Plug-in development 101, Part 2: Introducing rich-client applications

Learn the basics about plug-in development and rich-client applications

Chris Aniszczyk, Software Engineer, IBM, Software Group

Summary: Plug-in development in Eclipse is somewhat of an art form. If you're new to the concept of plug-ins, especially in the context of OSGi and Eclipse, it can be a burden learning the myriad tools Eclipse has to help you write plug-ins. This article will help you learn some basic plug-in development skills, with some best practices sprinkled in for good measure.

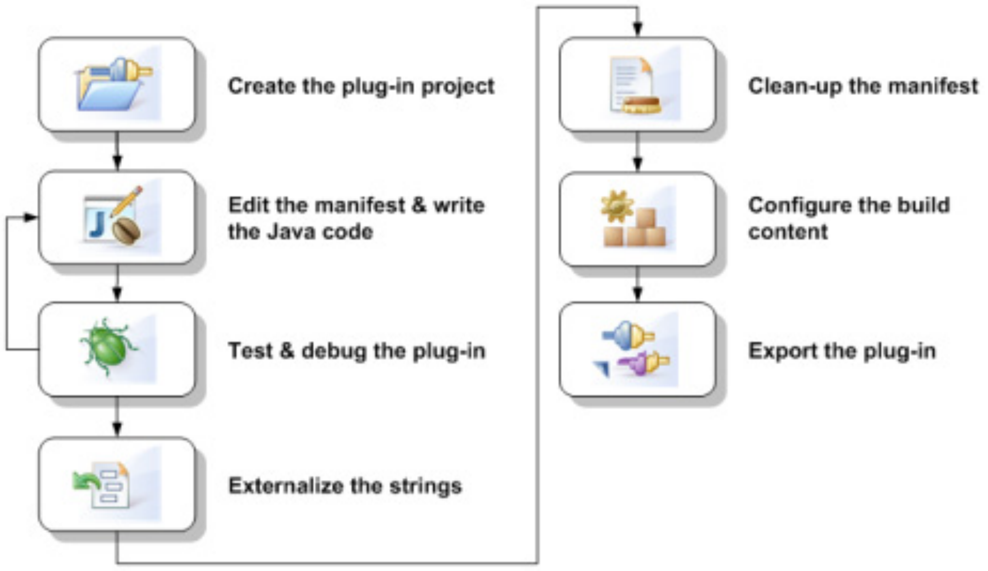
Date: 01 Apr 2008
Level: Intermediate
Activity: 1878 views
Comments: 0 (Add comments)

★★★★☆ Average rating (based on 13 votes)

This "Plug-in development 101" series of articles is all about developing plug-ins. But before we get started, we need to ensure that we have a proper environment in which to develop plug-ins. The first step is to download an Eclipse distribution that has the Plug-in Development Environment (PDE) in it from Eclipse.org. I recommend downloading the latest version of Eclipse Classic. In this series, we will use a milestone release of Eclipse V3.4 (M5). Once this done, you're ready to go. (See Resources to learn where to find Eclipse and additional background information if you are new to Eclipse.)

To make it easier to understand plug-in development, this article follows a workflow detailed in Figure 1. In Part 1, we discuss the first five steps of the workflow. Here, we cover the last two steps and focus on introducing rich-client applications.

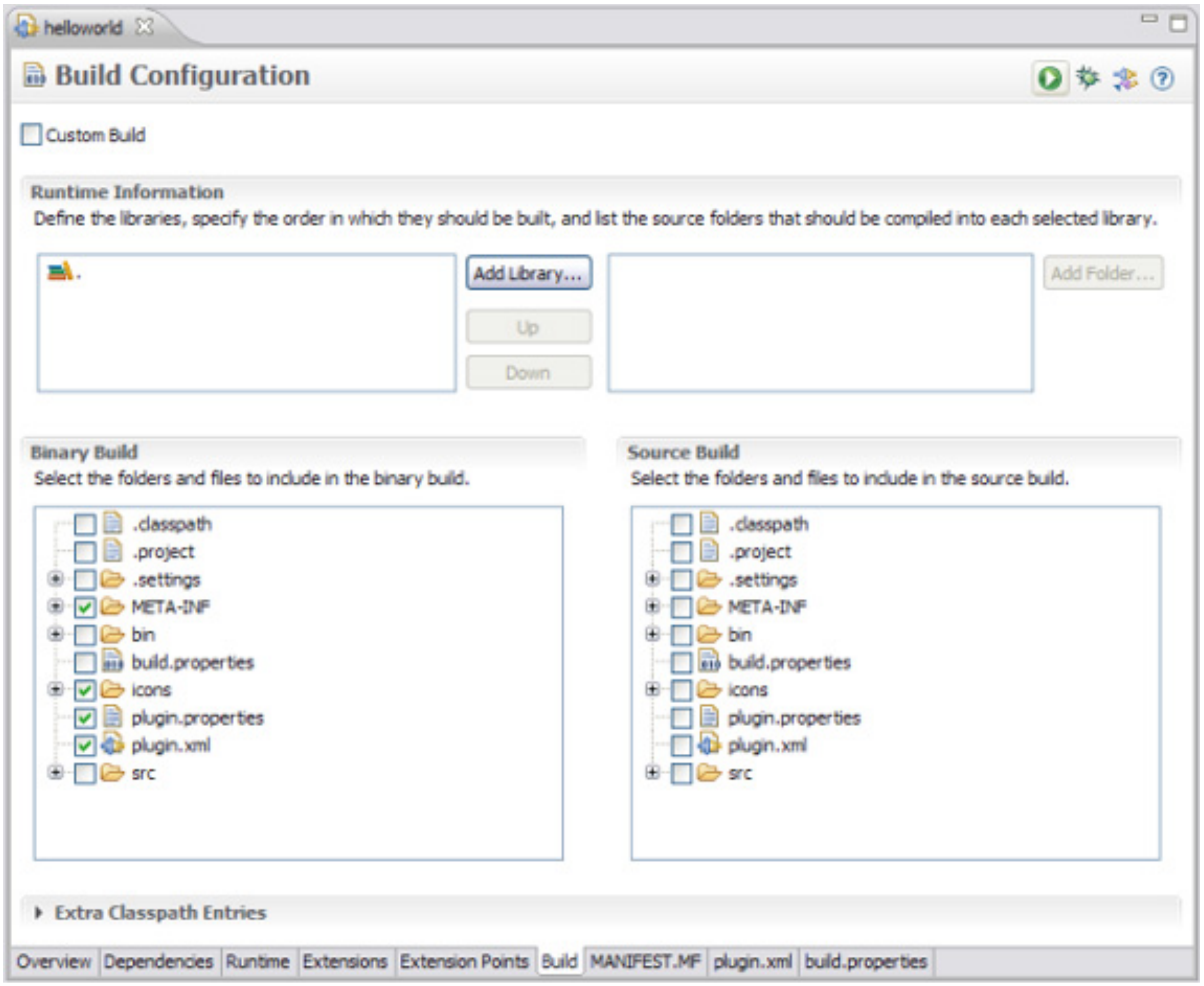
Figure 1. Plug-in development workflow


Create the plug-in project
Edit the manifest & write the Java code
Test & debug the plug-in
Externalize the strings
Clean-up the manifest
Configure the build content
Export the plug-in

Building

Configuring build content is an important step on our plug-in development adventure. In Eclipse, all plug-in development build-related configuration goes into the *build.properties* file.

