Architecture Patterns and Code Templates for Enterprise RCP Applications

Labs

The four labs in this tutorial will lead you through the process of creating an RCP application using architecture patterns relevant to enterprise development. Throughout the labs, the emphasis is on high-level architecture and design patterns rather than on lower-level coding details.

The pre-requisites for this lab are:

- Eclipse Classic 3.6.* or Eclipse for RCP and RAP Developers 3.6.*. Eclipse 3.7 milestones are ok as well.

- Target platform should be set to the default (your Eclipse installation). If you don't know what this means, then you're setup is most likely correct.

For each lab there will be a folder contains a starting point for that lab. There is also a `lab-final` folder containing the end result of all four labs. At the beginning of each lab you should point Eclipse at the appropriate folder and import the projects.

Please begin each lab by importing the appropriate projects. Even if you completed the previous lab, there are additions in each project set that you will need.

Lab 1 - Creating a properly shaped RCP application

The goal of this lab is to create a properly shaped base RCP application into which additional functionality can be installed.

You will start with a generic Eclipse RCP application containing no Perspectives, Views, etc. The goal is to make this application self-updating: so that while it contains nothing, it can update itself on the fly to be anything you want it to be. We also will demonstrate how to unit test parts of an OSGi application separately from the whole.

Here is how we will do this:

1. Import this generic "bootstrapper" application's projects into workspace.

2. Modify the bootstrapper to load a configurable perspective.

3. Create a product to manage branding and build configuration.

4. Export the product.

5. Create a test fragment containing a unit test.

Import projects into workspace.

For the labs to function properly, you will need to point Eclipse at the appropriate lab directory. Creating a workspace someplace else and importing the projects there will *not* work.

The lab folders are located under the tutorial home directory in the `webapps/root/labs` directory.

1. Open Eclipse and select the `lab-1` folder as the workspace.

2. Right-click in the **Navigator** view and select **Import...** from the context menu.

3. Select **General > Existing Projects into Workspace** from the wizard selection dialog. Click **Next**.

4. On the second page, make sure that the **Select root directory** radio button is selected.

5. Click **Browse** and the `lab-1` folder should already be selected. If it is not, browse to the `lab-1` folder. A set of projects should show up in the **Projects** list. They should already be checked.

6. Click **Finish** and the projects should appear in the **Navigator** view.

Modify the bootstrapper to load perspectives.

Your workspace should now contain a single bundle and feature representing a simple "Hello World" RCP application. We'll now add to this application.

1. Open the `ApplicationWorkbenchAdvisor` class in the `com.example.app.bootstrapper` bundle.

2. In the `getInitialWindowPerspectiveId` method, remove `return null` and add the following line:

   ←   return System.getProperty("initialWindowPerspectiveId");

   ←   This will allow us to configure the bootstrapper later on with a perspective by passing the perspective id on the command line or through an INI file.
←

Create a product to manage branding and build configuration.

A properly shaped RCP application utilizes a *Product Configuration* file to manage branding elements and the way in which the product should be built.

The Product Configuration file is placed into it's own project because a core bootstrapper bundle is potentially reusable by multiple products. Placing the configuration in a separate project is also necessary to perform a Maven Tycho build.

Create the Product Configuration file.

1. Select **File > New > Project..** from the main menu. Select **General > Project** from the list of wizards and click **Next**.

2. On the first page of the wizard, enter `com.example.app. product` as the **Project name**. Leave everything else as is and click **Finish**.

3. Right-click on the new project and select **New > Other** from the context menu. Select **Plug-in Development > Product Configuration** on the wizard selection dialog and click **Next**.

