OSGI STEPS:

Creating the bundle

Follow the steps below to create a Hello World bundle using OSGi and Eclipse.

1. In Eclipse, click on **File --> New --> Project**. A New Project dialog will open.
2. In the New Project dialog, select **Plug-in Project** and click Next. The Plug-in Project dialog will open.
3. In the Plug-in Project dialog, enter the following values:
   - Project Name: com.javaworld.sample.HelloWorld
   - Target Platform: OSGi framework --> Standard
4. Use default values for the remaining input and click Next. The Plug-in Context dialog will open.
5. Select the default values for the Plug-in Context dialog and click Next.
6. In the Templates dialog you'll find only one entry in Available Templates: Hello OSGi Bundle. Select it and click Finish.

Eclipse will take few seconds to generate template code for the Hello World bundle. It will create two files: Activator.java and MANIFEST.MF. We'll take a closer look at them both.

Activator.java

Your Activator.java file should look as shown in Listing 1.

Listing 1. Activator.java

```
package com.javaworld.sample.helloworld;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
public class Activator implements BundleActivator {
   public void start(BundleContext context) throws Exception {
      System.out.println("Hello world");
   }
   public void stop(BundleContext context) throws Exception {
      System.out.println("Goodbye World");
   }
}
```

If your bundle needs to be notified at the time of bundle startup or shutdown then you should create a class implementing the BundleActivator interface. Follow these rules when creating the class:

- The BundleActivator class must have a public constructor that takes no parameters. The OSGi framework can create a BundleActivator object by calling Class.newInstance().
- The container will call the start() method of your Activator class to start the bundle. The bundle can take this opportunity to perform resource initialization such as getting a database connection for future use. The start() method takes one argument, the BundleContext object. This object allows bundles to interact with the framework by providing access to OSGi-container-related information. If an exception is thrown for a particular bundle the container will mark that bundle as *stopped* and will not put it into service.
- The container will call the stop() method of your Activator class to report that it is shutting down a bundle. You can use this opportunity to perform cleanup tasks such as releasing the database connection.

Once your Activator class is ready you should relay its fully qualified name to the container using your MANIFEST.MF file.

MANIFEST.MF

The MANIFEST.MF file acts as deployment descriptor for your bundle. The format for this file is the same as that of a normal JAR file, so it consists of a set of headers with values. The OSGi specification defines a set of headers that you can use to describe your bundle to the OSGi container. The MANIFEST.MF file for your Hello World bundle should look as shown in Listing 2.

