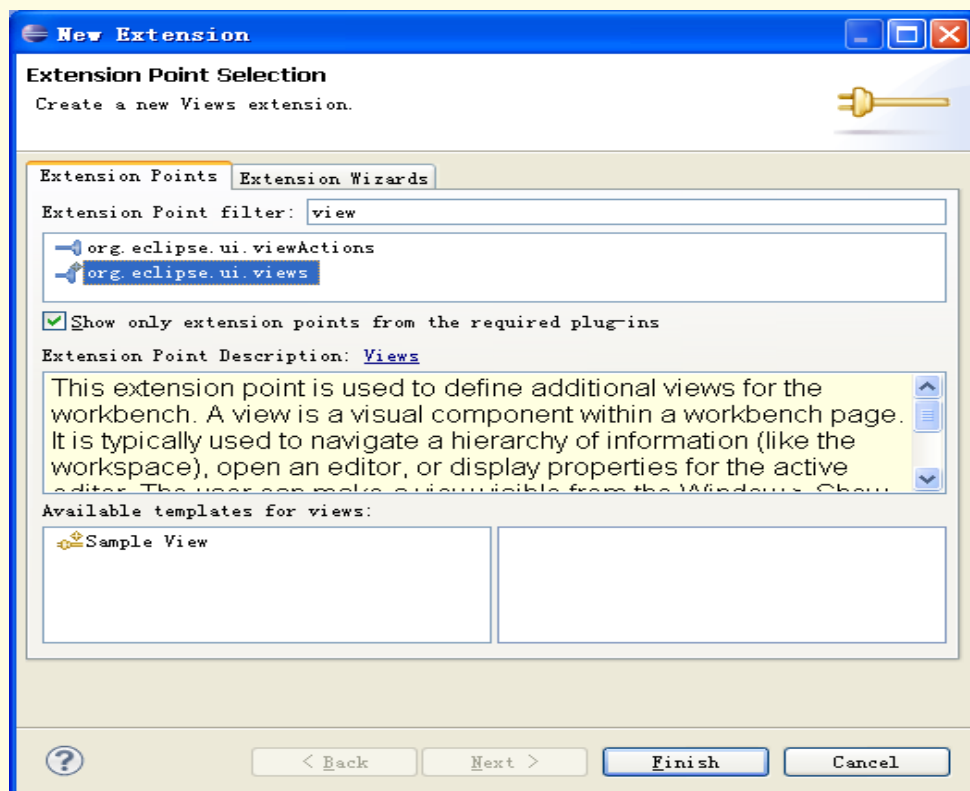
The following will explain how to add views to your application.

下面我们介绍怎样为应用添加视图。我们仍然在前面的示例应用中进行。

8.2. CREATE A VIEW 创建视图

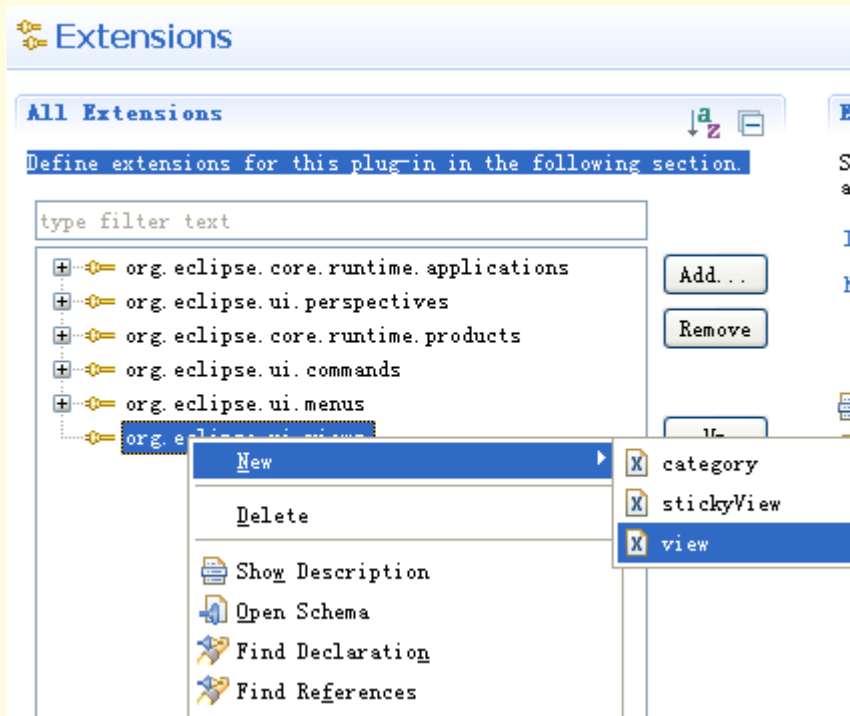
Select the file "plugin.xml" and the tab "Extensions". Press the "add" button and add "org.eclipse.ui.views" as an extension.

打开 plugin.xml，选择 Extensions 标签页，单击 Add 按钮添加 org.eclipse.ui.views 作为一个扩展点。



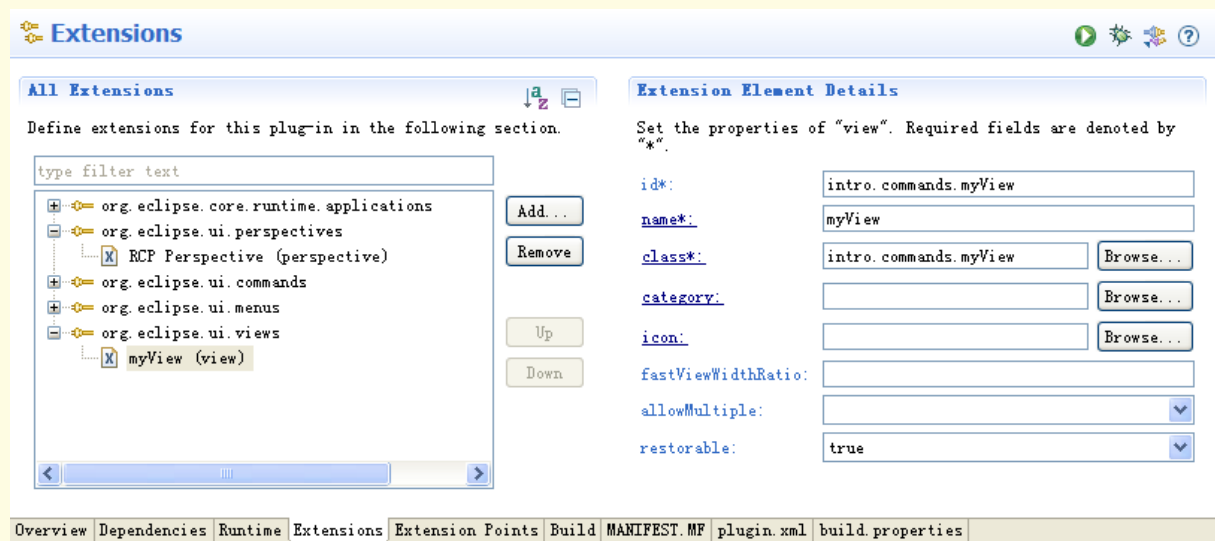
Right mouse-click on your new view extension and select New -> View

在新建的扩展上单击右键，从弹出菜单上选择 New→View:



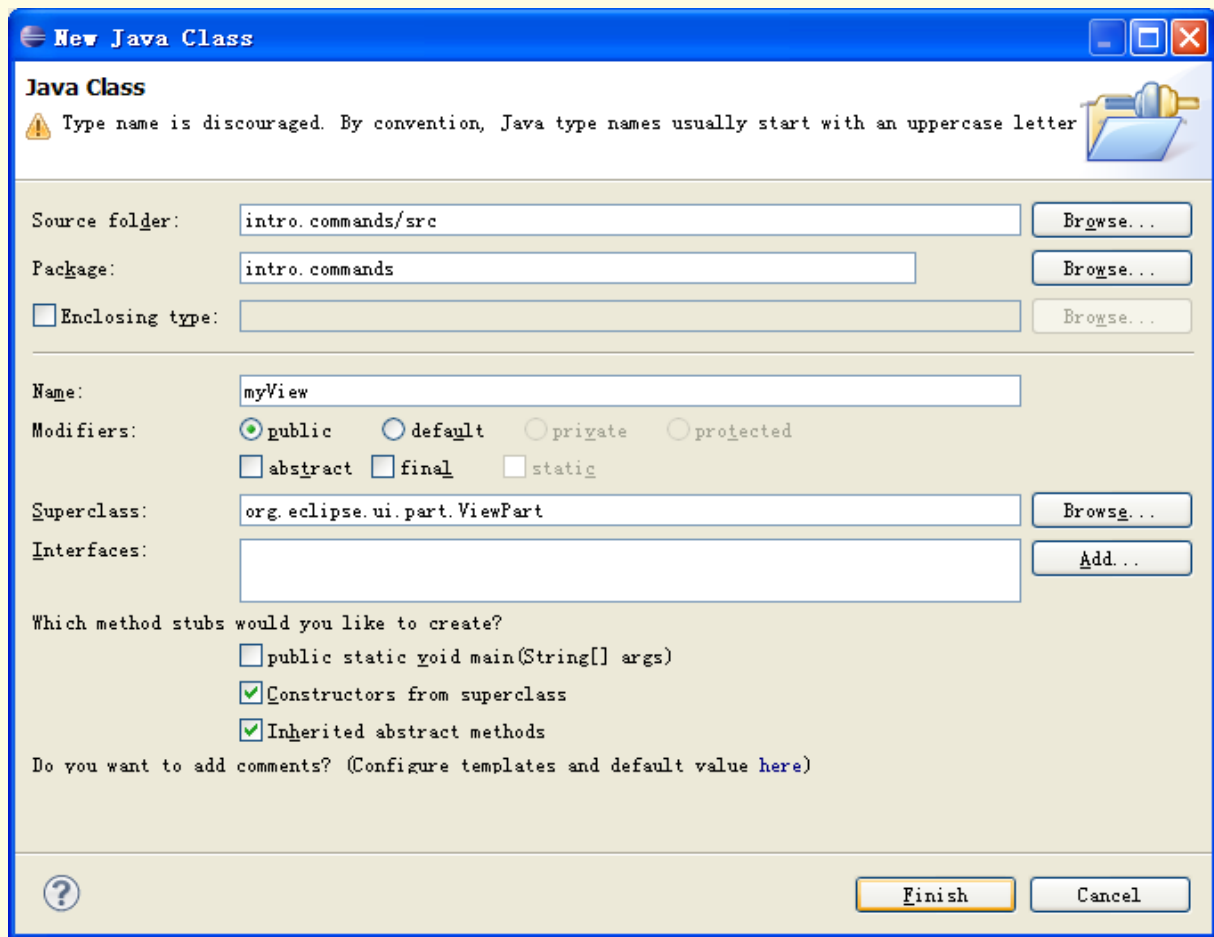
Maintain the id "intro.commands.myView" and the class "intro.commands.myView".

修改视图 ID 为 ntro.commands.myView，类名为 ntro.commands.myView



Create the class of the view by clicking on the class hyperlink.

单击 class 链接为该视图创建类:



Maintain the following code in your new class.

修改新创建的类代码如下：

```
package intro.commands;

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 myView extends ViewPart {

    @Override
    public void createPartControl(Composite parent) {
        Text text = new Text(parent, SWT.BORDER);
        text.setText("Imagine a fantastic user interface here");
    }

    @Override
    public void setFocus() {
```

```
}  
}
```

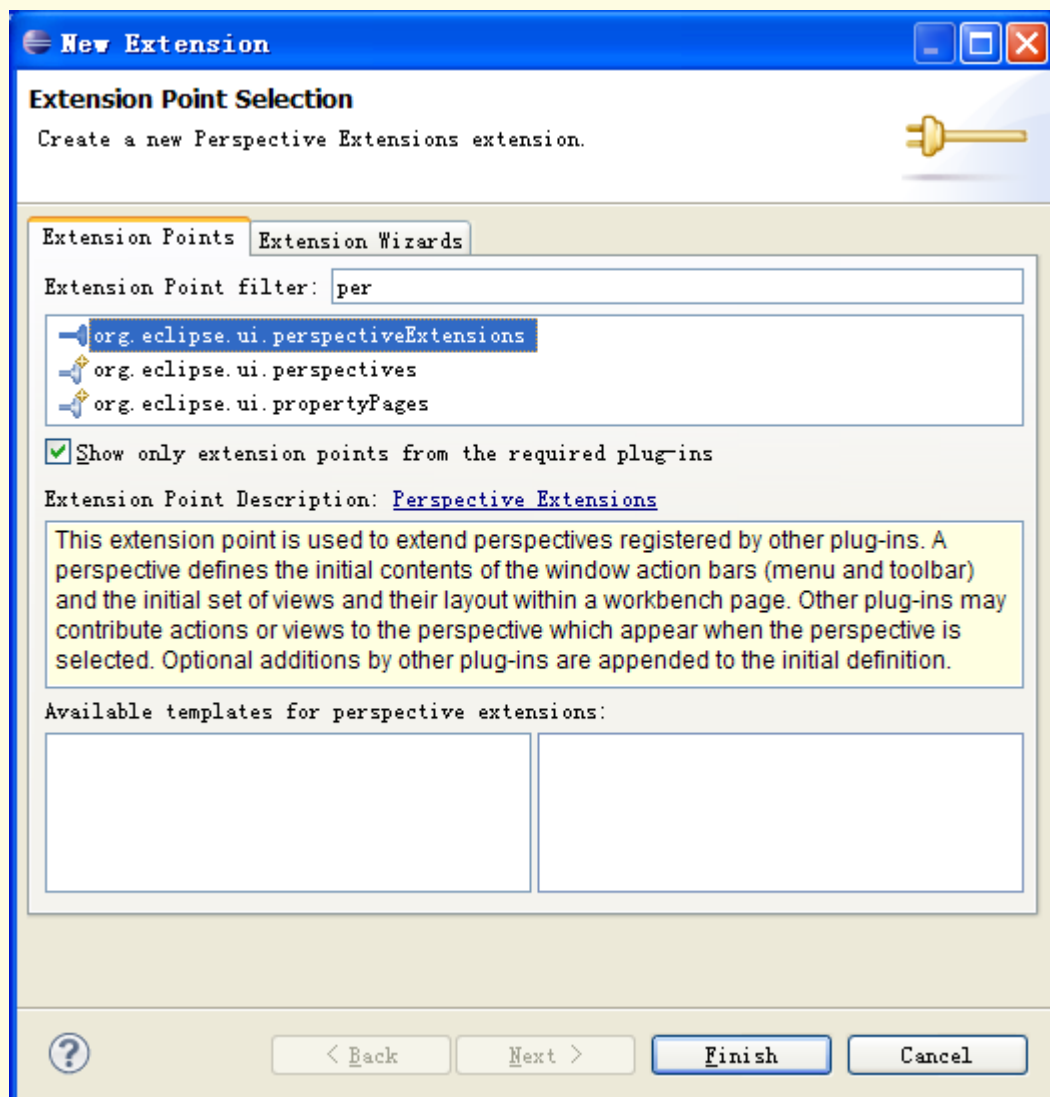
8.3. ADD THE VIEW TO YOUR PERSPECTIVE. 将视图添加到透视图

After creating the view you have to add this view to your perspective. We will do this by using another extension point.

在创建了视图后，你还必须将其添加到透视图。我们利用另一个扩展点来完成此任务。

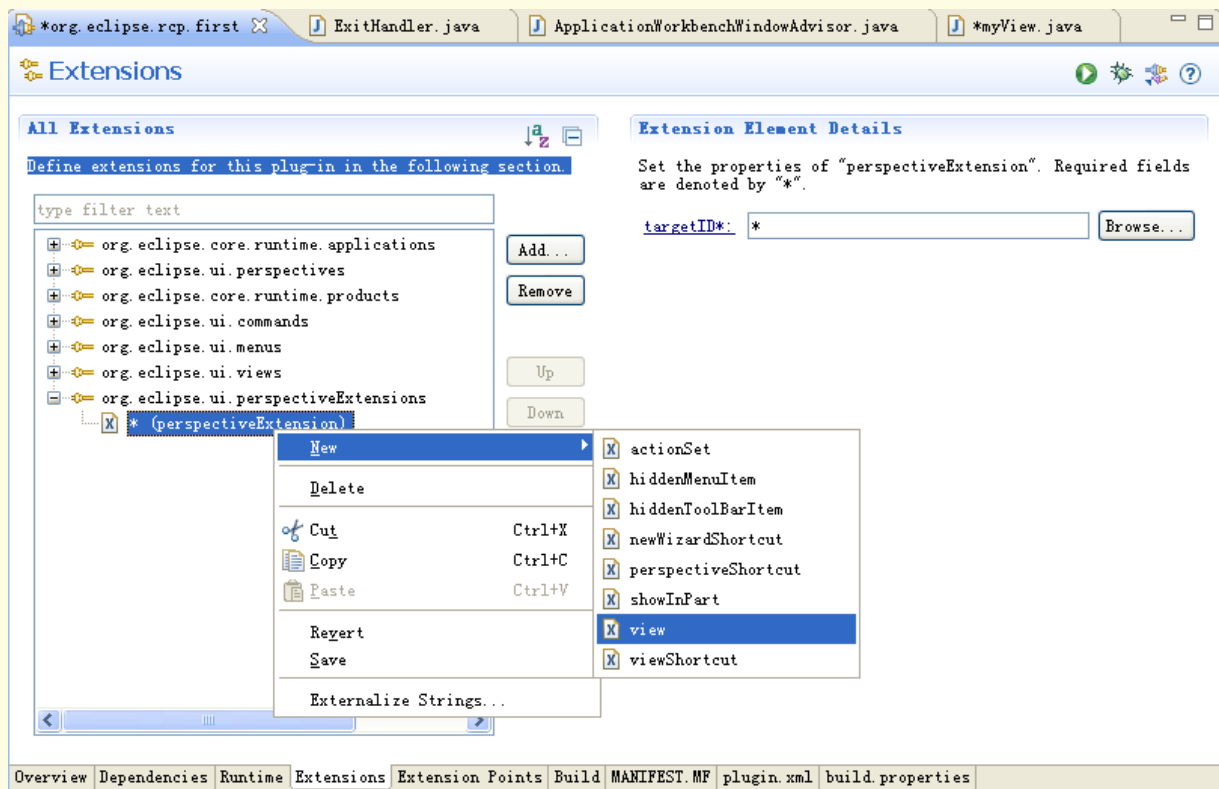
Select again "plugin.xml" and the tab "Extensions". Press add and add the extension "org.eclipse.ui.perspectiveExtensions".

再次打开 plugin.xml，选择 Extensions 标签，添加一个 org.eclipse.ui.perspectiveExtensions 扩展。



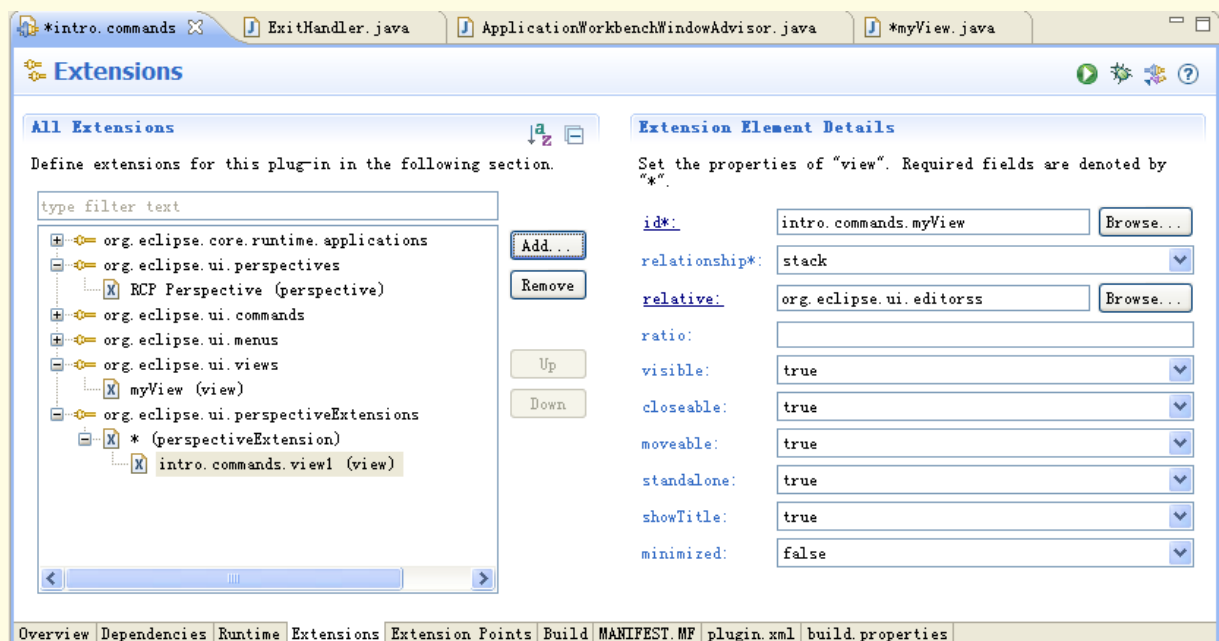
Right click it and select view.

在新建的扩展上单击右键，从弹出菜单中选择 New→View:



Maintain your view id "intro.commands.myView". Make the view relative to "org.eclipse.ui.editorss" which is the currently invisible editor area and make the view use all the space by selecting the maximum ratio of "0.95f".

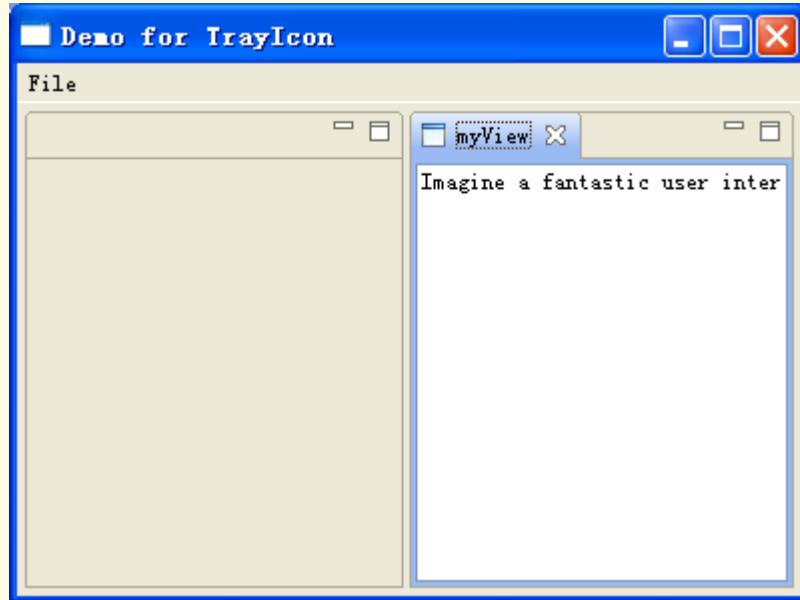
修改视图 ID 为 intro.commands.myView，栏 relative 的值为 org.eclipse.ui.editorss，其它参数按下图所示设置:



8.4. RESULT 结果

Run your application to see the result.

