

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.

Add the p2 bundles and feature.

1. Open the `example.product` file in the `com.example.app.product` project.
2. On the **Dependencies** tab, add the `org.eclipse.equinox.p2.user.ui` feature. This will cause the p2 bundles to be incorporated into the product build.
3. Select the `org.eclipse.equinox.p2.user.ui` feature in the list and click **Properties**. Enter `0.0.0` for in the **Version** field.
4. Open the `MANIFEST.MF` file in the `com.example.app.bootstrapper` bundle.
5. On the **MANIFEST.MF** tab, replace the `Require-Bundle` header with the following (keeping the leading spaces after the 1st line):

```
Require-Bundle: org.eclipse.ui,  
    org.eclipse.core.runtime,  
    org.apache.commons.codec,  
    org.apache.commons.httpclient,  
    org.eclipse.ecf,  
    org.eclipse.ecf.identity,  
    org.eclipse.ecf.filetransfer,  
    org.eclipse.ecf.provider.filetransfer,  
    org.eclipse.ecf.provider.filetransfer.httpclient,  
    org.eclipse.equinox.p2.artifact.repository,  
    org.eclipse.equinox.p2.core,  
    org.eclipse.equinox.p2.metadata,  
    org.eclipse.equinox.p2.metadata.repository,  
    org.eclipse.equinox.p2.operations,  
    org.eclipse.equinox.p2.repository,  
    org.eclipse.equinox.p2.repository.tools
```

Modify the Activator class to expose the BundleContext.

The p2 auto update code will need access to the `BundleContext` so that it can access OSGi services. To expose it, do the following:

1. Open the `Activator` class in the `com.example.app.bootstrapper` bundle.
2. Add a static variable called `context` of type `BundleContext`.
3. Assign the variable in `start`. Set it to `null` in `stop`.
4. Add a `getContext` static accessor method that returns the variable.

Modify the Application class.

1. Open the `Application` class in the `com.example.app.bootstrapper` bundle.
2. 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.
3. 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`.
4. 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:

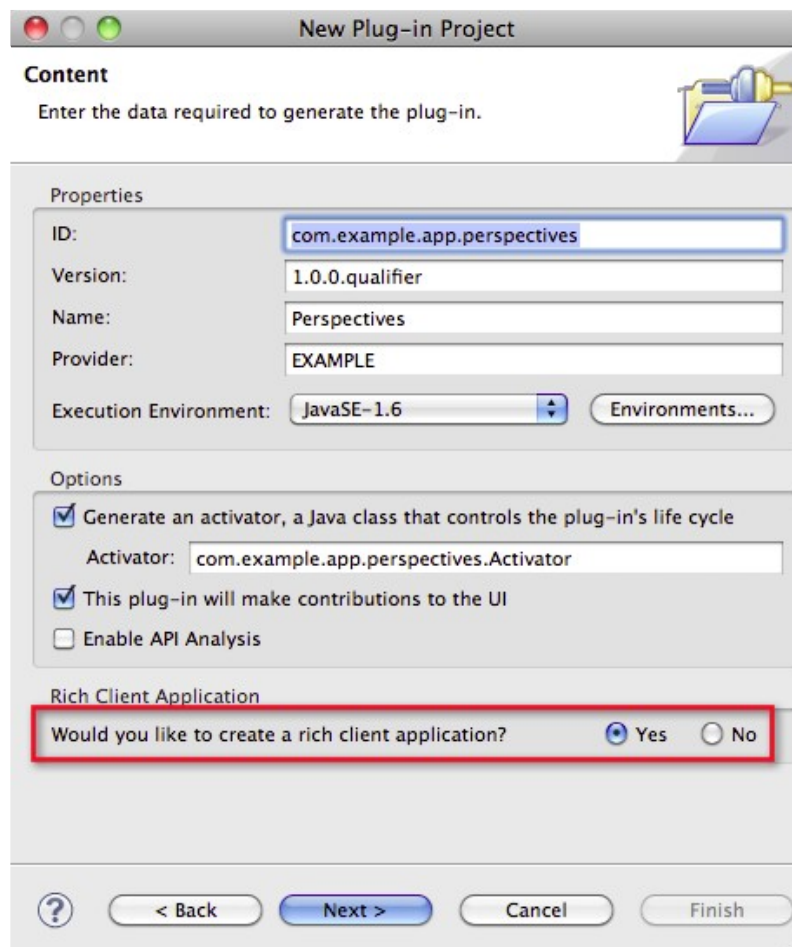
```
if (installNewFeature()) {  
    return IApplication.EXIT_RESTART;  
}
```

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.



The screenshot shows the 'New Plug-in Project' wizard dialog. The 'Content' section at the top says 'Enter the data required to generate the plug-in.' Below this is the 'Properties' section with the following fields: ID (com.example.app.perspectives), Version (1.0.0.qualifier), Name (Perspectives), Provider (EXAMPLE), and Execution Environment (JavaSE-1.6). The 'Options' section has three checkboxes: 'Generate an activator, a Java class that controls the plug-in's life cycle' (checked), 'This plug-in will make contributions to the UI' (checked), and 'Enable API Analysis' (unchecked). The 'Rich Client Application' section at the bottom has a question 'Would you like to create a rich client application?' with 'Yes' selected. The 'Next >' button is highlighted in blue.

New Plug-in Project

Content
Enter the data required to generate the plug-in.

Properties

ID:

Version:

Name:

Provider:

Execution Environment:

Options

☒ Generate an activator, a Java class that controls the plug-in's life cycle
Activator:

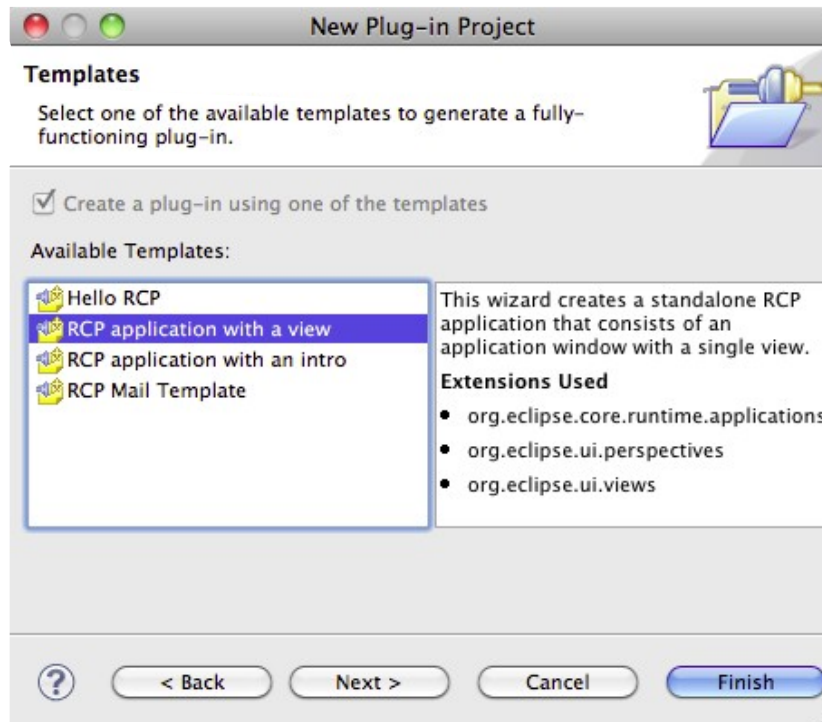
☒ This plug-in will make contributions to the UI

☐ Enable API Analysis

Rich Client Application

Would you like to create a rich client application? ☒ Yes ☐ No

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`.
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.

The URL in the auto-update code is hard coded to reference the update site in this project. It will actually being served up via an HTTP URL by the Jetty server.

Modify product configuration file to include the perspective id.

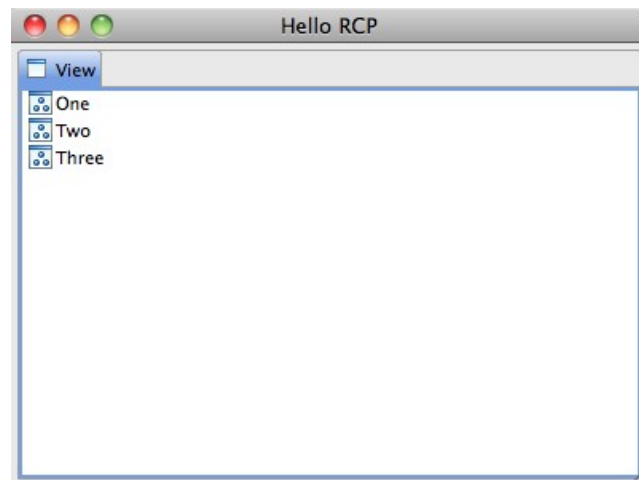
1. Open the `eclipse.product` file and switch to the **Launching** tab.
2. Under Launching Arguments > VM Arguments, enter the following:

```
-DinitialWindowPerspectiveId=com.example.app.perspectives.perspective
```

Install the bootstrapper application so that it has valid p2 metadata.

1. Export the application using the link in the `example.product` file. 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.