Part 2: Interacting with the Framework

EclipseZone — Getting Started with OSGi

Neil Bartlett <njbartlett@gmail.com>

Welcome back to the EclipseZone OSGi tutorial series.

Last time we looked at a simple Hello World bundle that printed a message when starting and stopping. It did that by implementing the `BundleActivator` interface and providing `start` and `stop` methods. Take another look at the code now, in particular the method signature of `start` and `stop`, and you'll notice that we were passed a parameter, the `BundleContext`. In this installment of the tutorial we will be looking at `BundleContext` and what we can do with it.

`BundleContext` is a magic ticket that the OSGi framework passes to our bundle. When code needs to interact with the framework in any way, you will use the `BundleContext` . In fact this is the only way to interact with the OSGi API, and the framework issues one of these tickets to each bundle through its `BundleActivator` when the bundle is started.

If you still have Equinox running from last time then you don't need to restart it. If it's not running, then remember the command to start it is:

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

Type `ss` at the prompt and you should see that the Hello World bundle from last time is still installed. That's the case even if you have shut down and restarted Equinox since then, because the OSGi framework persists its state between runs. For this exercise we will write a bundle that searches out and uninstalls Hello World. We could do this easily from the console using the `uninstall` command, but we want to see how it can be done programmatically using the OSGi API. So, create a new file called `HelloWorldKiller.java` and copy in the following code:

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

public class HelloWorldKiller implements BundleActivator {
    public void start(BundleContext context) {
        System.out.println("HelloWorldKiller searching...");
        Bundle[] bundles = context.getBundles();
        for(int i=0; i<bundles.length; i++) {
            if("HelloWorld".equals(bundles[i]
                                .getSymbolicName())) {
                try {
                    System.out.println("Hello World found, "
                                     + "destroying!");
                    bundles[i].uninstall();
                    return;
                } catch (BundleException e) {
                    System.err.println("Failed: "
                                     + e.getMessage());
                }
            }
        }
        System.out.println("Hello World bundle not found");
    }

    public void stop(BundleContext context) {
        System.out.println("HelloWorldKiller shutting down");
    }
}
```

Now create the manifest. Again, remember the blank line at the end is **very** important. Copy the following into `HelloWorldKiller.mf`:

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

```

Now compile and build the Jar:

```
> javac -classpath equinox.jar HelloWorldKiller.java
> jar -cfm HelloWorldKiller.jar HelloWorldKiller.mf HelloWorldKiller.class
```

Back at the OSGi console, install the new bundle using `install file:HelloWorldKiller.jar`, and then type `ss`. The status listing should now look like this:

```
id      State       Bundle
0       ACTIVE      system.bundle_3.2.1.R32x_v20060919
1       ACTIVE      HelloWorld_1.0.0
2       INSTALLED   HelloWorldKiller_1.0.0
```

Let's run the Hello World Killer by typing `start 2`. You should see the following output:

```
HelloWorldKiller searching...
Hello World found, destroying!
Goodbye EclipseZone Readers!
```

Notice that the last line of output comes from our original Hello World bundle. Because it was in the `ACTIVE` state before we ran the Killer, it had to be stopped before being uninstalled, and that caused the `stop` method of its `BundleActivator` to run.

Taking another look at the output of `ss`, Hello World has disappeared:

```
id      State       Bundle
0       ACTIVE      system.bundle_3.2.1.R32x_v20060919
2       ACTIVE      HelloWorldKiller_1.0.0
```

You might wonder if there is a security problem here. It appears that any bundle can uninstall any other bundle! Fortunately OSGi has a comprehensive security layer which gives fine-grained control over all interaction with the framework, so for example you could limit the right to uninstall bundles to a particular "management" bundle. However, getting security working is mostly a configuration issue, and in this series we're going to focus on the code.

That's it for this installment. Until next time, why not take a look at the `BundleContext` interface and see what else you can do with it? For example, you could try programmatically installing a new bundle using the `installBundle` method. Or you could get a list of all the currently installed bundles and print out the time and date they were last modified. To help you get started, check out the Javadocs for the OSGi Release 4 APIs .