运行应用，其结果如下：



8.5. ADD VIEW TO PERSPECTIVE VIA CODE 利用代码将视图添加到透视图

Alternatively to the usage of the extension point "org.eclipse.ui.perspectiveExtensions" you could have add the view also via coding to your perspective. To to this you could modify "Perspective.java" to the following.

除了利用扩展点 `org.eclipse.ui.perspectiveExtensions` 可以将视图添加至透视图而外，你也可以利用代码将视图添加至透视图。其方法是修改 `Perspective.java` 代码如下：

```
package intro.commands;

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

public class Perspective implements IPerspectiveFactory {

    public void createInitialLayout(IPageLayout layout) {

        layout.addView("intro.commands.myView", IPageLayout.TOP,
            IPageLayout.RATIO_MAX, IPageLayout.ID_EDITOR_AREA);

    }

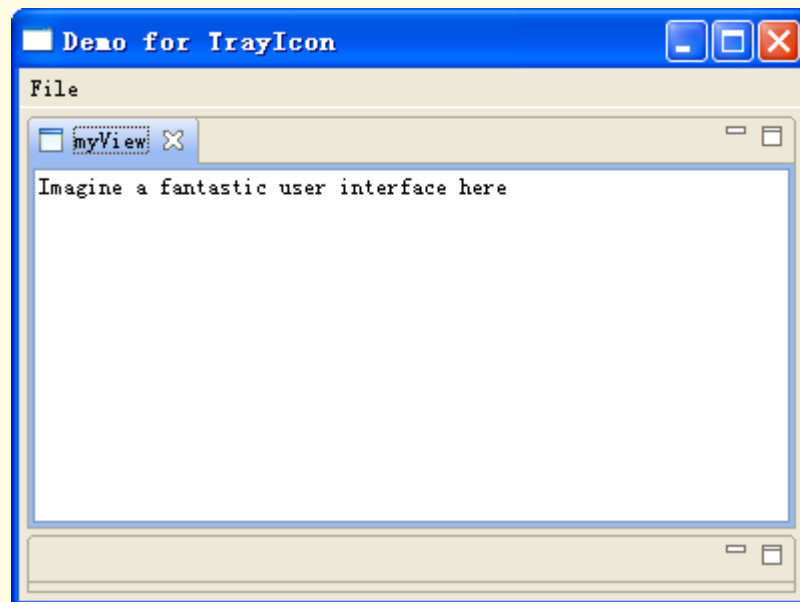
}
```

Tip

If possible use extension points over code.

提示：如果可能，请使用扩展点方式而不要使用代码方式。

利用代码方式添加视图之后，本例的运行效果如下：



9. WORKING WITH EDITORS, VIEW INTERACTION AND MODEL UPDATES 使用编辑器、视图进行交互及模型更新

9.1. OVERVIEW 概览

In the following we create a project which demonstrate the usage of editors and its interaction with an view.

We will create a view which shows several person. Once the user double-clicks on the name an editor is opened in which the user can edit the address of this person.

下面我们将创建一个项目，讲述编辑器的使用及与视图的交互。我们将创建一个显示某些联系人的视图，一旦用户点击某个人的名字，就会打开一个编辑器，可以在此编辑器中对该联系人的地址进行编辑。

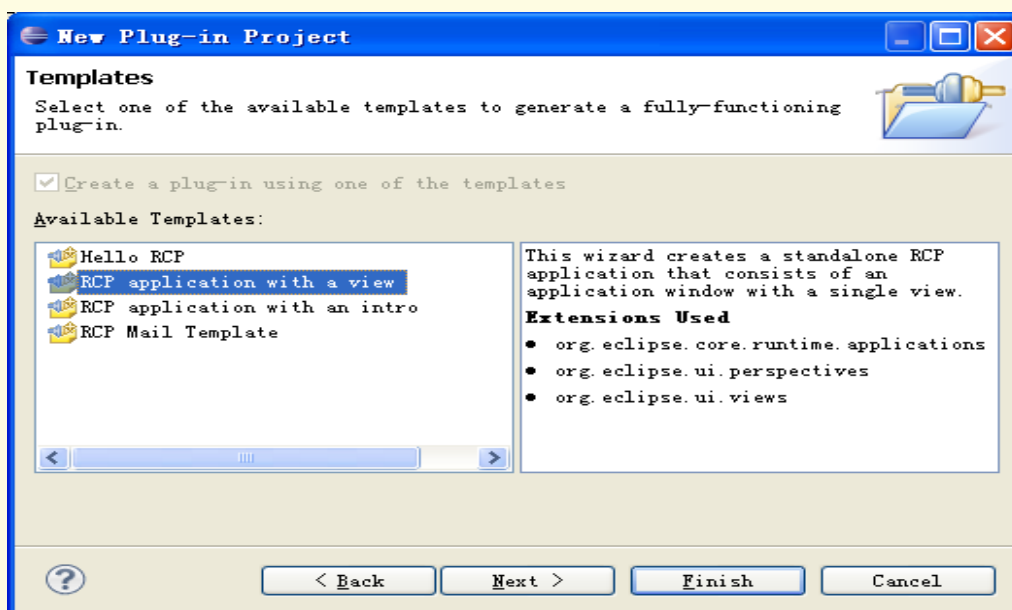
The following data model makes the assumption that the last name of a person is unique. This assumption is of course not true in the real world.

在下面的数据模型中，我们假定每个人的名字都是唯一的。当然在现实世界中此假定是不成立的。

9.2. CREATE PROJECT 创建项目

Create a new RCP project "org.eclipse.rcp.intro.editor". Use the "RCP application with a view" as a template.

利用 RCP application with a view 模板创建一个新的名为“org.eclipse.rcp.intro.editor”的 RCP 项目。



9.3. CREATE AND PREPARE THE DOMAIN MODEL *创建并准备域模型*

Create a package "org.eclipse.rcp.intro.editor.model". Create the following classes in this package.

创建一个名为 “org.eclipse.rcp.intro.editor.model” 的包，在此包中创建如下类：

类 Address 代码如下：

```
package org.eclipse.rcp.intro.editor.model;

public class Address {

    private String street;

    private String number;

    private String postalCode;

    private String city;

    private String country;

    public String getStreet() {

        return street;

    }

    public void setStreet(String street) {

        this.street = street;

    }

    public String getNumber() {

        return number;

    }

    public void setNumber(String number) {

        this.number = number;

    }

    public String getPostalCode() {

        return postalCode;

    }

    public void setPostalCode(String postalCode) {

        this.postalCode = postalCode;

    }

    public String getCity() {
```

```

        return city;
    }
    public void setCity(String city) {
        this.city = city;
    }
    public String getCountry() {
        return country;
    }
    public void setCountry(String country) {
        this.country = country;
    }
    public String toString() {
        return street + " " + number + " " + postalCode + " " + city + " "
            + country;
    }
}

```

类 **Person** 代码如下：

```

package org.eclipse.rcp.intro.editor.model;

public class Person {
    private String firstName;
    private String lastName;
    private Address address;
    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}

```

```

    }

    public String getLastName() {
        return lastName;
    }

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

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return firstName + " " + lastName;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
            + ((firstName == null) ? 0 : firstName.hashCode());
        result = prime * result
            + ((lastName == null) ? 0 : lastName.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;

```

```

        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Person other = (Person) obj;
        if (firstName == null) {
            if (other.firstName != null)
                return false;
        } else if (!firstName.equals(other.firstName))
            return false;
        if (lastName == null) {
            if (other.lastName != null)
                return false;
        } else if (!lastName.equals(other.lastName))
            return false;
        return true;
    }
}

```

类 **MyModel** 代码如下：

```

package org.eclipse.rcp.intro.editor.model;

import java.util.ArrayList;
import java.util.List;

public class MyModel {

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

    public List<Person> getPersons() {

        return persons;
    }

    public MyModel() {

        // Just for testing we hard-code the persons here:
        Person person = new Person("Lars", "Vogel");
        person.setAddress(new Address());
    }
}

```

```

        person.getAddress().setCountry("Germany");
        persons.add(person);

        person = new Person("Jim", "Knopf");
        person.setAddress(new Address());
        person.getAddress().setCountry("Germany");
        persons.add(person);
    }
}

```

Tip

Please note that we override the equals and the hashCode method). This allows us later to identify the right editor for each person.

提示：请注意我们在这儿重载了 **equals** 及 **hashCode** 方法。这样后面就可以在右侧的编辑器上标识每个人。

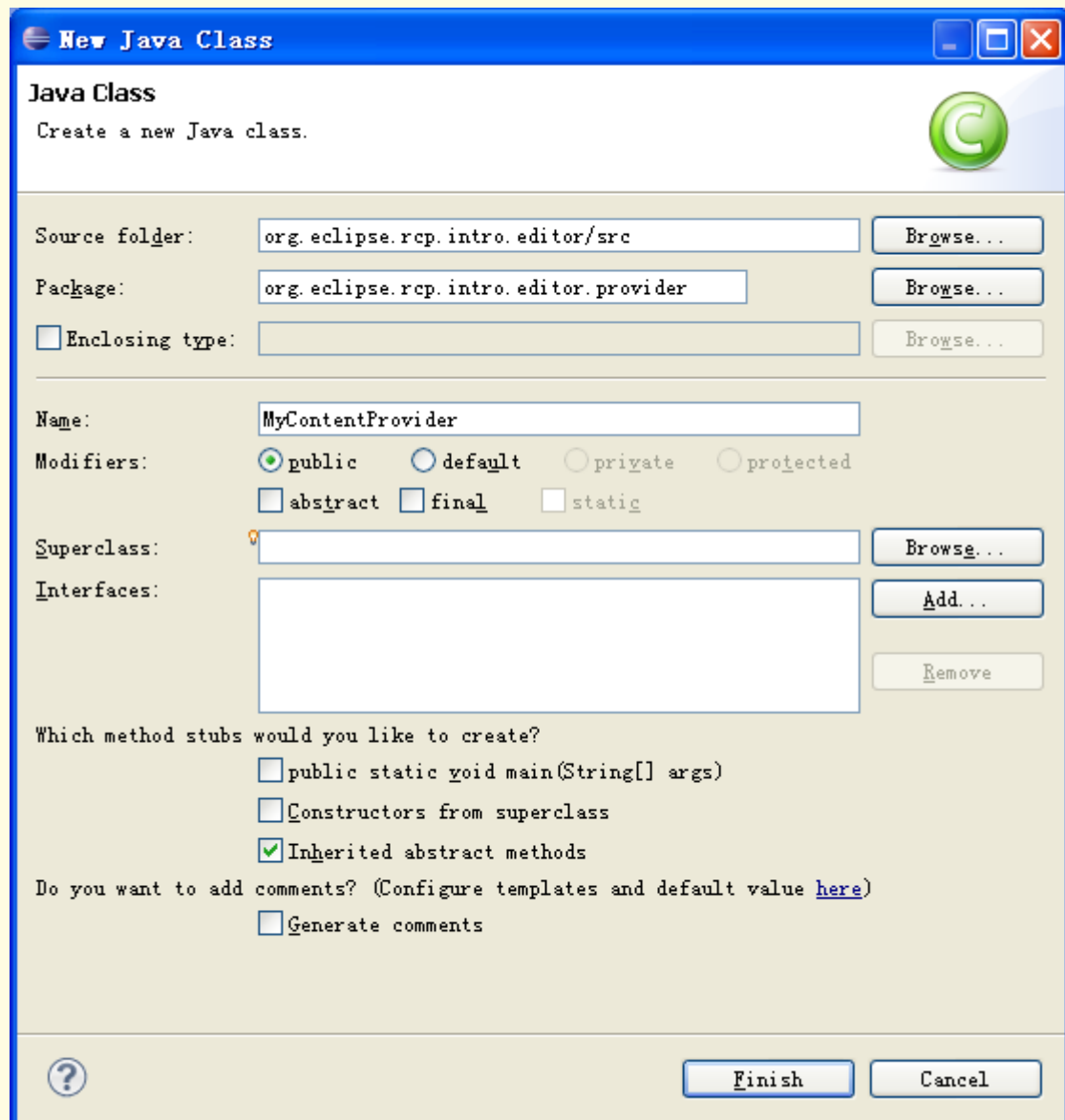
9.4. CONTENT AND LABEL PROVIDER *内容与标签提供者*

Create a new package "org.eclipse.rcp.intro.editor.provider "

创建一个新包名为 “**org.eclipse.rcp.intro.editor.provider**”

Create a new class "MyContentProvider" which implements the interface IStructuredContentProvider.

在此包中创建一个实现 **IStructuredContentProvider** 的类 **MyContentProvider**:



Adjust the coding according to the following example:

修改类代码内容如下:

```
package org.eclipse.rcp.intro.editor.provider;

import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.rcp.intro.editor.model.MyModel;

public class MyContentProvider implements IStructuredContentProvider,
    PropertyChangeListener {
    private final Viewer viewer;
```

```

public MyContentProvider(Viewer viewer) {
    this.viewer = viewer;
}

@Override
public Object[] getElements(Object inputElement) {
    MyModel content = (MyModel) inputElement;
    return content.getPersons().toArray();
}

@Override
public void dispose() {
}

@Override
public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {
}

@Override
public void propertyChange(PropertyChangeEvent arg0) {
    viewer.refresh();
}
}

```

Create a new class "MyLabelProvider" which implements the interface ILabelProvider. Use the following code.

再创建一个新类名为 **MyLabelProvider**，此类实现了 **ILabelProvider** 接口，其代码如下：

```

package org.eclipse.rcp.intro.editor.provider;

import org.eclipse.jface.viewers.ILabelProvider;
import org.eclipse.jface.viewers.ILabelProviderListener;
import org.eclipse.swt.graphics.Image;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.PlatformUI;
import org.eclipse.rcp.intro.editor.model.Person;

public class MyLabelProvider implements ILabelProvider {

```

```

@Override
public Image getImage(Object element) {
    return PlatformUI.getWorkbench().getSharedImages().getImage(
        ISharedImages.IMG_OBJ_ELEMENT);
}
@Override
public String getText(Object element) {
    Person person = (Person) element;
    return (person.getLastName());
}
@Override
public void addListener(ILabelProviderListener listener) {
}
@Override
public void dispose() {
}
@Override
public boolean isLabelProperty(Object element, String property) {
    return false;
}
@Override
public void removeListener(ILabelProviderListener listener) {
}
}

```

9.5. USE THE DOMAIN MODEL IN THE VIEW 在视图中使用域模型

Change the class View to use your new content providers.

利用你的新内容提供者修改类视图代码如下：

```

package org.eclipse.rcp.intro.editor;

import org.eclipse.jface.viewers.TableViewer;

```

```

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.ViewPart;
import org.eclipse.rcp.intro.editor.model.MyModel;
import org.eclipse.rcp.intro.editor.provider.MyContentProvider;
import org.eclipse.rcp.intro.editor.provider.MyLabelProvider;
public class View extends ViewPart {

    public static final String ID = "org.eclipse.rcp.intro.editor.view";

    private TableViewer viewer;

    /**
     * This is a callback that will allow us to create the viewer and initialize it.
     */
    public void createPartControl(Composite parent) {

        viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL
                                | SWT.V_SCROLL);

        viewer.setContentProvider(new MyContentProvider(viewer));
        viewer.setLabelProvider(new MyLabelProvider());
        viewer.setInput(new MyModel());
        getSite().setSelectionProvider(viewer);
    }

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

        viewer.getControl().setFocus();
    }
}

```

The view makes his viewer available as selection provider via the following line: "getSite().setSelectionProvider(viewer);". This make is possible for the command which opens the editor to get the selection of the view.

此视图通过 `getSite().setSelectionProvider(viewer)` 这一行使得在选择供应者时视图可用。这样就可以利用命令打开编辑器来获取视图选择。

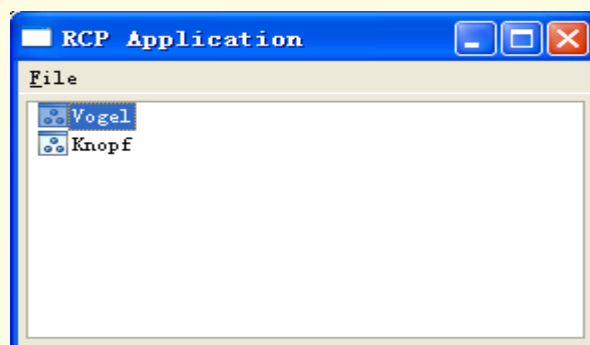
Tip

All workbench parts have a site, which can be accessed via the method `getSite()`. A site is a [Facade](#) which allows access to other parts of the workbench, e.g. the shell, the workbench window, etc. Whenever possible use the site to access Workbench objects.

提示：所有的工作台都有一个站点，可以通过 `getSite()` 方法来访问此站点。站点就像一个联络站，允许访问工件台的其它部分，例如外壳、工件台窗口等。无论何时，都可以利用站点访问工作台对象。

Run your application. The view should now display the last names for your content provider. Currently nothing happens if you click on the names.

现在，运行你的程序。视图将会显示你的内容提供者的名字。如果你在名字上点击，什么事也不会发生。



9.6. EDITOR AREA 编辑区

Make the editor area visible by changing your `Perspective.java`.

修改 `Perspective.java` 代码以便显示编辑区。

```
package org.eclipse.rcp.intro.editor;

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(true);

        layout.setFixed(true);

        layout.addStandaloneView(View.ID, false, IPageLayout.LEFT, 1.0f, editorArea);

    }

}
```

9.7. EDITOR INPUT 编辑器输入

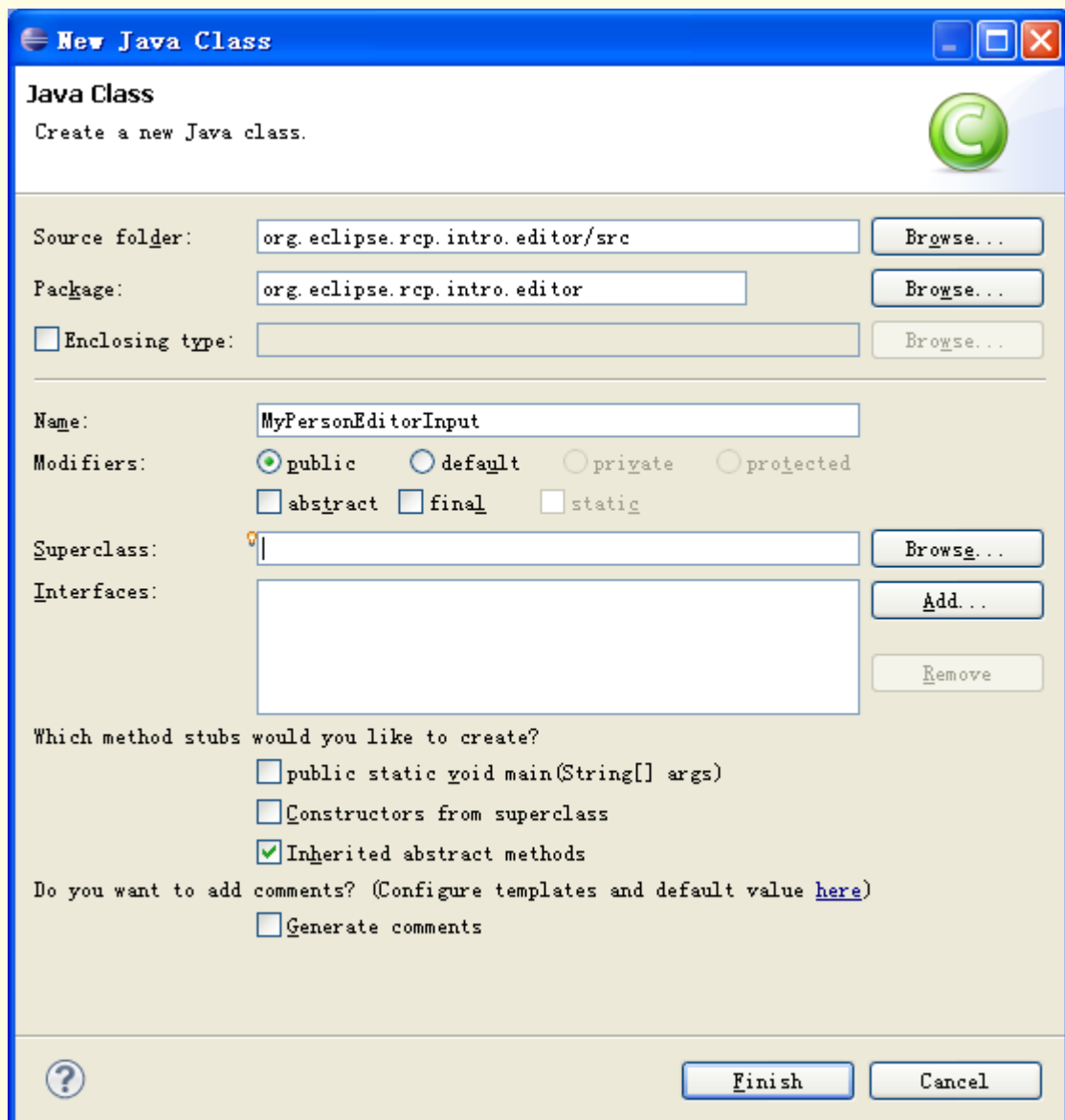
In package "org.eclipse.rcp.intro.editor" create a new class "MyPersonEditorInput" which implements IEditorInput. IEditorInput serves as the model for the editor.

在 org.eclipse.rcp.intro.editor 包中创建一个名为 MyPersonEditorInput 的类，此类实现了 IEditorInput。IEditorInput 用作编辑器的模型。

It is recommended to overwrite the method equals and hashCode in an implementation of IEditor. Based on the equals method the system will determine if the corresponding editor is already open or not.

Change the created code to the following.

建议在 IEditor 重载的实现中重载 equals 与 hashCode 方法。系统将根据 equals 方法来判断相应的编辑器是否已经打开。



类代码如下：

```
package org.eclipse.rcp.intro.editor;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IPersistableElement;
import org.eclipse.rcp.intro.editor.model.Person;

public class MyPersonEditorInput implements IEditorInput {

    private final Person person;

    public MyPersonEditorInput(Person person) {

        this.person = person;
    }

    public Person getPerson() {

        return person;
    }

    @Override
    public boolean exists() {

        return false;
    }

    @Override
    public ImageDescriptor getImageDescriptor() {

        return null;
    }

    @Override
    public String getName() {

        return person.toString();
    }

    @Override
    public IPersistableElement getPersistable() {

        return null;
    }

    @Override
```

```

    public String getToolTipText() {
        return person.toString();
    }

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

    @Override
    public boolean equals(Object obj) {
        if (super.equals(obj)) {
            return true;
        }
        if (obj instanceof MyPersonEditorInput) {
            return person.equals(((MyPersonEditorInput) obj).getPerson());
        }
        return false;
    }

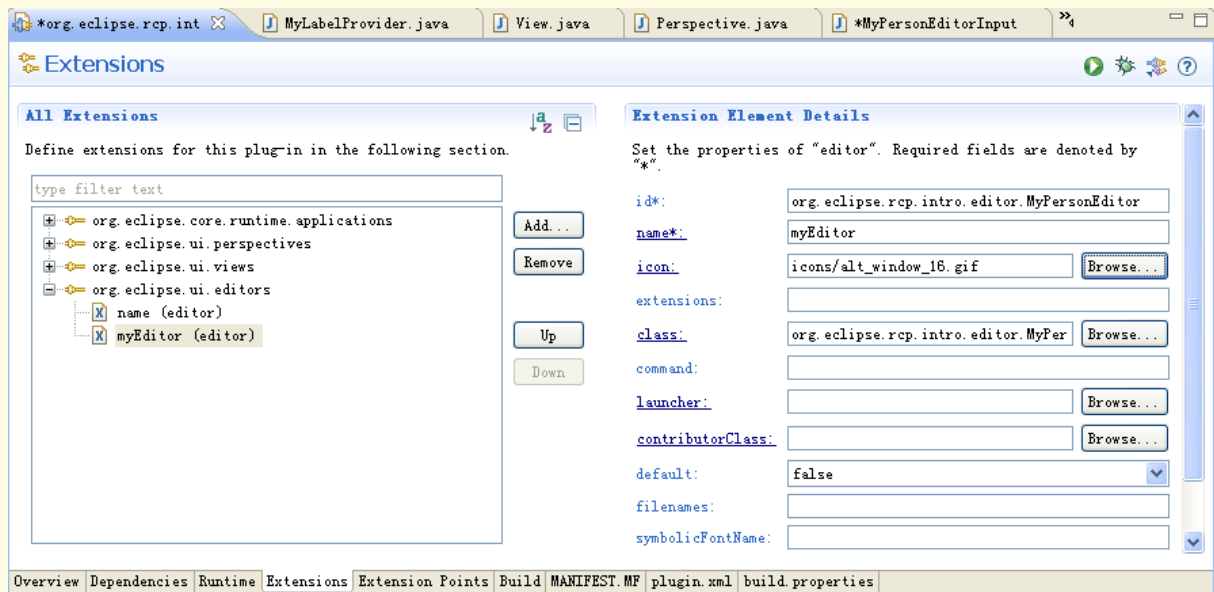
    @Override
    public int hashCode() {
        return person.hashCode();
    }
}

```

9.8. ADDING THE EDITOR 添加编辑器

Go to `plugin.xml` and select the tab `extensions`. Add the extension `org.eclipse.ui.editors`. Do not use a template. Use the ID `"org.eclipse.rcp.intro.editor.MyPersonEditor"`, any name you want and the class `"org.eclipse.rcp.intro.editor.MyPersonEditor"`.

打开 `plugin.xml`，选择 `Extensions` 标签页。添加 `org.eclipse.ui.editors` 扩展，新增一个 `editor`。不要使用模板。修改其 ID 为 `org.eclipse.rcp.intro.editor.MyPersonEditor`，名字可以随便取，例如 `myEditor`，类 `class` 栏中输入类名称 `org.eclipse.rcp.intro.editor.MyPersonEditor`



Tip

Make also sure to select an icon! Otherwise your editor will not work.

提示：确认为编辑器选择了一个图标，否则你的编辑器可能不会工作。

Click on the class hyperlink to create the class. Create the following class. Important is the class variable "ID" which we will later use to reference to this class. The variable ID must match the id defined in the editor extension.

单击 class 链接以创建一个类。注意类中的变量 ID 必须与刚才在创建扩展时设置的 ID 一致，以后将利用此 ID 来引用这个类。其代码如下：

```
package org.eclipse.rcp.intro.editor;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorSite;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.EditorPart;
import org.eclipse.rcp.intro.editor.model.Person;

public class MyPersonEditor extends EditorPart {

    public static final String ID = "org.eclipse.rcp.intro.editor.MyPersonEditor";
```

```

private Person person;

private Text text2;

public MyPersonEditor() {
}

@Override

public void doSave(IProgressMonitor monitor) {
    person.getAddress().setCountry(text2.getText());
}

@Override

public void doSaveAs() {
}

@Override

public void init(IEditorSite site, IEditorInput input)
    throws PartInitException {
    setSite(site);
    setInput(input);
    person = ((MyPersonEditorInput) input).getPerson();
    setPartName(person.getFirstName());
}

@Override

public boolean isDirty() {
    if (person.getAddress().getCountry().equals(text2.getText())) {
        return false;
    }
    return true;
}

@Override

public boolean isSaveAsAllowed() {
    return false;
}

@Override

```

```

public void createPartControl(Composite parent) {

    GridLayout layout = new GridLayout();

    layout.numColumns = 2;

    parent.setLayout(layout);

    Label label1 = new Label(parent, SWT.BORDER);

    label1.setText("Person: ");

    Label personName = new Label(parent, SWT.BORDER);

    personName.setText(person.toString());

    Label label2 = new Label(parent, SWT.BORDER);

    label2.setText("Country");

    text2 = new Text(parent, SWT.BORDER);

    text2.setText(person.getAddress().getCountry());

}

@Override

public void setFocus() {

}

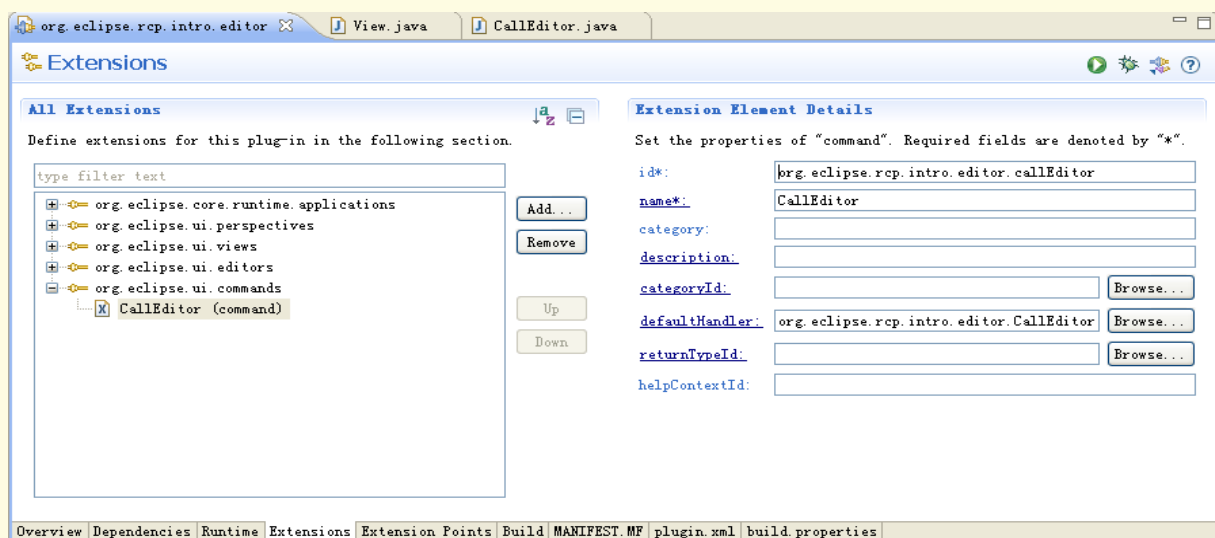
}

```

9.9. CREATING A COMMAND FOR CALLING THE EDITOR 创建编辑器调用命令

Create a command "org.eclipse.rcp.intro.editor.callEditor" with the default handler "org.eclipse.rcp.intro.editor.CallEditor".

创建一个命令"org.eclipse.rcp.intro.editor.callEditor"，将其 defaultHandler 设置为 org.eclipse.rcp.intro.editor.CallEditor



Create the following class "org.eclipse.rcp.intro.editor.CallEditor".

创建下面的类 `org.eclipse.rcp.intro.editor.CallEditor`:

```
package org.eclipse.rcp.intro.editor;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.core.commands.IHandler;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.handlers.HandlerUtil;
import org.eclipse.rcp.intro.editor.View;
import org.eclipse.rcp.intro.editor.MyPersonEditor;
import org.eclipse.rcp.intro.editor.MyPersonEditorInput;
import org.eclipse.rcp.intro.editor.model.Person;

public class CallEditor extends AbstractHandler implements IHandler {

    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {

        // Get the view
        IWorkbenchWindow window = HandlerUtil.getActiveWorkbenchWindow(event);
        IWorkbenchPage page = window.getActivePage();
        View view = (View) page.findView(View.ID);

        // Get the selection
        ISelection selection = view.getSite().getSelectionProvider()
            .getSelection();

        if (selection != null && selection instanceof IStructuredSelection) {
            Object obj = ((IStructuredSelection) selection).getFirstElement();

            // If we had a selection lets open the editor
            if (obj != null) {
```

```

        Person person = (Person) obj;
        MyPersonEditorInput input = new MyPersonEditorInput(person);
        try {
            page.openEditor(input, MyPersonEditor.ID);
        } catch (PartInitException e) {
            System.out.println(e.getStackTrace());
        }
    }
}
return null;
}
}

```

9.10. Calling the editor 调用此 Editor

Add a double-click listener to your view which call the editor.

为视图添加一个双击事件侦听以调用 editor.

```

package org.eclipse.rcp.intro.editor;
import org.eclipse.jface.viewers.DoubleClickEvent;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.handlers.IHandlerService;
import org.eclipse.ui.part.ViewPart;
import org.eclipse.rcp.intro.editor.model.MyModel;
import org.eclipse.rcp.intro.editor.provider.MyContentProvider;
import org.eclipse.rcp.intro.editor.provider.MyLabelProvider;
public class View extends ViewPart {
    public static final String ID = "org.eclipse.rcp.intro.editor.view";
    private TableViewer viewer;
    /**

```

```

    * This is a callback that will allow us to create the viewer and initialize
    * it.
    */
    public void createPartControl(Composite parent) {
        viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL |
SWT.V_SCROLL);
        viewer.setContentProvider(new MyContentProvider(viewer));
        viewer.setLabelProvider(new MyLabelProvider());
        viewer.setInput(new MyModel());
        getSite().setSelectionProvider(viewer);
        // New
        hookDoubleClickCommand();
    }
    // New
    private void hookDoubleClickCommand() {
        viewer.addDoubleClickListener(new IDoubleClickListener() {
            public void doubleClick(DoubleClickEvent event) {
                IHandlerService handlerService = (IHandlerService) getSite()
                    .getService(IHandlerService.class);
                try {
                    handlerService.executeCommand(
"org.eclipse.rcp.intro.editor.callEditor", null);
                } catch (Exception ex) {
                    throw new RuntimeException( "org.eclipse.rcp.intro.editor.callEditor
not found");
                }
            }
        });
    }
    /**
    * Passing the focus request to the viewer's control.
    */

```

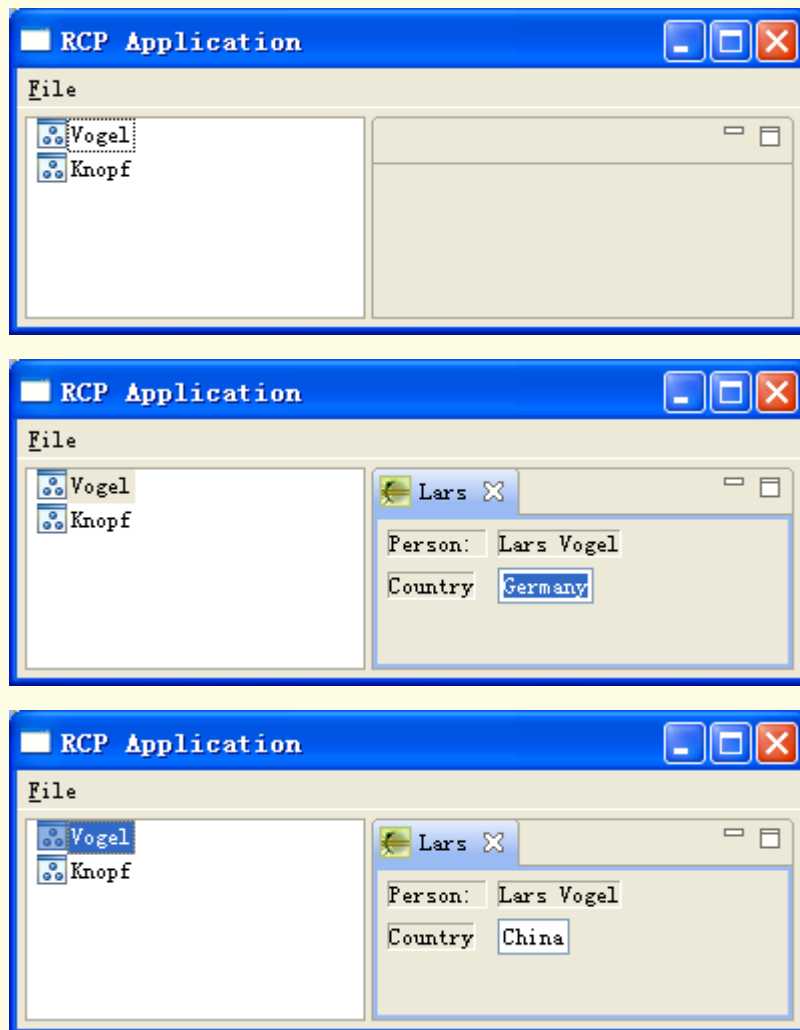
```

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

```

Run the application. If you double-click on an item in the view, the editor should open and the country should get displayed. If you close the editor and re-open it, changes in the address should be displayed.

现在运行应用，如果你双击联系人列表中的名字，就会打开编辑器并显示该联系人的姓名与国家。如果你修改 Country 值，关闭掉编辑器并再次打开它，则会显示改变后的地址。



10. DIALOG 对话框

10.1. OVERVIEW 概览

Eclipse supports several predefined dialogs. For example

Eclipse 支持一些预定义的对话框，例如：

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

Eclipse supports also user defined dialogs. Usually the class TitleAreaDialog is then extended.

Eclipse 也支持用户自定义的对话框。这种对话框通常是由 TitleAreaDialog 类派生的。

10.2. USING STANDARD DIALOGS 使用标准对话框

The following will describe how to use standard dialogs.

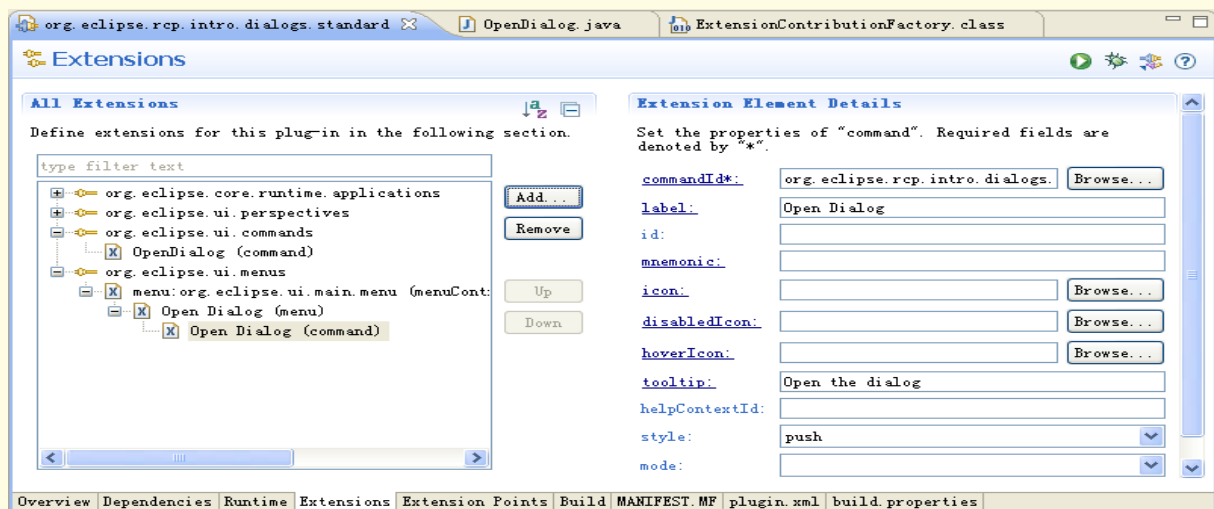
下面先讨论如何使用标准的对话框。

Create a new project "org.eclipse.rcp.intro.dialogs.standard". Use the "Hello RCP" as a template.

利用“Hello RCP”模板创建一个名为“org.eclipse.intro.dialogs.standard”的项目。

Add one command "org.eclipse.rcp.intro.dialogs.standard.openDialog" and create the default handler "org.eclipse.rcp.intro.dialogs.standard.handler.OpenDialog" for the command. Add the command to the menu.

为项目添加一个命令“org.eclipse.rcp.intro.dialogs.standard.openDialog”，在该命令的 defaultHandler 栏输入“org.eclipse.rcp.intro.dialogs.standard.handler.OpenDialog”，并将命令添加到菜单中。（创建菜单、添加命令的方法请参见 6.3）



Define the following code for the default handler.

为 `defaultHandler` 定义如下代码:

```
package org.eclipse.rcp.intro.dialogs.standard.handler;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.swt.graphics.FontData;
import org.eclipse.swt.graphics.RGB;
import org.eclipse.swt.widgets.ColorDialog;
import org.eclipse.swt.widgets.DirectoryDialog;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.FontDialog;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.handlers.HandlerUtil;

public class OpenFileDialog extends AbstractHandler {

    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {

        Shell shell = HandlerUtil.getActiveWorkbenchWindow(event).getShell();

        // File standard dialog
        FileDialog fileDialog = new FileDialog(shell);

        // 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();

        System.out.println(selected);

        // Directly standard selection
```

```

DirectoryDialog dirDialog = new DirectoryDialog(shell);
dirDialog.setText("Select your home directory");
String selectedDir = dirDialog.open();
System.out.println(selectedDir);

// Select Font
FontDialog fontDialog = new FontDialog(shell);
fontDialog.setText("Select your favorite font");
FontData selectedFont = fontDialog.open();
System.out.println(selectedFont);

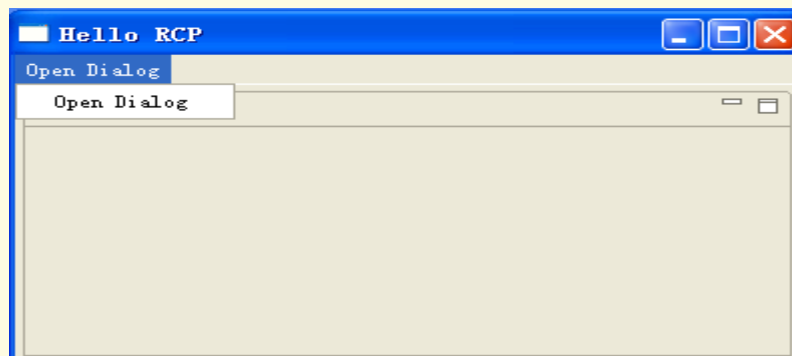
// Select Color
ColorDialog colorDialog = new ColorDialog(shell);
colorDialog.setText("Select your favorite color");
RGB selectedColor = colorDialog.open();
System.out.println(selectedColor);

// Now a few messages
MessageDialog.openConfirm(shell, "Confirm", "Please confirm");
MessageDialog.openError(shell, "Error", "Error occurred");
MessageDialog.openInformation(shell, "Info", "Info for you");
MessageDialog.openQuestion(shell, "Question", "Really, really?");
MessageDialog.openWarning(shell, "Warning", "I warn you");
return null;
}
}

```

Run the application. If you select your command the dialogs will be called after each other.

运行应用，单击刚才创建的菜单命令，标准对话框将会依次显示。



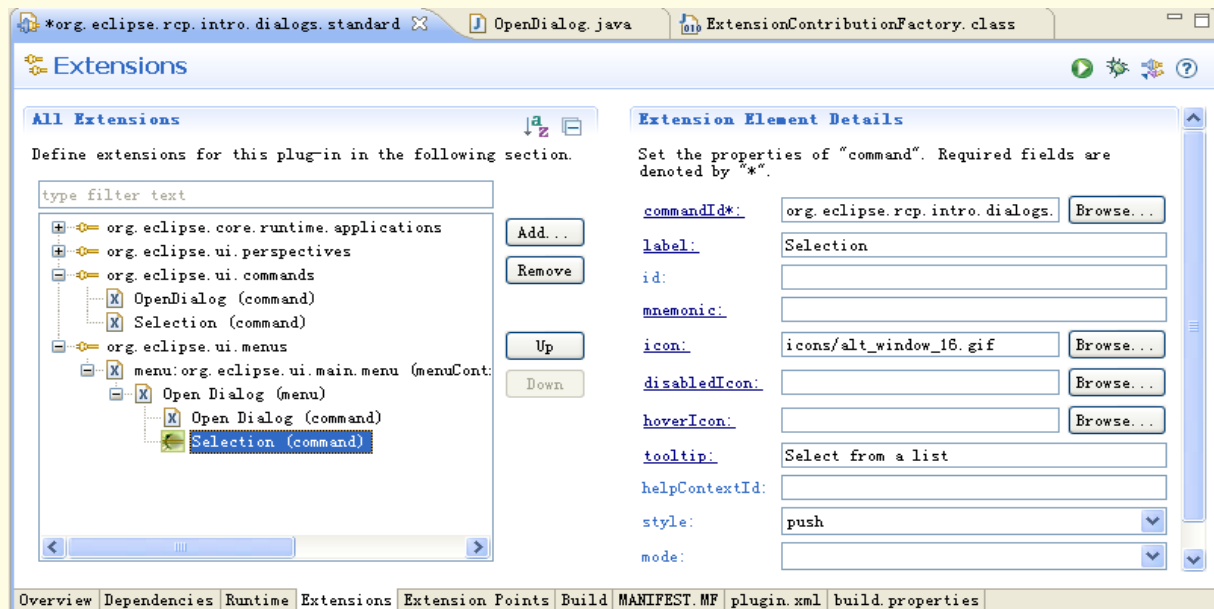
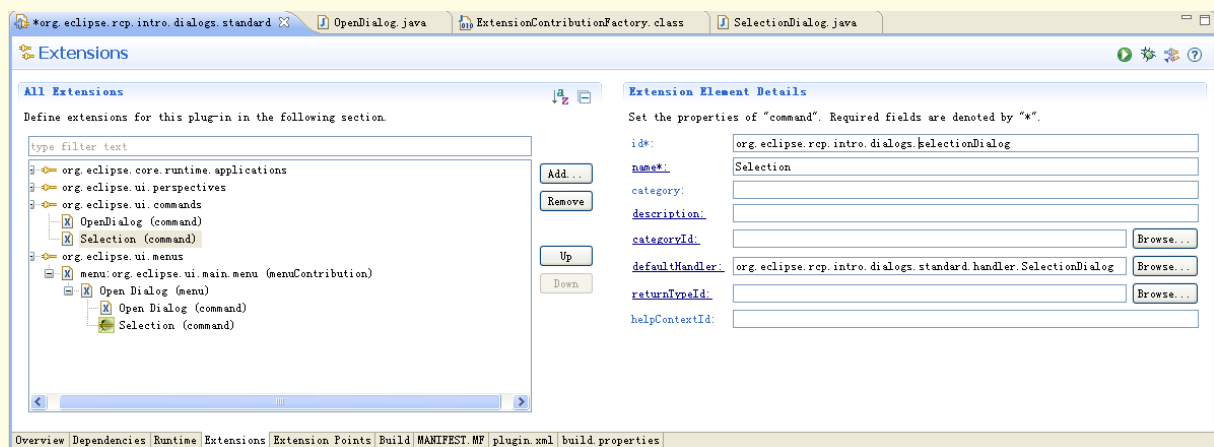
10.3. SELECTION 选择

Eclipse provides the `ElementListSelectionDialog` which allows to select elements from a list.

Eclipse 提供了一个 `ElementListSelectionDialog`，利用它用户可以在一个列表中进行选择。

Add one command "org.eclipse.rcp.intro.dialogs.selectionDialog" and create the defaultHandler "org.eclipse.rcp.intro.dialogs.handler.SelectionDialog" for the command. Add the command to the menu.

在上面的项目中添加一个命令“org.eclipse.rcp.intro.dialogs.selectionDialog”，在命令的 defaultHandler 栏中输入“org.eclipse.rcp.intro.dialogs.handler.SelectionDialog”，并将此命令添加到菜单中。



Define the following code in your handler.

在 handler 包中添加如下类代码：

```
package org.eclipse.rcp.intro.dialogs.standard.handler;

import org.eclipse.core.commands.AbstractHandler;
```

```

import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.dialogs.ElementListSelectionDialog;
import org.eclipse.ui.handlers.HandlerUtil;

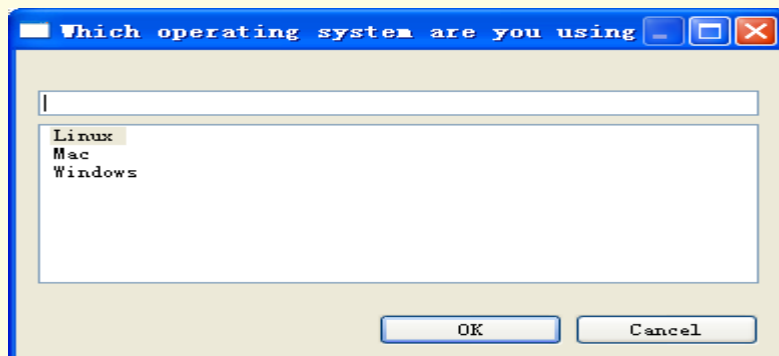
public class SelectionDialog extends AbstractHandler {
    private Object[] result;

    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {
        Shell shell = HandlerUtil.getActiveWorkbenchWindow(event).getShell();
        ElementListSelectionDialog dialog = new ElementListSelectionDialog(
            shell, new LabelProvider());
        dialog.setElements(new String[] { "Linux", "Mac", "Windows" });
        dialog.setTitle("Which operating system are you using");
        dialog.open();
        result = dialog.getResult();
        for (Object s : result) {
            System.out.println(s.toString());
        }
        return null;
    }
}

```

Run the application. If you select your command then the dialog will be displayed.

运行应用，从菜单中选择新创建的命令，将会显示如下图所示的操作系统选择对话框：



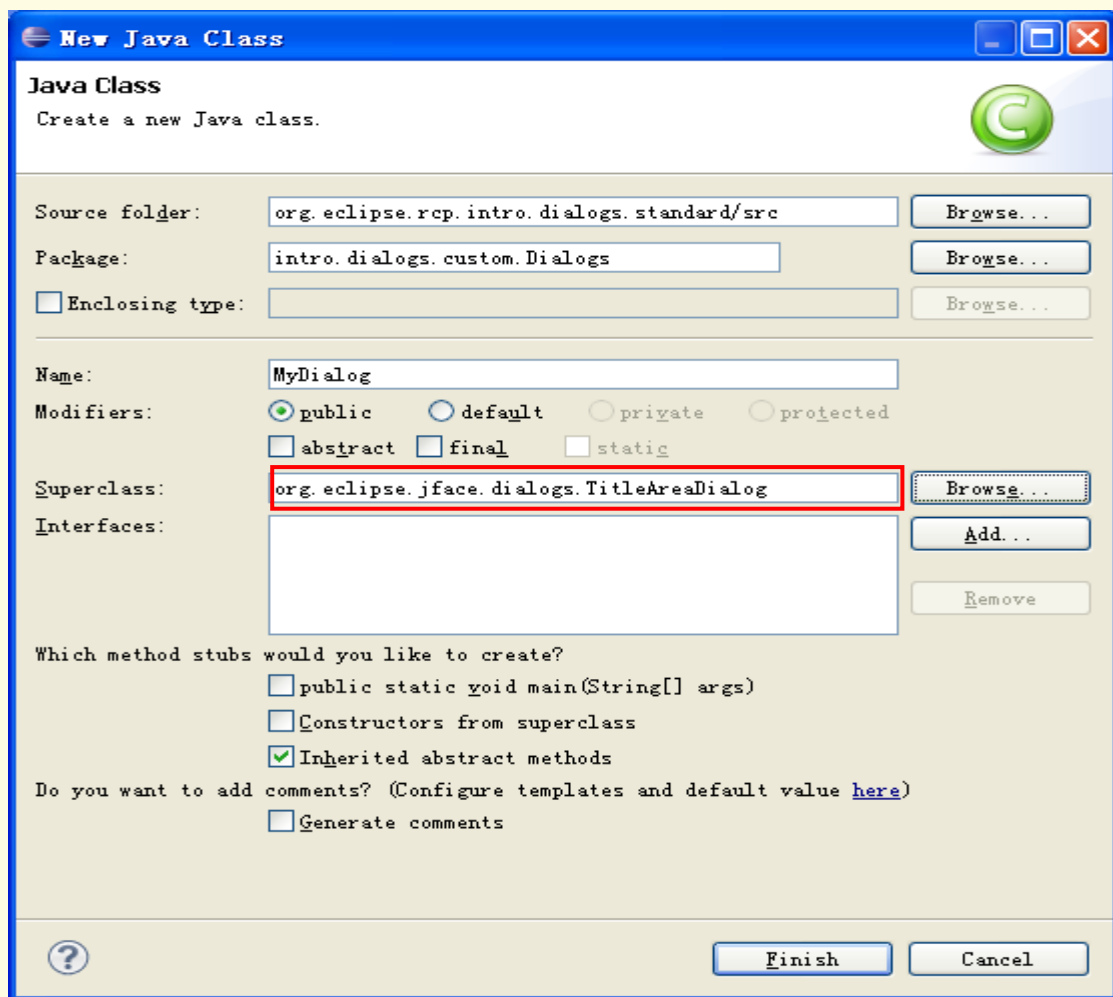
10.4. USER DEFINED DIALOGS 使用自定义对话框

The following will describe how to define your own dialogs and how to use it.

下面介绍自定义对话框的创建与使用。

Create the package "intro.dialogs.customDialogs" and create the following class MyDialog which extends TitleAreaDialog.

在项目中创建一个名为“intro.dialogs.custom.Dialogs”的包，并在此包中创建一个 MyDialog 的类，此类扩展自 TitleAreaDialog。



类代码如下：

```
package intro.dialogs.custom.Dialogs;

import org.eclipse.jface.dialogs.IDialogConstants;
import org.eclipse.jface.dialogs.IMessageProvider;
import org.eclipse.jface.dialogs.TitleAreaDialog;
import org.eclipse.jface.resource.JFaceResources;
import org.eclipse.swt.SWT;
```

```

import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
public class MyDialog extends TitleAreaDialog {
    private Text text1;
    private Text text2;
    public MyDialog(Shell parentShell) {
        super(parentShell);
    }
    protected Control createContents(Composite parent) {
        Control contents = super.createContents(parent);
        // Set the title
        setTitle("This is my first own dialog");
        // Set the message
        setMessage("This is a TitleAreaDialog", IMessageProvider.INFORMATION);
        return contents;
    }
    @Override
    protected Control createDialogArea(Composite parent) {
        // return super.createDialogArea(parent);
        GridLayout layout = new GridLayout();
        layout.numColumns = 2;
        parent.setLayout(layout);
        Label label1 = new Label(parent, SWT.NONE);
        label1.setText("First Name");
    }
}

```

```

        text1 = new Text(parent, SWT.BORDER);

        Label label2 = new Label(parent, SWT.NONE);

        label2.setText("Last Name");

        text2 = new Text(parent, SWT.BORDER);

        return parent;
    }

    @Override
    protected void createButtonsForButtonBar(Composite parent) {

        createOkButton(parent, IDialogConstants.OK_ID, IDialogConstants.OK_LABEL,
true);

        createButton(parent,
IDialogConstants.CANCEL_ID, IDialogConstants.CANCEL_LABEL, false);
    }

    protected Button createOkButton(Composite parent, int id, String label,
        boolean defaultButton) {

        // increment the number of columns in the button bar
        ((GridLayout) parent.getLayout()).numColumns++;

        Button button = new Button(parent, SWT.PUSH);
        button.setText(label);
        button.setFont(JFaceResources.getDialogFont());
        button.setData(new Integer(id));
        button.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent event) {
                if (text2.getText().length() != 0) {
                    buttonPressed(((Integer) event.widget.getData()).intValue());
                } else {
                    setErrorMessage("Please maintain the last name");
                }
            }
        });

        if (defaultButton) {
            Shell shell = parent.getShell();

```

```

        if (shell != null) {
            shell.setDefaultButton(button);
        }
    }

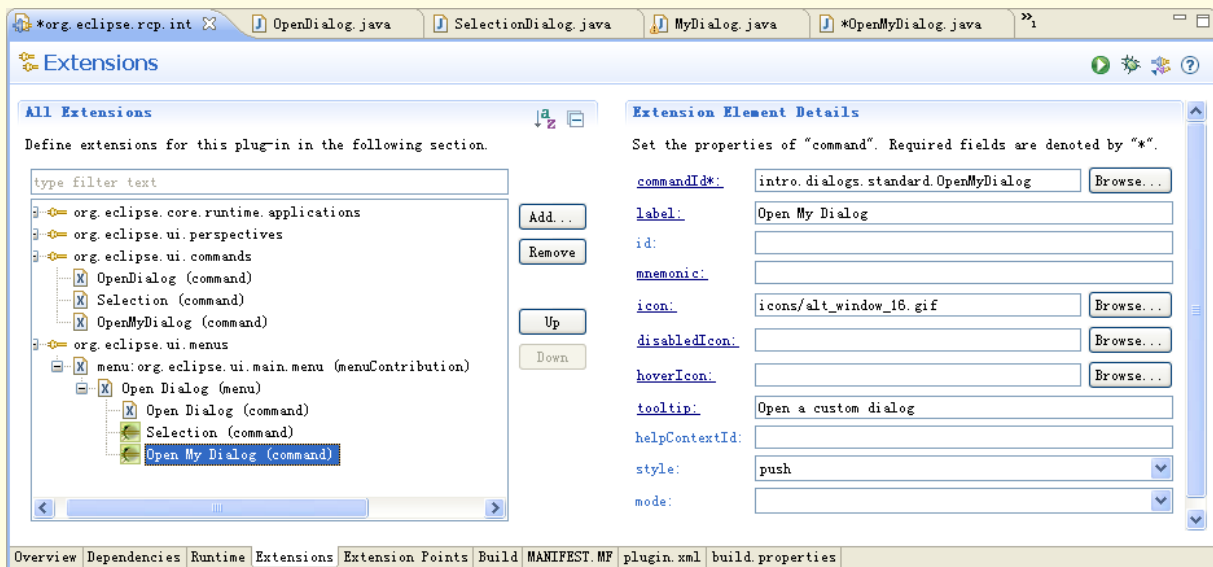
    setButtonLayoutData(button);

    return button;
}
}

```

Add one command "intro.dialogs.custom.openMyDialog" and create the defaultHandler "intro.dialogs.custom.handler.OpenMyDialog" for the command. Add the command to the menu.

添加一个命令 “intro.dialogs.custom.openMyDialog”，设置其 defaultHandler 为 “org.eclipse.rcp.intro.dialogs.standard.handler



Implement the following coding for the handler.

此 handler 的实现代码如下：

```

package org.eclipse.rcp.intro.dialogs.standard.handler;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.ui.handlers.HandlerUtil;
import intro.dialogs.custom.Dialogs.MyDialog;

public class OpenMyDialog extends AbstractHandler {

    @Override

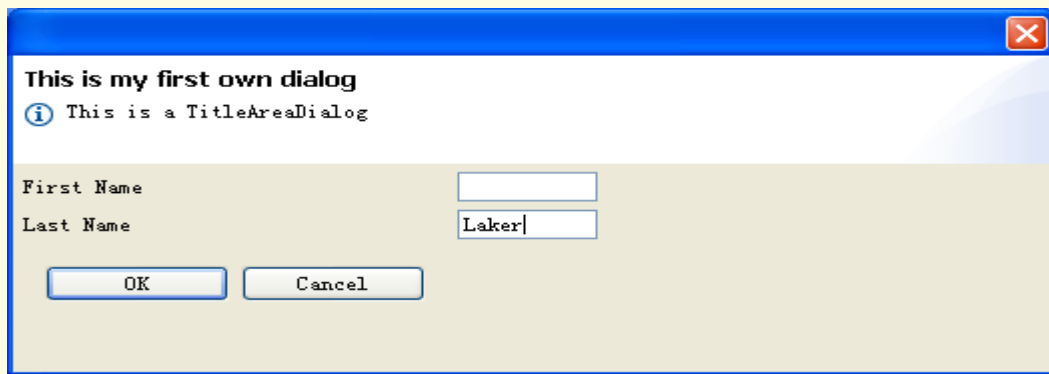
```



```
public Object execute(ExecutionEvent event) throws ExecutionException {  
    MyDialog dialog = new MyDialog(HandlerUtil.getActiveWorkbenchWindow(  
        event).getShell());  
    dialog.open();  
    return null;  
}  
}
```

Run the application. The dialog should open and you must maintain the last name otherwise the dialog will give you an error message.

运行此应用，点击菜单命令 **Open My Dialog** 打开自定义对话框。其运行效果如下图：



如果你在 **Last Name** 中不输入内容点击 **OK**，则会显示一个提示信息，如下图：



11. FIELD ASSIST 字段助手

Field Assist can be used to provide information about the possible inputs and status of a simple field, e.g. text field or combo box.

字段助手可以为简单字段（如文本字段或组合框）提供一个关于输入内容及状态的提示信息。

The `org.eclipse.jface.fieldassist` package provides assistance in two ways. Control decorations allow you to place image decorations to show the status of a field. Content proposal allows to provide a popup that which provides possible choices for this field. user. The following will demonstrate the content proposal.

`org.eclipse.jface.fieldassist` 包以两种方式提供帮助。**Decorations** 控件允许用户放置一个图像说明以显示字段状态，而内容建议则提供了一个弹出消息告诉用户那些可能的选项。下面将介绍内容建议。

Create a new project "`org.eclipse.rcp.intro.fieldAssist`". Use "RCP application with a view" as an example.

In our example the content proposal should get activated via certain keys (".", "#") as well as the key combination "Ctrl+Space".

利用“**RCPApplication with a view**”（带有一个视图的 RCP 应用）模板创建一个名为“**intro.fildassist**”的新项目。在我们的例子中，将通过特定键（“.”与“#”）来激活内容建议。

Change the `View.java` to the following.

修改 `View.java` 文件内容为如下：

```
package intro.filedassist;

import org.eclipse.jface.bindings.keys.KeyStroke;
import org.eclipse.jface.bindings.keys.ParseException;
import org.eclipse.jface.fieldassist.ContentProposalAdapter;
import org.eclipse.jface.fieldassist.SimpleContentProposalProvider;
import org.eclipse.jface.fieldassist.TextContentAdapter;
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 View extends ViewPart {

    public static final String ID = "intro.fieldAssist.view";

    public void createPartControl(Composite parent) {
```

```

Text text = new Text(parent, SWT.BORDER);
text.setText("Example");

// "." and "#" will also activate the content proposals
char[] autoActivationCharacters = new char[] { '.', '#' };

KeyStroke keyStroke;

try {

    keyStroke = KeyStroke.getInstance("Ctrl+Space");
    // assume that myTextControl has already been created in some way
    ContentProposalAdapter adapter = new ContentProposalAdapter(text,
        new TextContentAdapter(),
        new SimpleContentProposalProvider(new String[] {
            "ProposalOne", "ProposalTwo", "ProposalThree" })),
        keyStroke, autoActivationCharacters);
} catch (ParseException e) {
    e.printStackTrace();
}

}

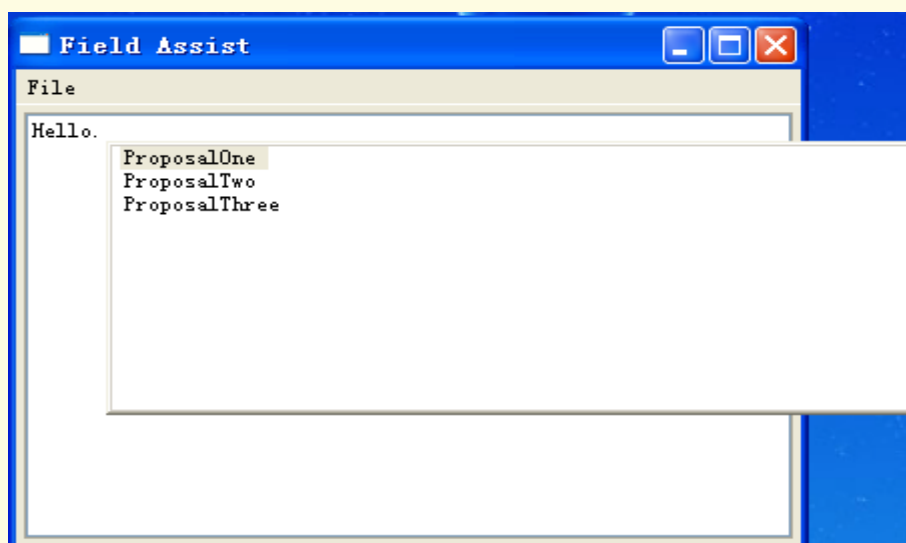
public void setFocus() {
}

}

```

Run the application and check the content proposal works.

运行此应用，输入内容并点击内容建议热键“.”或“#”以测试内容建议是否工作正常。



12. WIZARDS 向导

12.1. OVERVIEW 概览

Wizards provide a flexible ways to systematically gather user input and perform input validation. A wizard guides a user step by step through a process.

向导提供了一种灵活的方式以组织收集用户输入并执行有效性验证。向导将指导用户一步步的完成工作。

Eclipse implements a Wizard via class WizardDialog. The WizardDialog controls the process (Navigation, process bar, area for error and information messages). The Wizard content and is provided by the class Wizard and the pages are provided via class WizardPages.

