



Developing OSGi Enterprise Applications

using the free IBM® Rational® Development Tools for OSGi Applications

Lab Instructions

Authors:

Roland Barcia, ISSW Web 2.0 Lead Architect

Tim deBoer, RAD OSGi Architect

Jeremy Hughes, WAS OSGi Applications Architect

Alasdair Nottingham, WAS OSGi Applications Architect

Key reference notes for this lab

Lab objectives

In this lab you will learn:

- How to build OSGi bundles
- How to use OSGi Enterprise features in bundles, including Web, JPA 2.0, and Blueprint
- How to put OSGi bundles together into an application
- How to run the application in the Apache Aries (Incubating) project.

Prerequisite knowledge

- Basic Eclipse IDE for Java developers knowledge
- Basic Java Enterprise or OSGi background

Table of Contents

Section 1 - Overview	4
A. Copy the lab resources	4
B. Install Eclipse and the prerequisite Eclipse packages	4
C. Install IBM® Rational® Development Tools for OSGi Applications	6
Section 2 - Create the target definition.....	7
D. Download the target platform jars	7
E. Configure the target platform	7
Section 3 - Create API Bundle	10
F. Create an API Bundle Project	10
G. Import API Classes.....	12
H. Export Packages from Bundle	13
Section 4 - Create Persistence Bundle	15
I. Create Persistence (JPA) Bundle Project	15
J. Import API packages	18
K. Import Persistence Classes	21
L. Import persistence.xml	22
M. Extend persistence.xml	23
N. Enhance the persistence entities	24
O. Create Blueprint File.....	26
P. Create Blueprint Bean and Service	28
Q. Add Blueprint container managed transaction and persistence configuration ..	29
Section 5 - Create Web Bundle.....	30
R. Create Web Bundle Project	30
S. Import Packages.....	31
T. Importing Servlets	33
U. Importing Web Content	34
Section 6 - Create OSGi Application	35
V. Create an OSGi Application Project	35
Section 7 - Testing	38
W. Create an OSGi Framework run configuration	38
X. Create Blog Authors	40
Y. Create Blog Entries	40

Section 1 - Overview

In this lab you'll be creating a simple blogging website using the free IBM® Rational® Development Tools for OSGi Applications. The content you'll be creating has been adapted from the Apache Aries Blog Sample. Apache Aries is the open source project that delivers a set of pluggable Java components enabling an enterprise OSGi application programming model. This includes implementations and extensions of application-focused specifications defined by the OSGi Alliance Enterprise Expert Group (EEG) and an assembly format for multi-bundle applications, for deployment to a variety of OSGi based runtimes.

During this lab you'll be creating three OSGi bundles:

- A bundle that contains some API interfaces.
- A bundle that contains JPA persistence.
- A bundle that contains a simple web UI.

The content of the application has been split into these bundles to show the modularity of OSGi and is based on best practices for larger applications.

These bundles will be packaged in an OSGi application and tested on the Apache Aries Blog assembly target platform. This set of Apache Aries bundles and dependencies provide a test environment for OSGi applications.

This lab was created using the IBM® Rational® Development Tools for OSGi Applications and the Apache Aries project.

A. Copy the lab resources

1. Copy the /Resources directory to a place on your hard disk.

B. Install Eclipse and the prerequisite Eclipse packages

1. Download an Eclipse 3.6 package specific to your operating system from:

<http://eclipse.org/downloads/>

for example the 'Eclipse IDE for Java Developers' Windows 32 package:
eclipse-java-helios-win32.zip - then unzip it to your hard drive.

2. Ensure you have a JDK on your operating system path.
3. Start the IDE by running eclipse.exe (Windows)
4. When prompted for a workspace location, provide the location of an empty directory on disk.
5. Select **Help > Install New Software ...**

- When the Install window appears select the Helios update site:



- When the list of available software has refreshed, open **General Purpose Tools** and select **Marketplace Client**.
- Open **Web, XML and Java EE Development** and select the following:

Web, XML, and Java EE Development		
<input type="checkbox"/>	Apache MyFaces Trinidad Tag Support (Optional)	2.2.100.v20100521
<input type="checkbox"/>	Axis2 Tools	1.1.100.v20100521
<input type="checkbox"/>	CXF Web Services	1.0.0.v20100521
<input checked="" type="checkbox"/>	Dali Java Persistence Tools	2.3.0.v20100521
<input type="checkbox"/>	Dali Java Persistence Tools - EclipseLink Support (Optional)	2.3.0.v20100521
<input checked="" type="checkbox"/>	Eclipse Faceted Project Framework	3.2.0.v20100521
<input checked="" type="checkbox"/>	Eclipse Faceted Project Framework JDT Enablement	3.2.0.v20100521
<input checked="" type="checkbox"/>	Eclipse Java EE Developer Tools	3.2.0.v20100521
<input checked="" type="checkbox"/>	Eclipse Web Developer Tools	3.2.0.v20100521
<input checked="" type="checkbox"/>	Eclipse XML Editors and Tools	3.2.0.v20100521
<input checked="" type="checkbox"/>	Eclipse XSL Developer Tools	1.1.0.v20100521
<input checked="" type="checkbox"/>	JavaScript Development Tools	1.2.0.v20100521
<input checked="" type="checkbox"/>	JavaServer Faces Tools (JSF) Project	3.2.0.v20100521
<input type="checkbox"/>	JAX-WS DOM Tools	1.0.0.v20100521
<input type="checkbox"/>	JAX-WS Tools	1.0.0.v20100521
<input type="checkbox"/>	JST Server Adapters	3.2.0.v20100521
<input type="checkbox"/>	JST Server Adapters	3.2.0.v20100521
<input type="checkbox"/>	JST Server UI	3.2.0.v20100521
<input checked="" type="checkbox"/>	JST Web UI	3.2.0.v20100521
<input type="checkbox"/>	PHP Development Tools (PDT) SDK Feature	2.2.0.v20100521
<input checked="" type="checkbox"/>	Rich Ajax Platform (RAP) Tooling	1.3.0.v20100521

You should now have 12 items selected, including the **Marketplace Client** and those listed above.

9. Click on **Next** and **Next** again.
10. Accept the Eclipse Foundation Software User Agreement and click **Finish**.
11. Click the **Restart Now** button after the packages have been downloaded.

C. Install IBM® Rational® Development Tools for OSGi Applications

1. Select **Help > Eclipse Marketplace ...** then with 'Eclipse Marketplace' highlighted, click **Next**.
2. After the index has downloaded, type "OSGi Applications" into the search box. The first result returned should be:



- Click **Install** on this entry, then **Next**.
3. Ensure you are familiar with the license, accept it and click **Finish**.
 4. Click the **Restart Now** button after the packages have been downloaded.

Section 2 - Create the target definition

In this section you'll define an Eclipse Target Platform. The target platform is the platform you are developing for, the set of bundles prerequisite bundles your workspace requires at build time and run time.

The default Eclipse target platform is used for building and running Eclipse Plug-in projects. We need a target platform definition describing the bundles required to build and run an Apache Aries application. It will list the OSGi framework, Apache Aries, and related bundles needed to run the Blog application.

The target platform we will use is from the samples module in the Apache Aries project. If you don't already have it, download Maven v2.2.1 from <http://maven.apache.org/download.html> and follow the installation instructions for your operating system at the bottom of the page.

Use Maven to create the target platform:

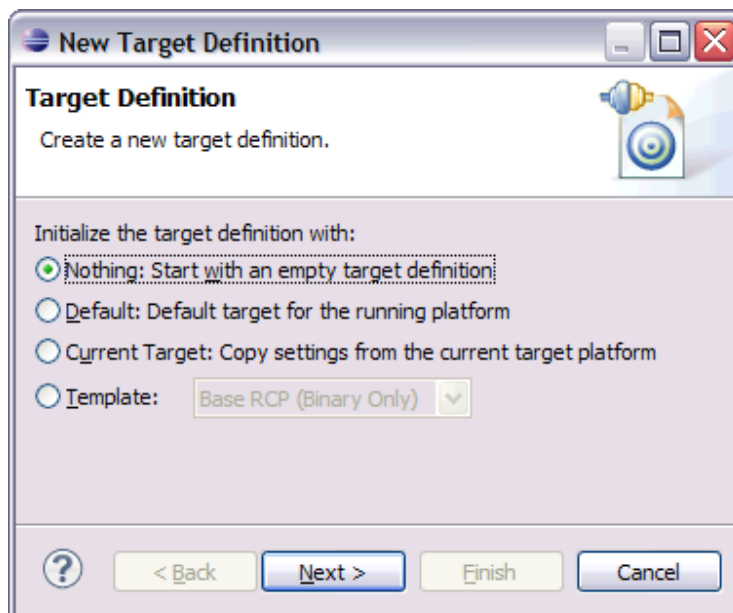
D. Download the target platform jars

1. Copy the **Resources/blog-assembly** directory to an empty directory on your hard disk.
2. At the command line in the **blog-assembly** directory you just copied, run the **mvn** command. You now have a subdirectory called **target** which contains the bundles required for the Eclipse target platform.

E. Configure the target platform

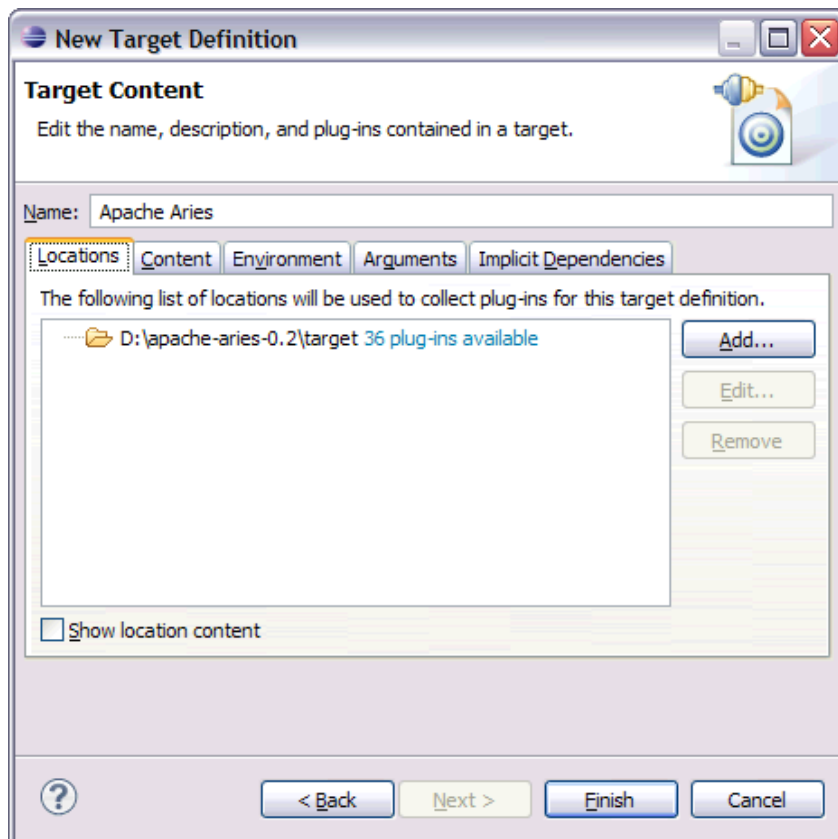
To configure Eclipse to the target you have just created:

