Explore Eclipse's OSGi console

Use and extend the console that drives Eclipse

Chris Aniszczyk, Software Engineer, SDI Corp.

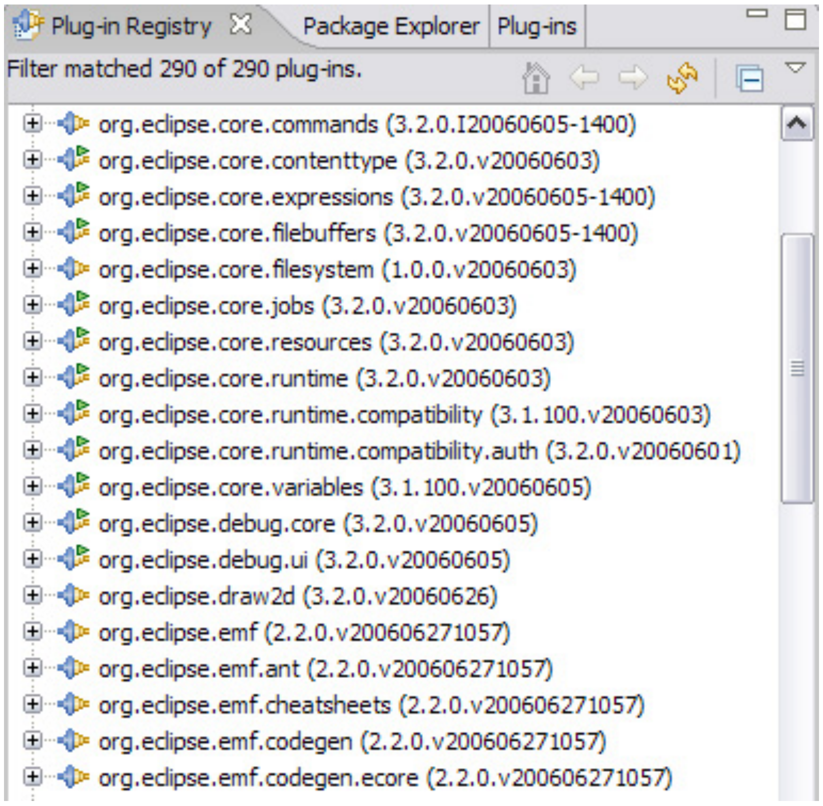
Summary: Get acquainted with the hidden gem known as the OSGi (Equinox) console and find out how it can be added to an Eclipse developer's toolbox. And learn how to extend the console to further add to the toolbox.

Date: 30 Jan 2007
Level: Intermediate
Activity: 4526 views
Comments: 0 (Add comments)

★★★★☆ Average rating

Starting with V3.0, Eclipse made a big leap by choosing Open Services Gateway Initiative (OSGi) to replace the rickety Eclipse plug-in technology found in earlier versions. This transition was almost transparent to users because plug-ins seemed to install and operate as plug-ins of yore.

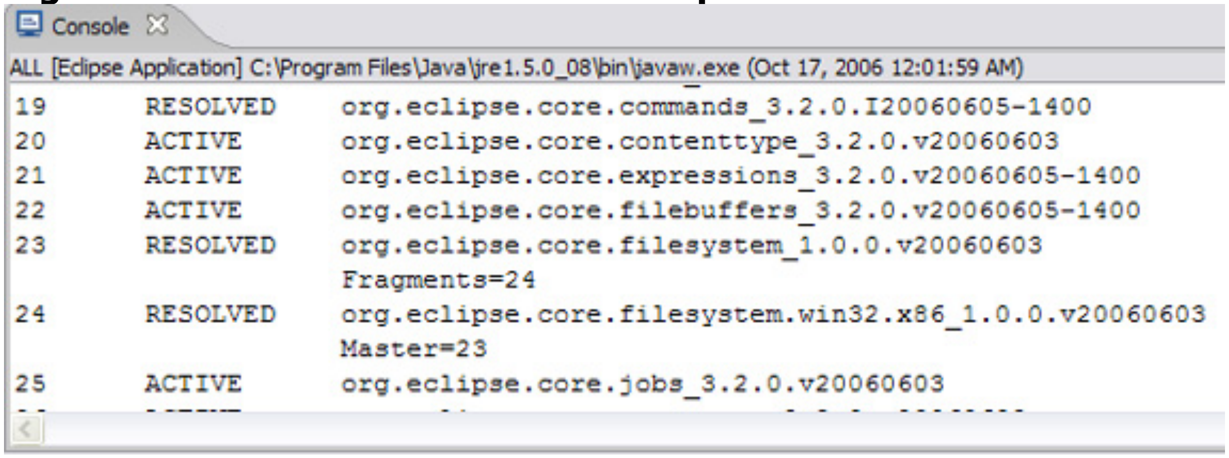
Figure 1. Plug-ins inside Eclipse



```
Filter matched 290 of 290 plug-ins.
org.eclipse.core.commands (3.2.0.I20060605-1400)
org.eclipse.core.contenttype (3.2.0.v20060603)
org.eclipse.core.expressions (3.2.0.v20060605-1400)
org.eclipse.core.filebuffers (3.2.0.v20060605-1400)
org.eclipse.core.filesystem (1.0.0.v20060603)
org.eclipse.core.jobs (3.2.0.v20060603)
org.eclipse.core.resources (3.2.0.v20060603)
org.eclipse.core.runtime (3.2.0.v20060603)
org.eclipse.core.runtime.compatibility (3.1.100.v20060603)
org.eclipse.core.runtime.compatibility.auth (3.2.0.v20060601)
org.eclipse.core.variables (3.1.100.v20060605)
org.eclipse.debug.core (3.2.0.v20060605)
org.eclipse.debug.ui (3.2.0.v20060605)
org.eclipse.draw2d (3.2.0.v20060626)
org.eclipse.emf (2.2.0.v200606271057)
org.eclipse.emf.ant (2.2.0.v200606271057)
org.eclipse.emf.cheatsheets (2.2.0.v200606271057)
org.eclipse.emf.codegen (2.2.0.v200606271057)
org.eclipse.emf.codegen.ecore (2.2.0.v200606271057)
```