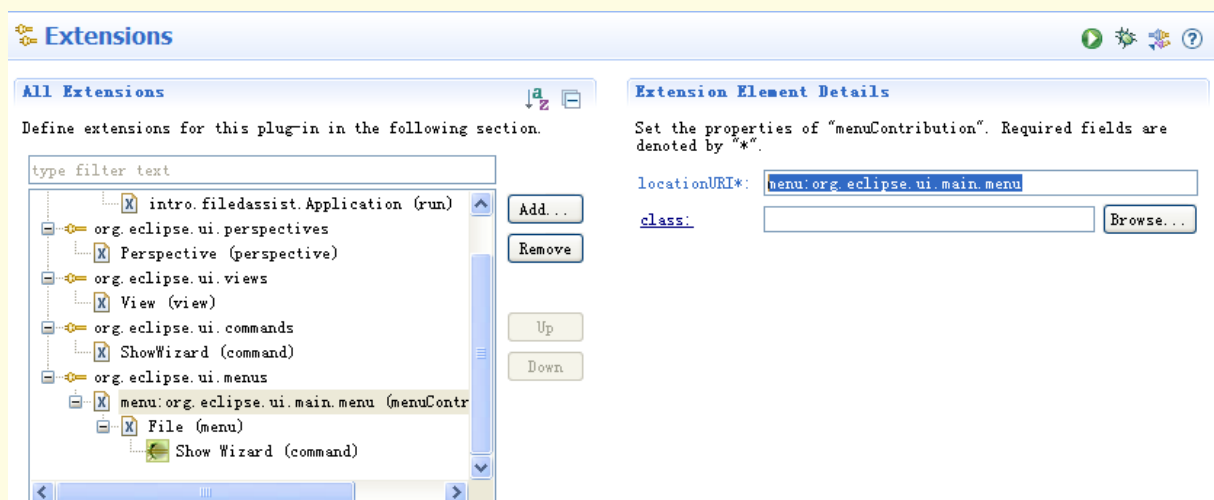
Eclipse 通过类 **WizardDialog** 实现了向导功能。**WizardDialog** 控件控制着进程（导航、进程条、错误及消息区）。向导的内容是由 **Wizard** 类所提供，页面则是通过类 **WizardPages** 提供。

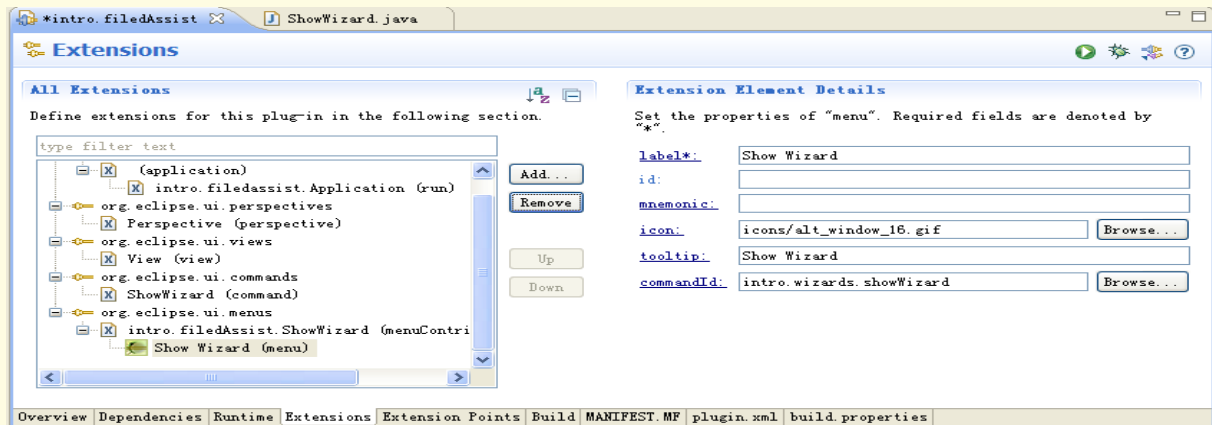
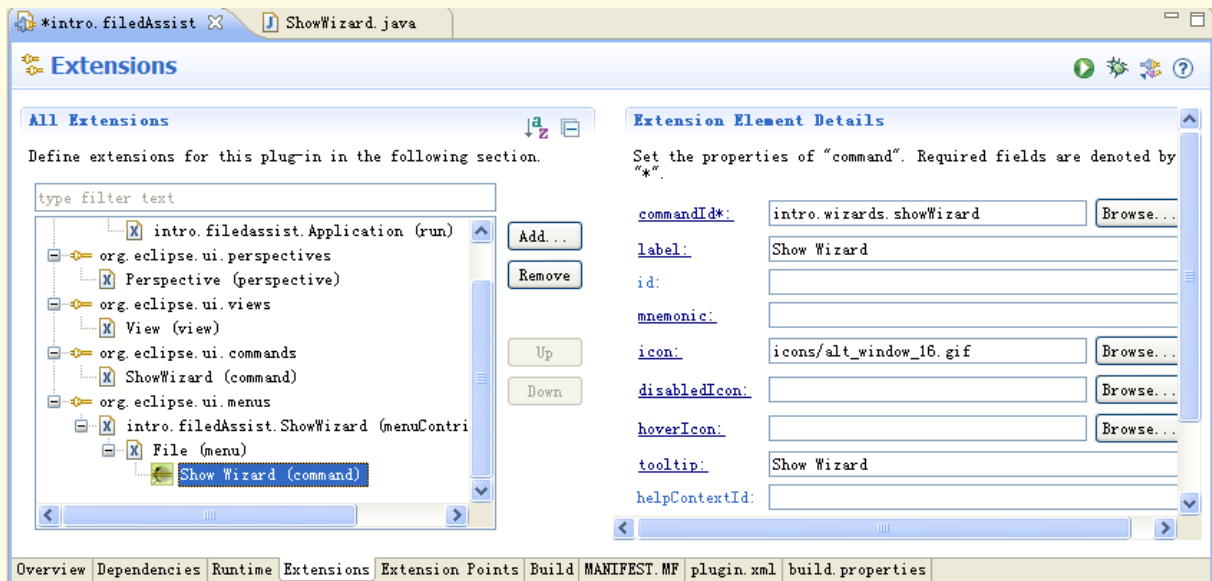
12.2. EXAMPLE 例子

Create a new project "org.eclipse.rcp.intro.wizards". Use the "Hello RCP" as a template.

Create the command "intro.wizards.showWizard" with the defaultHandler "intro.wizards.handler.ShowWizard". Add the command to the menu.

创建一个命令“intro.wizards.showWizard”，设置其 defaultHandler 为“intro.wizards.handler.ShowWizard”，并将其添加到菜单中。（注意新增的 menuContribution 中的 locationURI 必须为 menu:org.eclipse.ui.main.menu，否则菜单无法显示！）





Create a package "intro.wizards.wizard". Create the classes "MyPageOne" and "MyPageTwo" MyWizard which extends org.eclipse.jface.wizard.WizardPage.

Maintain the following code for the classes.

创建名为“intro.wizards.wizard”的包，并在此包中创建名为“MyPageOne”的类及“MyPageTwo”的类，这两个类都继承自 org.eclipse.jface.wizard.WizardPage。

其代码如下：

```
package intro.wizards.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");
    }
    @Override
    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() {
            @Override
            public void keyPressed(KeyEvent e) {
                }
            @Override
            public void keyReleased(KeyEvent e) {
                if (!text1.getText().isEmpty()) {
                    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();

}

}

```

类 **MyPageTwo** 代码如下：

```

package intro.wizards.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.Button;
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 {

    private Text text1;

    private Composite container;

    public MyPageTwo() {

        super("Second Page");

        setTitle("Second Page");

        setDescription("Now this is the second page");
    }
}

```

```

        setControl(text1);
    }
    @Override
    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("Say hello to Fred");
        text1 = new Text(container, SWT.BORDER | SWT.SINGLE);
        text1.setText("");
        text1.addKeyListener(new KeyListener() {
            @Override
            public void keyPressed(KeyEvent e) {
                // TODO Auto-generated method stub
            }
            @Override
            public void keyReleased(KeyEvent e) {
                if (!text1.getText().isEmpty()) {
                    setPageComplete(true);
                }
            }
        });
        GridData gd = new GridData(GridData.FILL_HORIZONTAL);
        text1.setLayoutData(gd);
        Label labelCheck = new Label(container, SWT.NONE);
        labelCheck.setText("This is a check");
        Button check = new Button(container, SWT.CHECK);
        check.setSelection(true);
        // Required to avoid an error in the system
    }
}

```



```

        setControl(container);
        setPageComplete(false);
    }
    public String getText1() {
        return text1.getText();
    }
}

```

Create a new class MyWizard which extends org.eclipse.jface.wizard.Wizard.

创建一个新类 **MyWizard**，该 类继承自 **org.eclipse.jface.wizard.Wizard**。类代码如下：

```

package intro.wizards.wizard;

import org.eclipse.jface.wizard.Wizard;

public class MyWizard extends Wizard {
    private MyPageOne one;
    private MyPageTwo two;
    public MyWizard() {
        super();
        setNeedsProgressMonitor(true);
    }
    @Override
    public void addPages() {
        one = new MyPageOne();
        two = new MyPageTwo();
        addPage(one);
        addPage(two);
    }
    @Override
    public boolean performFinish() {
        // just put the result to the console, imagine here much more
        // intelligent stuff.
        System.out.println(one.getText1());
        System.out.println(two.getText1());
    }
}

```

```

        return true;
    }
}

```

Implement the handler for the command.

实现命令的 handler 如下：

```

package intro.wizards.handler;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.jface.wizard.WizardDialog;
import org.eclipse.ui.handlers.HandlerUtil;
import org.eclipse.rcp.intro.wizards.wizard.MyWizard;

public class ShowWizard extends AbstractHandler {

    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {

        MyWizard wizard = new MyWizard();

        WizardDialog dialog = new WizardDialog(HandlerUtil
            .getActiveShell(event), wizard);

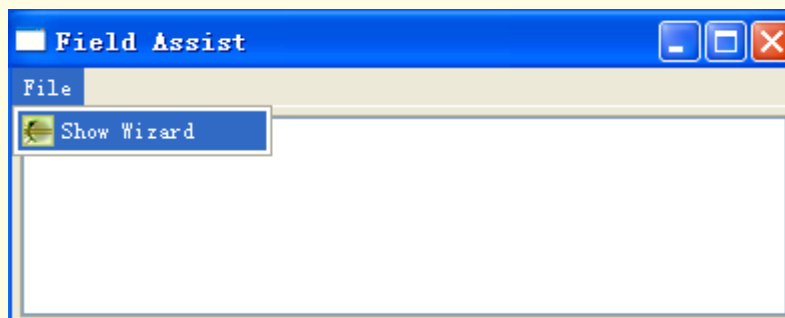
        dialog.open();

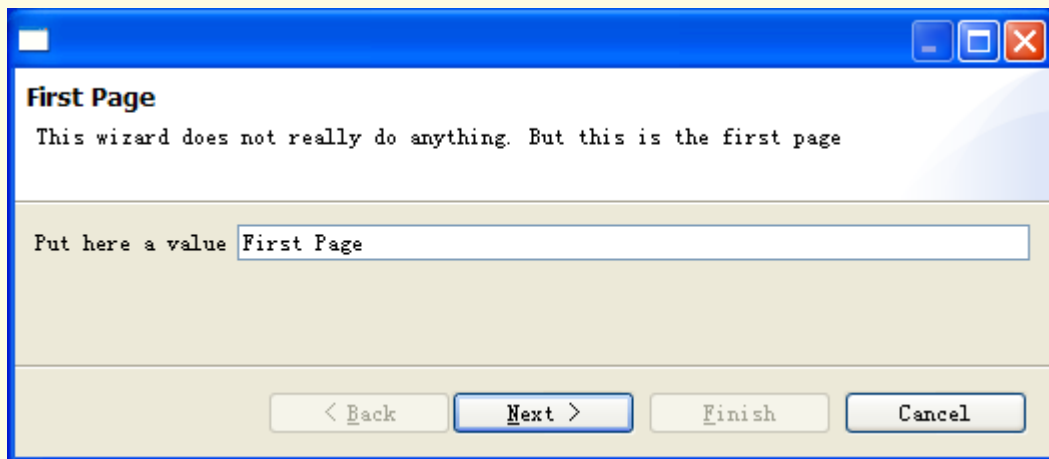
        return null;
    }
}

```

If you run your application and select the menu entry the wizard should get displayed and after pressing "Finish" you should see the result on the console.

如果现在运行应用，并从菜单中选择 **Show Wizard**，将会显示向导。当你在 **First Page** 完成输入，点击 **Next** 后，会显示 **Second Page**，再输入内容后点击 **Finish** 即结束向导。





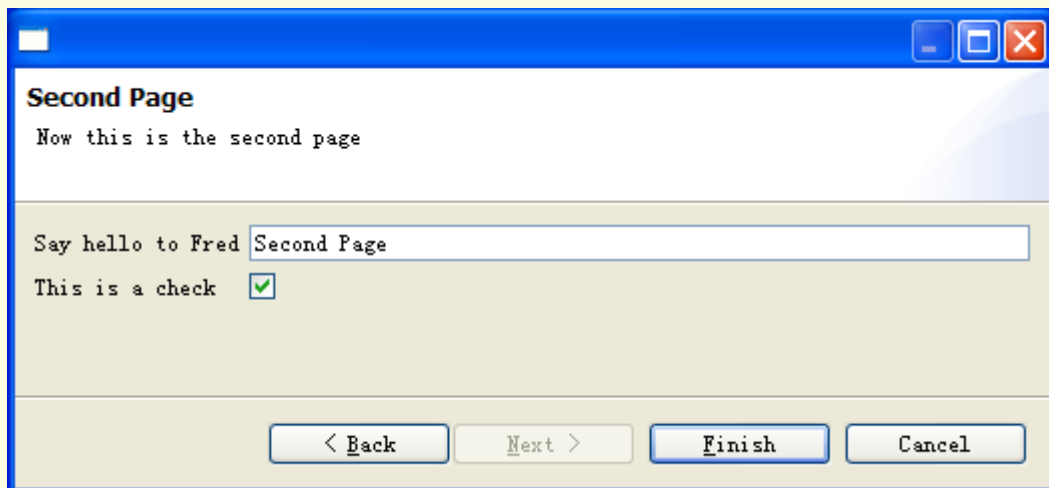
A Windows-style dialog box titled "First Page". The title bar is blue with standard minimize, maximize, and close buttons. The main area has a white header with the title and a light blue gradient background. Below the header, the text "This wizard does not really do anything. But this is the first page" is displayed. A text input field contains the text "First Page", preceded by the label "Put here a value". At the bottom, there are four buttons: "< Back", "Next >", "Finish", and "Cancel". The "Next >" button is highlighted with a blue border.

First Page

This wizard does not really do anything. But this is the first page

Put here a value

< Back Next > Finish Cancel



A Windows-style dialog box titled "Second Page". The title bar is blue with standard minimize, maximize, and close buttons. The main area has a white header with the title and a light blue gradient background. Below the header, the text "Now this is the second page" is displayed. A text input field contains the text "Second Page", preceded by the label "Say hello to Fred". Below the text input field, there is a label "This is a check" followed by a checked checkbox. At the bottom, there are four buttons: "< Back", "Next >", "Finish", and "Cancel". The "Next >" button is highlighted with a blue border.

Second Page

Now this is the second page

Say hello to Fred

This is a check ☒

< Back Next > Finish Cancel

13. ADDING A STATUS LINE 添加状态行

13.1. SETUP STATUS LINE 设置状态行

Create a new RCP project "Statusline". Use the "Hello RCP" as the template.

Go into the ApplicationWorkbenchWindowAdvisor and change method preWindowOpen(). The relevant line in the coding is: "configurer.setShowStatusLine(true); "

打开 ApplicationWorkbenchWindowAdvisor，修改 preWindowOpen() 方法。控制是否显示状态行的代码是 "iconfigurer.setShowStatusLine(true)"，修改后的完整代码如下：

```
package intro.filedassist;

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

public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    public ApplicationWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer
configurer) {

        super(configurer);
    }

    public ActionBarAdvisor createActionBarAdvisor(
        IActionBarConfigurer configurer) {

        return new ApplicationActionBarAdvisor(configurer);
    }

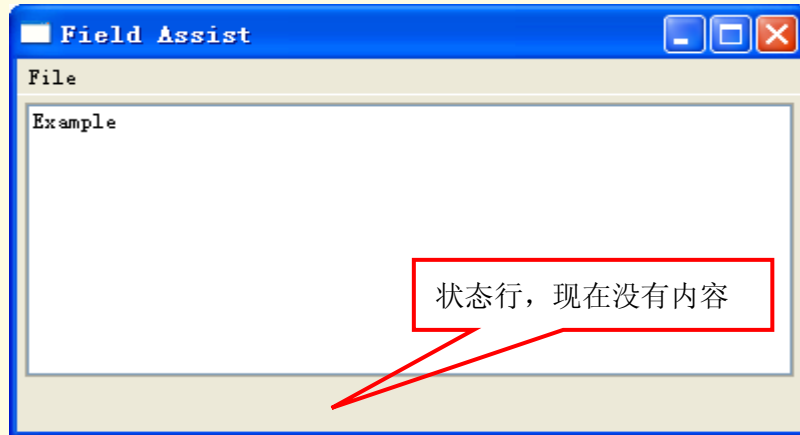
    public void preWindowOpen() {

        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowCoolBar(false);
        configurer.setShowStatusLine(false);
        configurer.setTitle("Field Assist");
        configurer.setShowStatusLine(true);
    }
}
```

}

If you run the application you should already see a status line. At this point the status line does not contain text.

如果运行应用，可以看到状态行已经显示，只是状态行上现在没有任何内容。



13.2. SHARED STATUS LINE 共享状态行

The shared message area can be used by all parts of the application to write messages to this area.

The following will explain how you can add a status line to your application. It will also indicate how the status line can be filled from a view or editor.

通过共享消息区，可以使得应用的所有单元将消息写到该区域。下面我们来介绍如何为应用添加状态行。在此还将介绍如何在视图或编辑器中填充状态行。

Tip

As the whole RCP application has access to the information in the shared status line the information in the shared status line might be overwritten.

提示：由于整个 RCP 应用都会访问共享状态行上的消息，因此状态行上的显示消息可能会被覆盖。

Lets now fill the status line. We can do this for example via the `postWindowOpen()` method in `ApplicationWorkbenchWindowAdvisor`. You have two options to add this method:

现在我们来填充状态行。可以在 `ApplicationWorkbenchWindowAdvisor` 中通过 `postWindowOpen()` 方法来做到这一点。方法如下：

You can either just paste this method into the class.

你可以将此方法粘贴到需要的类中；

Or use the menu "Source" -> Implement / Overwrite Methods and then select "`postWindowOpen()`" to create the method and then adjust it to your needs.

或者你可以利用菜单 **Source→Implement/Overwrite Methods**，然后选择 **postWindowOpen()** 来创建此方法并添加需要的代码。

```
package statusline;

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

public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

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

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

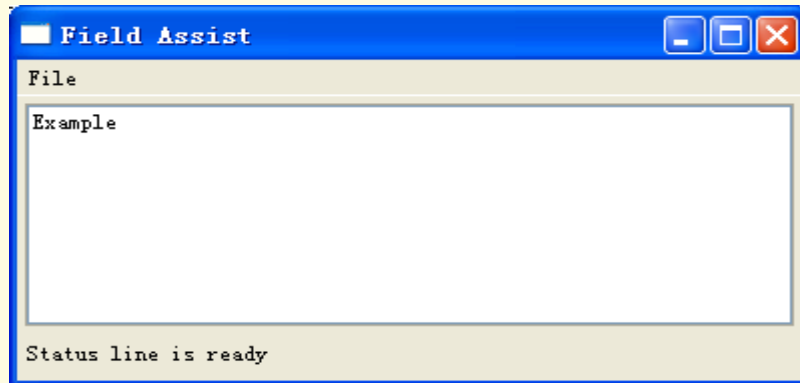
    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowCoolBar(false);
        configurer.setShowStatusLine(true);
        configurer.setTitle("Hello RCP");
    }

    // This is the new method
    @Override
    public void postWindowOpen() {
        IStatusLineManager statusline = getWindowConfigurer()
            .getActionBarConfigurer().getStatusLineManager();
        statusline.setMessage(null, "Status line is ready");
    }
}
```

```
}  
}
```

Run your application. You should now see the following.

再次运行，其结果如下：



Add a view with the ID "StatusLineView" to your application. You can access the status line, via the following.

为应用添加一个 ID 为 “StatusLineView” 的视图，通过如下代码，你也可以访问状态行：

```
IActionBars bars = getViewSite().getActionBars();  
bars.getStatusLineManager().setMessage("Hello");
```

In our example we create a button which set the status line.

在这个例子中，我们还创建了一个按钮以设置状态行。

```
package intro.fileassist;  
  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.events.SelectionEvent;  
import org.eclipse.swt.events.SelectionListener;  
import org.eclipse.swt.widgets.Button;  
import org.eclipse.swt.widgets.Composite;  
import org.eclipse.ui.IActionBars;  
import org.eclipse.ui.part.ViewPart;  
  
public class StatusLineView extends ViewPart {  
    public static final String ID="intro.fileAssist.StatusLineView";  
    boolean pressed = false;  
    public StatusLineView() {  
    }  
}
```

```

@Override

public void createPartControl(Composite parent) {

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

        public void widgetSelected(SelectionEvent e) {

            IActionBars bars = getViewSite().getActionBars();
            if (pressed) {

                bars.getStatusLineManager().setMessage("I would
                    like to say hello to you.");
            } else {

                bars.getStatusLineManager().setMessage("Thank you for using me");
            }
            pressed = !pressed;
        }

        public void widgetDefaultSelected(SelectionEvent e) {

        }

    });
}

@Override

public void setFocus() {

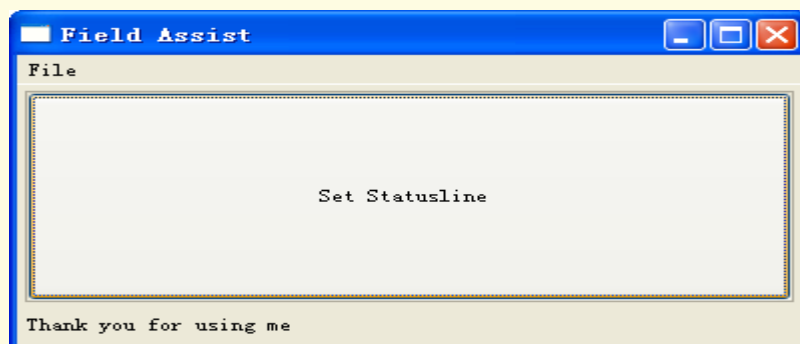
}

}

```

If you run it the result should look like the following.

如果运行，其结果应该如下。单击 **Set Status Line** 可以改变状态行的内容：



Tip

From an editor you can access the status line via the following:

提示：在编辑器中可以通过如下代码来访问状态行：

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

14. PERSPECTIVES 透视图

Perspectives group together and organize UI element that relate to a specific task.