Figure 2. Build configuration (build.properties)



Binary Build: [✓ META-INF], [✓ icons], [✓ plugin.properties], [✓ plugin.xml]; Source Build: none selected.

Automated builds?

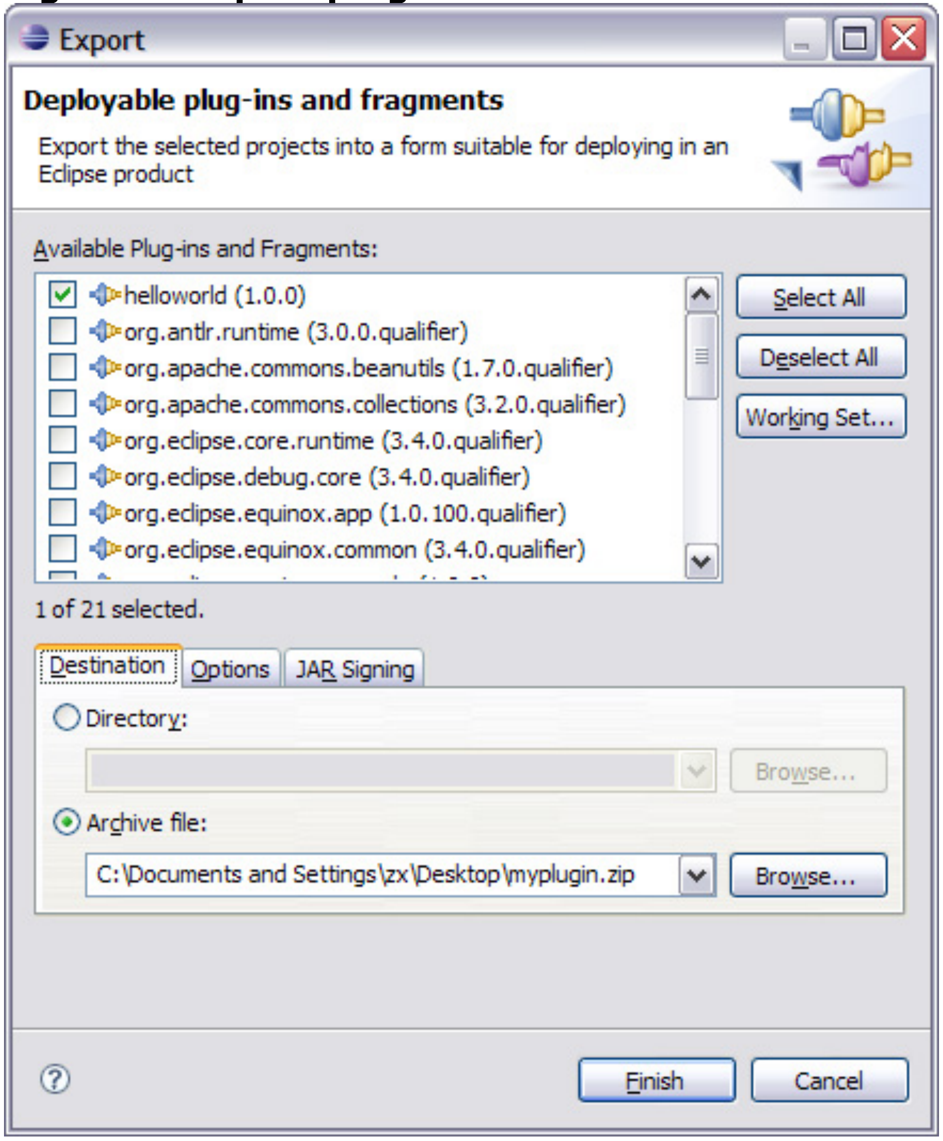
Setting up an automated build for a plug-ins is outside the scope of this article, but since it's such a frequent question, here's some background. The most common way to set up automated builds is with the *PDE Build* component in the Eclipse SDK. The downside to PDE Build is its complication, which beginners find daunting. A quick way to see how to set up an automated build is reviewing the plug-ins and tools at the Pluginbuilder Web site (see Resources).

Sample build-configuration content centers around the MANIFEST.MF, plugin.xml, and icon files. It may also include *plugin.properties* for internationalization support or artifacts like license files. It's important to notice the distinction of what is included in a *binary build* vs. a *source build*. Typically, when you export a plug-in from Eclipse, it's just a binary plug-in export that can be used by your friends in their Eclipse installations. A Source Build should include the source from the plug-in you're working on. You can elect to do a source build during a typical export operation.

Exporting

The final step in a typical plug-in developer's workflow involves exporting the plug-in you've created. Eclipse's PDE makes this an easy process using a specialized export wizard. To access this wizard (see Figure 3), simply click **File > Export** and select **Deployable Plug-ins and Fragments** under the **Plug-in Development** category.

Figure 3. Export plug-in wizard



Export

Deployable plug-ins and fragments

Export the selected projects into a form suitable for deploying in an Eclipse product

Available Plug-ins and Fragments:

- [✓] helloworld (1.0.0)
- [ ] org.antlr.runtime (3.0.0.qualifier)
- [ ] org.apache.commons.beanutils (1.7.0.qualifier)
- [ ] org.apache.commons.collections (3.2.0.qualifier)
- [ ] org.eclipse.core.runtime (3.4.0.qualifier)
- [ ] org.eclipse.debug.core (3.4.0.qualifier)
- [ ] org.eclipse.equinox.app (1.0.100.qualifier)
- [ ] org.eclipse.equinox.common (3.4.0.qualifier)

1 of 21 selected.

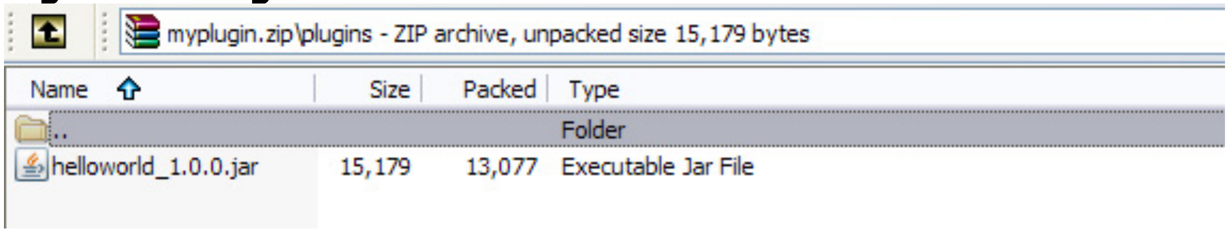
Destination | Options | JAR Signing

Directory: [ ]

[✓ Archive file:] C:\Documents and Settings\zx\Desktop\myplugin.zip

The first option to select within this wizard is what plug-ins we're interested in exporting. In this case, we just want our simple HelloWorld plug-in. The next step is to select a destination for our plug-in. We can opt to put the plug-in inside a ZIP file or within a directory. The other options within this wizard include abilities to do plug-in signing and bundling source with your plug-in. For now, ignore these and simply click **Finish** on the wizard to export your plug-in (see Figure 4). Note that the Overview page in the plug-in manifest editor provides an easy hyperlink to launch the wizard.

Figure 4. Plug-in on disk



myplugin.zip\plugins - ZIP archive, unpacked size 15,179 bytes

Name	Size	Packed	Type
..			Folder
helloworld_1.0.0.jar	15,179	13,077	Executable Jar File

That's it! That's all it takes to get a plug-in from an Eclipse workspace to a consumable form on a hard disk. Once in this form, plug-ins can be distributed easily to colleagues and friends. This ends our focus on plug-in development workflow, but now that we know the basics, we can start looking at what it takes to create a rich-client application within Eclipse.

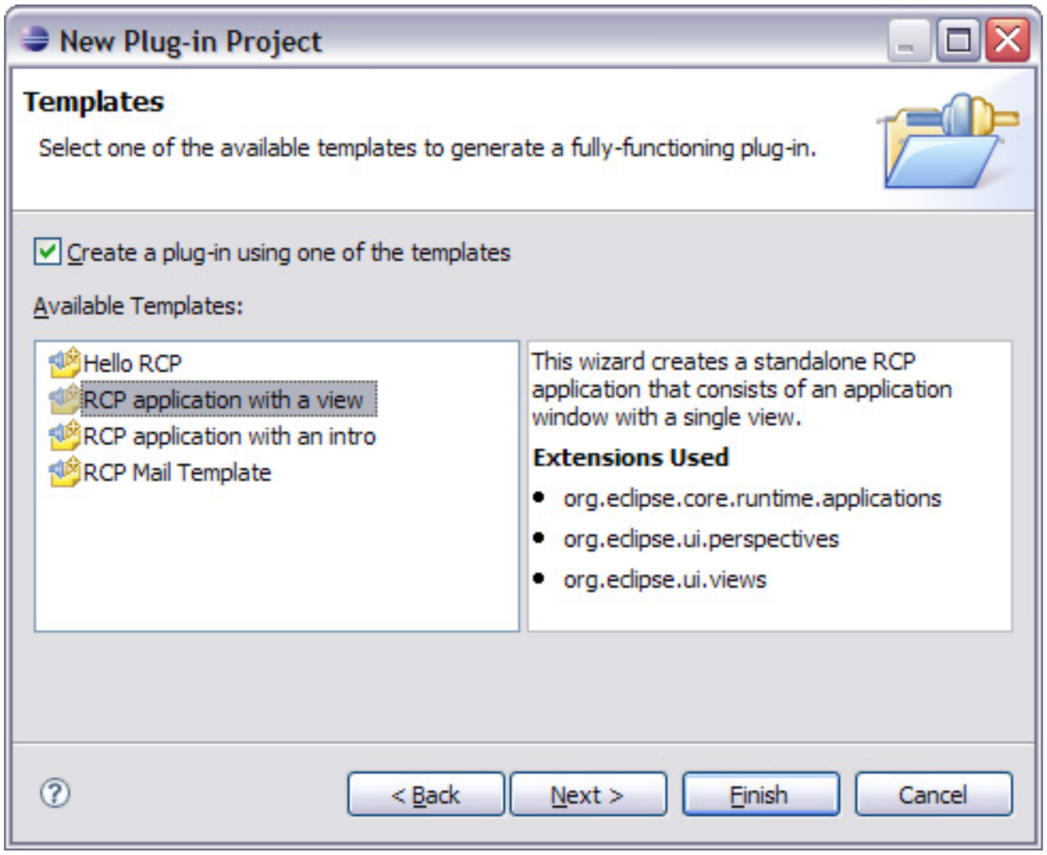
Rich Client Platform

Rich-client applications are enabled in Eclipse using the Rich Client Platform (RCP). Traditionally, the Eclipse platform is designed to serve as an open tools platform. However, it is architected so its components could be used to build just about any client application. The minimal set of plug-ins needed to build a rich-client application is collectively known as the Rich Client Platform. See Resources to learn more.

Products

To create a rich-client application within Eclipse, we need to work with a concept called *product configurations*. Product configurations are what PDE provides as a way for plug-in developers to build rich-client applications. To understand product configurations, we need to have a sample rich-client application to work with. We'll create one by taking advantage of the PDE template mechanism we went over in Part 1. Create a plug-in project named `rcp` and make sure to indicate that its a rich-client application, then select the **RCP Application with a view** template (see Figure 5).

Figure 5. Rich-client application template



Select one of the available templates to generate a fully-functioning plug-in.

[✓] Create a plug-in using one of the templates

Available Templates:

- Hello RCP
- RCP application with a view
- RCP application with an intro
- RCP Mail Template

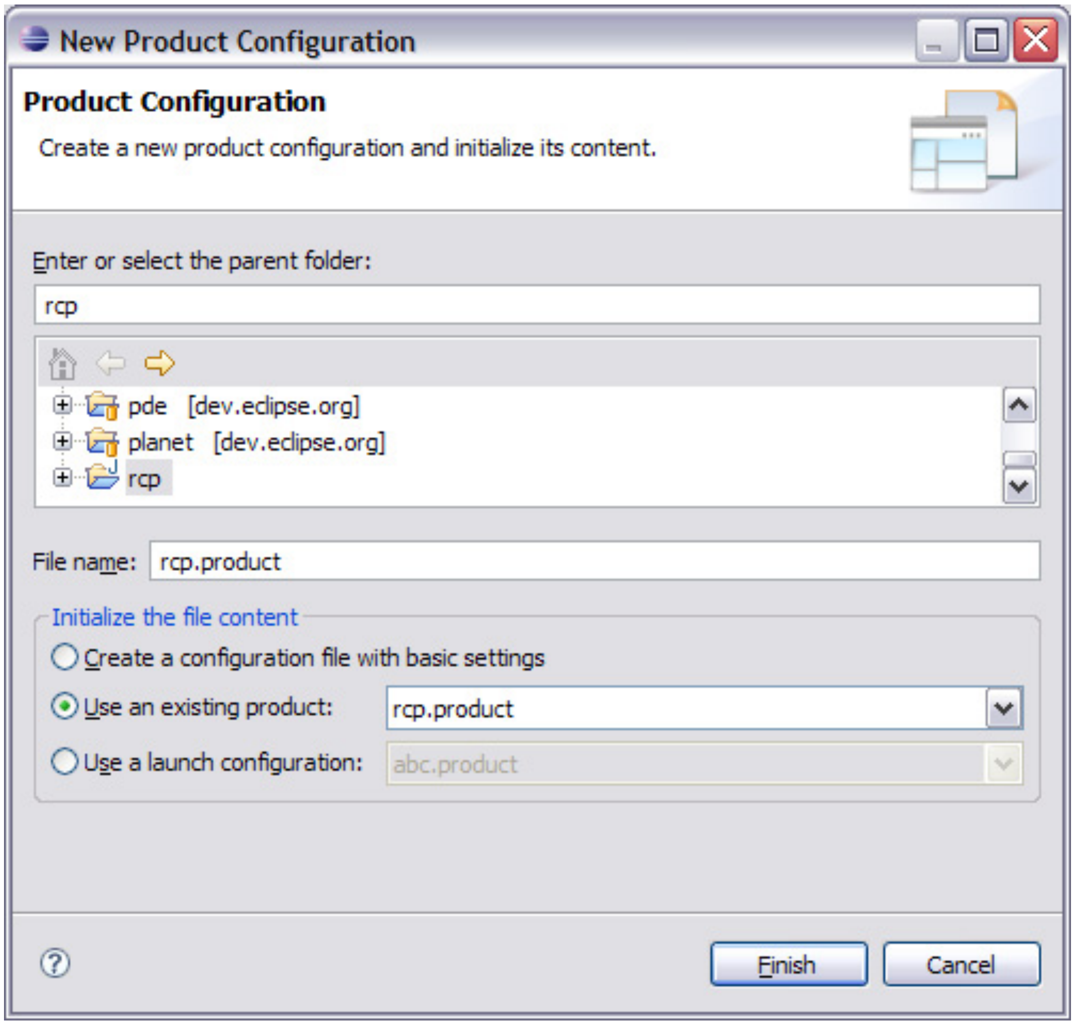
This wizard creates a standalone RCP application that consists of an application window with a single view.