3. Select **Window > Preferences ...**
4. **Plug-in Development > Target Platform** then click **Add ...**



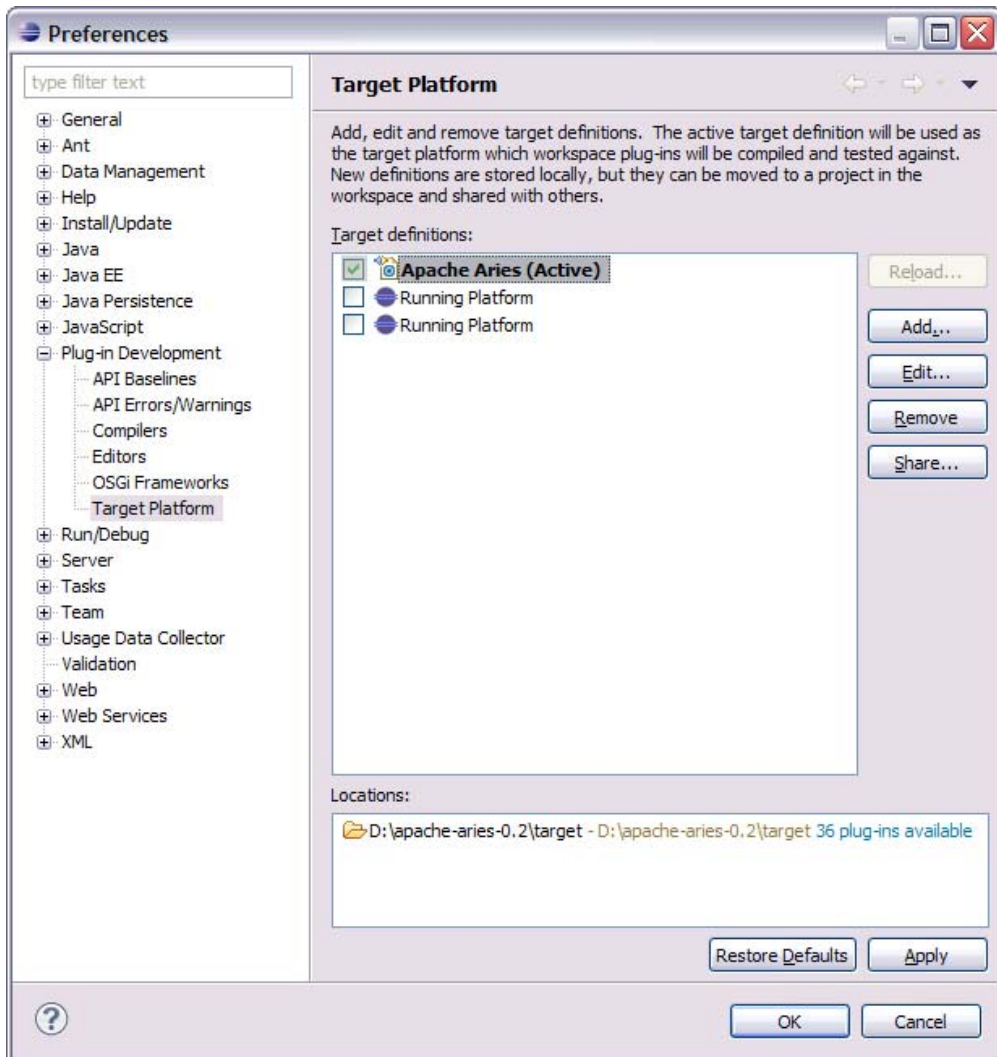
5. We're going to start with an empty target definition and add the target directory of bundles we have just created. Click **Next**.

6. Enter "**Apache Aries**" in the Name field.
7. Add the location of the **target** directory by clicking **Add ...**
8. Out of the four options presented, select **Directory**
9. Click **Next**
10. Enter the location of the **target** directory created earlier.
11. Click **Finish**. Eclipse should have detected 36 plug-ins and added them to the target definition.



12. Click **Finish**

13. Now select Apache Aries as the active target platform.



14. Click **OK**.

Section 3 - Create API Bundle

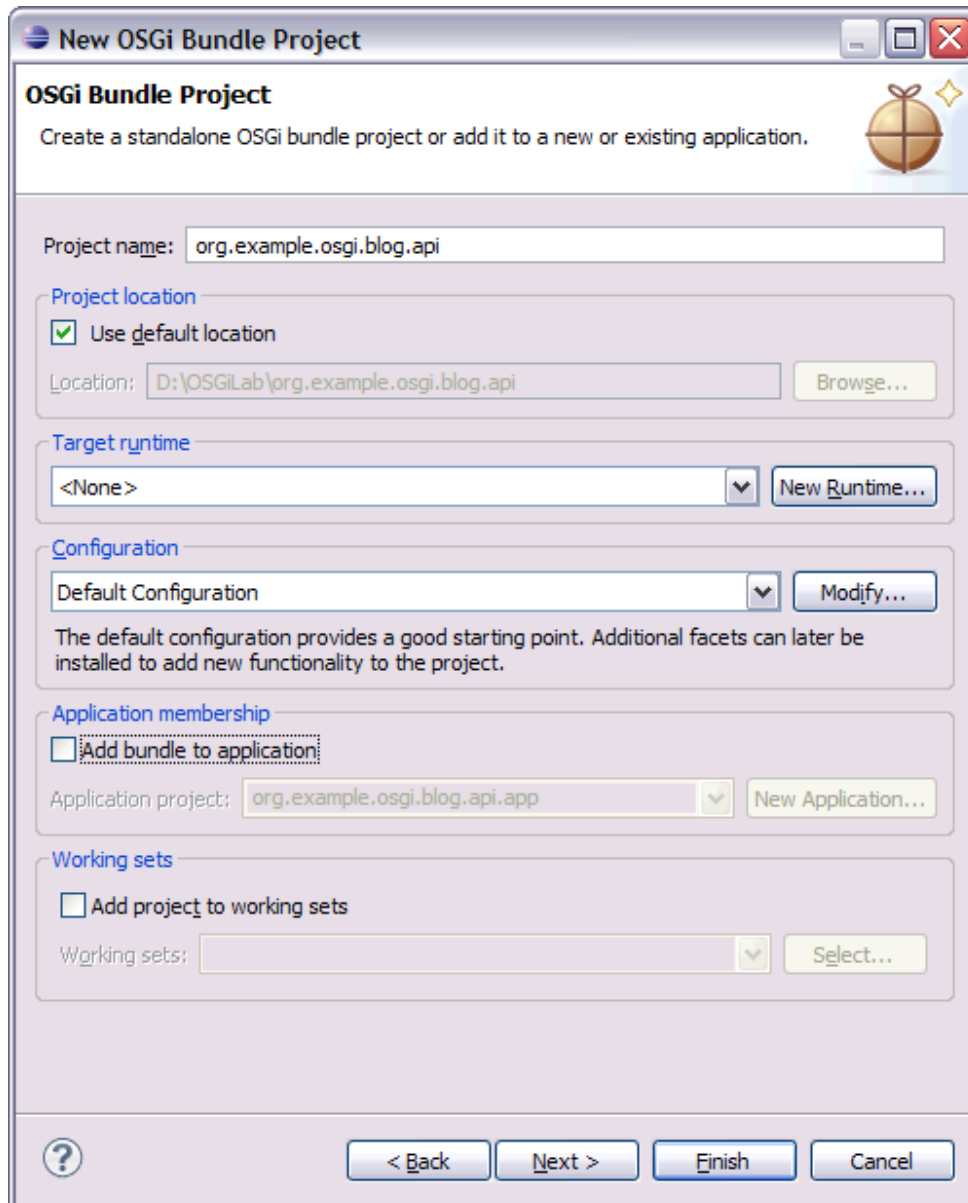
In this section you'll create a simple bundle that contains API interfaces used by the rest of the application. The OSGi standard provides an environment for the modularization of applications into bundles. OSGi bundles are normal Java jar files with extra metadata in the Manifest files. The OSGi runtime uses this metadata to find dependencies, manage the life-cycle of the bundle, and many other services.

F. Create an API Bundle Project

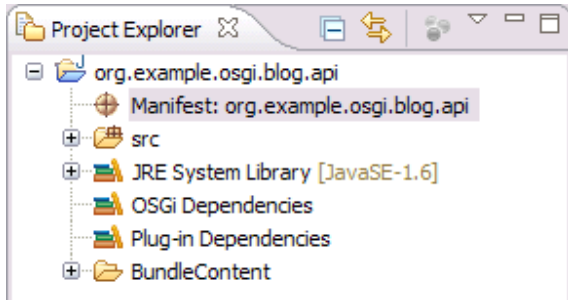
First, we'll create a bundle that contains the API classes that will be used later by both our persistence and web bundles. This is a best practice, as it would allow alternate implementation bundles to be created without duplicating the API classes or requiring changes to any client bundles.

1. Select **File > New > Project...**
2. Select **OSGi > OSGi Bundle Project**. Click **Next**.
3. In the **Project Name** field, type **org.example.osgi.blog.api**.

4. Uncheck **Add bundle to application**. We'll be creating an application later.



5. Click **Finish**.
6. Use the **Project Explorer** view in the **Java EE** perspective to view the additional information the OSGi Applications tools adds.

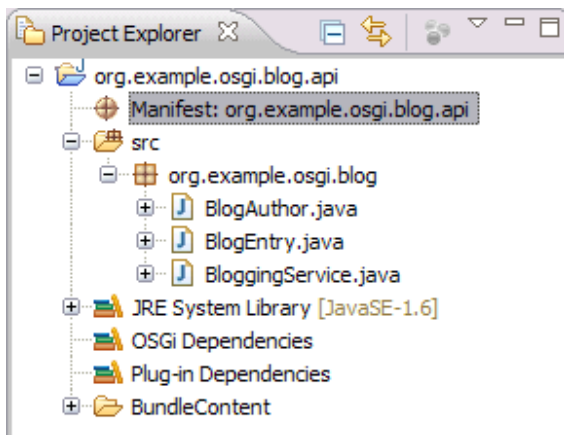


G. Import API Classes

We'll be copying some pre-canned resources from the desktop during this lab, to avoid unnecessary time typing. Perform the following steps:

1. Open the **Resources** folder on the desktop.
2. Open the **api** folder.
3. Drag or copy the **org** folder to the **org.example.osgi.blog.api/src** folder within the workspace.

Expand the **org.example.osgi.blog** package within Eclipse and explore the three new interfaces that have been created.¹ **BlogAuthor** and **BlogEntry** will be the persistence objects, and **BloggingService** is our manager class for creating and querying these objects.