透视图将与特定任务有关的 UI 元素进行分组、管理。

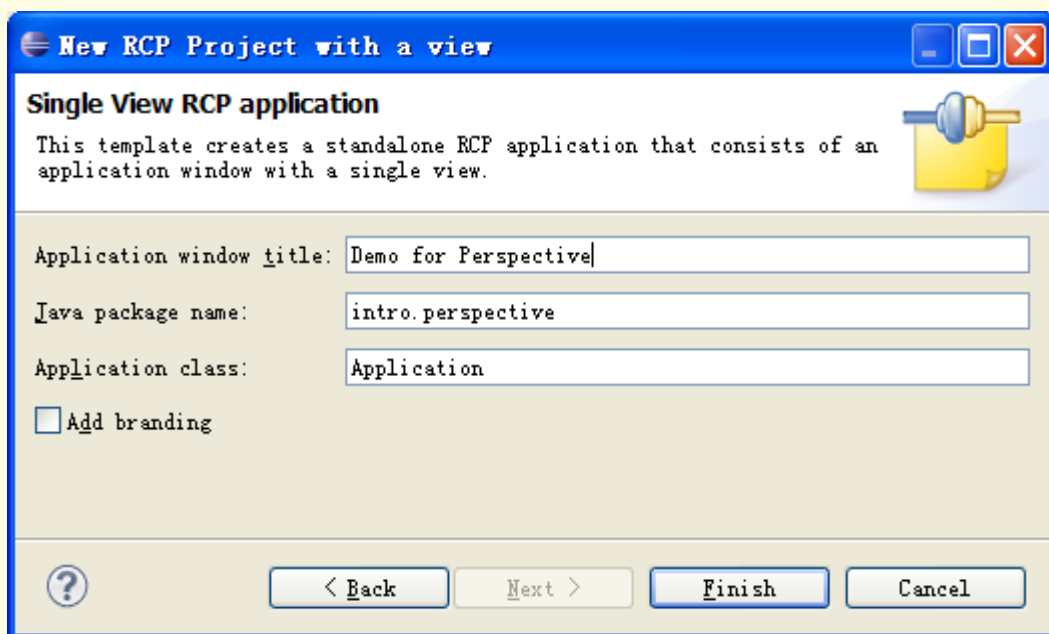
Eclipse RCP allows you to add easily perspectives to your application. The following presents an example

Eclipse RCP 可以很方便的将透视图添加到用户应用。下面用一个例子进行介绍。

14.1. ADDING A PERSPECTIVE TO YOUR APPLICATION 为应用添加透视图

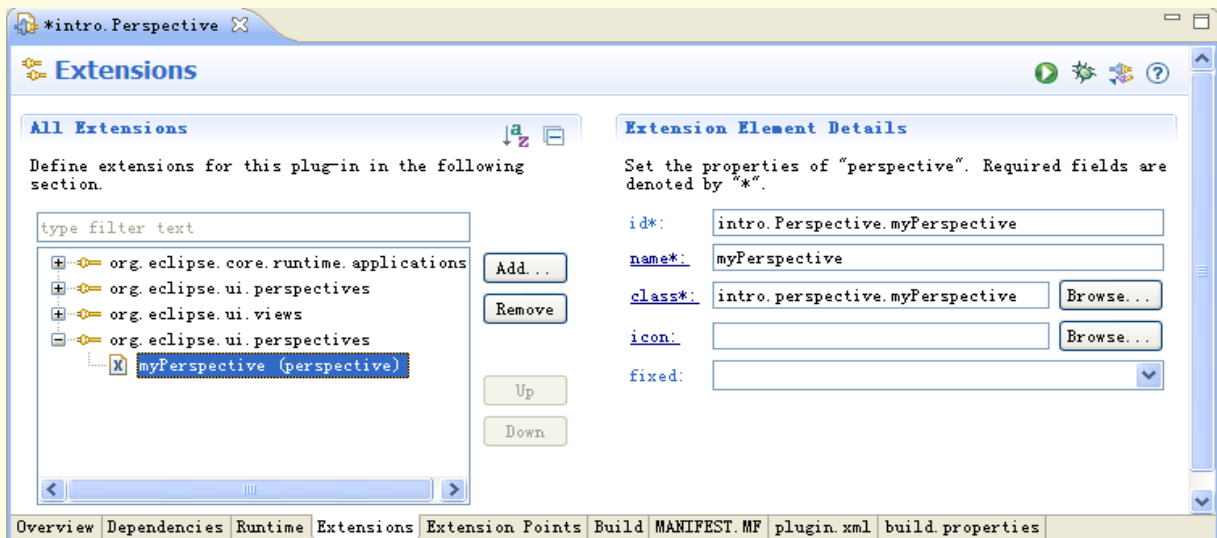
Create a new RCP project called "intro.Perspective". Use the "RCP application with a view" as a template.

利用 RCP application with a view 模板创建一个新的名为“intro.Perspective”的项目。



In plugin.xml add a new perspective extension point.

打开 plugin.xml 文件，选择 Extensions 标签页，新添加一个 org.eclipse.ui.perspectives 扩展点，在其下添加一个新扩展：

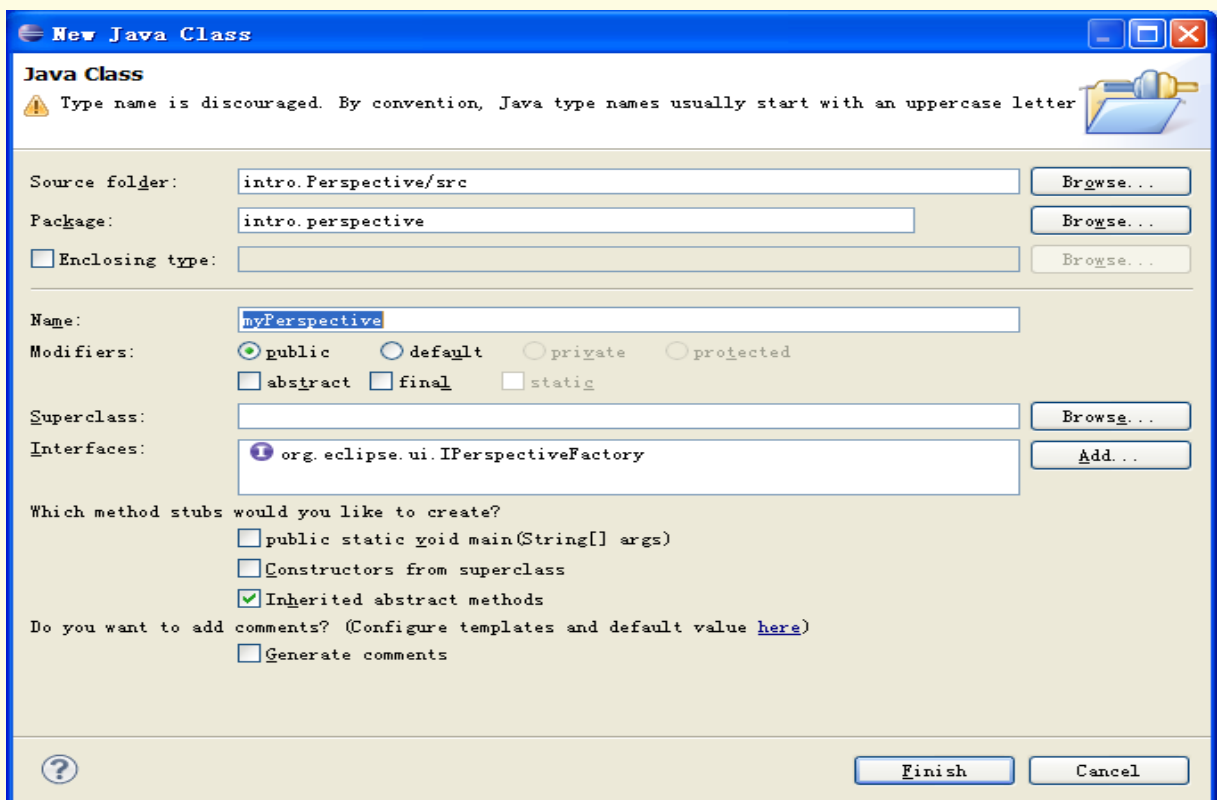


Give the perspective the id "intro.Perspective.myPerspective " and the name "myPerspective". The name is the name under which the perspective will be visible. Change the class name to intro.Perspective.myPerspective.myPerspective".

在此扩展的 ID 栏中输入 “intro.Perspective.myPerspective”，在其 name(名称)栏中输入 “myPerspective”，将类名改 “intro.Perspective.myPerspective”：

Click on the "class*" link to create the class.

单击 “class*” 链接创建类：



The method createInitialLayout() in your new class is responsible for creating the new perspective. We re-use the existing view in the coding.

这个新类中的 `createInitialLayout()` 方法就负责创建新透视图。我们修改代码利用现有的视图;

```
package intro.perspective;

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

public class myPerspective implements IPerspectiveFactory {

    @Override

    public void createInitialLayout(IPageLayout layout) {

        // TODO Auto-generated method stub

        String editorArea = layout.getEditorArea();

        layout.setEditorAreaVisible(true);

        layout.setFixed(false);


        layout.addStandaloneView(View.ID, false, IPageLayout.LEFT, 1.0f, editorArea);

    }

}
```

Now the perspective is define but not yet reachable view the application.

现在透视图已经定义好了，但还不能管理视图。

14.2. SELECT THE PERSPECTIVE 选择透视图

14.2.1. Select perspective via the toolbar / coolbar 通过工具条/快捷键选择透视图

You can activate the switch between perspectives the `ApplicationWorkbenchWindowAdvisor` in method `preWindowOpen()` with `configurer.setShowPerspectiveBar(true)`;

你可以利用 `ApplicationWorkbenchWindowAdvisor` 中的 `preWindowOpen()` 方法来激活透视图切换开关。

```
package intro.perspective;

import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.swt.graphics.Point;
import org.eclipse.ui.IWorkbenchPreferenceConstants;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
```

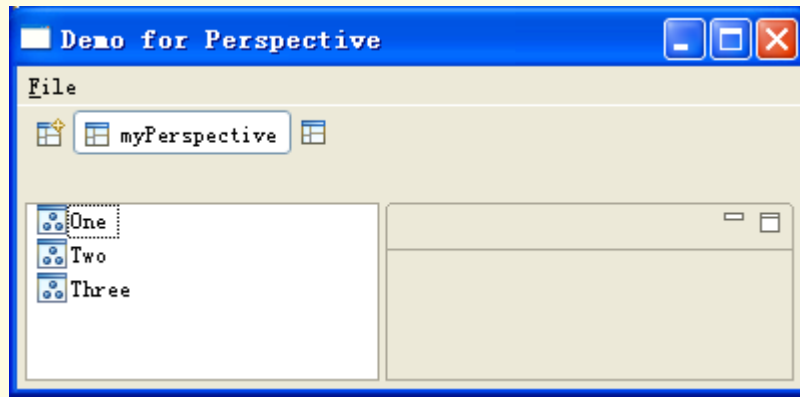
```

import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;
public class ApplicationWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {
    public ApplicationWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer
configurer) {
        super(configurer);
    }
    public ActionBarAdvisor createActionBarAdvisor(
        IActionBarConfigurer configurer) {
        return new ApplicationActionBarAdvisor(configurer);
    }
    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setInitialSize(new Point(400, 300));
        configurer.setShowCoolBar(false);
        configurer.setShowStatusLine(false);
        configurer.setTitle("Demo for Perspective");
        configurer.setShowPerspectiveBar(true);
        // Set the preference toolbar to the left place
        // If other menus exists then this will be on the left of them
        IPreferenceStore apiStore = PlatformUI.getPreferenceStore();
        apiStore.setValue(IWorkbenchPreferenceConstants.DOCK_PERSPECTIVE_BAR,
"TOP_LEFT");
    }
}

```

You should now be able to select your perspective interactively.

现在你可以利用工具条来交互选择不同的透视图了。



14.2.2. Select perspective via the menu 通过菜单选择透视图

You can re-use the Eclipse perspective switch in a menu via the following standard command "org.eclipse.ui.perspectives.showPerspective".

你可以通过如下标准命令 `org.eclipse.ui.perspectives.showPerspective` 在菜单中重用 Eclipse 透视图切换：

See [Using Eclipse Commands](#) for details on using Eclipse standard commands.

有关使用 Eclipse 标准命令的内容请参看使用 Eclipse 命令。

15. PRODUCTS AND BRANDING 产品与商标

15.1. PRODUCT CONFIGURATION 产品配置

So far your RCP applications can only get started from Eclipse itself. We will now create a stand-alone program, which is called "product".

到目前为止，你的 RCP 应用还只能从 Eclipse 中启动。现在我们要创建一个能完全独立运行的程序——即通常称为产品的应用。

Tip

In Eclipse terms a product is everything that goes with your application, including all the plugs-ins it depends on, a program to run the application and any branding (icons, etc.) of the application.

提示：用 Eclipse 的术语来说，产品就是与应用有关的所有东西，包括所依赖的全部插件，运行应用的程序以及应用的任何商标（图标，等等）。

To create a product you need a product configuration. This configuration file contains all the information about required packages, configuration file, etc. The configuration file is used for exporting / creating your product.

要创建产品，需要有一个产品配置。这个配置文件包含了关于所需的包、配置文件等等的全部信息。此配置文件用于导出/创建你的产品。

15.2. CREATE A PROJECT 创建项目

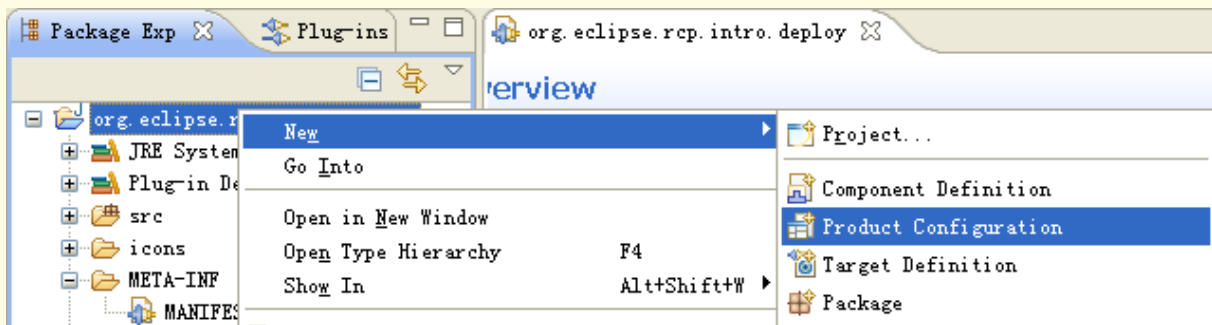
Create a new RCP project "org.eclipse.rcp.intro.deploy" based on the "RCP application with a view" template. Validate that this plugin runs without problems.

现在利用“RCP application with a view”模板创建一个名为“org.eclipse.rcp.intro.deploy”的项目，请检查运行没有任何问题。

15.3. CREATE YOUR PRODUCT CONFIGURATION 创建你的产品配置

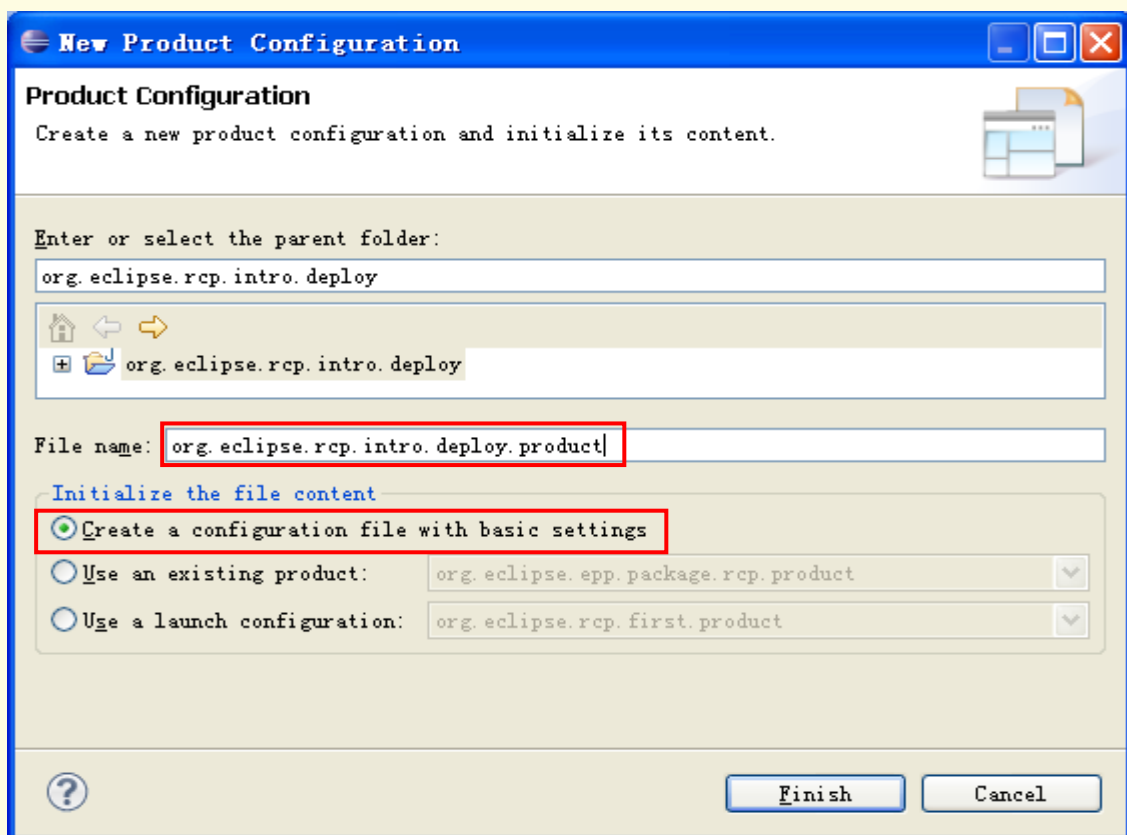
Right-click on your project and select New -> Product Configuration.

用右键点击你的项目，从弹出菜单选择 New→Product Configuration



Name your product configuration "org.eclipse.rcp.intro.deploy.product". Select "Create a configuration file with basis settings". Press finish.

为你的产品配置取名为 `org.eclipse.rcp.intro.deploy.product`，并选择 **Create a configuration file with basis settings**（用基本选项创建一个配置文件），单击 **Finish**。



This will create your product configuration file "org.eclipse.rcp.intro.deploy.product" and open the "Overview" Tab of your product configuration.

很快系统会为你创建名为 `org.eclipse.rcp.intro.deploy.product` 的产品配置文件，并打开产品配置的 **Overview**（概览）标签页。

15.4. MAINTAIN THE OVERVIEW TAB 概览页维护

The overview tab for the product configuration file "org.eclipse.rcp.intro.deploy.product" should still be open.

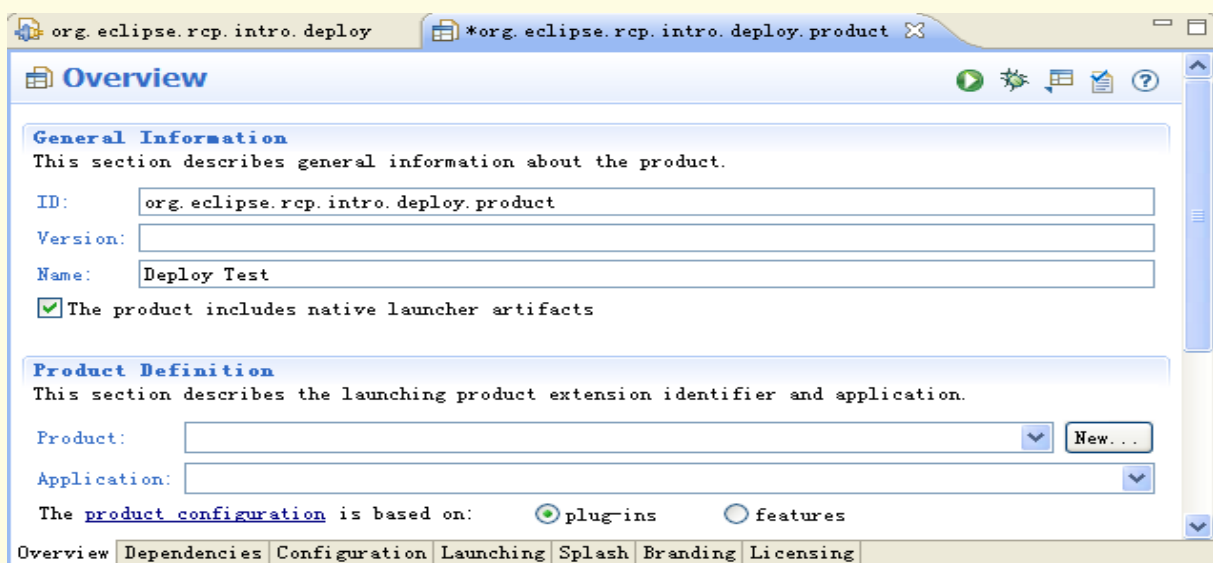
Use the id "org.eclipse.rcp.intro.deploy.product" and name "DeployTest" for your product. The name of your product is the name which will be displayed in the title of the window.

现在，产品配置文件的概览页应仍处于打开状态。请在 ID 栏中输入 `org.eclipse.rcp.intro.deploy.product`，name（名称）栏输入 `Deploy Test`。此产品名称将作为窗口的标题显示。

Tip

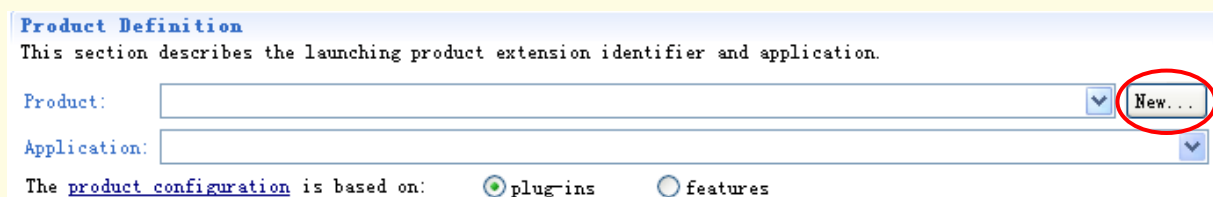
You can change this default name in class `ApplicationWorkbenchWindowAdvisor` in the method `preWindowOpen()` via the `configurer.setTitle("New title");`

提示：你可以在类 `ApplicationWorkbenchWindowAdvisor` 的 `preWindowOpen()` 方法中通过 `configurer.setTitle("新标题")` 修改此缺省名称。



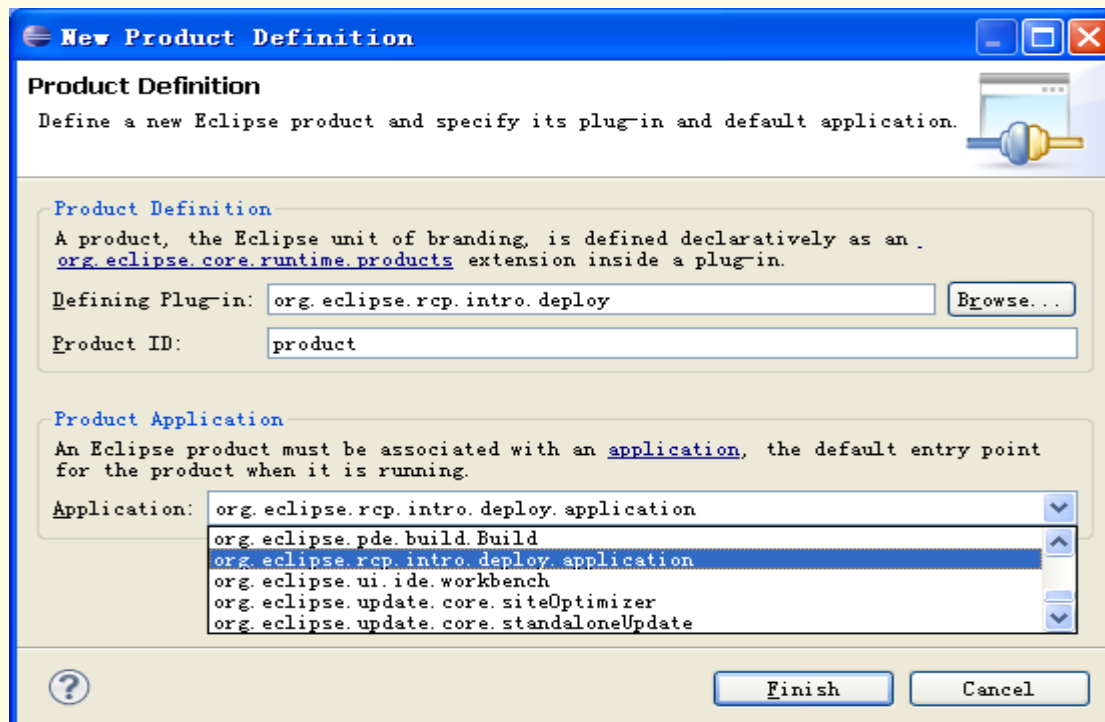
Press New under Product Definition.