Listing 2. Manifest for the Hello World bundle

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HelloWorld Plug-in
Bundle-SymbolicName: com.javaworld.sample.HelloWorld
Bundle-Version: 1.0.0
Bundle-Activator: com.javaworld.sample.helloworld.Activator
Bundle-Vendor: JAVAWORLD
Bundle-Localization: plugin
Import-Package: org.osgi.framework;version="1.3.0"
```

Let's take a closer look at what each of these headers is used for:

Bundle-ManifestVersion

The Bundle-ManifestVersion header tells the OSGi container that this bundle follows the rules of the OSGi specification. A value of *2* means that the bundle is compliant with OSGi specification Release 4; a value of *1* means that it is compliant with Release 3 or earlier.

Bundle-Name

The Bundle-Name header defines a short, human-readable name for the bundle.

Bundle-SymbolicName

The Bundle-SymbolicName header specifies a unique, non-localizable name for the bundle. This is the name you will use while referring a given bundle from other bundles.

Bundle-Version

The Bundle-Version header specifies the version of the bundle.

Bundle-Activator

The Bundle-Activator header specifies the name of the optional listener class to be notified of bundle *start* and *stop* events. In Listing 2 the value of this header is com.javaworld.sample.helloworld.Activator.

Bundle-Vendor

The Bundle-Vendor header contains a human-readable description of the bundle vendor.

Bundle-Localization

The Bundle-Localization header contains the location in the bundle where localization files can be found. The Hello World bundle doesn't contain any locale-specific files, but the eclipse IDE still generates this header.

Import-Package