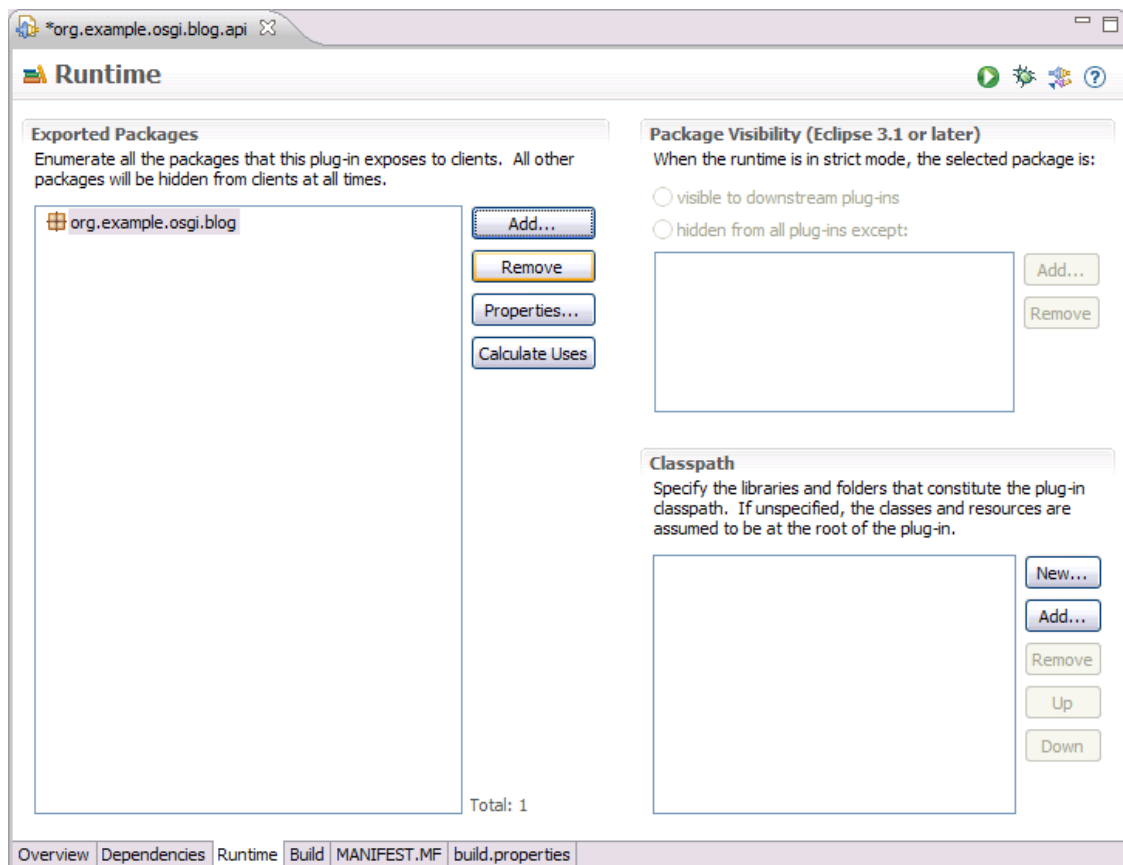


¹ You may need to select the src folder in Eclipse and hit the F5 key to refresh the view for the artifacts to be made visible.

H. Export Packages from Bundle

Packages must be exported from a bundle before they are available for downstream bundles to use.

1. Expand the **org.example.osgi.blog.api** project in the **Project Explorer** view.
2. Open the manifest editor by double-clicking on the **Manifest: org.example.osgi.blog.api** node.
3. Select the **Runtime** tab.
4. Under **Exported Packages**, click **Add**.
5. Select **org.example.osgi.blog** and click **OK**.
6. Click on the **Save** toolbar button or hit **Ctrl-S** to Save.



Let's take a look at the source to see what our manifest looks like:

7. Click on the **MANIFEST.MF** tab at the bottom to switch to the source view.
8. Notice the **Export-Package:** entry that we've just added.

9. When you are finished, close the editor.

Bundle-Name: Defines a human-readable name for this bundle, simply assigns a short name to the bundle.

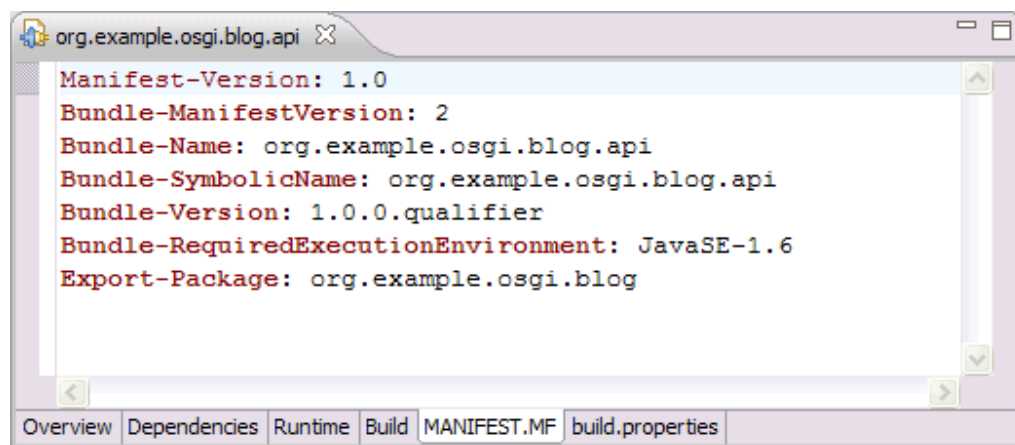
Bundle-SymbolicName: The only required header, this entry specifies a unique identifier for a bundle, based on the reverse domain name convention.

Bundle-ManifestVersion: This header indicates the OSGi specification to use for reading this bundle.

Bundle-Version: Designates a version number to the bundle.

Bundle-RequiredExecutionEnvironment: Specifies the runtime version, in this case Java 6.

Export-Package: Expresses what Java packages contained in a bundle will be made available to the outside world.