Because Eclipse is now built on OSGi, the plug-ins we see in Figure 1 are full-fledged OSGi bundles. (Figure 2 shows the running bundles inside an Eclipse instance using the OSGi console.) By using OSGi, Eclipse supports an industry-endorsed open standard and can now take advantage of the facilities provided by OSGi, including security, HTTP service, useradmin, and others. It seems Eclipse's gamble on OSGi is paying off since we are seeing Eclipse use continue to grow while reported conflicts among plug-ins are decreasing.

Figure 2. OSGi bundles inside of Eclipse



```
ALL [Eclipse Application] C:\Program Files\Java\jre1.5.0_08\bin\javaw.exe (Oct 17, 2006 12:01:59 AM)
19      RESOLVED    org.eclipse.core.commands_3.2.0.I20060605-1400
20      ACTIVE      org.eclipse.core.contenttype_3.2.0.v20060603
21      ACTIVE      org.eclipse.core.expressions_3.2.0.v20060605-1400
22      ACTIVE      org.eclipse.core.filebuffers_3.2.0.v20060605-1400
23      RESOLVED    org.eclipse.core.filesystem_1.0.0.v20060603
                    Fragments=24
24      RESOLVED    org.eclipse.core.filesystem.win32.x86_1.0.0.v20060603
                    Master=23
25      ACTIVE      org.eclipse.core.jobs_3.2.0.v20060603
```

Eclipse, Equinox, OSGi, oh my!