单击 **Production Dification**（产品定义）下面的 **New...** 按钮



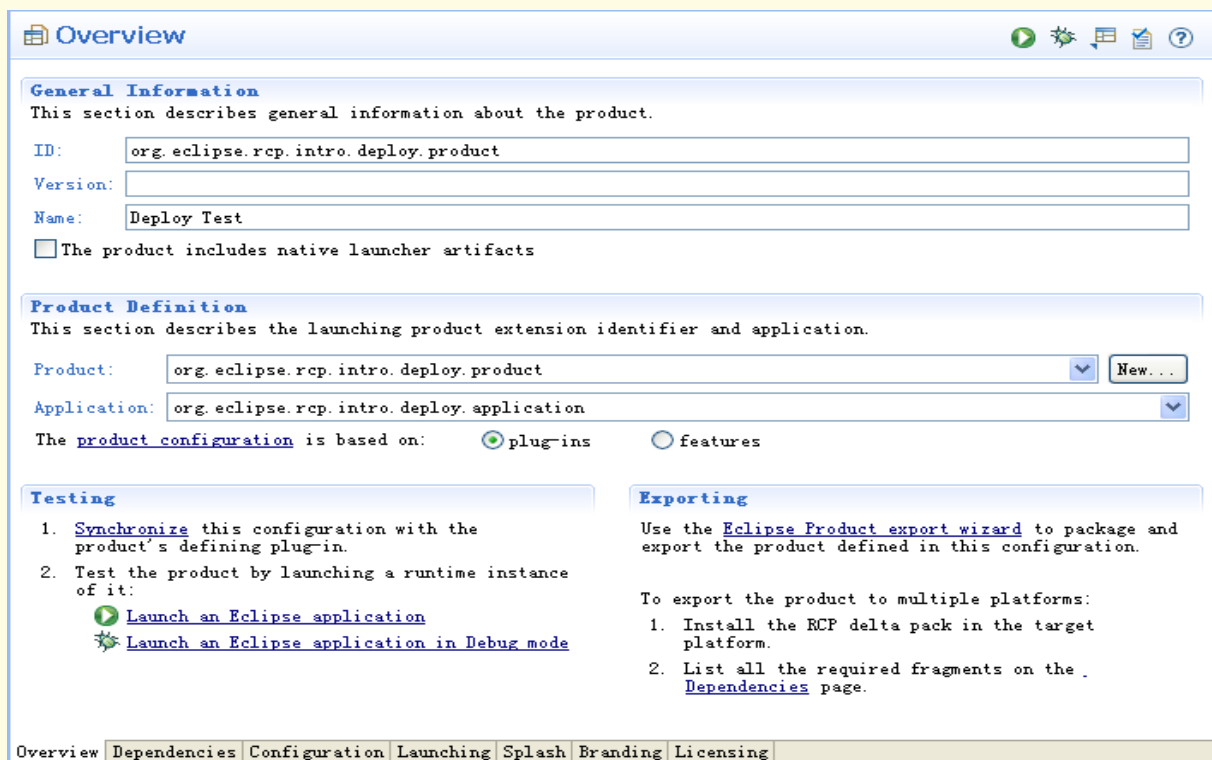
Maintain the following values. Select the application of your plugin "org.eclipse.rcp.intro.deploy.application". Press finish.

修改下列值，从 **Application** 列表中选择你的应用 `org.eclipse.rcp.intro.deploy.application`，单击 **Finish**



As a result the "Product Definition" part should now be filled.

设置结果会显示在 **Products Definition**（产品定义）部分：



Tip

The product identifier is stored as extension in the plugin in which you maintain your product configuration file. Open your plugin.xml and have a look at the "org.eclipse.core.runtime.products" extension point.

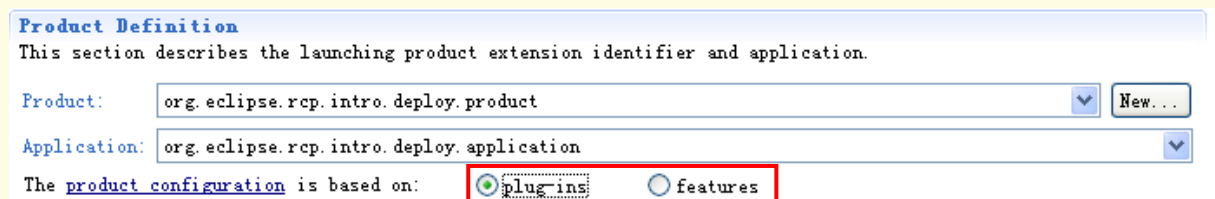
提示：产品的标识符保存在插件扩展中。你可以在此对你的产品配置文件进行维护。打开 `plugin.xml` 文件，看看 `org.eclipse.core.runtime.products` 扩展点的内容。

15.5. DEPENDENCIES 依赖性

Tip

A product can either be based on plugins or features. This setting is done on the overview tab.

提示：产品可以是基于插件，也可以是基于特征。此属性通过配置文件的概览页进行设置：



See [Eclipse Feature Project - Tutorial](#) for details on how to create a feature project.

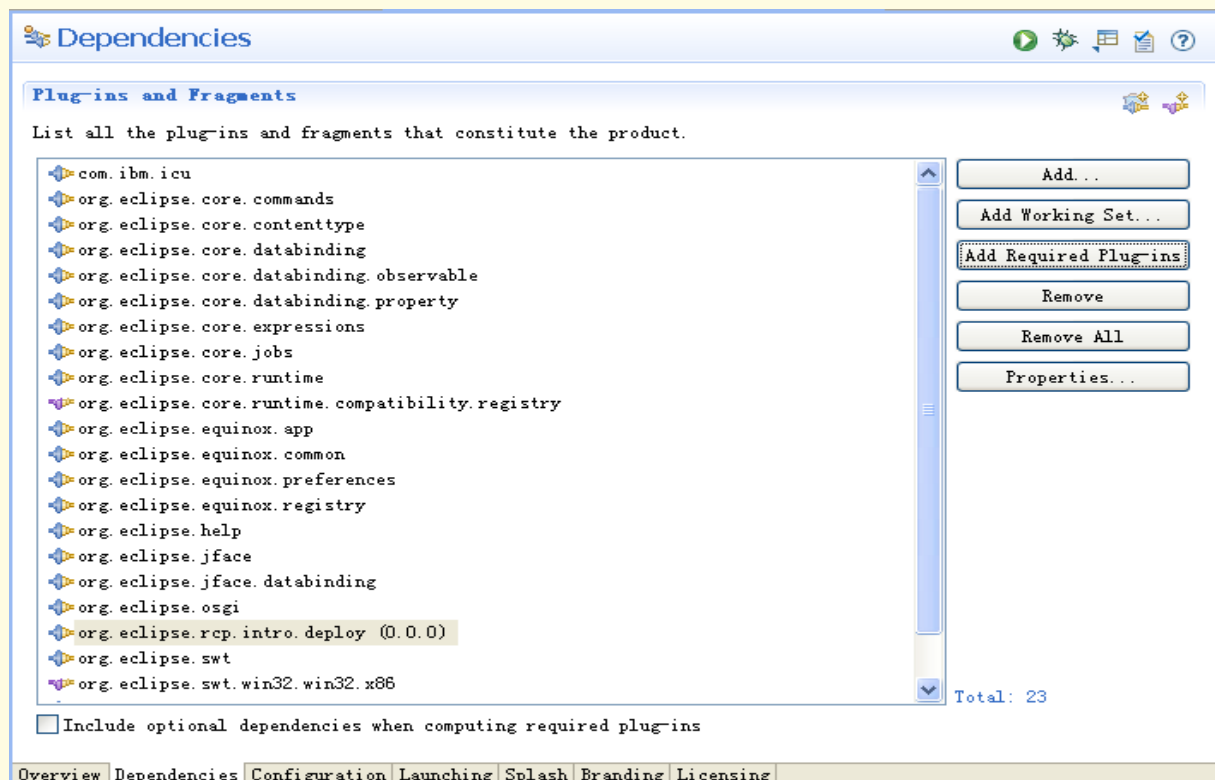
创建特征项目的方法请参看 [Eclipse 特征项目](#)。

In this example we are using a plugin based product configuration.

在此例子中我们使用基于插件的项目配置。

Switch to the "Dependencies" Tab and click "Add". Select the plugin "org.eclipse.rcp.intro.deploy". Press "Add Required Plug-ins". Save.

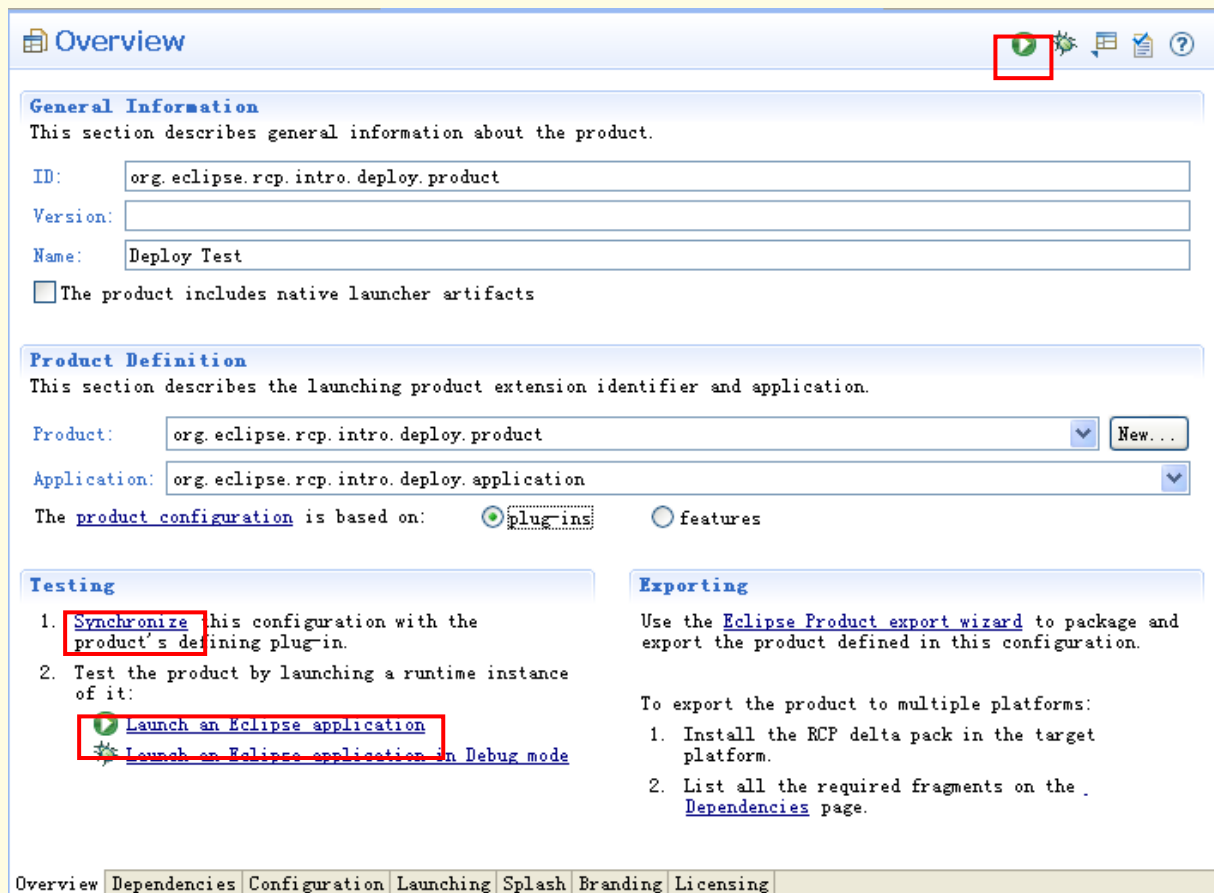
切换到 **Dependencies**（依赖性）标签页，单击 **Add**，选择插件 `org.eclipse.rcp.intro.deploy`，再单击 **Add Required Plug-ins**（添加所需插件），如图：



15.6. LAUNCH YOUR PRODUCT 启动你的应用

On the overview tab press synchronize and then press the launch application (or the big run button).

在 Overview 标签页上单击 Synchronize，再单击 Launch an Eclipse application 链接（或点击按钮）：



Tip

Synchronize will align your product configuration with the other existing configuration files. .

提示：通过同步将使得你的产品配置文件与其它已有配置文件同步。

Tip

If you receive the error "Dependent plug-in could not be loaded" or "Application could not be found" then you may have to adjust your Dependencies configuration. Open the Dependencies and press "Add required Plug-ins".

提示：如果你看到了“Dependent plug-in could not be loaded”（所依赖插件无法加载）或者一个“Application could not be found”（无法找到应用）的错误消息，你可能需要调整启动配置文件。打开 Dependencies（依赖性）页面并单击“Add required Plug-ins”（添加所需插件）。

Tip

The launch configuration for your product is automatically created and updated based on your product configuration, e.g. the selected plugins. Changes in the launch configuration will not automatically update the product configuration.

提示：产品的启动配置是自动创建的，并会根据你的产品配置（例如所选的插件）更新。在启动配置中所做的修改并不会自动更新产品配置。

15.7. SPLASH SCREEN 溅射屏幕

The tab "Splash" allows you to specify the splash screen. Your application will use the splash.bmp picture which must be in the plugin.xml directory. The file must be a bmp file and it must be called splash.bmp.

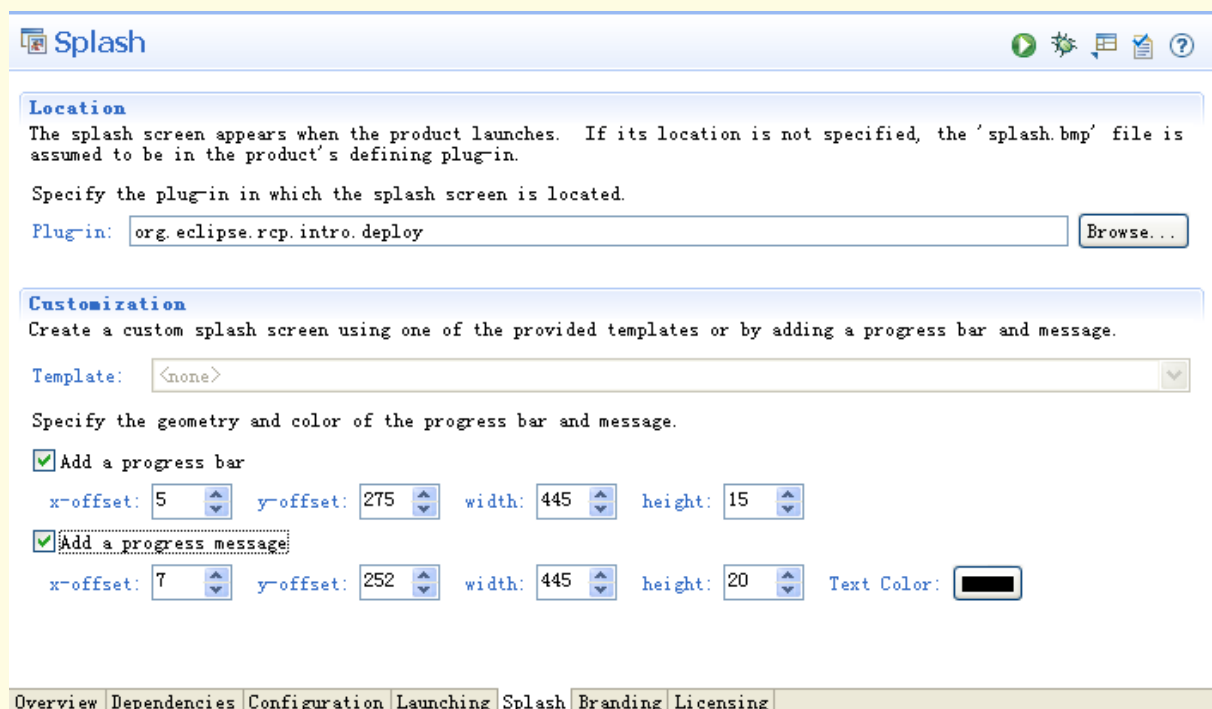
你可以利用 **Splash** 标签标签页来指定一个溅射屏幕。你的应用将会使用 **splash.bmp**,而此图像文件必须在 **plugin.xml** 文件所在的目录中，文件名也必须为 **splash.bmp**。

You can add a message and a process bar to your splash screen by selecting the corresponding settings.

通过选择相应的设置，你也可以为你的溅射屏添加消息和进度条。

Create a "splash.bmp" via your favority graphics tool and save it in the "org.eclipse.rcp.intro.deploy" project. Maintain the the following.

用你喜欢的绘图工具创建一个名为 **splash.bmp** 的图像，将其保存到 **org.eclipse.rcp.intro.deploy** 项目中，进行如下维护：



Splash

Location
The splash screen appears when the product launches. If its location is not specified, the 'splash.bmp' file is assumed to be in the product's defining plug-in.
Specify the plug-in in which the splash screen is located.
Plug-in:

Customization
Create a custom splash screen using one of the provided templates or by adding a progress bar and message.
Template:

Specify the geometry and color of the progress bar and message.

☒ Add a progress bar
x-offset: y-offset: width: height:

☒ Add a progress message
x-offset: y-offset: width: height: Text Color:

Overview Dependencies Configuration Launching **Splash** Branding Licensing

If you now start your application you should see a slashescreen. For example I used a screenshot of windows desktop.

如果现在启动应用，你将会看到一个溅射屏幕。



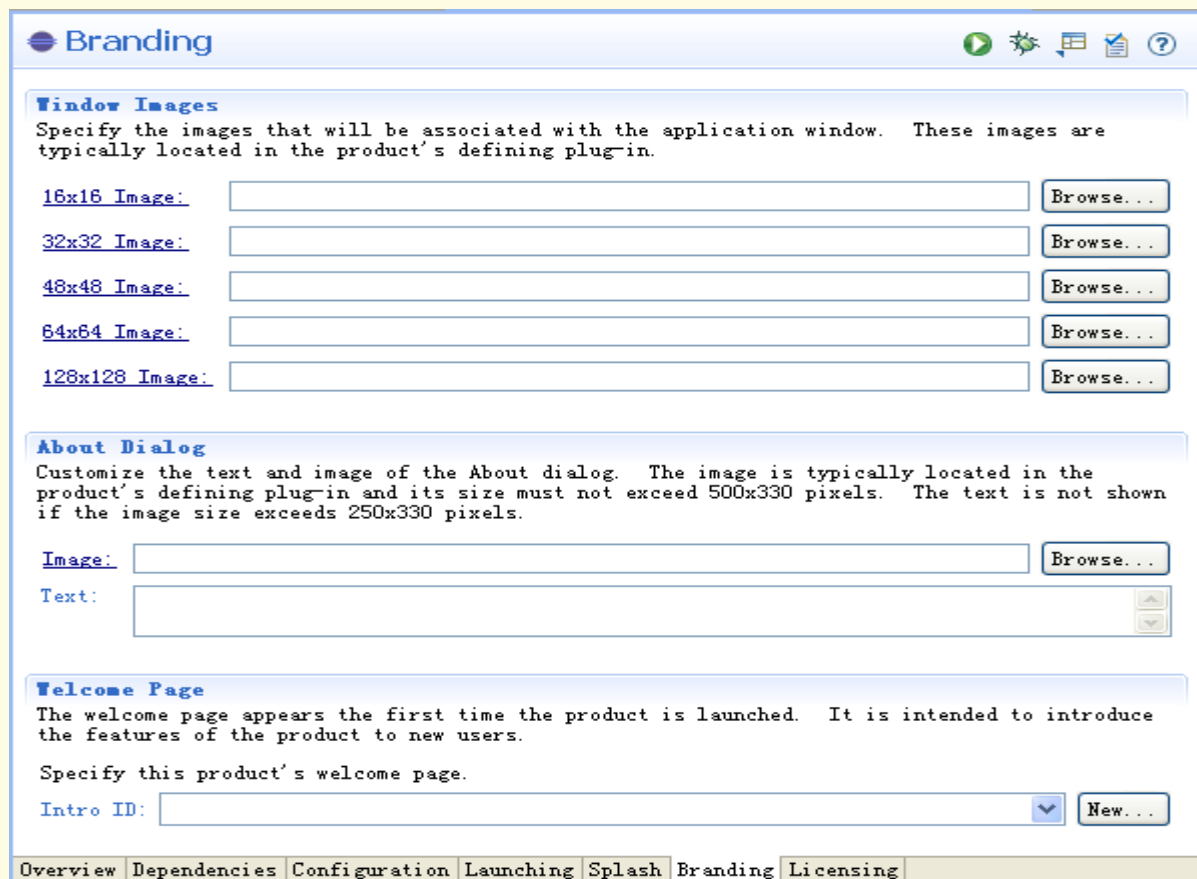
15.8. BRANDING YOUR PRODUCT 制作产品商标

In this example I do not use this.

在本例中我并不讲如何制作产品商标。

The standard Eclipse About Dialog can be branded. You can add an icon and / or and a text to your application.

标准的 Eclipse About 对话框是可以商标化。你可以为你的应用添加一个图标和/或文字。



Add the standard command "org.eclipse.ui.help.about" to your menu to open the about dialog. See [Eclipse Commands Tutorial](#) for information how to use Eclipse standard commands.

将标准命令 “org.eclipse.ui.help.about” 添加到你的菜单中可以打开一个 **About(关于)**对话框。有关使用 Eclipse 标准命令的方法，请参见 [Eclipse 命令教程](#)。

15.9. CUSTOMIZING THE START ICON AND LAUNCHER ARGUMENTS 定制开始图标与启动

参数

In this example I do not use this.