The screenshot shows a window titled 'org.example.osgi.blog.api' with a scrollable text area containing the following manifest entries:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: org.example.osgi.blog.api
Bundle-SymbolicName: org.example.osgi.blog.api
Bundle-Version: 1.0.0.qualifier
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Export-Package: org.example.osgi.blog
```

At the bottom of the window, there is a tabbed interface with the following tabs: Overview, Dependencies, Runtime, Build, MANIFEST.MF (selected), and build.properties.

Section 4 - Create Persistence Bundle

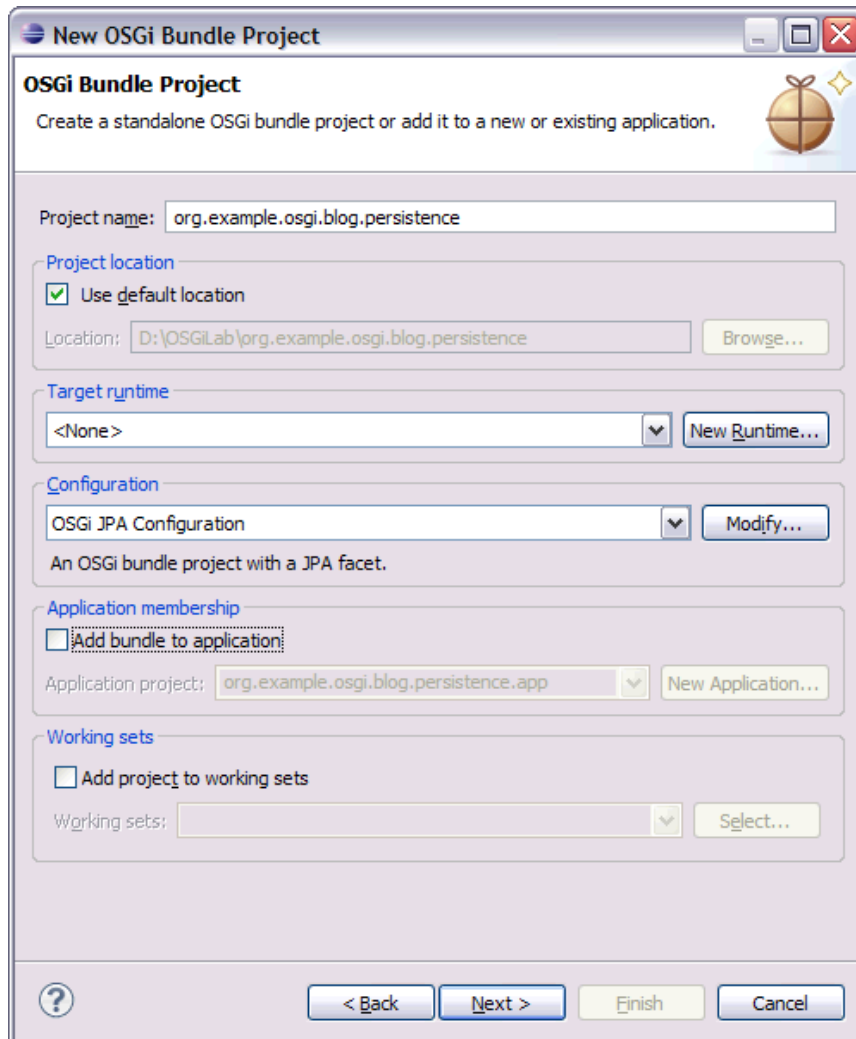
In this section we'll create a persistence bundle, implement some persistence classes, and create a blueprint service to expose the persistence classes to the rest of the application.

I. Create Persistence (JPA) Bundle Project

First, let's create a bundle that provides persistence support for the interfaces we just created.

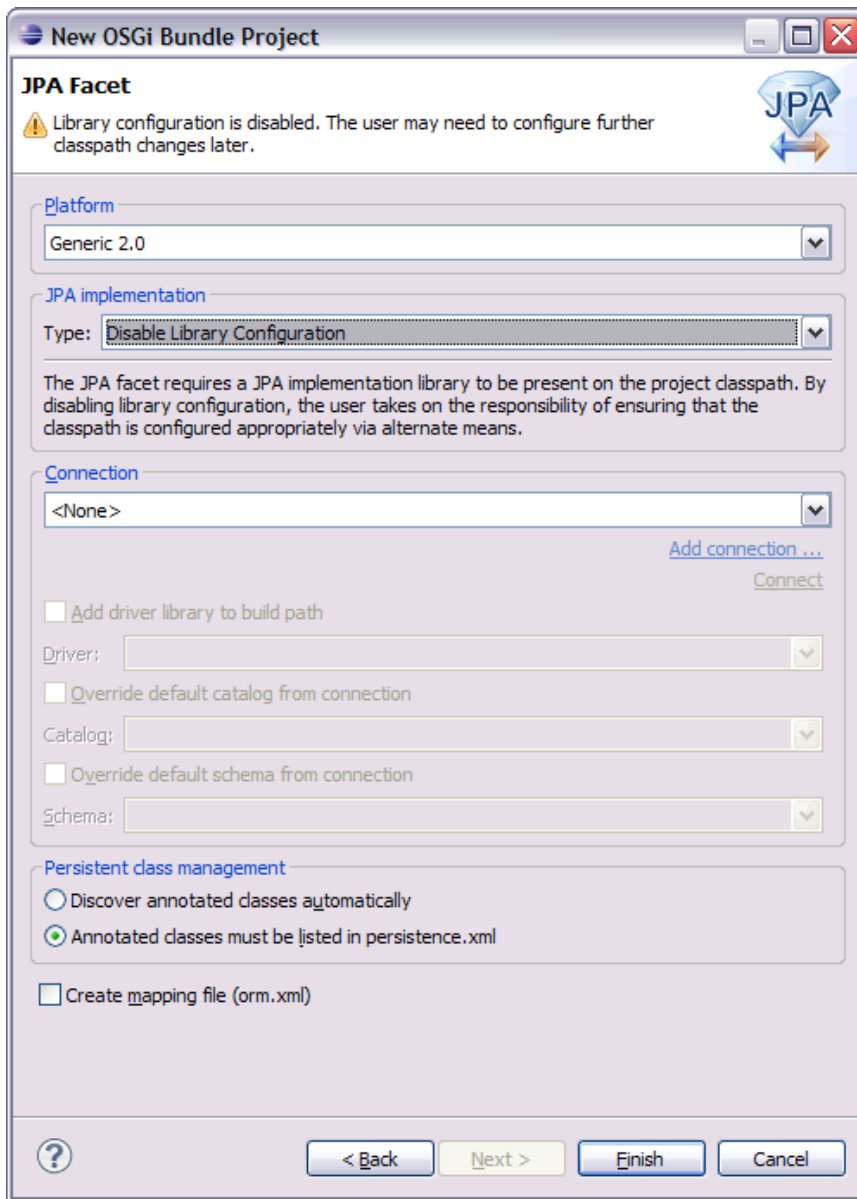
1. Select **File > New > Project**.
2. Select **OSGi > OSGi Bundle Project**. Click **Next**.
3. In the **Project Name** field, type **org.example.osgi.blog.persistence**
4. Change **Configuration** to **OSGi JPA Configuration**.

5. Uncheck **Add bundle to application**.



6. Click **Next** three times to see the **JPA Facet** page. There is an error mark indicating no JPA implementation library has been specified. The JPA implementation is provided by the **Target Platform** so can be disabled in this page.

7. Change the **JPA implementation** to 'Disable Library Configuration'



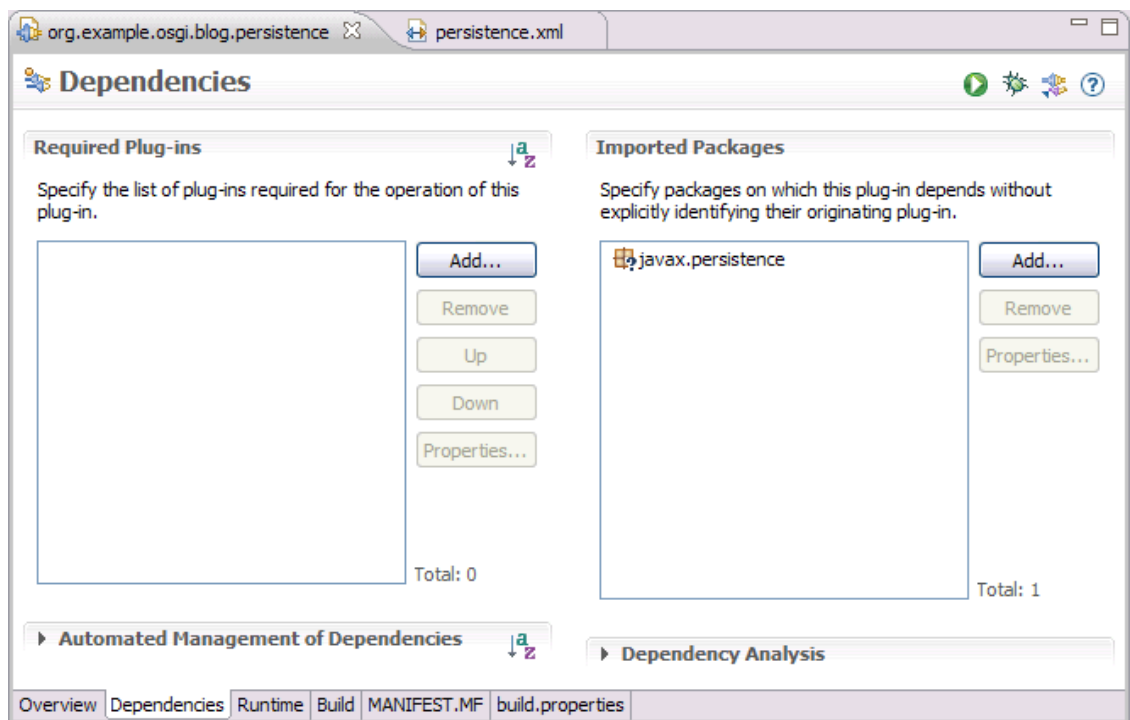
8. Click **Finish**.

J. Import API packages

All packages must be imported before they are visible to your bundle. For the persistence bundle, we'll need both the API package we just created and the standard JPA persistence packages.

Note: there is an issue with the runtime build used in this lab that will cause the versions of the JPA 2.0 packages to be incorrect. We'll simply remove the version from these packages to avoid any problems.

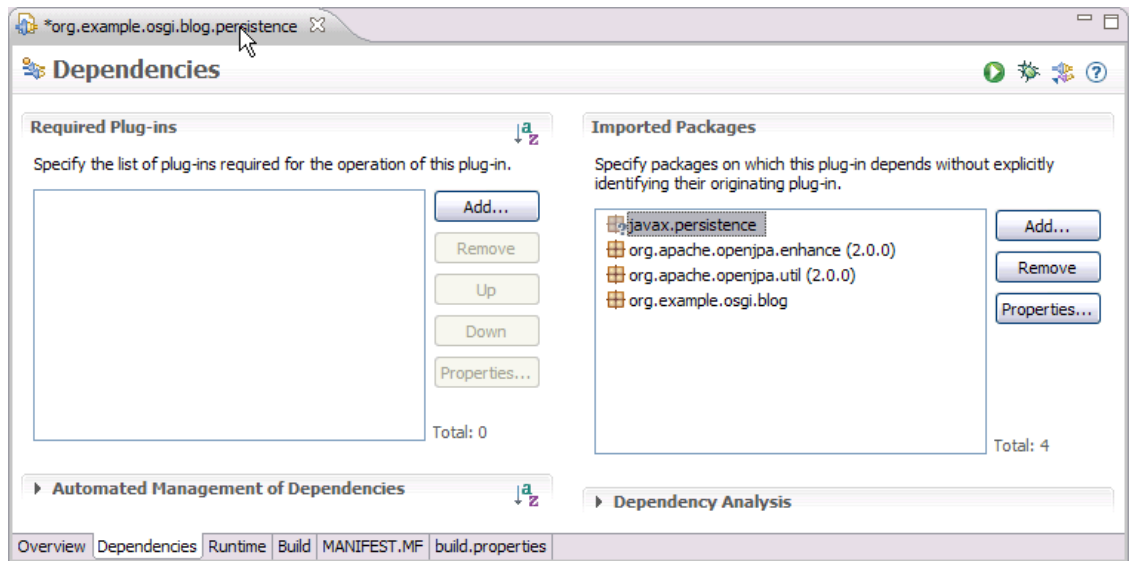
1. Expand the **org.example.osgi.blog.persistence** project in the **Project Explorer** view.
2. Open the manifest editor by double-clicking on the **Manifest: org.example.osgi.blog.persistence** node.
3. Select the **Dependencies** tab. The **javax.persistence** package has already been added to the **Imported Packages**.



4. Under **Imported Packages**, click **Add...**
5. Select **org.example.osgi.blog**. Click **OK**.

Section N discusses entity enhancement using Apache OpenJPA. After entities have been enhanced, they will depend on two packages from the OpenJPA project. To avoid a `ClassNotFoundException` at runtime, those OpenJPA packages must be imported as well.

6. Add the following additional packages to the **Imported Packages** list: **org.apache.openjpa.enhance** and **org.apache.openjpa.util**



7. Click on the **Save** toolbar button or hit **Ctrl-S** to Save.

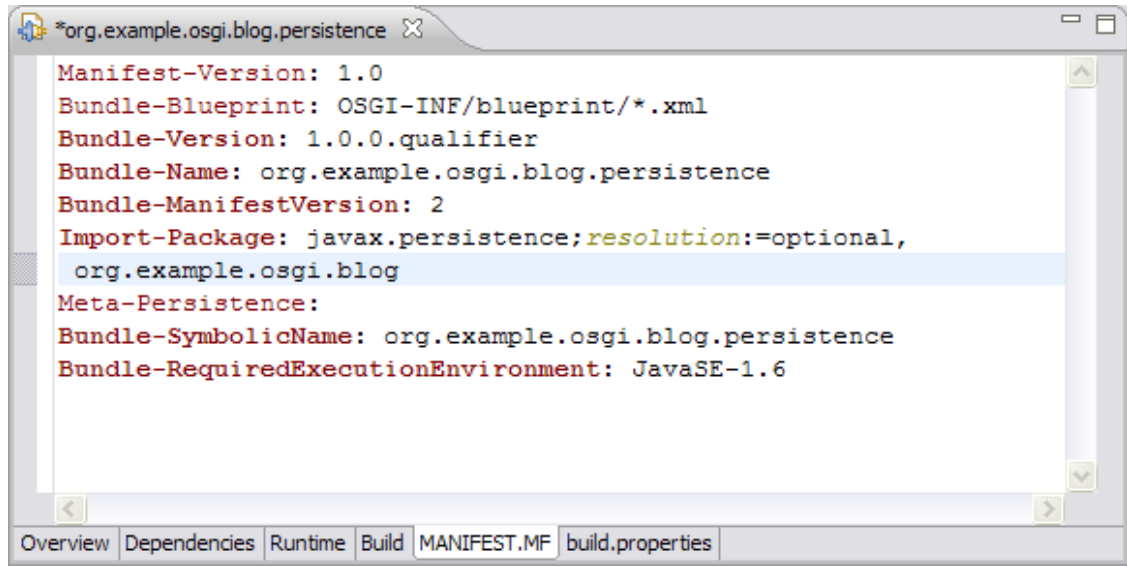
Let's take a look at the source to see what our manifest looks like:

8. Click on the **MANIFEST.MF** tab at the bottom to switch to the source view.
9. Notice the **Import-packages:** entry that we've just added.
10. Notice the **Meta-Persistence:** field that was automatically added by the tools. This marks the bundle as containing JPA content.
11. When you are finished, close the editor.

Most of the fields are the same from the previous bundle, however, there are a couple of new ones:

Meta-Persistence: This is a pointer to the location of the persistence code, in our case JPA's persistence.xml

Import-Packages: Indicates what Java packages will be required from the outside world, in order to fulfill the dependencies needed in a bundle. This includes our package from the previous bundles and packages from the JPA bundles.



K. Import Persistence Classes

Let's import some classes to save time, just like we did with the API bundle.

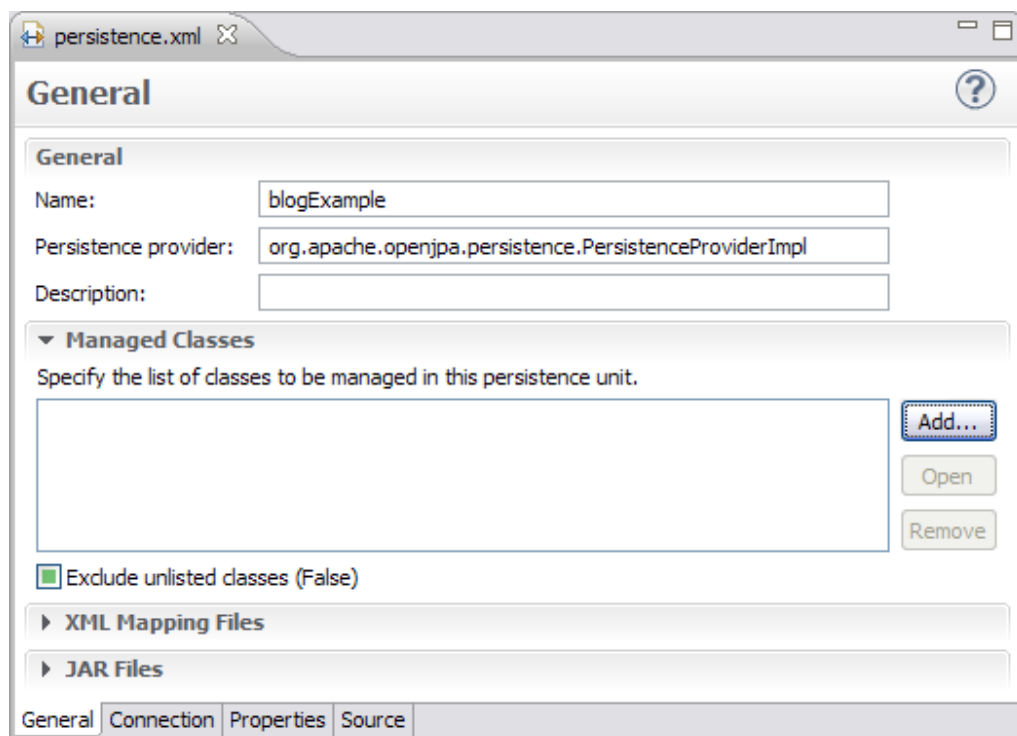
1. Open the **Resources** folder on the desktop.
2. Open the **persistence** folder.
3. Drag or copy the **org** folder into the **org.example.osgi.blog.persistence/src** folder within the workspace.
4. Expand the **org.example.osgi.blog.persistence.internal** package and explore the three new classes that have been created.² **AuthorImpl** and **EntryImpl** are persistence objects, and **BloggngServiceImpl** is a JPA manager bean that we'll later turn into a blueprint service.

² You may need to select the src folder in Eclipse and hit the F5 key to refresh the view for the artifacts to be made visible.

L. Import persistence.xml

The persistence.xml file that's automatically created in the workspace is empty. To avoid typos while entering the provider or datasource name, we'll import a copy of the file that contains this information.

1. Open the **Resources** folder on the desktop.
2. Open the **persistence** folder.
3. Drag or copy **persistence.xml** into the **org.example.osgi.blog.persistence/src/META-INF** folder within the workspace. If you are prompted to overwrite, click **Yes**.
4. Open the **org.example.osgi.blog.persistence/src** folder in the **Project Explorer**. Select either the **src** folder or the **META-INF** folder and hit the F5 key to induce Eclipse to refresh its content to pick up the new persistence.xml you just dropped in the **META-INF** folder.
5. Expand the **org.example.osgi.blog.persistence/src/META-INF** folder within Eclipse and double-click on **persistence.xml**. Expand the **Managed Classes** section and you should see a view as shown in the figure below.



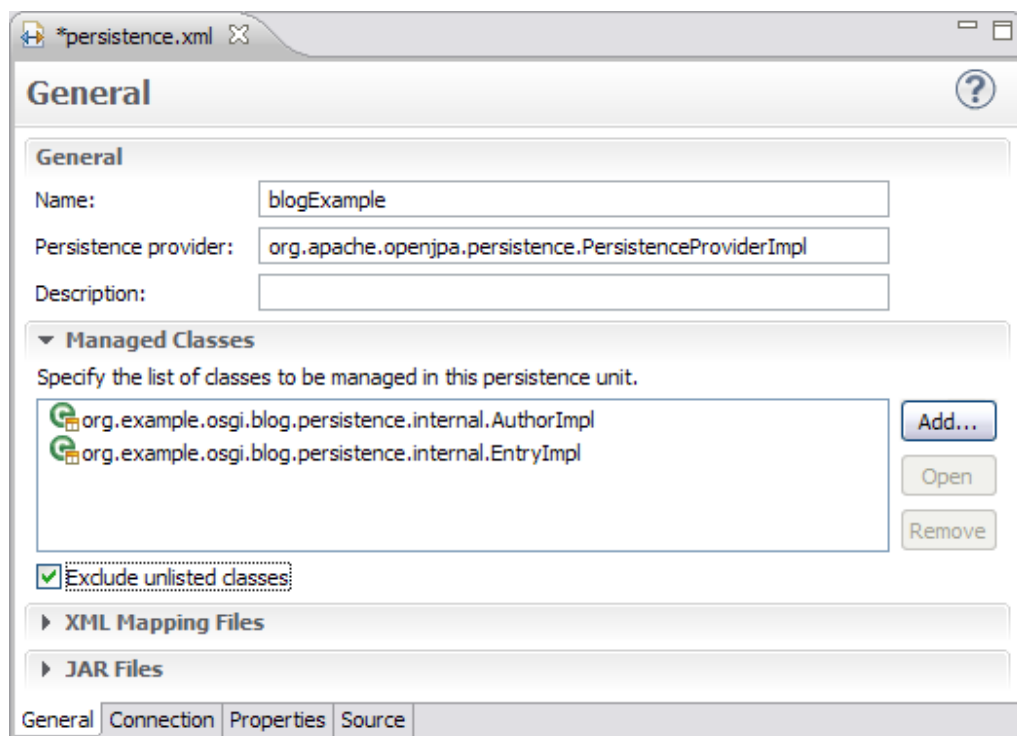
M. Extend persistence.xml

There are two errors listed in the **Markers** view:

✘ Class "AuthorImpl" is mapped, but is not included in any persistence unit	AuthorImpl.java
✘ Class "EntryImpl" is mapped, but is not included in any persistence unit	EntryImpl.java

To complete the persistence.xml, we need to do is list our persistence classes in the persistence.xml and ensure at run time JPA will only use the listed classes.

1. In the **Managed Classes** section click **Add ...**
2. Type **AuthorImpl** and select the class found. Click **OK**.
3. Add **EntryImpl** in the same way.
4. Check the box marked **Exclude unlisted classes** which defaults to false.



5. Click on the **Save** toolbar button or hit **Ctrl-S** to Save.

The red markers have gone.

N. Enhance the persistence entities

Enhancement is the process of modifying the entity classes such that the JPA runtime can monitor the entity objects. The entity classes' bytecode is 'enhanced' with calls to the JPA provider. This is done as an additional step after the entity classes have been built, and can be automated by configuring an Eclipse Builder to run the OpenJPA enhancer.

1. Right click the **org.example.osgi.blog.persistence** bundle then select **Properties...**
2. Select **Builders** from the list of properties then click the **New...**
3. Select **Program** from the list of configuration types and click **OK**

A launch configuration window will be displayed.

4. In the **Name** field enter **Enhance Entity Classes**
5. In the **Location** field enter the location of your **java** executable. If you have the **JAVA_HOME** environment variable configured correctly then you can use:

```
${env_var:JAVA_HOME}/bin/java.exe (for Windows)
${env_var:JAVA_HOME}/bin/java (for Unix/Linux)
```

6. In the **Arguments** field enter:

```
-Djava.ext.dirs=${target_home} -cp
${workspace_loc:/org.example.osgi.blog.api/bin};${workspace_loc:/o
rg.example.osgi.blog.persistence/src}
org.apache.openjpa.enhance.PCEnhancer (for Windows)
```

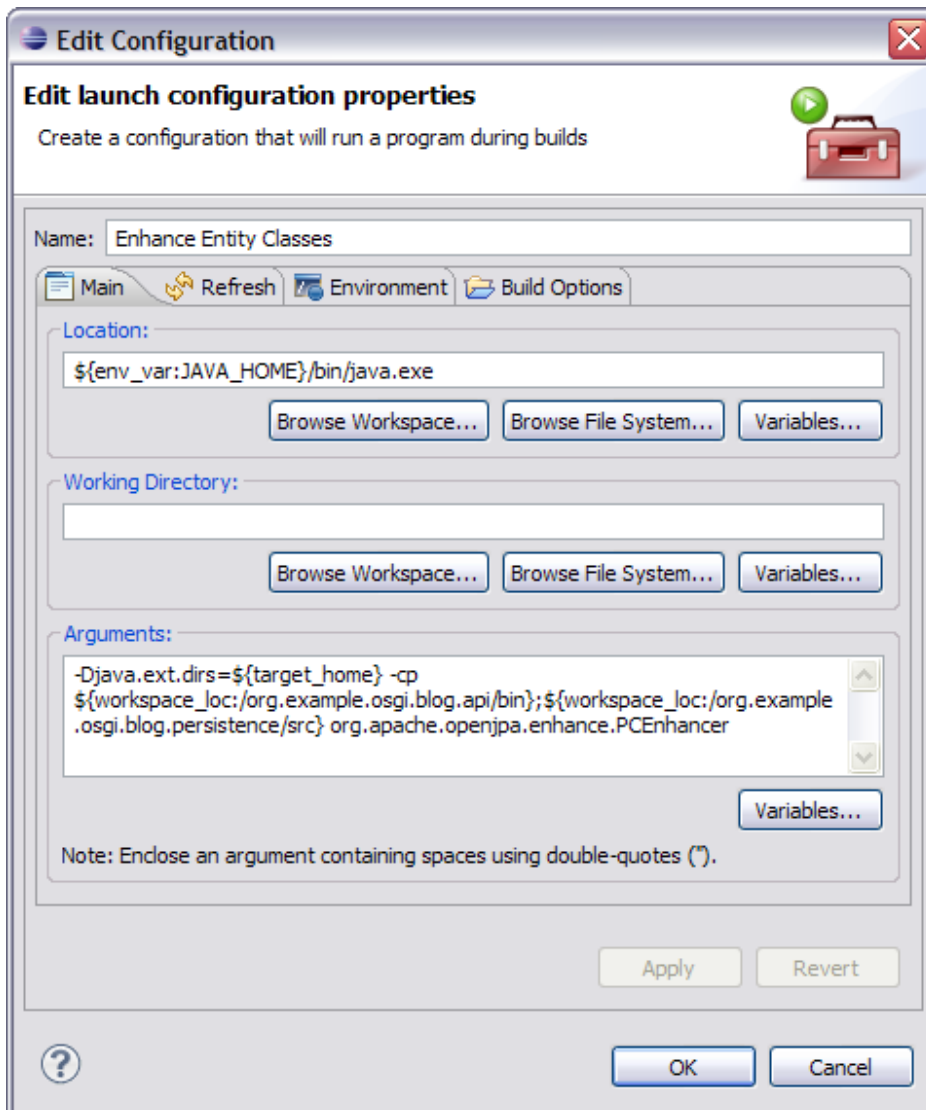
```
-Djava.ext.dirs=${target_home} -cp
${workspace_loc:/org.example.osgi.blog.api/bin}:${workspace_loc:/o
rg.example.osgi.blog.persistence/src}
org.apache.openjpa.enhance.PCEnhancer (for Unix/Linux)
```

To prevent typo's you can copy and paste one of the above from the **Resources/enhancement-run-configuration-snippet.txt** file.

Note: If your `${target_home}` variable contains a space character then you need to enclose the variable in ". If your `${workspace_loc}` variable include a space character then you need to enclose the complete value of the `-cp` flag in " e.g. (Windows)