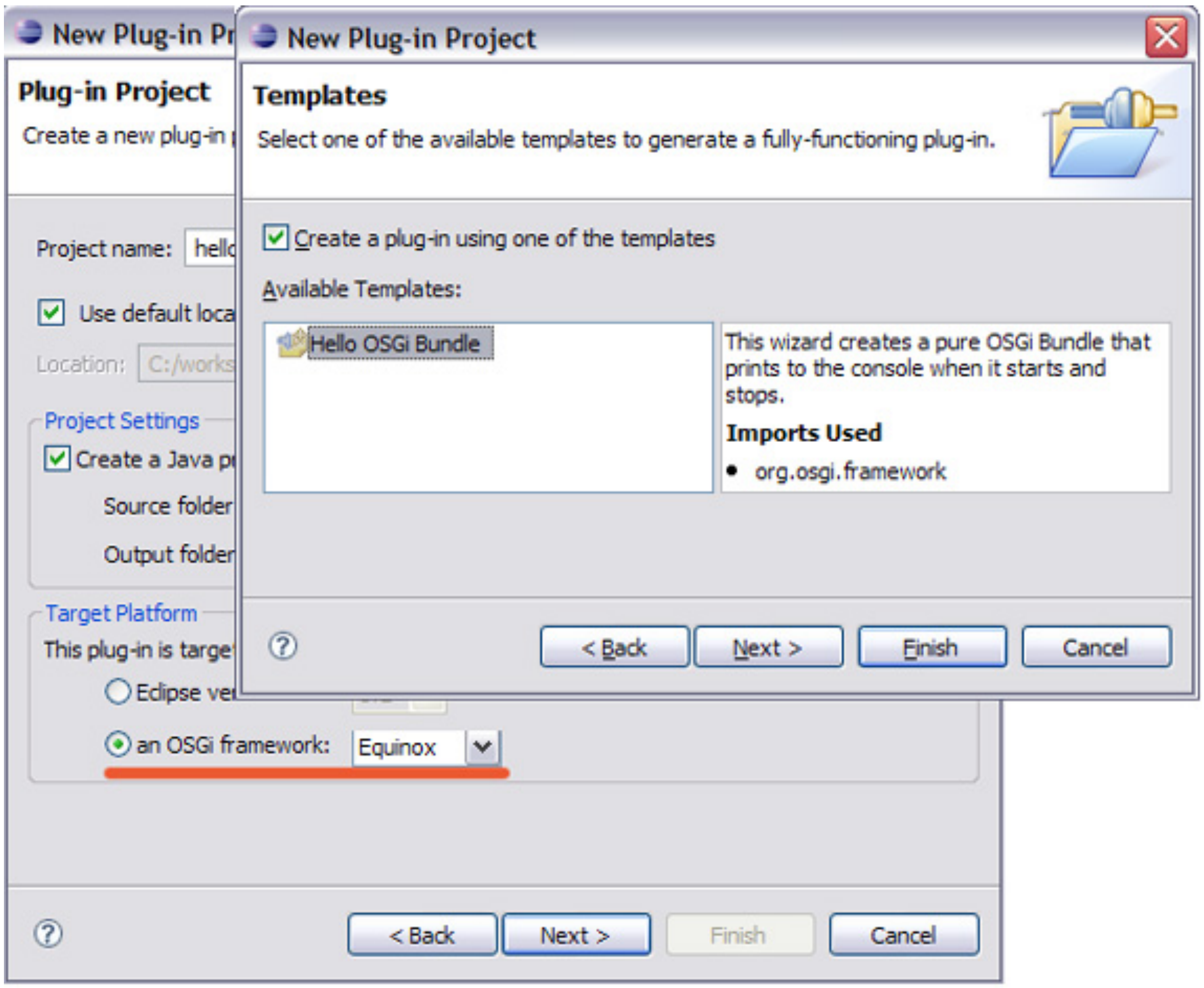
The OSGi Alliance is an independent, nonprofit corporation responsible for OSGi technology. It is akin to the Eclipse Foundation in function. The OSGi Alliance is responsible for producing specifications describing OSGi technology. In brief, OSGI technology provides a service-oriented component-based platform for application development. Various implementations are based on these specifications. One of the most popular implementations is Equinox, which is Eclipse's implementation of the specification. Another popular implementation of OSGi is Apache's Felix project.

Before we go any further, this article assumes you have a working knowledge of Eclipse and OSGi. If you don't, I suggest reading Scott Delap's article "Understanding how Eclipse plug-ins work with OSGi" before diving into the OSGi console.

Create your OSGi bundle

The first step on this adventure is to create a simple OSGi bundle in Eclipse using the Plug-in Development Environment (PDE). To do this, we need to create a new plug-in project using PDE (**File > New > Project > Plug-in Project**). In the process of creating your new plug-in project, make sure you set the proper options. First, select your plug-in target platform as an OSGi Framework, specifically Equinox. Finally, for the sake of brevity, use the **Hello OSGi Bundle** template provided by PDE (see Figure 3). We now have our bundle that we'll use throughout this article.

Figure 3. PDE Hello OSGi bundle wizard



New Plug-in Project

Templates

Select one of the available templates to generate a fully-functioning plug-in.

✓ Create a plug-in using one of the templates

Available Templates:

Hello OSGi Bundle

This wizard creates a pure OSGi Bundle that prints to the console when it starts and stops.