4. Enter `example.product` in the **File name** field. Select the **Use an existing product** radio button and make sure the product is set to `com.example.app.bootstrapper.product.` Click **Finish**.
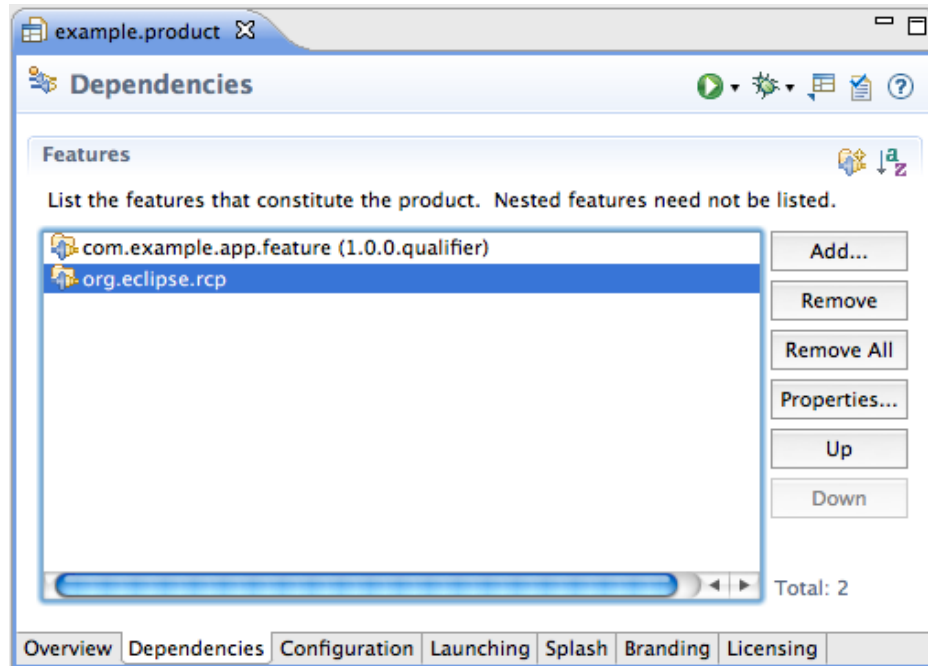
Configure the product.

1. The Product Configuration file should have opened with the **Overview** tab showing. Verify that the product runs by clicking on the **Launch an Eclipse application** link in the **Testing** section.

2. Again on the **Overview** tab, select the **features** radio button in the **Product Definition** section. This will allow us to configure our product as a set of features instead of a set of bundles.

3. Switch to the **Dependencies** tab. Add our bootstrapper feature along with `org.eclipse.rcp`. Save the file and run the application again by clicking the link on the **Overview** tab.
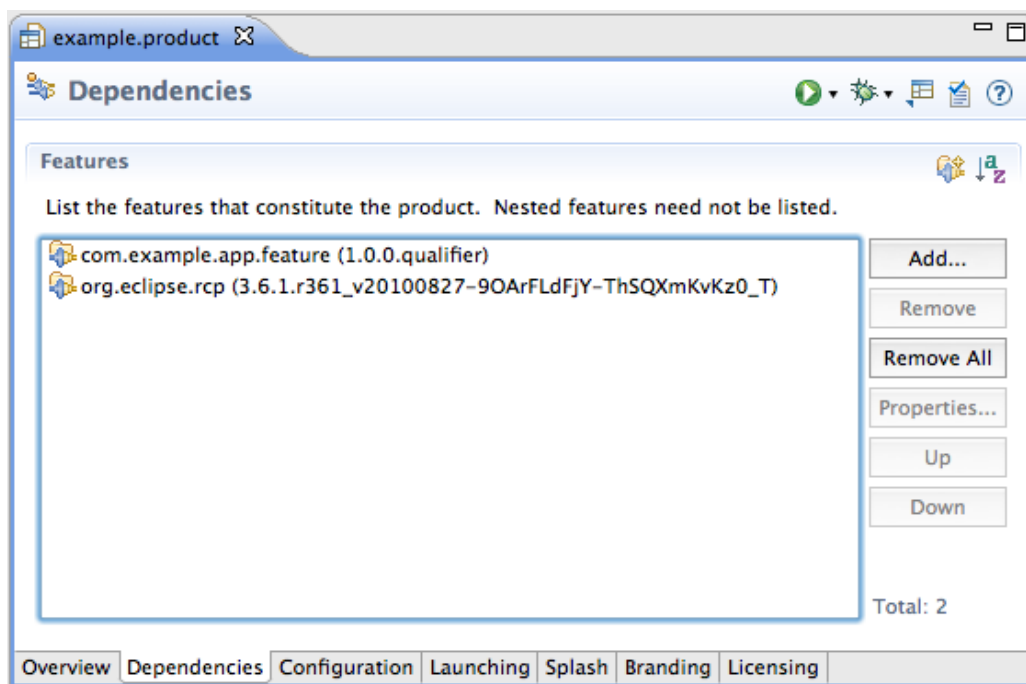
← If you see the bootstrapper application appear, your product configuration is valid.

4. Generally we don't want to limit the RCP feature to a specific version. On the **Dependences** tab, select the `org.eclipse.rcp` feature and click **Properties**. In the dialog, enter `0.0.0` in the **Version** field. Note that no version is now displayed next to the feature.

Verify that the product runs by clicking on the **Launch an Eclipse application** link in the **Testing** section.

Export the product.

1. On the **Overview** tab, click the **Eclipse Product export wizard** link in the **Exporting** section.

2. In the Export dialog, enter `example` in the **Root directory** field. Also select a destination in the **Directory** field. Your Desktop would be a good destination.
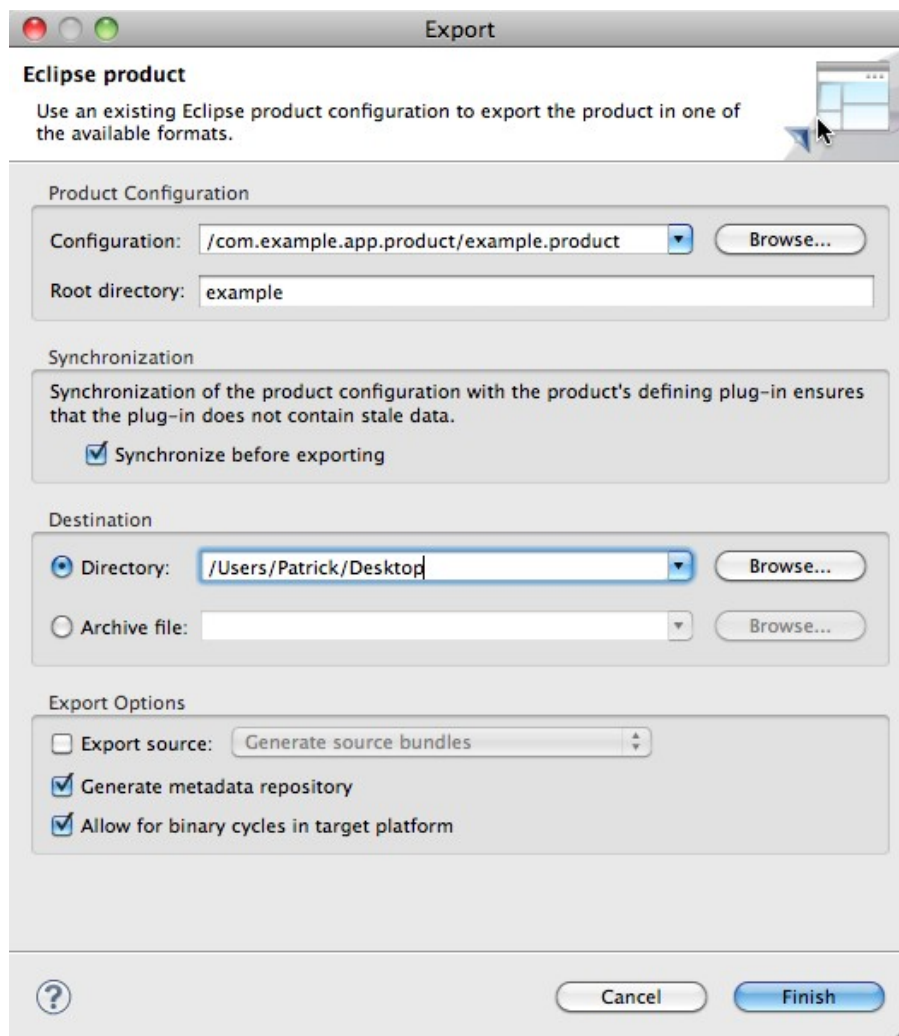
3. Click **Finish** and the export process should begin.