The Import-Package header defines imported packages for the bundle. You'll learn more about this when I discuss dependency management, later in the article.

The Hello World bundle is ready, so let's execute it to see the output.

Executing a bundle

As I mentioned earlier, the Eclipse IDE has an embedded Equinox OSGi container that you can use to execute or debug OSGi bundles. Follow these steps to execute the Hello World bundle:

1. Click on **Run --> Run**.
2. Eclipse will open the dialog called "Create, manage and run configuration." In that dialog, double-click the Equinox OSGi Framework button and it will open a runtime configuration dialog box.
3. In that dialog, change the value of the Name field to Hello World Bundle.
4. You will notice that in the Plug-ins section under the Workspace plug-in there is an entry for the

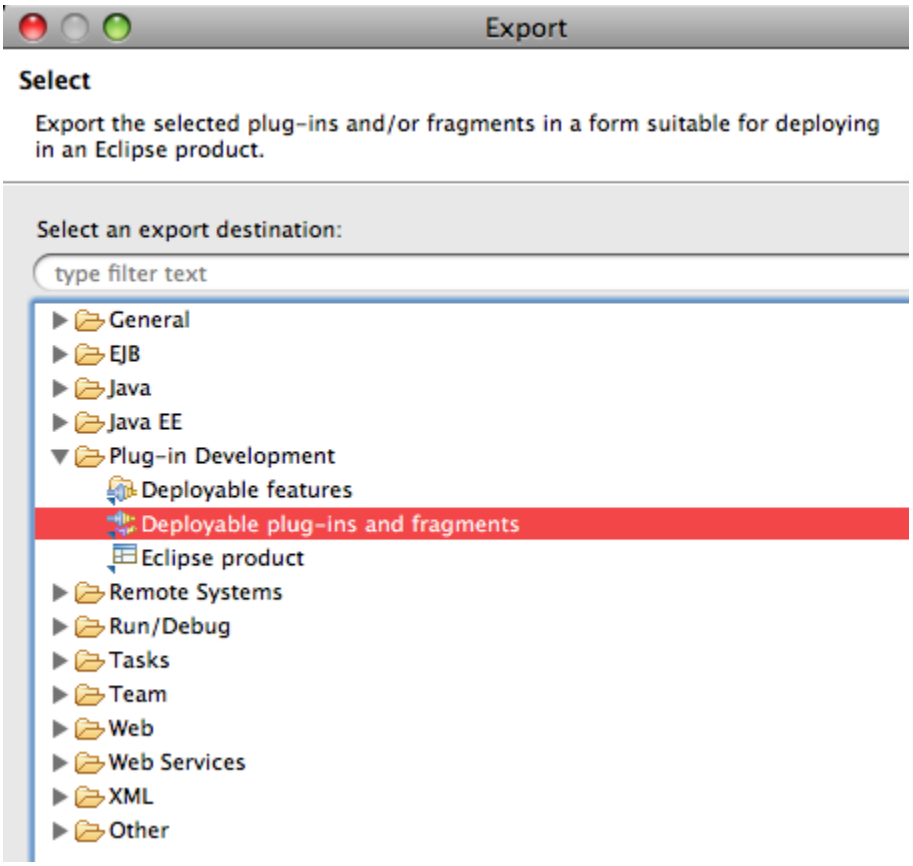
   com.javaworld.sample.HelloWorld

   plugin, which is checked. Under Target Platform, make sure that the checkbox next to the

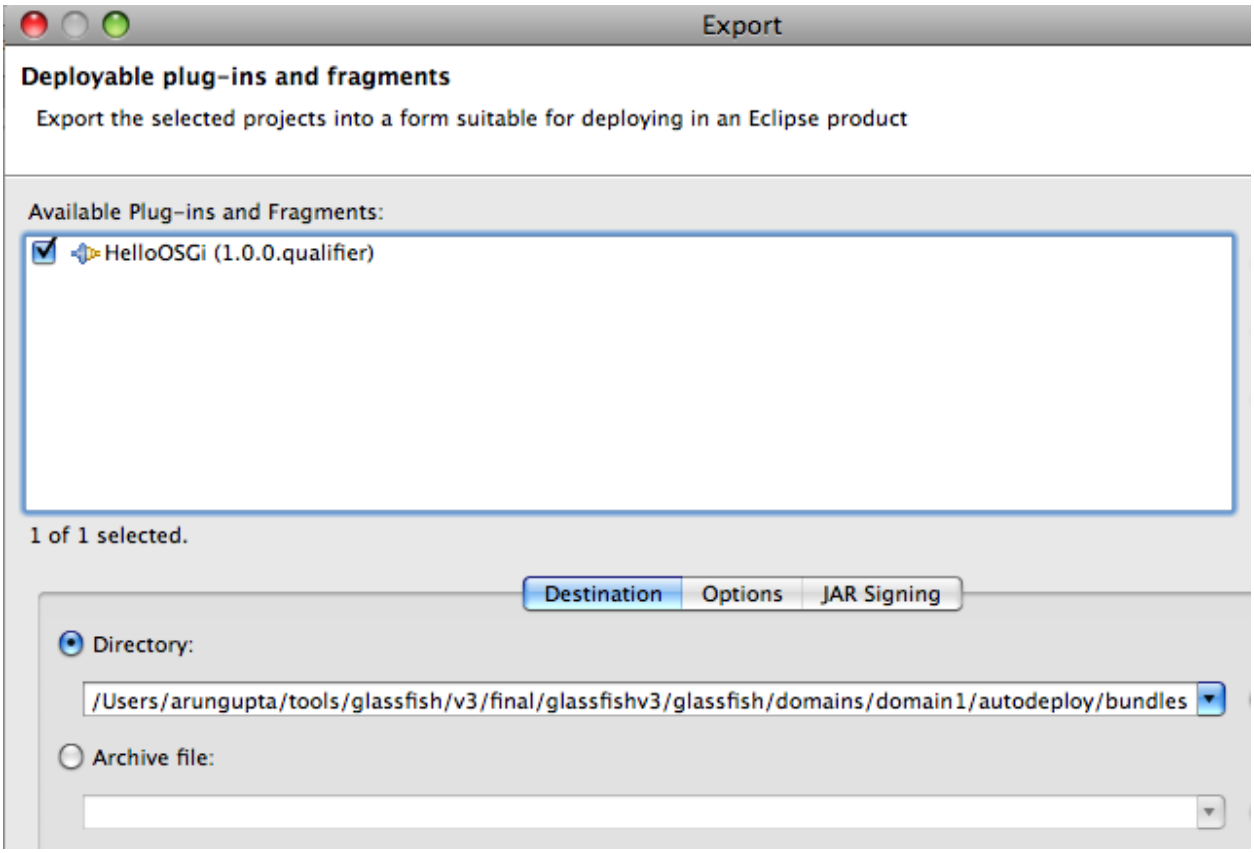
   org.eclipse.osgi plugin is also checked.

5. Now click the Run button. You should see a "Hello world" message in the IDE's console view. Note that Eclipse actually opens the OSGi console in its console view.

Select "File", "Export ...", "Plug-in Development", and select "Deployable Plug-ins and fragments":



and click on "Next >".
Make sure "HelloOSGi" bundle is selected and specify the "domains/domain/autodeploy/bundles" directory of your GlassFish installation as specified below:


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:
HelloOSGi (1.0.0.qualifier)
1 of 1 selected.
Destination
Options
JAR Signing
Directory:
/Users/arungupta/tools/glassfish/v3/final/glassfishv3/glassfish/domains/domain1/autodeploy/bundles
Archive file: