Lab 2 - Auto-updating an RCP application

The goal of this lab is to create a feature that will add functionality to our RCP application. The new feature will be automatically installed into the application using p2.

Start by pointing Eclipse at the `webapps/root/labs/lab-2` folder contained in the tutorial root. Import the projects into the workspace.

In this lab you will do the following:

1. Add p2 auto-update logic to the bootstrapper bundle.

2. Create a feature that will add a perspective to our application.

3. Publish the feature to an update site and install the feature into our application.

Add p2 auto-update logic to the bootstrapper bundle.

To add auto-update logic to the bootstrapper bundle we need to have access to the p2 framework itself. These dependencies have been added for you in order to save time.

1. Open the `example.product` file in the `com.example.app.bootstrapper.product` project. Examine the **Dependencies** tab and you'll see that the `org.eclipse.equinox.p2.user.ui` feature has been added.

   Note that in the Eclipse 3.7 release there will be a new feature called `org.eclipse.equinox.p2.rcp.feature`. This feature will not contain the IDE-specific portions of p2 and will be preferable for most RCP applications.

2. Open the `MANIFEST.MF` file in the `com.example.app.bootstrapper` bundle. Examine the **Dependencies** tab and note the additional bundles that are now required.

3. Open the `Application` class in the `com.example.app.bootstrapper` bundle.

4. With a text editor, open the `p2-auto-update.txt` file in the `extra-files` folder. Copy the contents and paste them at the bottom of the `Application` class.

5. There will be lots of compile errors because of missing imports. Right-click in the class and choose **Source > Organize Imports** from the context menu. There are a few imports with multiple suggestions. Choose `java.net.URI`, `java.util.Collection`, `org.eclipse.jface.operation.IRunnableWithProgress`, and `java.net.URI`.

6. Examine this code to get an idea of what p2 auto-update logic looks like. `doInstallOperation` is the method that will install features from a p2 repository. The p2 repository URL is identified by a system argument that we will supply later on in this lab.

7. Finally, we need to initiate the auto update process in the `start` method (around line 54). Add the following code at the very beginning of the `try` block:

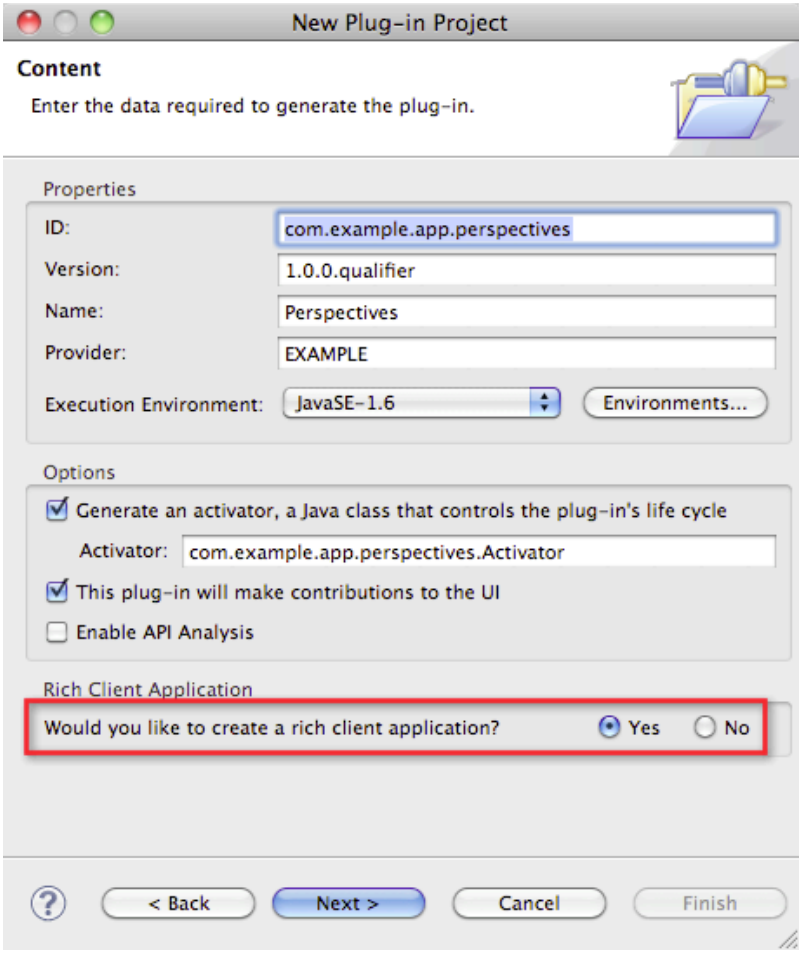
```
if (installNewFeature()) return IApplication.EXIT_RESTART;
```

Create a feature that will add a perspective to our application.

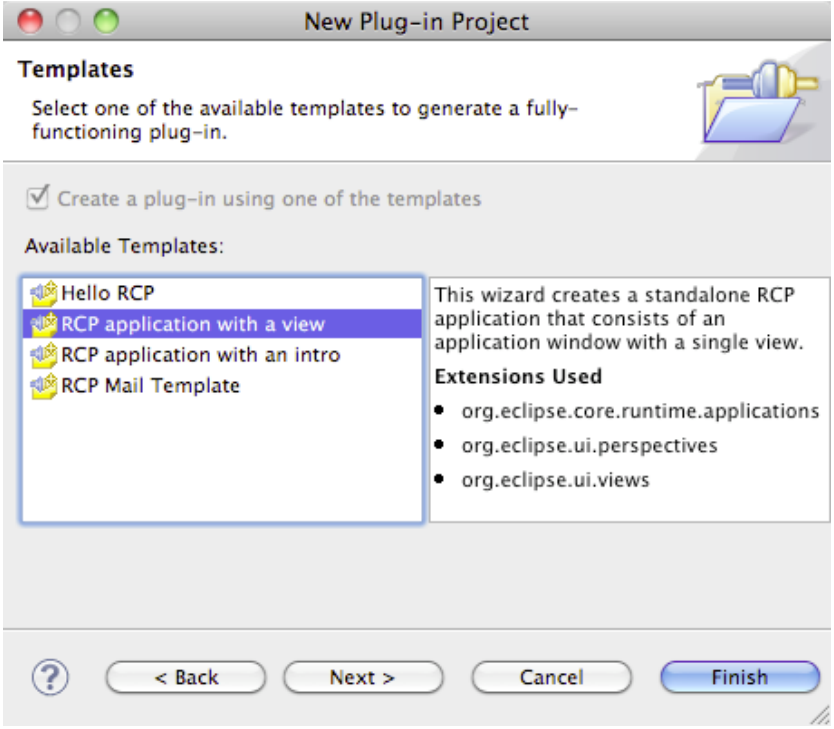
Create a bundle that contributes a perspective.

1. Create a new Plug-in project called `com.example.app.perspectives`. On the second page of the wizard, make sure to select the **Yes** radio button for **Would you like to create a rich client application?** Click **Next**.

   Note that we are not really going to create an RCP app here. This is just the easiest way to create a new bundle that contributes a perspective.



2. On the third page of the wizard, select the **RCP application with a view** template. Click **Finish**.



3. In the new project, delete the following classes:

```
Application
ApplicationActionBarAdvisor
ApplicationWorkbenchAdvisor
ApplicationWorkbenchWindowAdvisor
```

4. Open the manifest for the new project. Select the **Extensions** tab and remove the `org.eclipse.core.runtime.applications` extension.

Create a feature that will contribute the new functionality to our application.

1. Create a new Feature project called `com.example.app.perspectives.feature`. Click **Next**.

2. On the second page of the wizard, select the `com.example.app.perspectives` bundle from the list. Click **Finish**.

Create an update site project that can be used to populate a p2 repository.

1. Create a new project. Select **Plug-in Development > Update Site Project** from the wizard selection dialog. Click **Next**.

2. Name the project **com.example.app.perspectives.p2**. Click **Finish**.

3. In the `site.xml` file, add the `com.example.app.perspectives.feature` to the list.

Publish the feature to an update site and install the feature into our application.

Publish the feature.

1. In the `com.example.app.perspectives.p2` project, right click on the `site.xml` file and select **PDE Tools > Build Site** from the context menu. This will cause a p2 repository to be created inside the project.

   If you do not see this menu option, make sure you are in the Java Perspective.

Modify product configuration file to include the perspective id and the URL for fetching the application Feature.

1. Open the `eclipse.product` file and switch to the **Launching** tab.

2. Under Launching Arguments > VM Arguments, enter the following:

   ```
   -DinitialWindowPerspectiveId=com.example.app.perspectives.perspective
   -DupdateSiteUrl=http://localhost:8080/labs/lab-2/com.example.app.perspectives.p2/
   ```

   These arguments specify the perspective to load and also the URL identifying the p2 repository to search for new features.

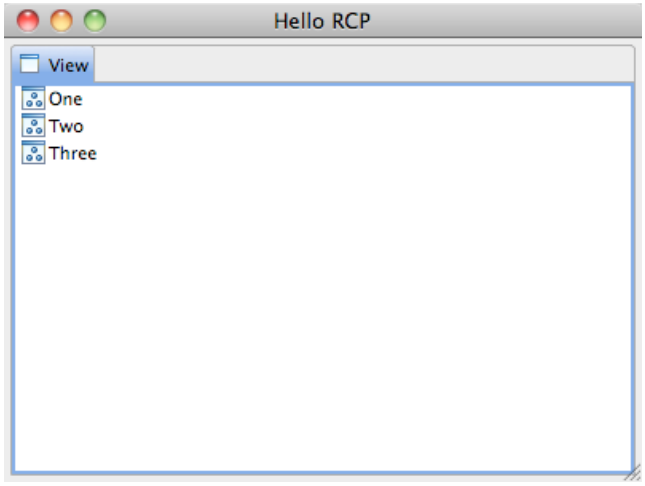
Install the bootstrapper application and run it. Verify that the feature in the update site is loaded by the auto-update process.

1. Still in the `export.product` file, switch to the **Overview** tab. Click on the **Eclipse Product export wizard** link in the **Exporting** section. Make sure that the **Generate metadata repository** checkbox is selected. This is the key to making sure that the application is created with a valid p2 profile.

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. After the export is complete, you should have two folders in the destination directory, one containing the exported application and one called `repository`. You can ignore the `repository` folder.

   The final step is to run the bootstrapper application. The auto update logic should discover and install the new feature. The contributed perspective should now appear.



4. Shut down the application and delete the `example` and `repository` folders. This lab is now complete.