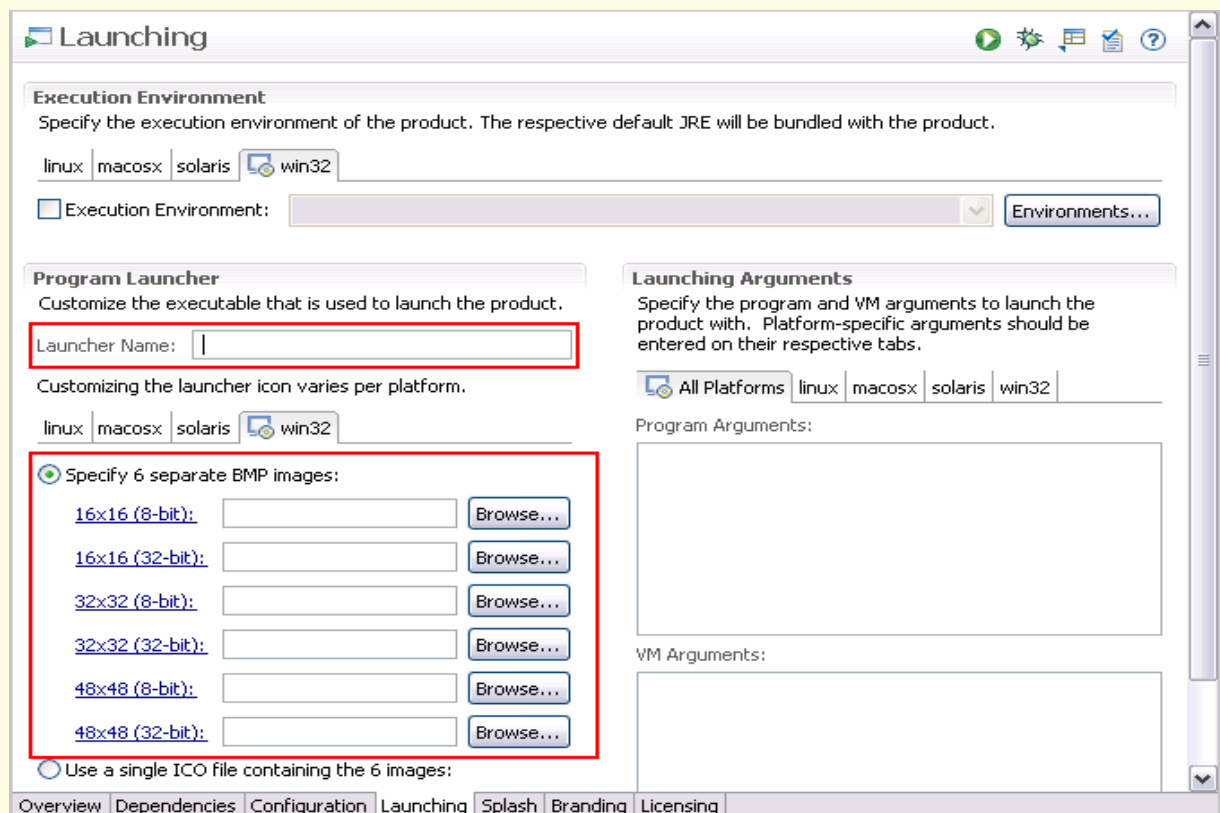
在本例中，我不讲这一点。

The launcher is the executable program that is created during the deployment of the product. Per default "eclipse.exe" with an "eclipse" icon is created. To change this select the launcher tab of your product configuration.

启动器是一个在部署期间所创建的可执行程序。每个缺省的“eclipse.exe”都会带有一个“eclipse”图标。要改变这一点，你需要在你的产品配置文件的 **Launching** 标签页进行修改。

Here you can specify the name of the application and the icon which should be used. Make sure the deep of the icons is correctly maintained otherwise Eclipse will not use these icons.

在此你可以指定应用名称及所要使用的图标。请注意图标文件的尝试需要正确，否则 Eclipse 无法使用这些图标。



In the launcher section you can also specify parameters to your Eclipse program or JVM arguments.

你需要在 **Launcher** 部分指定你的 **Eclipse** 程序或 **JVM** 的参数

15.10. DONE 完成了!

You created your product configuration. The next chapter will explain how to use this configuration to create an product, e.g. a stand- alone program which can be started outside of Eclipse.

你已经完成了你的产品配置。在下一章中将教你如何利用此配置创建一个产品——即独立运行程序，它可以脱离 **Eclipse** 运行。

16. DEPLOY YOUR PRODUCT 部署你的产品

To create a standalone RCP program you deploy your product. The result can be shared with other users.

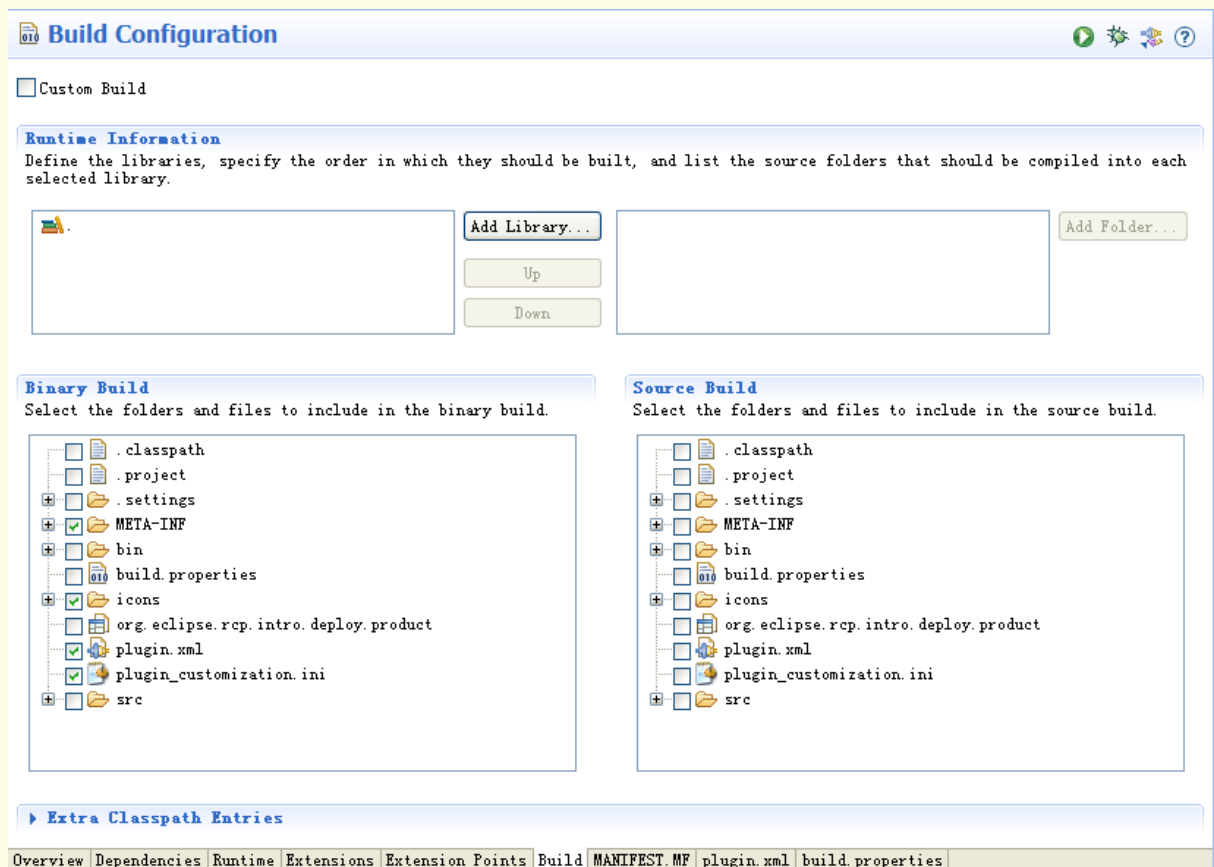
要创建独立运行的 RCP 应用，你可以“部署”你的产品。此结果使你可以与它人共享。

Eclipse will automatically include all your compiled classes into the build output. You have to manage manually the other files. If you using icons / splash screens images, etc. you have to add them to the build output.

Eclipse 可以将所有的已经编译好的类打包输出。其它文件则需要手工管理。如果你使用了 icons/splash（图标或溅射）屏幕图像，你就需要将这些内容添加进去并建立输出文件。

Select your plugin.xml and select the build tab. Make sure the META-INF directory and the file plugin.xml is selected, select your icon directory and any other graphic file which should be included in your output.

打开你的 plugin.xml 文件，再选择 Build（构建）标签页。确认 META-INF 目录与 plugin.xml 文件已经选中，再选择你图标文件目录和其它需要包含到输出中的图像文件。



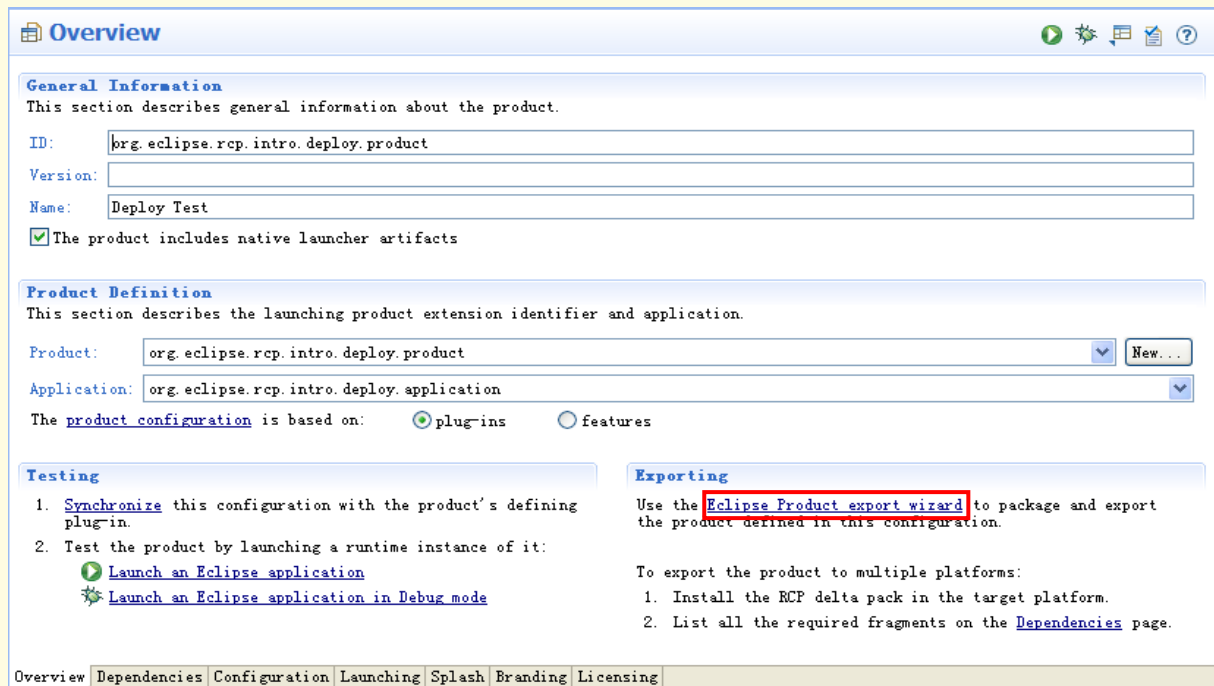
Tip

Without META-INF directory and the file "plugin.xml" in build the exported product will not start. If after deployment your graphics are missing check this tab if you really include them in the deployment.

提示：如果不选择 META-INF 目录及 plugin.xml 文件，所生成的输出将不能运行。如果在生成的部署中没有图像，请检查是否勾选了相应的文件。

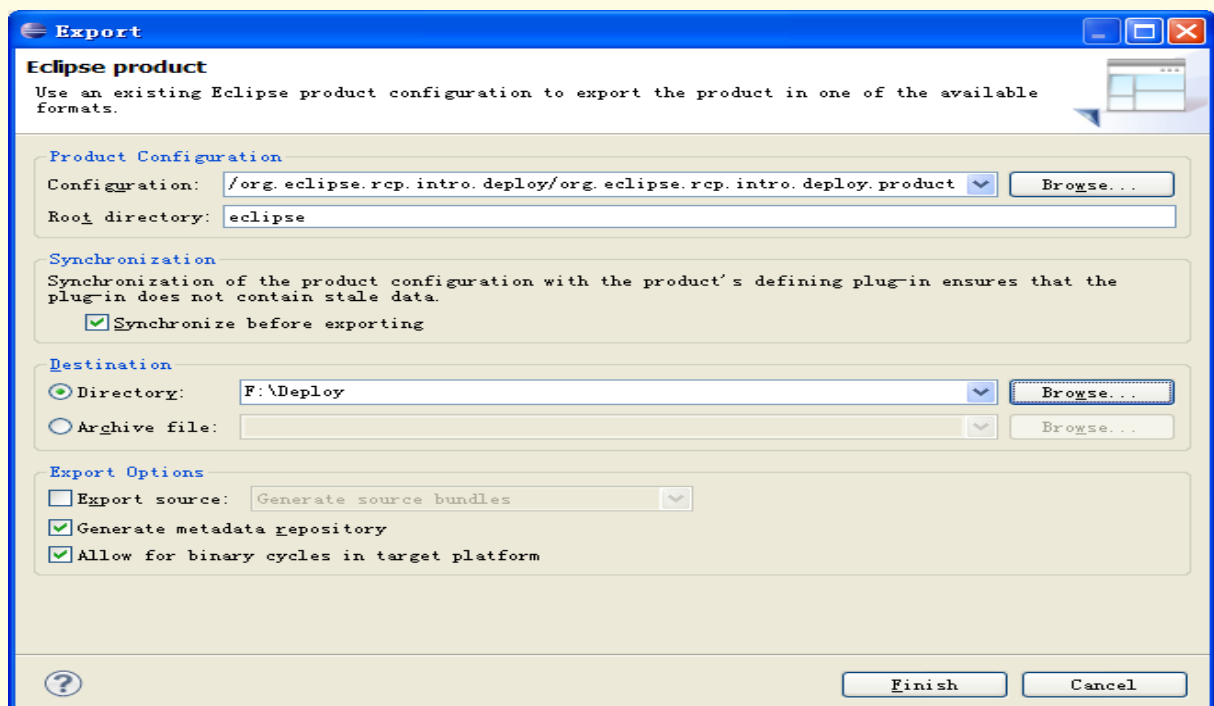
Switch now to your product configuration file and select the tab Overview. Click on the "Eclipse Product export wizard" to export your product.

现在切换到你的产品配置文件，选择 Overview（概览）标签页。点击“Eclipse Products export Wizard”（Eclipse 产品导出向导）以导出你的产品。



Maintain the target directory and press finish.

输入目标目录，点击 Finish。



This should create a directory in the specified place with includes a file "eclipse.exe" which starts your application. Double click on this to start your application.

此操作将在指定位置创建一个目录，目录中包含有一个名为“Eclipse.exe”的文件用来启动你的应用。双击此文件来运行你的应用。

If you take the content of this directory and unzip on another machine (which has Java installed) your program should run there too.

如果你将此目录打包后在另一台已经安装了 Java 的主机上展开，程序将可以运行。

Tip

The export dialog allows to created a Archive file which you can the send directly to your users.

提示：导出对话框允许你创建一个归档文件以便你直接发送给其它人。

To learn how to automate the creation of a product please see [Eclipse PDE Build](#) .

要学习如何自动创建产品请参看 [Eclipse PDE Build](#)。

17. TIPS AND TRICKS 提示与技巧

17.1. SAVE USERS LAYOUT 保存用户布局

To remember the user's layout and window size for the next time you start your application, add `configurer.setSaveAndRestore(true);` to the initialize method of `ApplicationWorkbenchAdvisor`.

要保存用户退出时的布局与窗口大小，以便再次启动时使用，请在 `ApplicationWorkbenchAdvisor` 类的初始化方法 `initialize()` 中添加语句 `configurer.setSaveAndRestore(true);` 修改后的完整代码如下：

```
public class ApplicationWorkbenchAdvisor extends WorkbenchAdvisor {
    private static final String PERSPECTIVE_ID = "AddActiontoView.perspective";
    public WorkbenchWindowAdvisor createWorkbenchWindowAdvisor(
        IWorkbenchWindowConfigurer configurer) {
        return new ApplicationWorkbenchWindowAdvisor(configurer);
    }
    public String getInitialWindowPerspectiveId() {
        return PERSPECTIVE_ID;
    }
    @Override
    public void initialize(IWorkbenchConfigurer configurer) {
        super.initialize(configurer);
        configurer.setSaveAndRestore(true);
    }
}
```

Eclipse has a pre-defined command to reset the perspective. See [Eclipse Commands](#).

Eclipse 有一个预定义的命令用于进行透视图复位。请参看 Eclipse 命令。

17.2. PLUGIN ID IN APPLICATION 应用中的插件 ID

The plugin ID is usually needed in several places, so it is a good idea to declare it as static in `Application.java`. This ID must be the same as defined in the field "ID" of the overview tab of `plugin.xml`. The ID is case sensitive.

In our example if the RCP application is called "org.eclipse.rcp.intro.first" therefore add the following statement to Application.java.

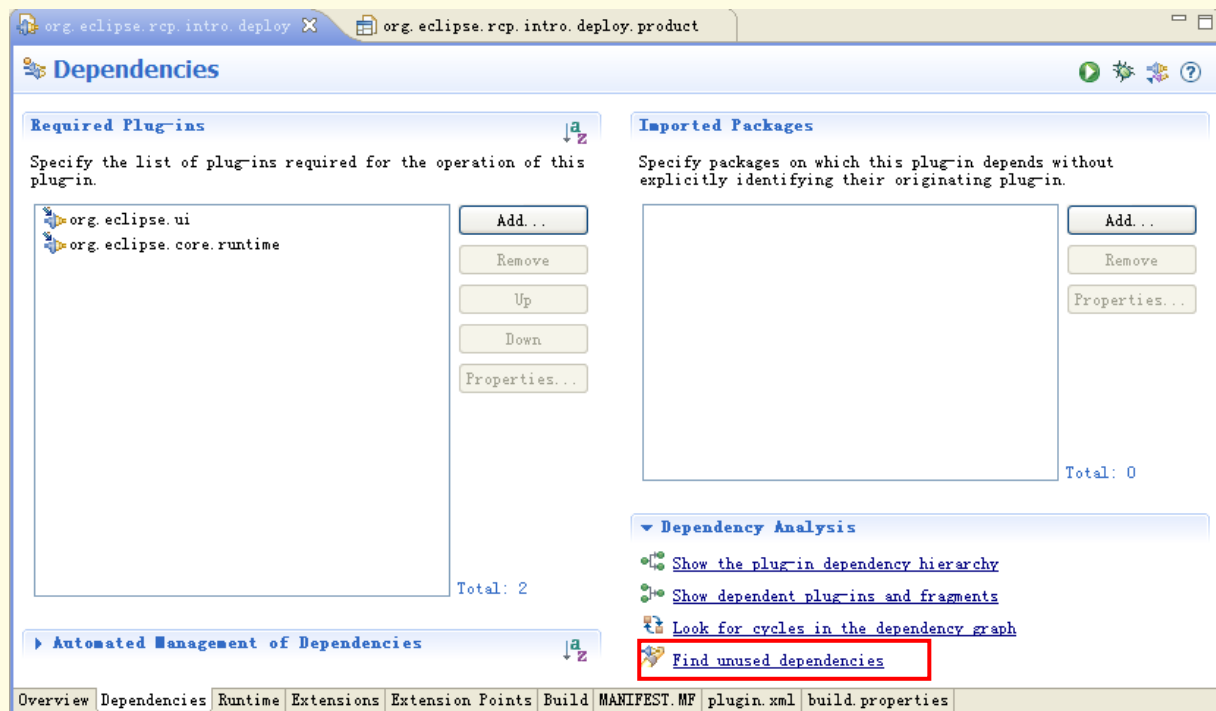
许多时候通常需要插件 ID，因此最好在 Application.java 中将插件 ID 声明成静态（static）。此 ID 必须与在 plugin.xml 文件的概览页中的 ID 字段值相同。此 ID 是大小写敏感的。在我们的例子中，如果 RCP 应用名为“rcp.intro.first”，则需要将如下语句添加到 Application.java 中：

```
public static final String PLUGIN_ID = "rcp.intro.first";
```

17.3. FINDING UNUSED DEPENDENCIES 查找无用的依赖

In the file plugin.xml (tab dependencies) you define on which plugins your plugin depends. Of course you should only define the required plugins here. You can check if you have any dependency maintained which is actually not used, by selecting Dependency Analysis -> Find unused dependencies.

在 plugin.xml 文件的标签页 Dependencies，你定义了自己的插件依赖于那些插件。自然你在此只需要定那些所需插件。你可以对你所选择的插件进行检查看其是否实际并没有使用。方法是在 Dependencies 标签页点击 Dependency Analysis 下的 Find unused dependencies(查找未用依赖)。



18. NEXT STEPS 下一步

后面内容，不再翻译

In general you find lots of good Eclipse articles on [eclipse.org Resources](#)

To learn how to use JFace Views you can use [Eclipse JFace TableViewer - Tutorial](#)

To learn how to use Eclipse Preferences and Preference Pages you can use [Eclipse Preferences](#)

Good luck in your journey of learning Eclipse the platform!

19. Thank you

Thank you for practicing with this tutorial.

20. Questions and Discussion

For questions and discussion around this article please use the [www.vogella.de Google Group](#). Also if you note an error in this article please post the error and if possible the correction to the Group.

I believe the following is a very good guideline for asking questions in general and also for the Google group [How To Ask Questions The Smart Way](#).

21. Links and Literature

21.1. Source Code

<http://www.vogella.de/code/codeeclipse.html> Source Code of Examples

21.2. Eclipse Resources

http://www.eclipse.org/articles/Article-SWT-graphics/SWT_graphics.html SWT Graphics

<http://www-128.ibm.com/developerworks/opensource/library/os-eclink/> View linking, e.g. how can view 1 react to selection changes in view 2

<http://www.eclipse.org/articles/article.php?file=Article-Understanding-Layouts/index.html> Understanding Layout in SWT, updated version by Wayne Beaton

<http://www.eclipse.org/articles/article.php?file=Article-AddingHelpToRCP/index.html> Adding help to an Eclipse RCP application

<http://wiki.eclipse.org/index.php/JFaceSnippets> JFace Snippets (Code Examples for certain tasks)

<http://www.eclipse.org/swt/snippets/> SWT Snippets (Code Examples for certain tasks)

21.3. Other Resources

[Eclipse.org Homepage](#)

Articles about Java, Eclipse and Webdevelopment from www.vogella.de

Articles about Eclipse development from www.vogella.de

Articles about Web development from www.vogella.de