Extensions Used

- org.eclipse.core.runtime.applications
- org.eclipse.ui.perspectives
- org.eclipse.ui.views

The next step is to create a product configuration file to help assemble the rich-client application we previously created. To create a new product configuration, right-click on the plug-in project and select **New > Product Configuration** to launch the new product configuration wizard (see Figure 6). Accept all the defaults, use `rcp.product` for the name of your product configuration file and click **Finish** to launch the product configuration editor. The next sections will go over the various pages of the product configuration editor.

Figure 6. New product configuration wizard


New Product Configuration
Product Configuration
Create a new product configuration and initialize its content.

Enter or select the parent folder: rcp

pde [dev.eclipse.org]
planet [dev.eclipse.org]
rcp

File name: rcp.product

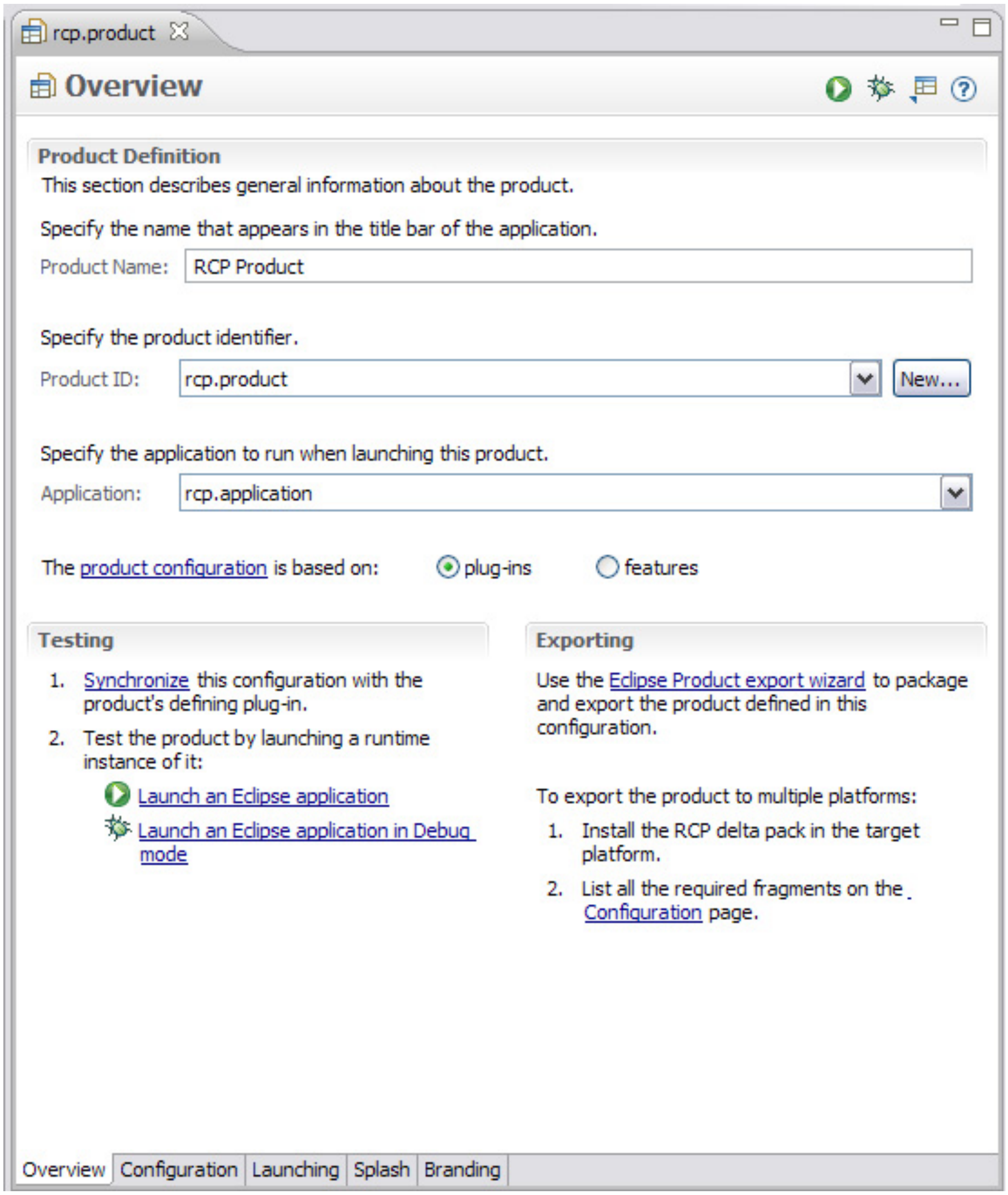
Initialize the file content

( ) Create a configuration file with basic settings
(●) Use an existing product: rcp.product
( ) Use a launch configuration: abc.product

Overview

Similar to the plug-in manifest editor, the first page presented within the product configuration editor is the Overview tab (see Figure 7). It gives a quick synopsis of the product configuration, convenient links to test and export products, and the ability to select whether the product is based on plug-ins or features. To quickly test applications, select the **Launch an Eclipse application** link within the **Testing** section and see what the rich-client application looks like.

Figure 7. Overview


rcp.product

Overview

Product Definition

This section describes general information about the product.

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

Product Name: RCP Product

Specify the product identifier.

Product ID: rcp.product

Specify the application to run when launching this product.

Application: rcp.application

The product configuration is based on: [✓ plug-ins] features

Testing

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

Exporting

Use the Eclipse Product export wizard to package and export the product defined in this configuration.