```
"${workspace_loc:/org.example.osgi.blog.api/bin}:${workspace_loc:/
org.example.osgi.blog.persistence/src}"
```

7. Click **Apply**.



8. Click **OK**
9. Click **OK** to finish editing the project's properties.

Now, Eclipse will ensure the entities are enhanced by running the Apache OpenJPA enhancer to enhance the entity classes.

The persistence support in this bundle is complete.

O. Create Blueprint File

The OSGi Blueprint service is a standardization of Dependency Injection Programming Models. Companies like IBM, SpringSource (now VMWare), and others have worked on this standard.

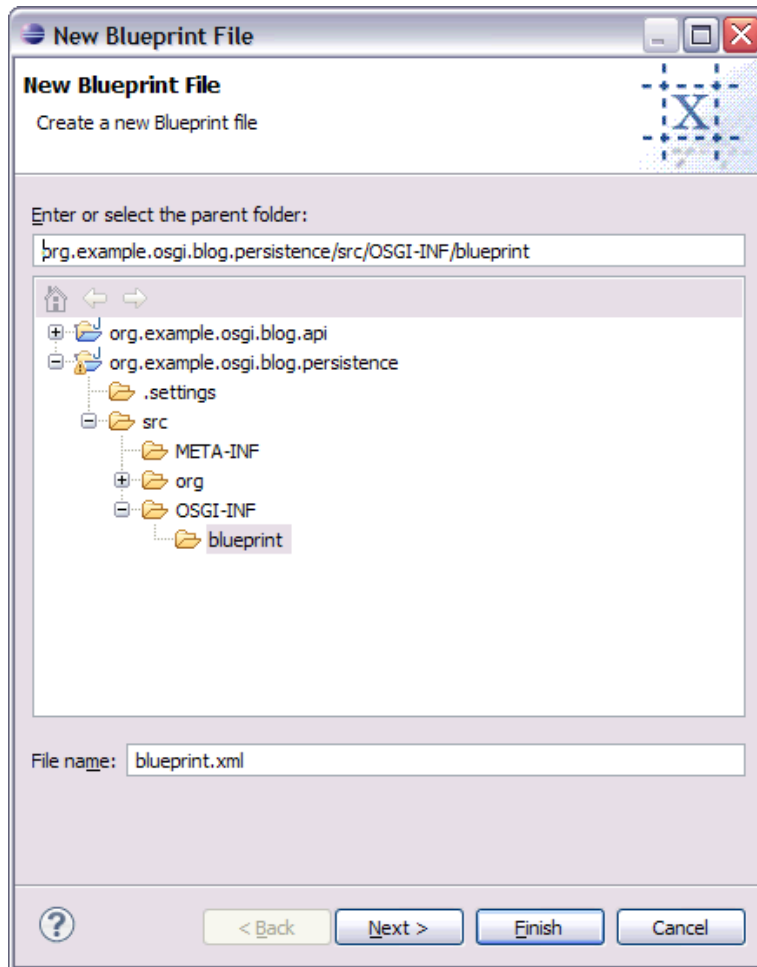
It's time to create a blueprint service to expose the blogging service bean. This will make the blogging service easily accessible to other bundles.

The blueprint file wizard will create an empty blueprint file with the correct xml namespace. The blueprint XML file is the descriptor that defines the dependency injection behavior of your application.

If the file is in a non-default location, it will also add an entry into the bundle's manifest.

1. Select the **org.example.osgi.blog.persistence** project in the navigator.
2. Select **File > New > Other...**

3. Select **OSGi** > **Blueprint File**. Click **Next**.



4. Leave the default location for the file and click **Finish**.

After the wizard has finished, the newly created file will be opened in the XML editor.

P. Create Blueprint Bean and Service

The Blueprint specification offers a declarative way to publish a bean class as a service in the OSGi service registry. To do this, declare a bean element with an id, then a service element declaring the API interface(s) the bean implements and refer to the bean's id:

1. Select the **Source** tab and declare the BloggingServiceImpl class as a bean by adding a <bean> element to the <blueprint> element with the following attributes:

```
<bean id="persistenceImpl"
      class="org.example.osgi.blog.persistence.internal.BloggingServiceImpl">
</bean>
```

2. Then declare a <service> element with a ref attribute containing the bean's id, and the interface it implements:

```
<service ref="persistenceImpl"
         interface="org.example.osgi.blog.BloggingService">
</service>
```

3. Click on the **Save** toolbar button or hit **Ctrl-S** to save the blueprint file.



Q. Add Blueprint container managed transaction and persistence configuration

The JPA configuration discussed earlier, by way of the **Meta-Persistence** header allows for the JPA persistence unit to be configured through a persistence.xml file. A Blueprint bundle can gain the benefits of *container* managed JPA. The Blueprint container will fully manage the PersistenceContext and inject it into the Blueprint managed bean.

To use the Apache Aries container managed JPA support, the Blueprint bean needs to have its transaction and persistence settings defined. To have the blueprint container manage the PersistenceContext and inject it into the **persistenceImpl** bean:

1. Within the **Source** tab add the following namespace attribute to the top-level **blueprint** element:
xmlns:jpa="http://aries.apache.org/xmlns/jpa/v1.0.0"
2. Add the following element as a child within the **<bean ...></bean>** element:
`<jpa:context property="entityManager" unitname="blogExample" />`

To ensure all the methods of the **persistenceImpl** bean run under a global transaction established by the Blueprint container:

1. Within the **Source** tab add the following namespace attribute to the top-level **blueprint** element:
xmlns:tx="http://aries.apache.org/xmlns/transactions/v1.0.0".
2. Add the following element as a child within the **<bean ...></bean>** element:
<tx:transaction method="*" value="Required"/>.