4. Open the `example` directory and run the application by clicking on the `eclipse` executable. Verify that the application runs correctly.
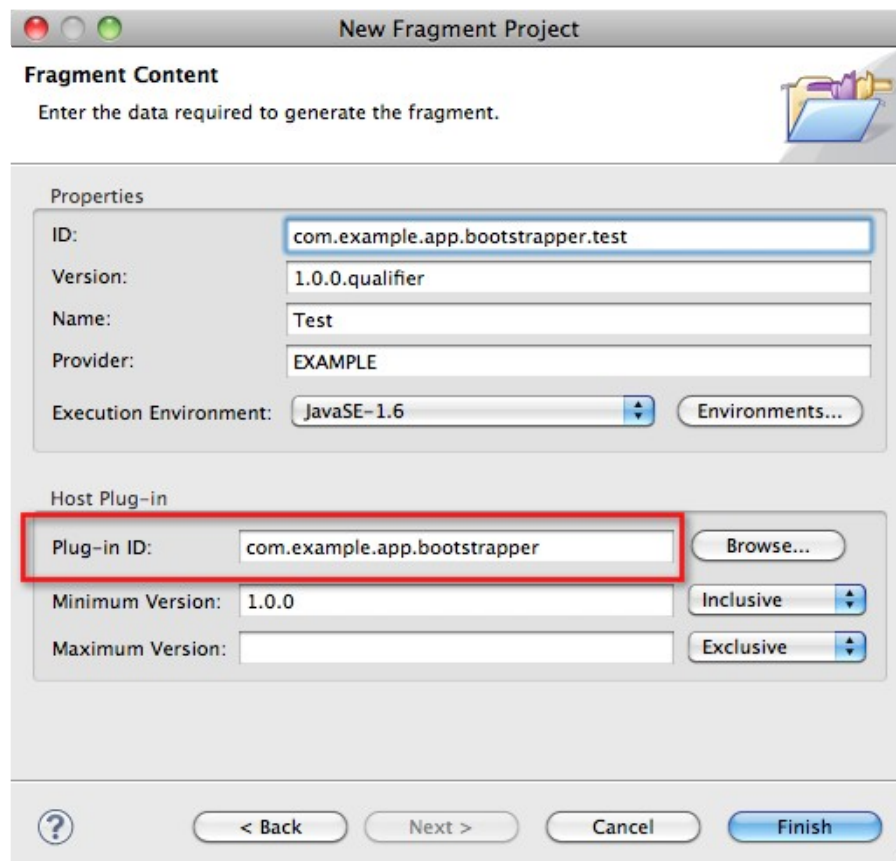
   Note that two directories are created in the destination. The `example` directory and also a `repository` directory containing your application in the form of a p2 repository. We'll explore how p2 repositories can be utilized in the next lab.

5. Shut down the example application and delete the exported directories.

Create a test fragment containing a unit test.

Create the test fragment project.

1. Select **File > New > Project..** from the main menu. Select **Plug-in Development > Fragment Project** from the list of wizards and click **Next**.

2. On the first page of the wizard, enter `com.example.app.bootstrapper.test` as the **Project name**. Leave everything else as is and click **Next**.

3. On the second page, enter `com.example.app.bootstrapper` in the **Host Plug-in > Plug-in ID** field. This is the bundle that contains the code that we will be testing. Click **Finish**.

Create a method to test and a unit test to exercise it.

1. In the `com.example.app.bootstrapper` bundle, open the `Activator` class. Add a method called `getMessage` with package visibility:

   ```
   String getMessage() {
           return "Hello!";
   }
   ```

2. In the `com.example.app.bootstrapper.test` fragment, create a package called `com.example.app.bootstrapper`.

   Note that it's important that this package have the same name as the package containing the class to test. At runtime, the test will then live in the same package and have access to non-public class members.

3. Right-click on the new package and select **New > JUnit Test Case** from the context menu. If this menu is not there, make sure you are in the Java Perspective. Create a unit test class called `ActivatorTest`.

4. Create a test method called `testGetMessage` and write an assertion verifying the expected string value.

5. Right-click on the test case and select **Run As > JUnit Plug-in Test** from the context menu. Lab 1 is now complete.