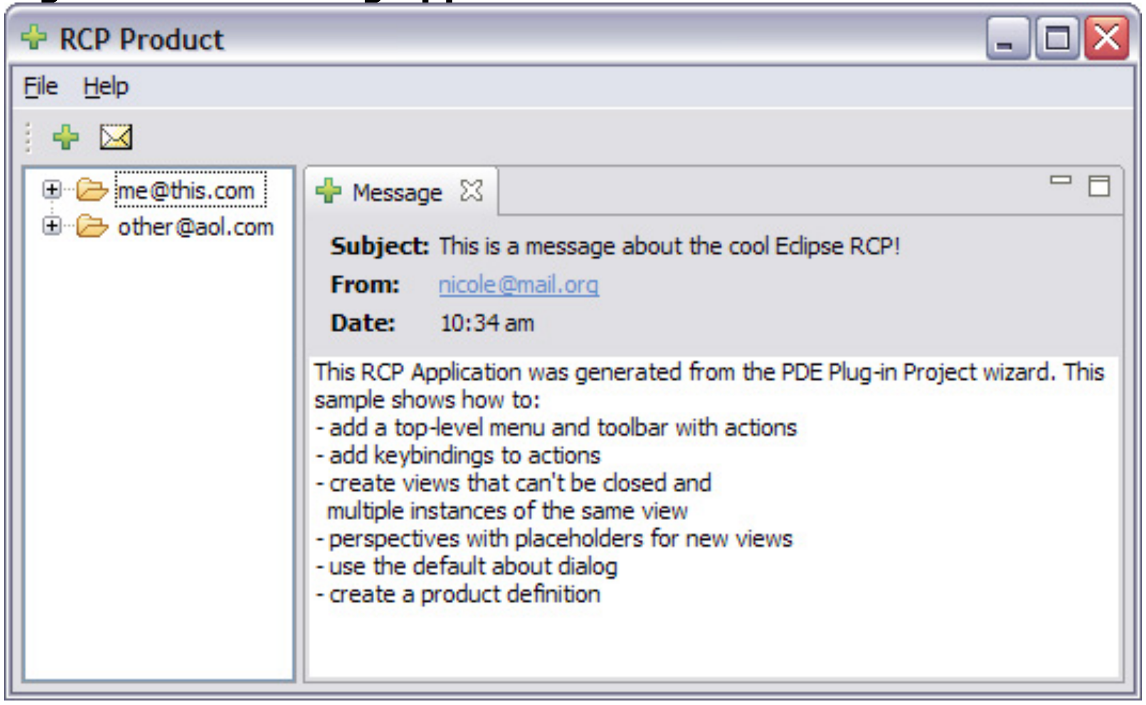
To export the product to multiple platforms:

1. Install the RCP delta pack in the target platform.
2. List all the required fragments on the Configuration page.

That's all there is to the Overview tab within the product configuration editor. If you're interested in the basic operations you can do on product configurations, head back to Overview.

It is also possible to launch your application from here. For example, click the **Launch an Eclipse application** and you should see your application pop up (see Figure 8).

Figure 8. Launching applications



Subject: This is a message about the cool Eclipse RCP!
From: nicole@mail.org
Date: 10:34 am

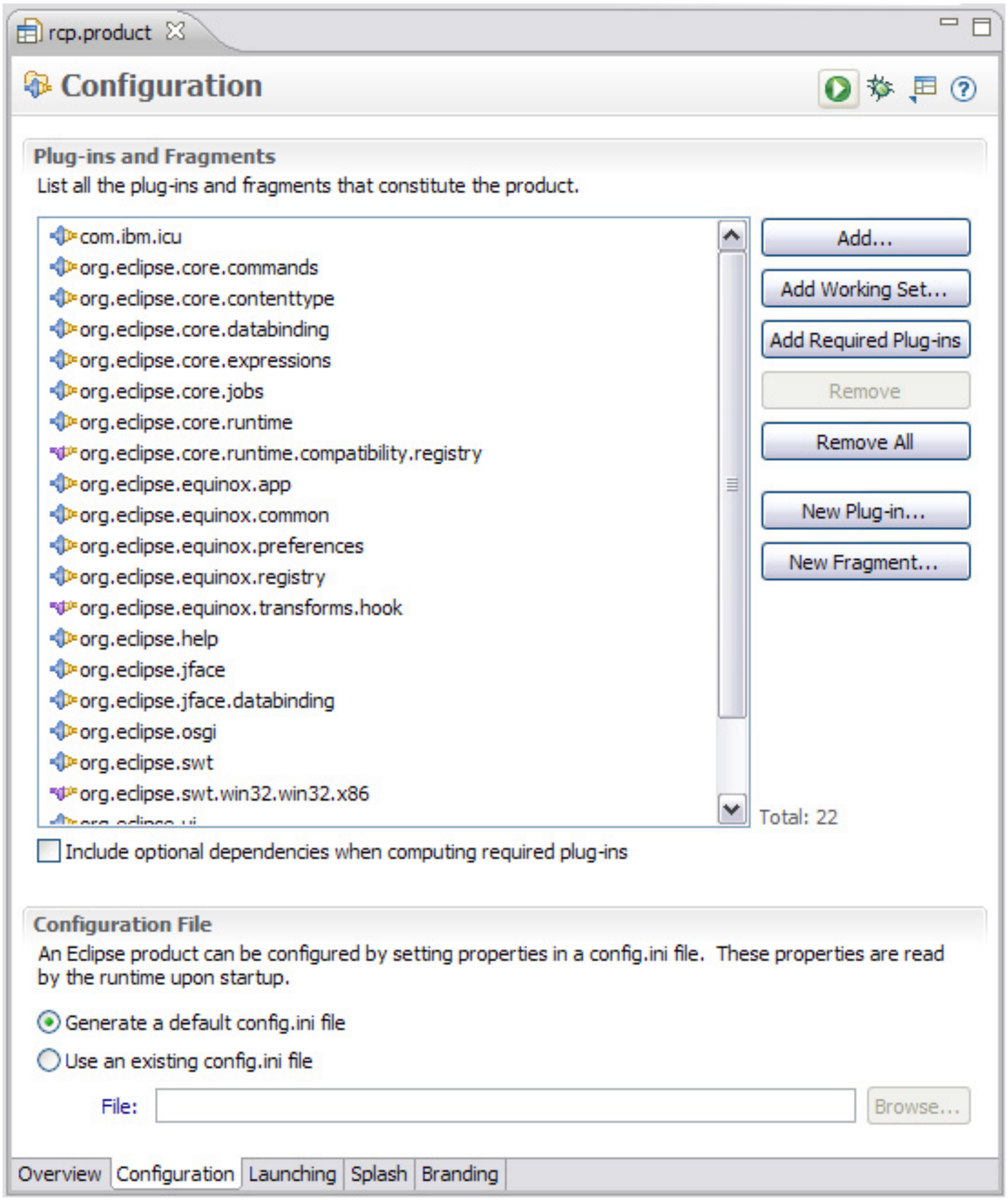
This RCP Application was generated from the PDE Plug-in Project wizard. This sample shows how to:
- add a top-level menu and toolbar with actions
- add keybindings to actions
- create views that can't be closed and
  multiple instances of the same view
- perspectives with placeholders for new views
- use the default about dialog
- create a product definition

Configuration

The Configuration tab contains the basic building blocks needed to run your product. This includes information such as the artifacts needed to run your product. The first section, Plug-ins and Fragments, simply lists the plug-ins and fragments needed to run your product. As a helpful tip, there will be times when you're working on your product and dependencies will be introduced. When this happens, it's always keen to select the **Add Required Plug-ins** to calculate if anything needs to be added to your product configuration.