```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:tx="http://aries.apache.org/xmlns/transactions/v1.0.0"
           xmlns:jpa="http://aries.apache.org/xmlns/jpa/v1.0.0">
  <bean id="persistenceImpl"
        class="org.example.osgi.blog.persistence.internal.BloggingServiceImpl">
    <tx:transaction method="*" value="Required"/>
    <jpa:context property="entityManager" unitname="blogExample" />
  </bean>
  <service ref="persistenceImpl"
           interface="org.example.osgi.blog.BloggingService">
  </service>
</blueprint>
```

3. Click on the **Save** toolbar button or hit **Ctrl-S** to save the blueprint file.

The blueprint file is now complete. The persistence bundle is also now complete. Note that we did not export any packages from the persistence bundle, since it will not be used directly by our web bundle.

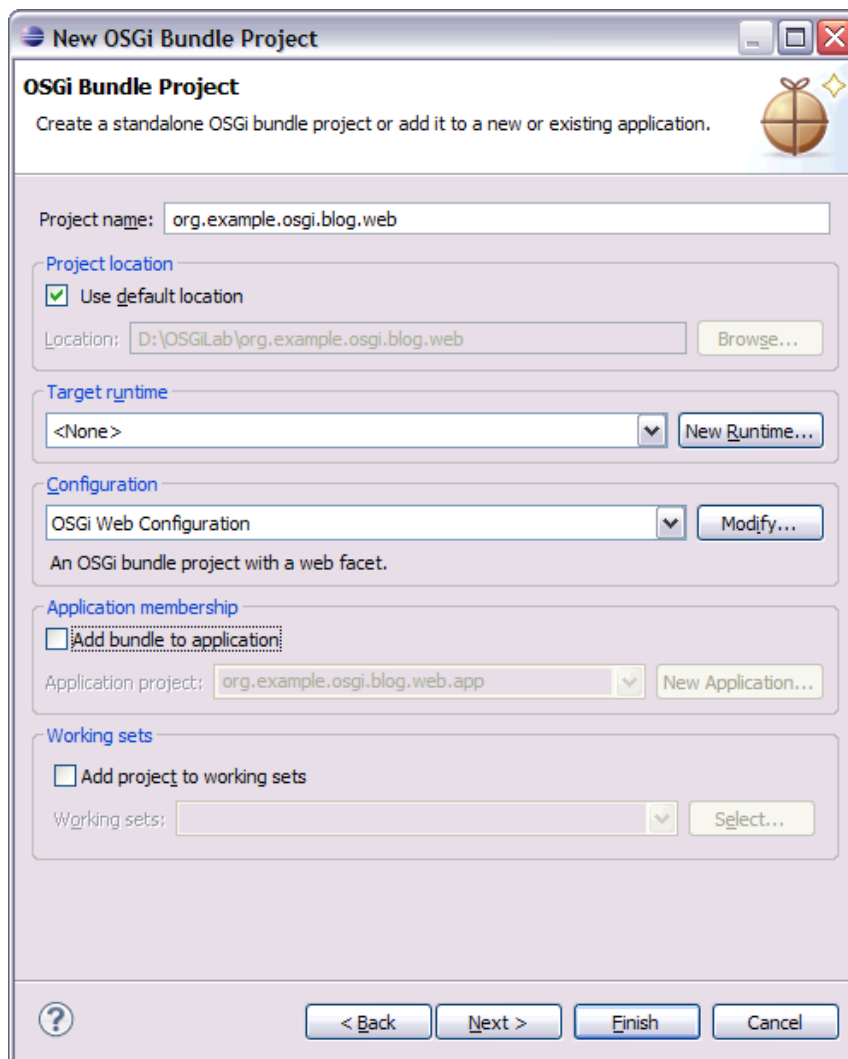
Section 5 - Create Web Bundle

Web applications can be deployed as OSGi bundles as well. In this section we'll create a web bundle that allows us to access the blog service via the web.

R. Create Web Bundle Project

First, we'll create a web bundle.

1. Select **File > New > Project**.
2. Select **OSGi > OSGi Bundle Project** Click **Next**.
3. In the **Project Name** field, type **org.example.osgi.blog.web**.
4. Change **Configuration** to **OSGi Web Configuration**.
5. Uncheck **Add bundle to application**.



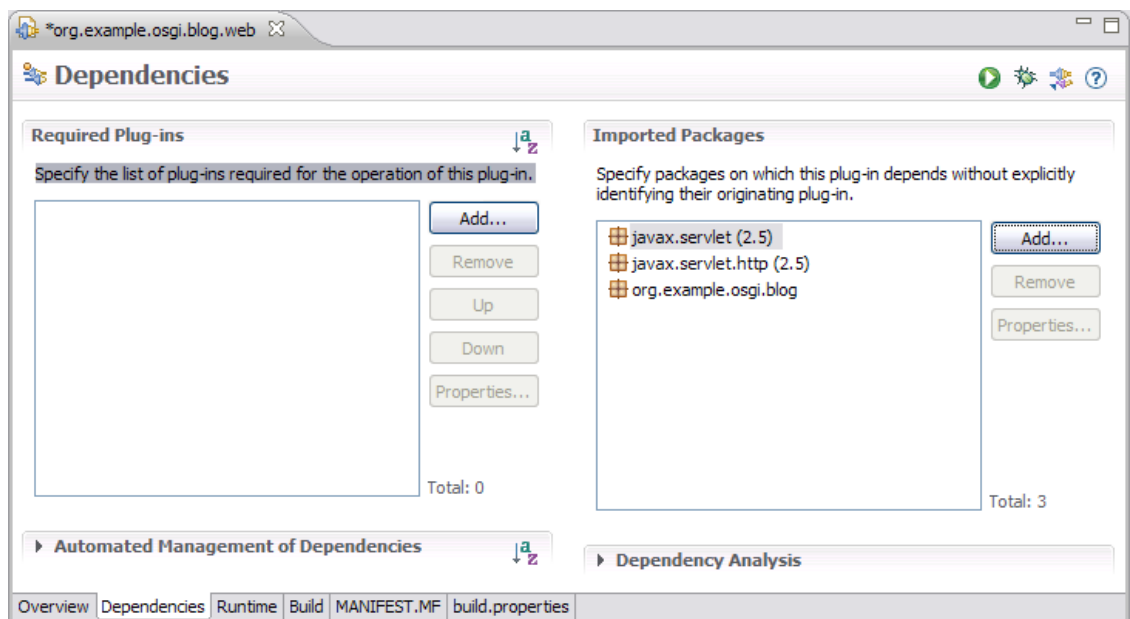
6. Click **Finish**.

S. Import Packages

Just like the persistence bundle, packages that this bundle requires need to be imported. The web bundle will need the standard javax.servlet packages as well as our blog API package.

By default the Bundle project with OSGi Web configuration imports the following packages: javax.el, javax.servlet, javax.servlet.http, javax.servlet.jsp, javax.servlet.jsp.el, javax.servlet.jsp.tagext. Not all of these are required for our web bundle project.

1. Expand the **org.example.osgi.blog.web** project in the **Project Explorer** view.
2. Open the manifest editor by double-clicking on the **Manifest: org.example.osgi.blog.web** node.
3. Select the **Dependencies** tab.
4. Under **Imported Packages**, select the three jsp packages. Click **Remove**
5. Select the **javax.el** package. Click **Remove**
6. Under **Imported Packages**, click **Add ...**
7. Enter **org.example.osgi.blog** Click **OK**.



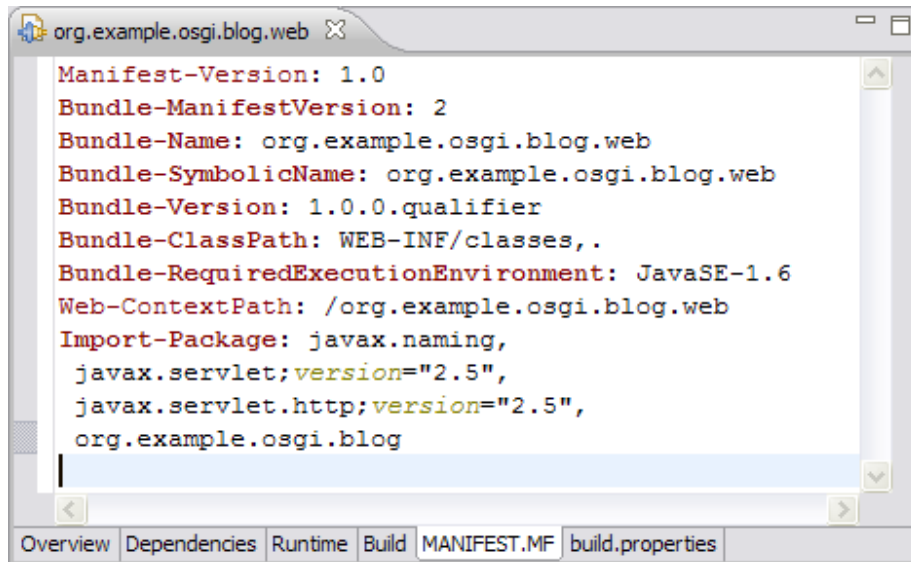
8. Click on the **Save** toolbar button or hit **Ctrl-S** to Save.

Let's take a look at the source to see what our manifest looks like:

9. Click on the **MANIFEST.MF** tab at the bottom to switch to the source view.
10. Note the **Import-Package** entry that we've just added. The web bundle depends on the regular servlet API and blog API classes, but does not have any dependency on the persistence bundle or its packages.
11. Note the **Web-ContextPath** field that was automatically added by the tools. This defines the web context root for the web bundle.

The Bundle-Classpath header only lists the WEB-INF/classes directory. Due to a current issue with the web container integration with Jetty, static web content will only be served out if the static web content directory is listed in the Bundle-Classpath header.

12. Add `..` to the end of the Bundle-Classpath header.



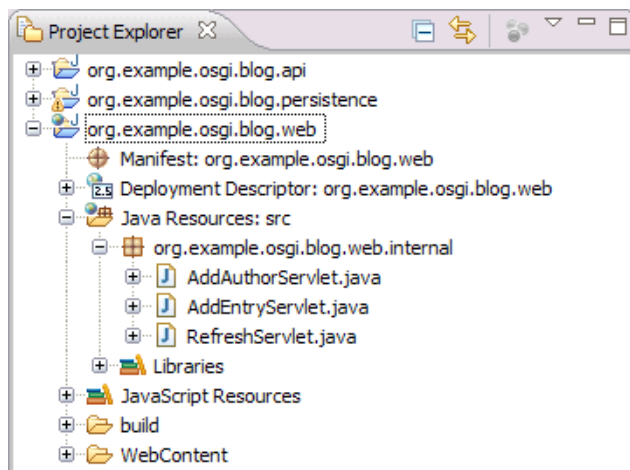
```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: org.example.osgi.blog.web
Bundle-SymbolicName: org.example.osgi.blog.web
Bundle-Version: 1.0.0.qualifier
Bundle-ClassPath: WEB-INF/classes,..
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Web-ContextPath: /org.example.osgi.blog.web
Import-Package: javax.naming,
    javax.servlet;version="2.5",
    javax.servlet.http;version="2.5",
    org.example.osgi.blog
```

13. When you are finished, close the editor.

T. Importing Servlets

Let's import some classes to save time, just like the other bundles.

1. Due to a current issue with the Java EE perspective, first switch to the **Java** perspective.
2. Open the **Resources** folder on the desktop.
3. Open the **web** folder.
4. Drag or copy the **org** folder into the **org.example.osgi.blog.web/src** folder within the workspace.
5. Expand the **org.example.osgi.blog.web.internal** package within Eclipse and explore the three new classes that have been created. **AddAuthorServlet** and **AddEntryServlet** are utility servlets for creating new authors and entries respectively, and **RefreshServlet** loads current blog information from the manager service.



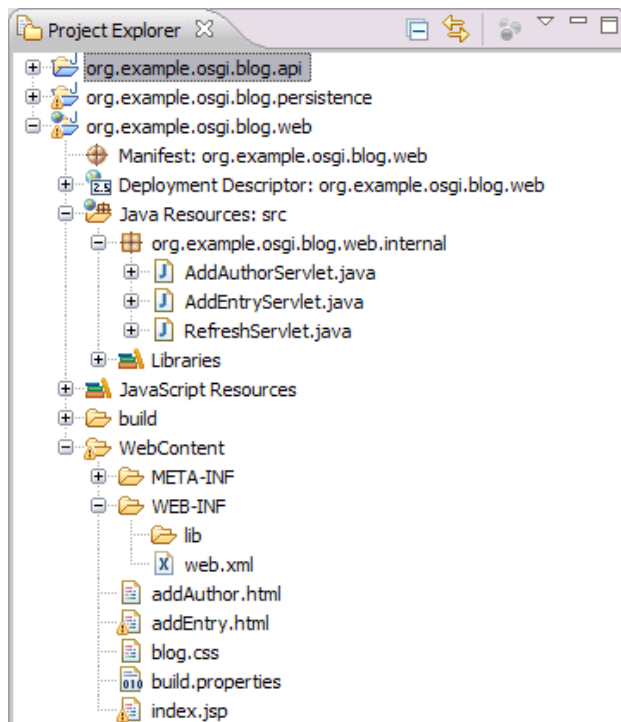
6. Open up any of the three servlets to take a look at the code. In each one, the blog service is found by using code like the following:

```
InitialContext in = new InitialContext();
BloggingService bs = (BloggingService)
    in.lookup("osgi:service/" + BloggingService.class.getName());
```

U. Importing Web Content

Let's import some web content to save time as well.

1. Open the **Resources** folder on the desktop.
2. Open the **web** folder.
3. Drag or copy the files, `addAuthor.html`, `addEntry.html`, `blog.css` and `index.jsp`, into the **org.example.osgi.blog.web/WebContent** folder within the workspace.
4. Open the **WEB-INF** folder on the file system.
5. Drag or copy the **web.xml** file into the **org.example.osgi.blog.web/WebContent/WEB-INF** folder within the workspace.
6. Explore the new HTML, JSP, and web.xml files. These are a simple web front end that interacts with the servlets we just created.



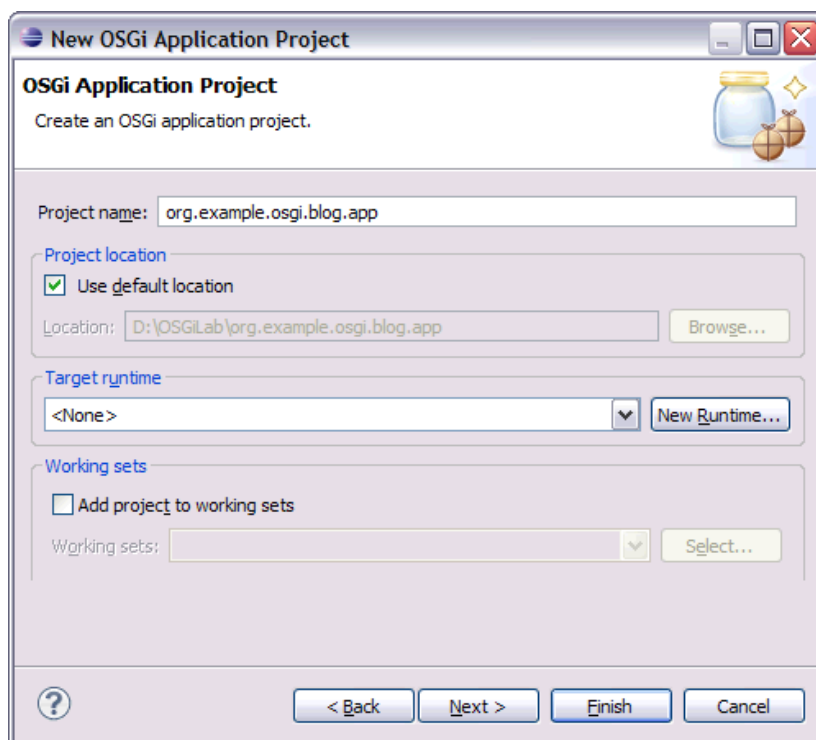
The web bundle is now complete.

Section 6 - Create OSGi Application

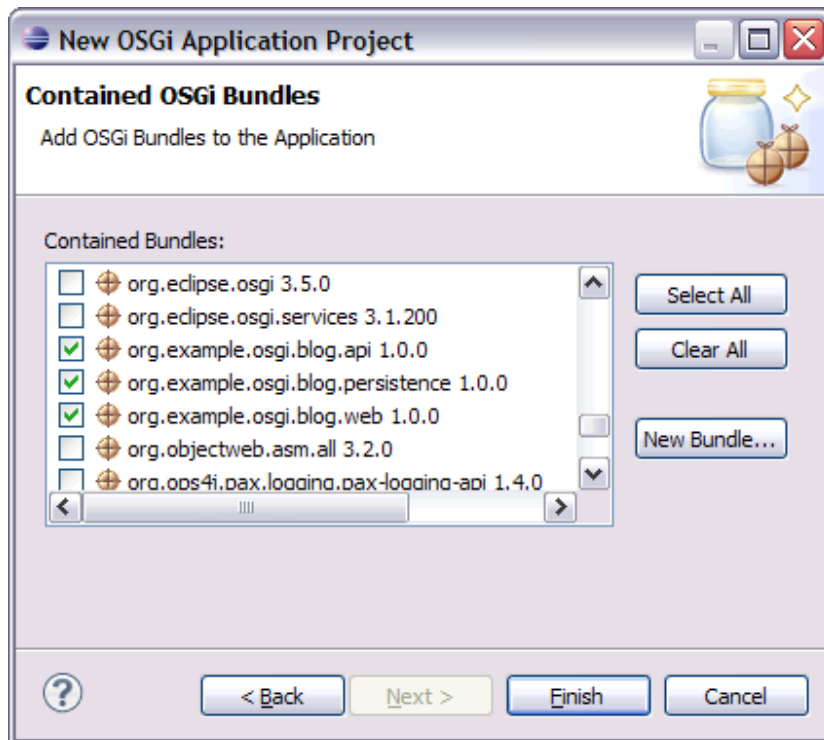
V. Create an OSGi Application Project

The final step is to create an application that contains all of our bundles.

1. Select **File > New > Project**.
2. Select **OSGi > OSGi Application Project**. Click **Next**.
3. In the **Project Name** field, type **org.example.osgi.blog.app**
4. Click **Next**



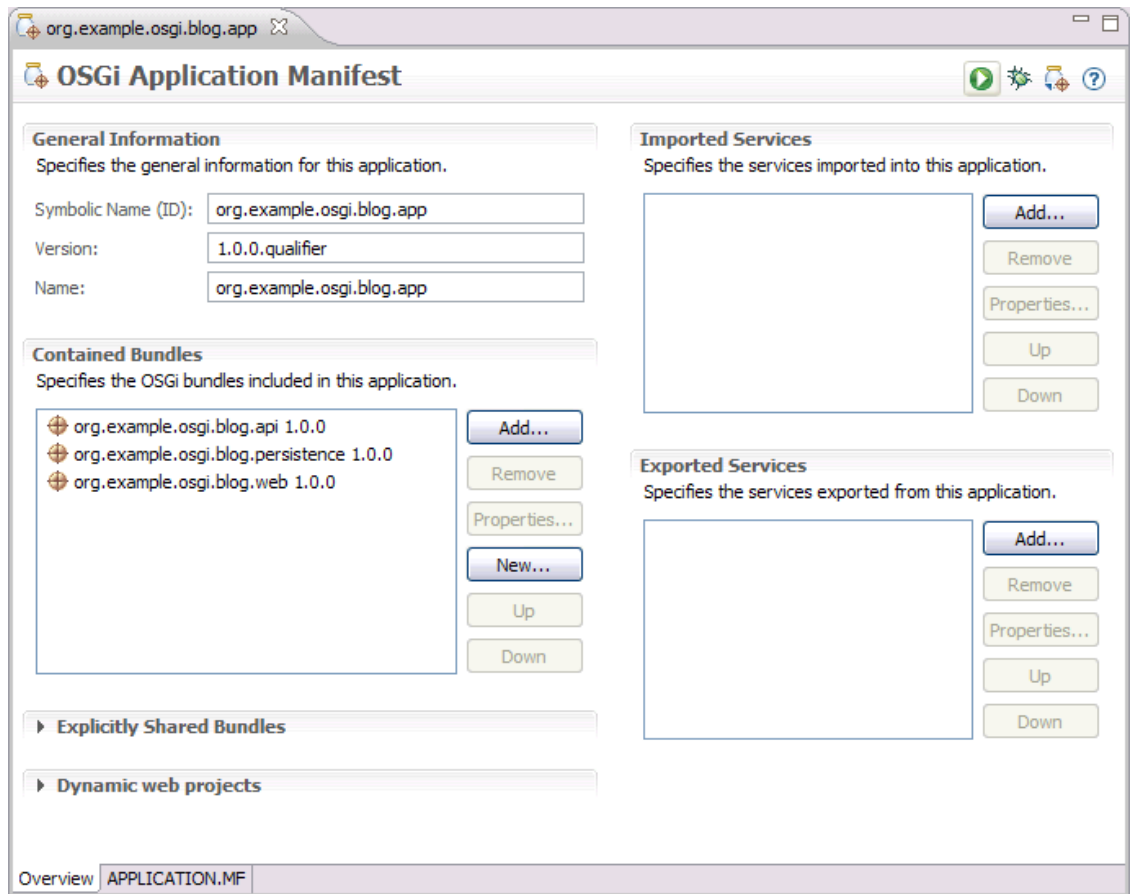
5. Select the three bundles beginning **org.example.osgi.blog** in the list of contained bundles.



6. Click **Finish**.

Let's take a look at the application manifest to see what has been created:

7. Expand the **org.example.osgi.blog.app** project in the **Project Explorer** view if it is not already expanded.
8. Double-click on the **Manifest: org.example.osgi.blog.app** in the **Navigator**.



9. Switch between the **Design** and **Source** tabs to see the contents of the application and how it is defined in the manifest.

10. When you are finished, close the editor.

The OSGi Application is complete.

Section 7 - Testing

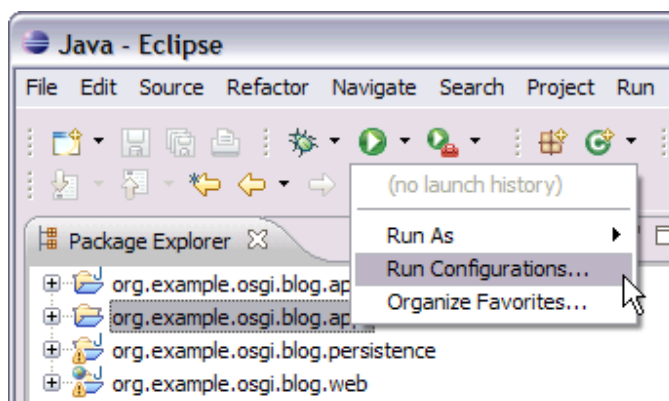
Now that the application is complete, let's test it.

The OSGi Application can be run directly in Eclipse by creating an OSGi Framework run configuration containing the projects the OSGi Application contains. Alternatively, the OSGi Application can be exported to an archive (.eba file) and deployed to the Apache Aries Blog assembly **load** directory.

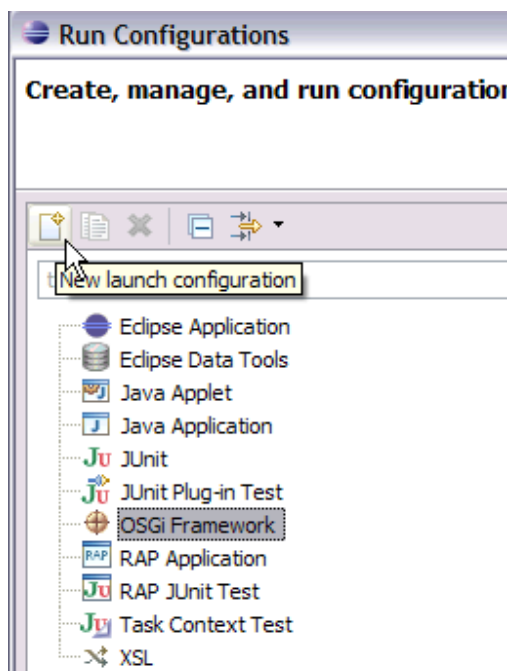
We'll look at running the application from Eclipse:

W. Create an OSGi Framework run configuration

1. Click the **Run** icon then select **Run Configurations ...**



2. Select **OSGi Framework** then click the New icon:



A new run configuration for running an OSGi Framework will be displayed.

3. In the **Name** field enter **Blog example**.

Browse through the bundles selected from the **Workspace** to ensure all three Blog example projects are selected, and all the bundles in the **Target Platform** are also selected. Leave the Start level and Auto start settings to their defaults.

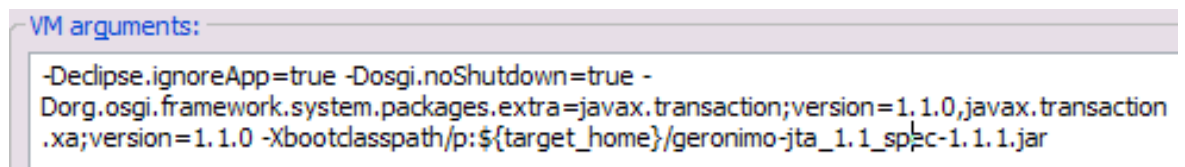
The Apache Aries bundles require the Java Transaction API (JTA) packages to be available at version 1.1 or greater. Some JTA classes are available from the JRE and hence by default are exposed into the OSGi framework via the System Bundle (bundle ID 0). Packages exposed in this way have a version of 0.0.0. To override this:

4. Select the **Arguments** tab.

5. In the **VM arguments** entry field add the following two arguments:

```
-Dorg.osgi.framework.system.packages.extra=javax.transaction;version=1.1.0,javax.transaction.xa;version=1.1.0
```

```
-Xbootclasspath/p:${target_home}/geronimo-jta_1.1_spec-1.1.1.jar
```



To prevent typo's you can copy and paste one of the above from the **Resources/OSGi-framework-run-configuration-snippet.txt** file.

The first argument ensures the javax.transaction packages are exported from the System Bundle at version 1.1.0. The second argument prepends the Apache Geronimo jar that contains the JTA v1.1 specification classes to the JVM's bootclasspath. Together these arguments ensure application bundles can import the JTA packages at the correct version.

6. Click **Apply**.

7. Click **Run**

The OSGi framework starts in debug mode.

8. Click inside the console window and type **ss** to list the bundles in the framework. All 39 bundles (numbered 0-38) should be marked ACTIVE.

9. Go to a web browser and use the following link to go to the Blog:

<http://localhost:8080/org.example.osgi.blog.web/>

Note: there is a timing issue which shows itself as an HTTP 500 error with a NPE stack trace in org.apache.jsp.index_jsp._jspService. This can occur due to the application bundles starting before Apache Aries. To avoid it, increment the start level for the workspace bundles in the run configuration.

X. Create Blog Authors

Let's add some authors to the database.

1. To add an author, click on the **Add Author** link.
2. Enter a new **name** and **email address** for the author.
3. Click **Submit**. The web browser returns to the index page and shows that the author was successfully added.
4. Repeat several times to add more authors.

Y. Create Blog Entries

Let's create some blog entries as well.

1. To add an author, click on the **Add Entry** link.
2. Enter the **email address** of an author created in the previous step, enter a **title**, and **text** for the blog post.
3. Click **Submit**. The web browser should return to the index page and show that the entry was successfully added.
4. Repeat several times to add more entries.