Part 1: Your First Bundle

EclipseZone — Getting Started with OSGi

Neil Bartlett <njbartlett@gmail.com>

Over the next week or two, EclipseZone will be running a series of short posts on OSGi. Taken together they should form a smooth path into mastering the art of OSGi programming, but each post will introduce just one new technique and it should be possible to work through in under ten minutes. Also, we want to show how simple OSGi development can be, so we will not be using Eclipse for development - just a text editor and the basic command line tools will do. So, welcome to the "Getting started with OSGi" series.

Actually this first post will be a little longer than the others, because we need to set up a very basic working environment. Before getting started, we need an OSGi framework to run on. There are three open source implementations to choose from: Apache Felix, Knopflerfish, and Equinox. The code we're going to write will be identical no matter which one you choose, but the instructions for running it will be a little different. Since this is EclipseZone we will use Equinox, the runtime that Eclipse itself is built on. You can pull a copy of it right out of your existing Eclipse installation: just find the file `org.eclipse.osgi_3.2.1.R32x_v20060919.jar` and copy it to an empty directory (NB the version string might be a little different depending on what version of Eclipse you have). If you don't have a copy of Eclipse anywhere, then you can download just that Jar file from http://download.eclipse.org/eclipse/equinox/.

To keep the commands short, let's rename the Jar file to `equinox.jar`. Now bring up a Command Prompt in our development directory and run the command:

```
> java -jar equinox.jar -console
```

In a few seconds, the `osgi>` prompt should appear. Congratulations, you are now running OSGi!

The `osgi>` prompt gives us access to commands in Equinox to control the framework. If you like, type `help` to see a list of commands, and have a play with them. Done that? Now type `ss`. This is the most frequently used command; it stands for "short status" and it shows us the list of bundles that are installed, and what their current status is. (A "bundle" is a module in OSGi terminology. Or if you are an Eclipse developer, you may know them as plug-ins; bundles and plug-ins are basically the same things.) Equinox should print out the following:

```
Framework is launched.

id      State       Bundle
0       ACTIVE      system.bundle_3.2.1.R32x_v20060919
```

This tells us that there is one bundle installed and active, and it is the System Bundle. This is a special bundle in OSGi that is always present, and it represents the framework itself. Now, we're going to write our own bundle. In the same directory as before, create a file called `HelloActivator.java` and copy the following code into it:

```
import org.osgi.framework.*;

public class HelloActivator implements BundleActivator {
  public void start(BundleContext context) {
    System.out.println("Hello EclipseZone Readers!");
  }

  public void stop(BundleContext context) {
    System.out.println("Goodbye EclipseZone Readers!");
  }
}
```

A bundle also needs a manifest file that declares various metadata about the bundle, e.g. its name, version, etc. So create a file called `HelloWorld.mf` and copy the following text into it. **Make very sure** that this file ends with a blank line, otherwise the `jar` command line tool will truncate the file.

```
Manifest-Version: 1.0
Bundle-Name: HelloWorld
Bundle-Activator: HelloActivator
Bundle-SymbolicName: HelloWorld
Bundle-Version: 1.0.0
Import-Package: org.osgi.framework
```

Now open a new Command Prompt (because we want to leave OSGi running) and build the Jar with the following commands:

```
> javac -classpath equinox.jar HelloActivator.java

> jar -cfm HelloWorld.jar HelloWorld.mf HelloActivator.class
```

Going back into the OSGi console, type `install file:HelloWorld.jar`. The reply should be "Bundle id is 1" . Type `ss` again and you will see the following:

```
Framework is launched.

id      State       Bundle
0       ACTIVE      system.bundle_3.2.1.R32x_v20060919
1       INSTALLED   HelloWorld_1.0.0
```

Our HelloWorld bundle is installed... but it's not yet active. We'll look into what these states mean in a later post, but for now we just need to start the bundle by typing `start 1` . The "1" is the ID of the bundle from the first column. When you do this you should see the message "Hello EclipseZone Readers!". Now type `stop 1` and you will see "Goodbye EclipseZone Readers!". Repeat this until you get bored. Don't forget to do `ss` occasionally to see the state of the bundle changing.

What's happening here? Our code implements the `BundleActivator` interface, allowing the framework to notify us of important lifecycle events. When the bundle is started, the framework calls the `start` method, and when the bundle is stopped, the framework calls the `stop` method. The other thing going on here is the line in the manifest file "`Bundle-Activator: HelloActivator`" , which tells the framework which class in our bundle is the activator. Normally the name we give is a fully-qualified class name, but we were lazy and used the default package.

And that concludes our first installment. See you next time.