The next section, Configuration File, represents an OSGi-specific artifact called the config.ini file. About 99.9 percent of the time, you want to leave this setting alone and have Eclipse generate the file. All this file contains really is what you have specified in the previous plug-ins and fragments section, but in a special format Eclipse can understand when launching.

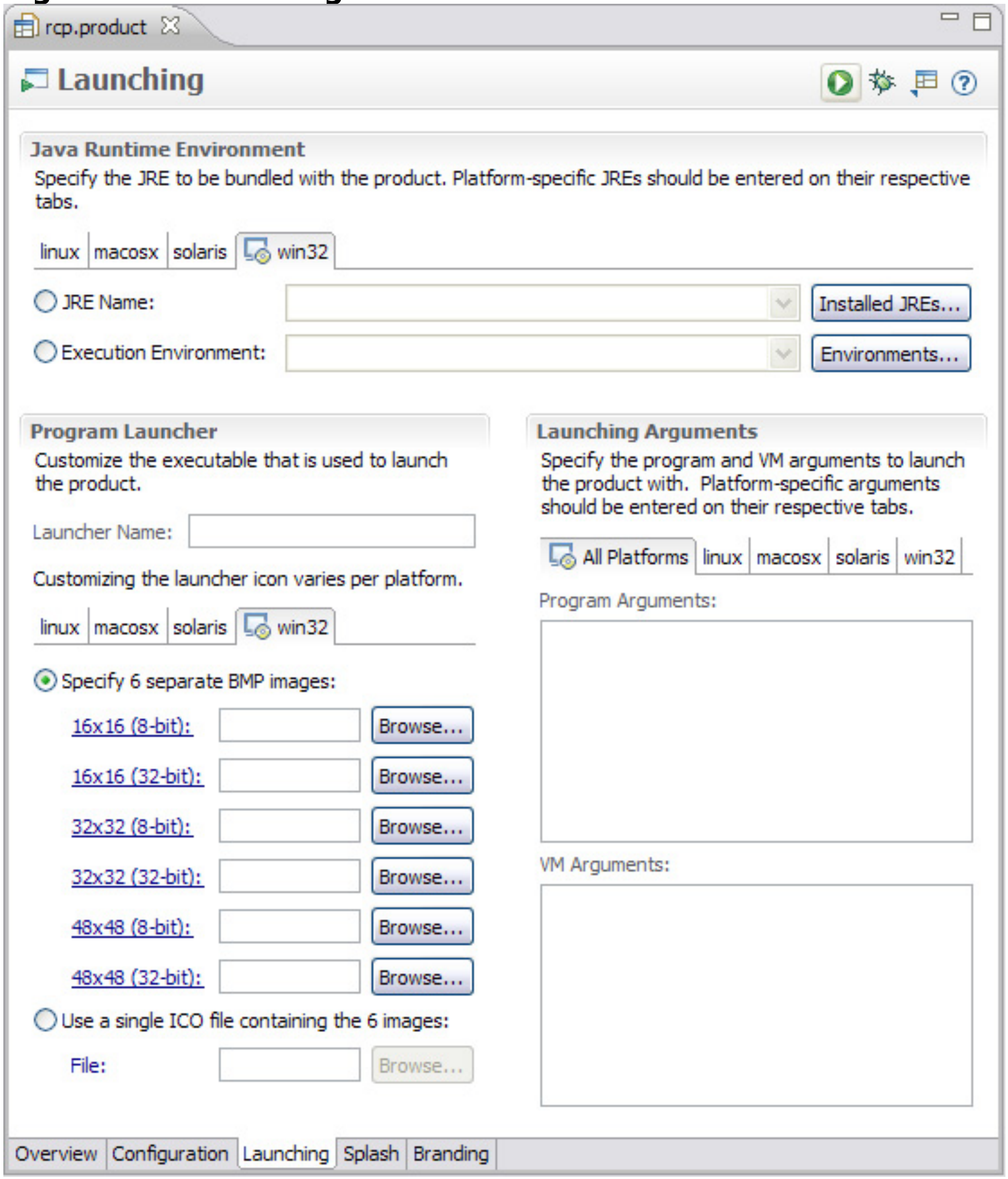
Figure 9. Configuration


rcp.product
Configuration
Plug-ins and Fragments
List all the plug-ins and fragments that constitute the product.
com.ibm.icu
org.eclipse.core.commands
org.eclipse.core.contenttype
org.eclipse.core.databinding
org.eclipse.core.expressions
org.eclipse.core.jobs
org.eclipse.core.runtime
org.eclipse.core.runtime.compatibility.registry
org.eclipse.equinox.app
org.eclipse.equinox.common
org.eclipse.equinox.preferences
org.eclipse.equinox.registry
org.eclipse.equinox.transforms.hook
org.eclipse.help
org.eclipse.jface
org.eclipse.jface.databinding
org.eclipse.osgi
org.eclipse.swt
org.eclipse.swt.win32.win32.x86
Add...
Add Working Set...
Add Required Plug-ins
Remove
Remove All
New Plug-in...
New Fragment...
Total: 22
Include optional dependencies when computing required plug-ins
Configuration File
An Eclipse product can be configured by setting properties in a config.ini file. These properties are read by the runtime upon startup.
Generate a default config.ini file
Use an existing config.ini file
File:
Browse...
Overview
Configuration
Launching
Splash
Branding

Launching

The Launching tab contains all the information that deals with launching your Eclipse-based product. The **Java Runtime Environment** section allows you the convenience of bundling platform-specific JREs. The **Program Launcher** section allows you to customize the executable that launches the product. For example, it's common that developers want something other than *eclipse.exe* for the name of their launcher executables. On top of naming the launcher executable, you can also brand it using platform-specific icons. The **Launching Arguments** section allows you to specify platform-specific launch parameters for your product. This may be useful if you want any special behavior on an operating system, such as Mac OS X.

Figure 10. Launching



rcp.product

Launching

Java Runtime Environment

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

linux | macosx | solaris | win32

JRE Name:

Execution Environment:

Program Launcher

Customize the executable that is used to launch the product.

Launcher Name:

Customizing the launcher icon varies per platform.

linux | macosx | solaris | win32

[✓ Specify 6 separate BMP images:]

16x16 (8-bit):
16x16 (32-bit):
32x32 (8-bit):
32x32 (32-bit):
48x48 (8-bit):
48x48 (32-bit):

Use a single ICO file containing the 6 images:

File:

Launching Arguments

Specify the program and VM arguments to launch the product with. Platform-specific arguments should be entered on their respective tabs.

All Platforms | linux | macosx | solaris | win32

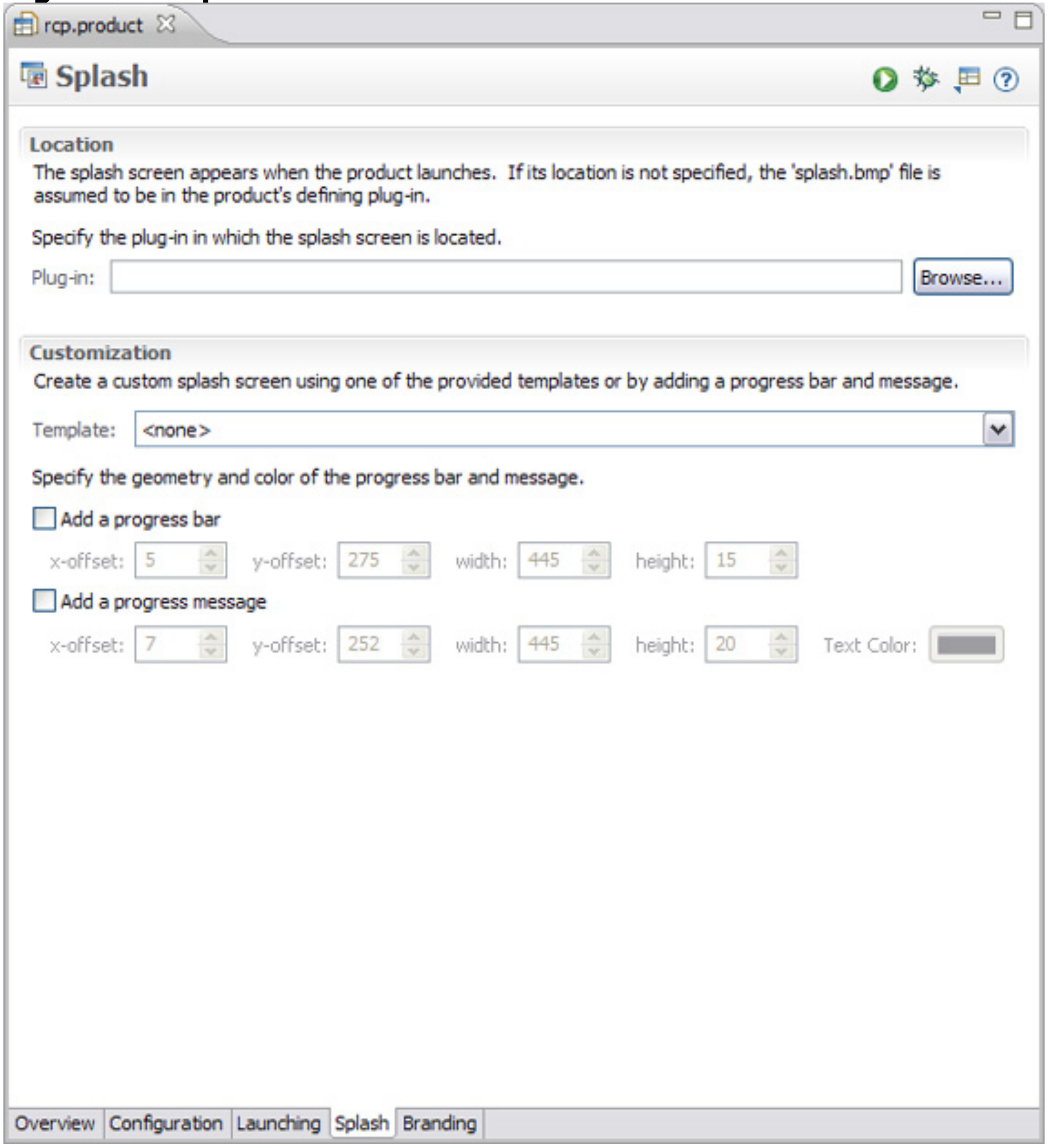
Program Arguments:

VM Arguments:

Splash

The Splash tab allows you to optionally configure a splash screen for your product (see Figure 11). For example, when you launch Eclipse, you get a simple splash screen that says Eclipse and shows loading plug-in progress. By default, your example RCP application came with a simple bitmap-based splash screen. However, to demonstrate the interesting things you can do with your splash screen, we'll take advantage of the login splash screen template (in the Customization section). Once you select the log-in template, save your product configuration and launch your application.

Figure 11. Splash



rcp.product

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:

Browse...

Customization

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

Template: <none>

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

Add a progress bar

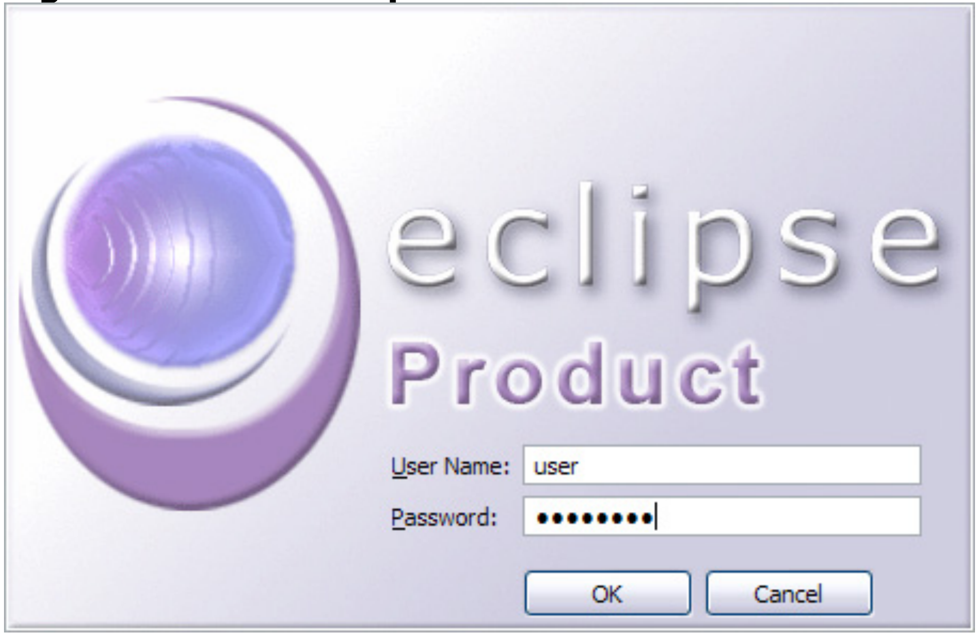
x-offset: 5 y-offset: 275 width: 445 height: 15

Add a progress message

x-offset: 7 y-offset: 252 width: 445 height: 20 Text Color:

Notice the new splash screen with a login and password.

Figure 12. Custom splash screen

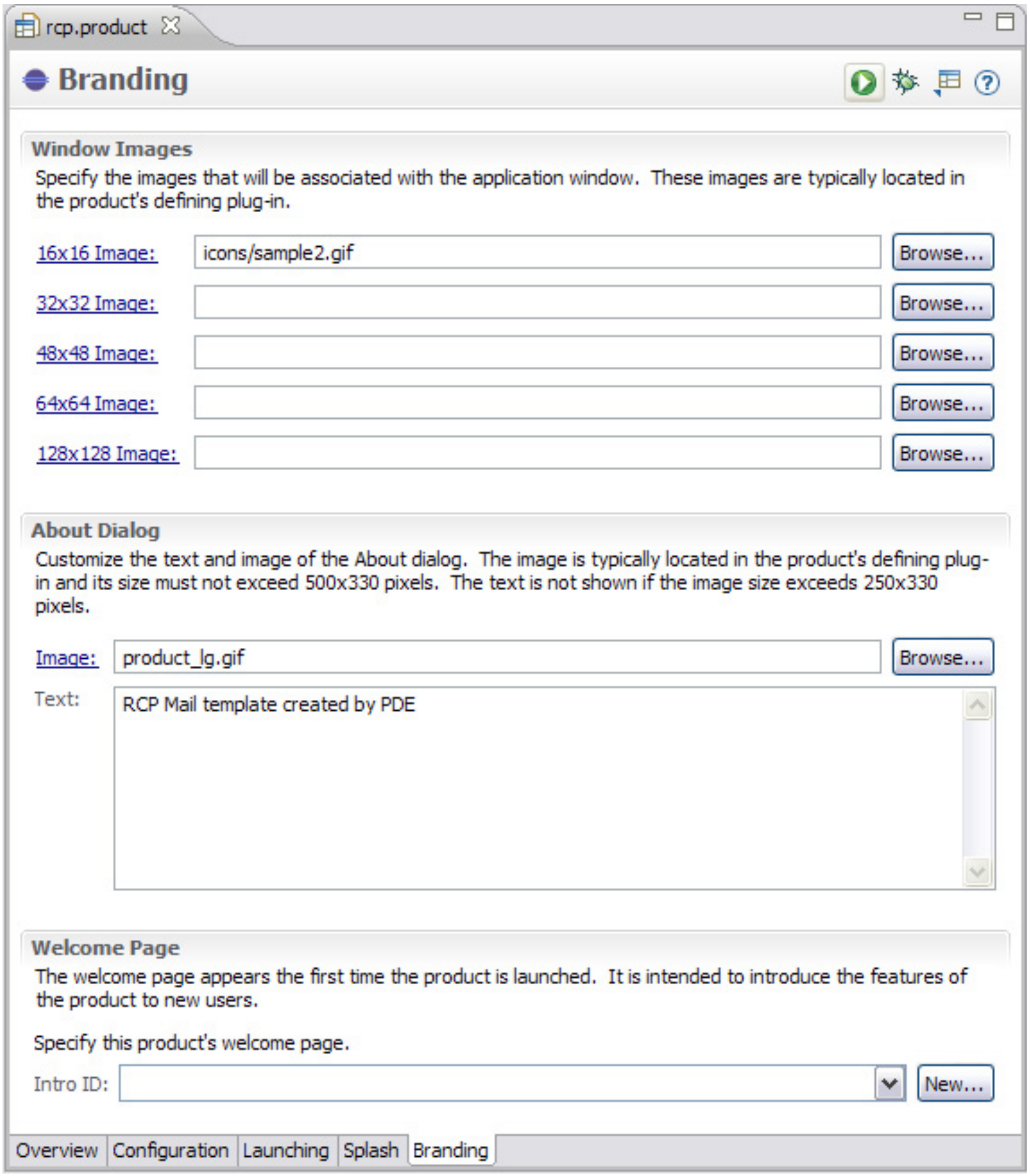


Branding

The Branding tab (see Figure 13) allows you to do three things: customize window images, create a custom about dialog, and create an optional welcome page. Window images are simply the images associated with your application shell windows. For example, when you launch Eclipse, the title bar has a little 16x16 Eclipse icon. These are the types of icons you can customize. Most software applications have some form of an about dialog to list things like licenses, authorship, and version information. The product configuration editor lets you reuse the existing Eclipse about dialog, but customize it with your own images and information.

The welcome page is something you can create to aid your users in learning your application. For example, to see the default Eclipse Welcome page, simply select the **Help > Welcome** menu item. If you desire something similar for your application, you can start the process in the **Welcome Page** section of the Branding tab. Since creating a welcome page is outside the scope of this article, to learn more, please consult Resources to discover more information about Eclipse user-assistance technologies.

Figure 13. Branding



Window Images

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

16x16 Image:	icons/sample2.gif	Browse...
32x32 Image:		Browse...
48x48 Image:		Browse...
64x64 Image:		Browse...
128x128 Image:		Browse...

About Dialog

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

Image: product_lg.gif

Text: RCP Mail template created by PDE

Welcome Page

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

Specify this product's welcome page.

Intro ID:

Conclusion

On the whole, this "Plug-in development 101" series' mission was to give an introduction to the basics of plug-in development with some best practices sprinkled in. We accomplished that in Part 1 by creating a sample plug-in and going through a typical plug-in development workflow. In Part 2, we finished going through our plug-in development workflow and created a rich-client application. Once the workflow is mastered, it becomes much easier to develop plug-ins and Eclipse RCP-based applications.

Now, go forth and use your newly found knowledge to create plug-ins and Eclipse-based applications.

Resources

Learn

- Pluginbuilder.org helps automate the construction of Eclipse plug-ins.
- The OSGi Alliance created the plug-in standard used by Eclipse.
- Learn more about Eclipse Plug-in Versioning at the Eclipse Foundation wiki.
- See the Eclipse EE guide at the Eclipse Foundation wiki.
- To learn more about the Eclipse Rich Client Platform, visit Eclipse.org.
- To learn more about the Eclipse SDK, see the Eclipse Foundation documentation titled "Platform Extension Points."
- Need help creating a welcome page for your RCP application? Read "Get to know Eclipse User Assistance."
- Need help debugging in Eclipse? Read "Debugging with the Eclipse Platform."
- Good background information on internationalizing plug-ins can be found in the Eclipse Foundation's articles "How to Internationalize your Eclipse Plug-In" and "How to Test Your Internationalized Eclipse Plug-In."
- Check out the "Recommended Eclipse reading list."
- Browse all the Eclipse content on developerWorks.
- New to Eclipse? Read the developerWorks article "Get started with Eclipse Platform" to learn its origin and architecture, and how to extend Eclipse with plug-ins.
- Expand your Eclipse skills by checking out IBM developerWorks' Eclipse project resources.
- To listen to interesting interviews and discussions for software developers, check

out developerWorks podcasts.

- Stay current with developerWorks' Technical events and webcasts.
- Watch and learn about IBM and open source technologies and product functions with the no-cost developerWorks On demand demos.
- Check out upcoming conferences, trade shows, webcasts, and other Events around the world that are of interest to IBM open source developers.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Download an Eclipse distribution containing the Plug-in Development Environment (PDE) from Eclipse.org.
- The first stop to make when seeking plug-ins is the Eclipse Foundation's project list.
- The second stop to make when seeking plug-ins is Eclipse Plug-in Central (EPIC).
- Check out the latest Eclipse technology downloads at IBM alphaWorks.
- Download Eclipse Platform and other projects from the Eclipse Foundation.
- Download IBM product evaluation versions, and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- The Eclipse Platform newsgroups should be your first stop to discuss questions regarding Eclipse. (Selecting this will launch your default Usenet news reader application and open eclipse.platform.)
- The Eclipse newsgroups has many resources for people interested in using and

extending Eclipse.

- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Chris Aniszczyk is an Eclipse committer at IBM Lotus who works on OSGi-related development. His primary focus these days is improving Eclipse's Plug-in Development Environment (PDE) and spreading the Eclipse love inside of IBM's Lotus organization. He is an open source enthusiast at heart, specializing in open source evangelism. He evangelizes about Eclipse in his blog, and he's honored to represent the Eclipse committers on the Eclipse Foundation's board of directors. He's always available to discuss open source and Eclipse over a frosty beverage.