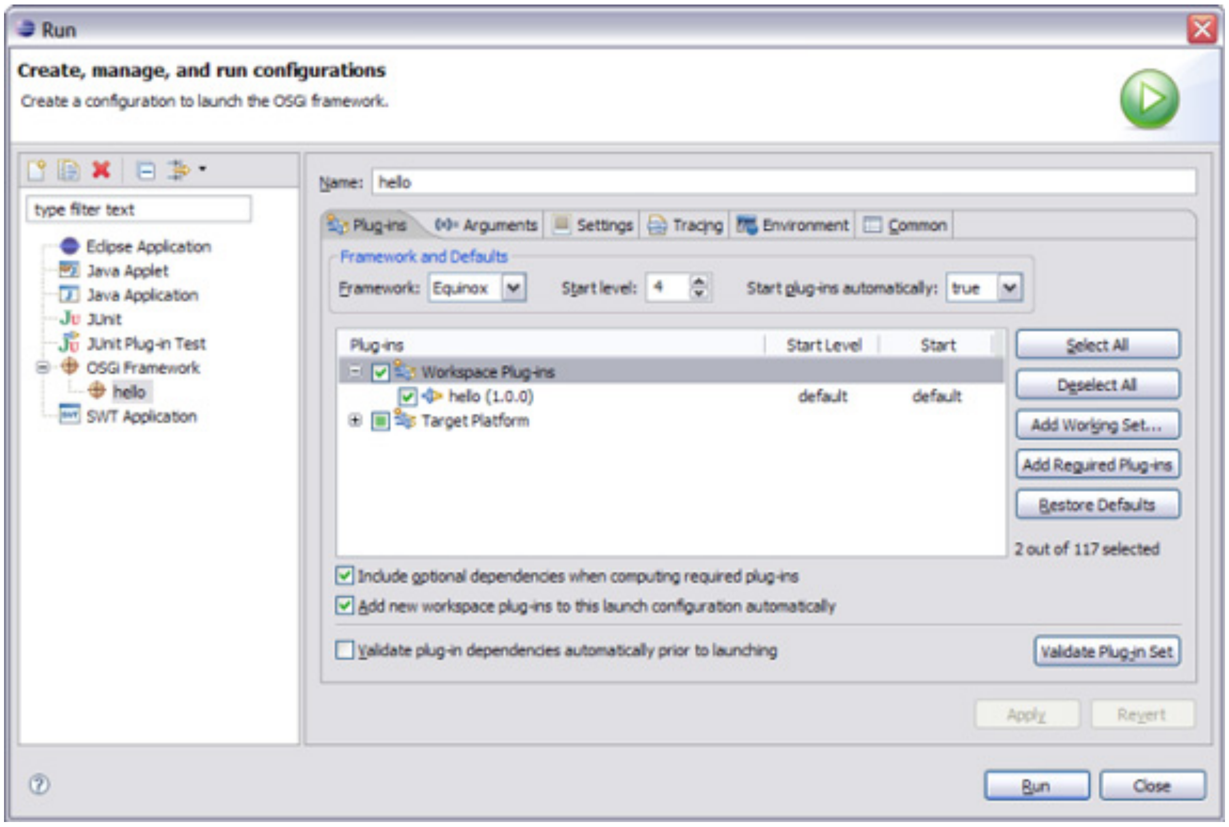
Imports Used

- org.osgi.framework

Hello OSGi console

Since we now have our Hello bundle, we can go ahead and launch the framework to get an OSGi console. To launch the framework, we can take advantage of PDE's OSGi Framework launch configurations. First, go to the launch configuration menu (**Run > Run ...**) and create an OSGi Framework launch configuration for our Hello bundle (see Figure 4). Also, make sure that only the required bundles needed to run the Hello bundle are selected. An easy way to accomplish this is to press **Deselect All** in the launch configuration and check off the Hello bundle, followed by pressing **Add Required Plug-ins**.

Figure 4. PDE's OSGi framework launch configuration



Name: hello

Framework: Equinox Start level: 4 Start plug-ins automatically: true

Plug-ins	Start Level	Start
✓ Workspace Plug-ins		
✓ hello (1.0.0)	default	default
Target Platform		

2 out of 117 selected

✓ Include optional dependencies when computing required plug-ins
✓ Add new workspace plug-ins to this launch configuration automatically
Validate plug-in dependencies automatically prior to launching

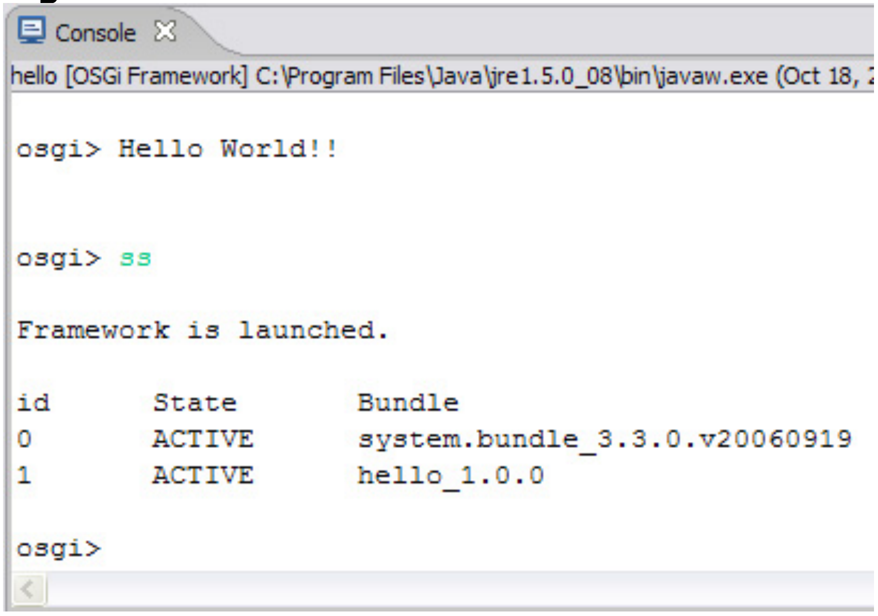
Plug-ins and bundles: What's the difference?

Wander by a basement where Eclipse developers are hacking together applications and you'll hear the magic words *plug-ins* and *bundles* bandied about. Is there a distinction? From a marketing vice president's perspective, the terms are synonymous. A bundle is a plug-in and a plug-in is a bundle. We are stuck with both terms, it seems. However, from an uber-nerd's perspective, there's some contention. To be accurate, an Eclipse plug-in is an OSGi bundle that takes advantage of the extension registry (i.e., have a plug-in.xml at the root of the bundle). An OSGi bundle is, well, an OSGi bundle.

After our launch configuration is completed and ready to go, we can launch our bundle using the **Run** button in the launch configuration dialog. Once this is completed, you should see a result similar to Figure 5.

In Figure 5, we see that our Hello bundle was started (using the HelloWorld message printed in the console indicating our bundle was started) and that we're presented with an `osgi>` prompt. The OSGi prompt is similar to a DOS or a Bash prompt in that you can enter commands that act on the OSGi instance. In this case, we issued the `ss` command, which displays a quick status of everything. I encourage you to try this command in a normal Eclipse instance to realize that everything is just an OSGi bundle under the covers. To get an OSGi console for a normal Eclipse instance, simply launch Eclipse with the `-console` parameter.

Figure 5. Our first launch

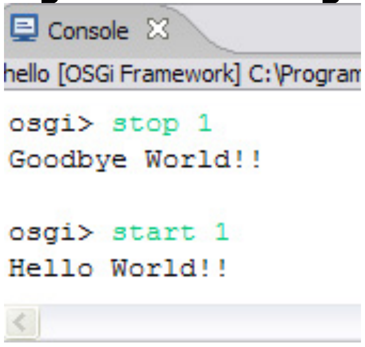


```
Console
hello [OSGi Framework] C:\Program Files\Java\jre1.5.0_08\bin\javaw.exe (Oct 18, 

osgi> Hello World!!


osgi> ss

Framework is launched.

id      State       Bundle
0       ACTIVE      system.bundle_3.3.0.v20060919
1       ACTIVE      hello_1.0.0

osgi>
```

Starting and stopping bundles

In the OSGi dynamic environment, you can start and stop bundles easily. To test this, let's use our simple Hello bundle. Simply stop the bundle with the `stop` command, then start the bundle with the `start` command. You should see results similar to Figure 6.

Figure 6. Starting and stopping bundles

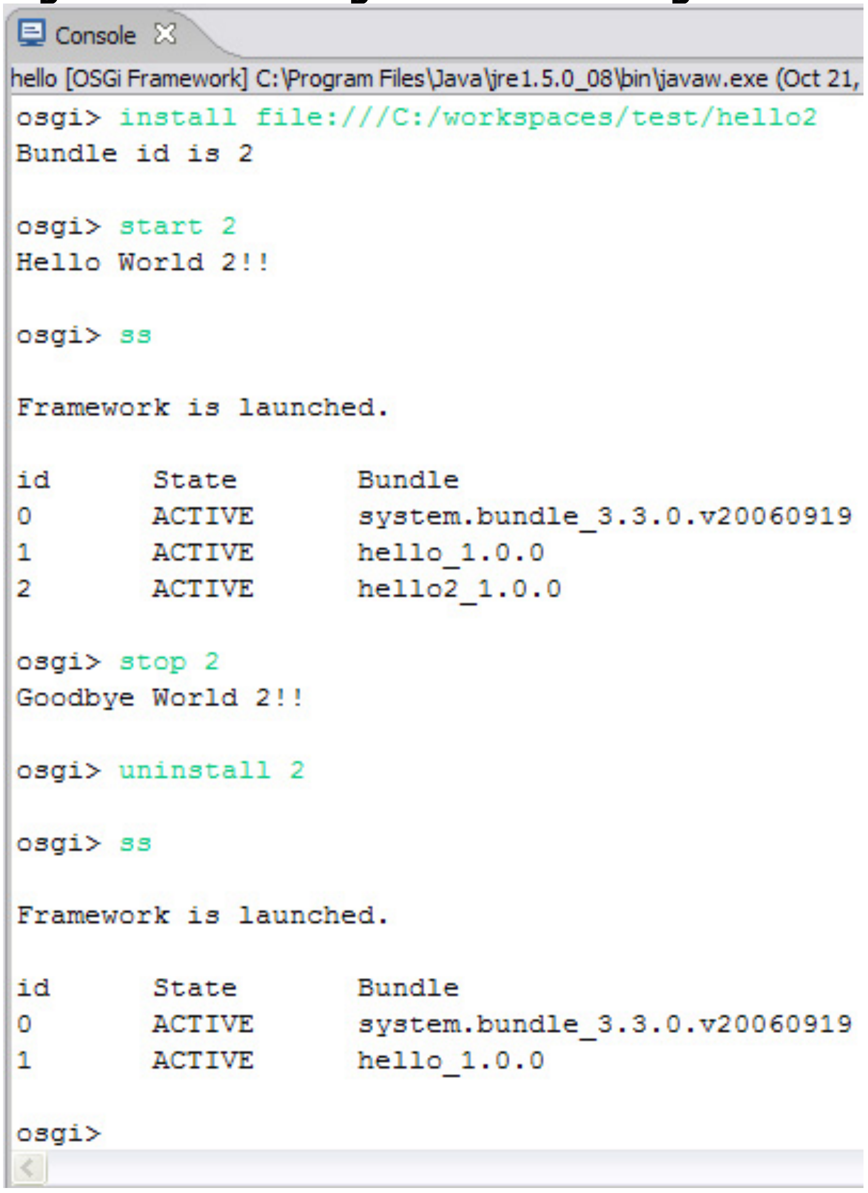


```
Console
hello [OSGi Framework] C:\Program
osgi> stop 1
Goodbye World!!

osgi> start 1
Hello World!!
```

Adding, removing and updating bundles

Another powerful aspect of an OSGi system is the ability to add, remove, and update bundles in a running OSGi instance -- all without restarting the Java™ Virtual Machine. Figure 7 demonstrates installing and uninstalling bundles.

Figure 7. Installing and uninstalling bundles

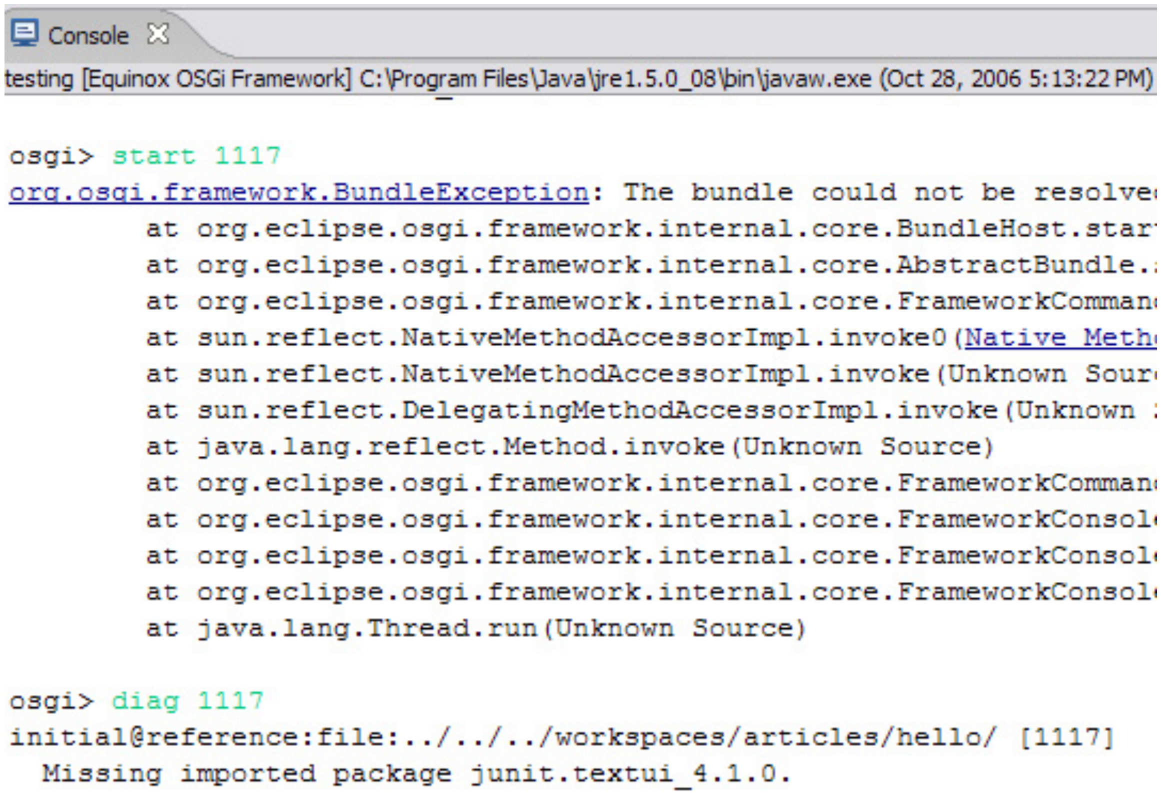


```
hello [OSGi Framework] C:\Program Files\Java\jre1.5.0_08\bin\javaw.exe (Oct 21,
osgi> install file:///C:/workspaces/test/hello2
Bundle id is 2

osgi> start 2
Hello World 2!!

osgi> ss

Framework is launched.

id      State       Bundle
0       ACTIVE      system.bundle_3.3.0.v20060919
1       ACTIVE      hello_1.0.0
2       ACTIVE      hello2_1.0.0

osgi> stop 2
Goodbye World 2!!

osgi> uninstall 2

osgi> ss

Framework is launched.

id      State       Bundle
0       ACTIVE      system.bundle_3.3.0.v20060919
1       ACTIVE      hello_1.0.0

osgi>
```

Diagnostics

There are times when a bundle or plug-in attempts to start but has errors during initialization. The OSGi console provides a useful command -- `diag` -- that helps you debug problems associated with bundle initialization. For an example, let's examine Figure 8, we see that in an attempt to start our Hello bundle, we get an error. To help diagnose the error, we run the `diag` command against our bundle and see that we are missing an imported package in our runtime environment.

Figure 8. Console diagnostics


Console

testing [Equinox OSGi Framework] C:\Program Files\Java\jre1.5.0_08\bin\javaw.exe (Oct 28, 2006 5:13:22 PM)

```
osgi> start 1117
org.osgi.framework.BundleException: The bundle could not be resolve
        at org.eclipse.osgi.framework.internal.core.BundleHost.star
        at org.eclipse.osgi.framework.internal.core.AbstractBundle.
        at org.eclipse.osgi.framework.internal.core.FrameworkComman
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Meth
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Sour
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown
        at java.lang.reflect.Method.invoke(Unknown Source)
        at org.eclipse.osgi.framework.internal.core.FrameworkComman
        at org.eclipse.osgi.framework.internal.core.FrameworkConsol
        at org.eclipse.osgi.framework.internal.core.FrameworkConsol
        at org.eclipse.osgi.framework.internal.core.FrameworkConsol
        at java.lang.Thread.run(Unknown Source)

osgi> diag 1117
initial@reference:file:../../../workspaces/articles/hello/ [1117]
  Missing imported package junit.textui_4.1.0.
```

Summary of useful console commands

Table 1. Useful console commands

Command	**Description**
start	Starts a bundle given an ID or symbolic name
stop	Stops a bundle given an ID or symbolic name
install	Adds a bundle given a URL for the current instance
uninstall	Removes a bundle given a URL for the current instance
update	Updates a bundle given a URL for the current instance
active	Lists all active bundles in the current instance
headers	List the headers for a bundle given an ID or symbolic name
ss	Lists a short status of all the bundles registered in the current instance
services <filter>	Lists services given the proper filter
diag	Runs diagnostics on a bundle given an ID or symbolic name

There are many other OSGi commands available. The ones listed here are the ones I find most useful. To get a list of all the commands, just type `help` in the console.

Extending the console

People say that the beauty of Eclipse lies in its extensibility. The console is extensible in a similar manner. This is important because as a developer, you may provide a service of some kind to users. By extending the console, you can enable advanced users or administrators to debug problems regarding your service.

Instead of using familiar extension points, the console has a simple extensibility mechanism. Let's work through several examples to illustrate the extensibility of the console.

uname, OSGi style

Anyone who has worked with UNIX®-flavored systems is familiar with the `uname` command, which prints the name, version, and other information about the running operating system. In an OSGi context, there can be various implementations of OSGi console (such as Knopflerfish) in the same way there are various flavors of UNIX.

The most important part of extending the OSGi console is the `CommandProvider`interface. Clients interested in extending the console must implement this. Once it is implemented, the next step is to start method names with a `"_"`. These methods will represent the commands available in the console. It's that simple! See Listing 1 for an example.

Listing 1. OSGi uname

```
public class Activator implements BundleActivator, CommandProvider {

        private BundleContext context;

        public void start(BundleContext context) throws Exception {
                this.context = context;
                Hashtable properties = new Hashtable();
                context.registerService\
                (CommandProvider.class.getName(), this, properties);
        }

        public String getHelp() {
                StringBuffer buffer = new StringBuffer();
                buffer.append("\tuname - returns framework information\n");
                return buffer.toString();
        }

        public void stop(BundleContext context) throws Exception {}

        public void _uname(CommandInterpreter ci) throws Exception {
                String vendor = context.getProperty(Constants.FRAMEWORK_VENDOR);
                String version = context.getProperty(Constants.FRAMEWORK_VERSION);
                String osName = context.getProperty(Constants.FRAMEWORK_OS_NAME);
                String osVersion = context.getProperty(Constants.FRAMEWORK_OS_VERSION);
                System.out.println("\n " + vendor + " "
                                + version + " (" + osName + " "
                                + osVersion + ")");
        }
}
```

Bundles and existentialism

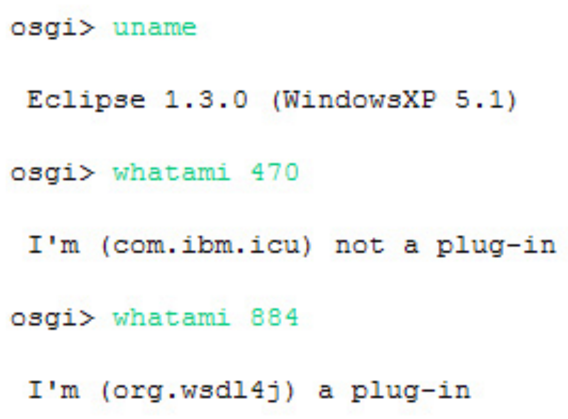
Bundles may never question their existence, but here's a simple example that prints whether a bundle is a vanilla bundle or an Eclipse plug-in. (Remember, both are still bundles!) Listing 2 builds on the previous example by adding a new method and modifying the `getHelp()` method.

Listing 2. Modify `getHelp()`

```
...
public String getHelp() {
                StringBuffer buffer = new StringBuffer();
                buffer.append("\twhatami - \
                returns whether the bundle is a plug-in or not\n");
                buffer.append("\tuname - returns framework information\n");
                return buffer.toString();
        }

public void _whatami(CommandInterpreter ci) throws Exception {
                try {
                        long id = Long.parseLong(ci.nextArgument());
                        Bundle bundle = context.getBundle(id);
                        URL url = bundle.getEntry("plugin.xml");
                        if(url != null) {
                                System.out.println("\n I'm \
                                (" + bundle.getSymbolicName() + ") a plug-in");
                        } else {
                                System.out.println("\n I'm \
                                (" + bundle.getSymbolicName() + ") not a plug-in");
                        }
                } catch (NumberFormatException nfe) {
                        System.out.println("\n Error processing command");
                }
        }
...
```

The results of our labor can be seen in Figure 9.

Figure 9. Results from extending the console



```
osgi> uname

 Eclipse 1.3.0 (WindowsXP 5.1)

osgi> whatami 470

 I'm (com.ibm.icu) not a plug-in

osgi> whatami 884

 I'm (org.wsdl4j) a plug-in
```

Conclusion

This article demonstrates how to use the OSGi console and how to extend the console. Along the way, we've looked at the console and several examples on how to extend it. You now have greater familiarity with the console and have an idea how to use it in your day-to-day Eclipse development. Maybe using the console will even bring back memories of playing Doom and Quake.

Figure 10. The Quake console

```
THE NECROPOLIS
MESHING PROGS/ARMOR.MDL...
YOU GOT THE SHELLS
YOU GOT THE GRENADE LAUNCHER
YOU RECEIVE 25 HEALTH
YOU GET 2 ROCKETS
YOU GOT THE ROCKETS
YOU GOT THE NAILGUN
YOU GOT THE NAILS
YOU GOT THE NAILS
YOU RECEIVE 15 HEALTH
YOU RECEIVE 15 HEALTH
YOU GOT THE ROCKETS
VERSION 1.09 SERVER (24778 CRC)

INTRODUCTION
]+MLOOK
MWHEELDOWN IS UNBOUND, HIT F4 TO SET.
WROTE QUAKE00.TGA
]
                                        (GL 0.95) 1.09
```

Download

Description	Name	Size	Download method
Source code	os-ecl-osgiconsole.hello.zip	49KB	HTTP

Information about download methods

Resources

Learn

- Visit the OSGi Alliance to learn more about the Open Services Gateway Initiative (OSGi).
- Learn about Eclipse's OSGi implementation of Equinox.
- The Eclipse.org PDE is a great place to learn about the plug-in development environment provided by Eclipse.
- Knopflerfish is another OSGi implementation.
- Read "Understanding how Eclipse plug-ins work with OSGi" by Scott Delap.
- For an introduction to Eclipse and OSGi, read "Leave Eclipse plug-in headaches behind with OSGi."
- Check out Eclipsecon 2007, the premier Eclipse conference.
- For an excellent introduction to the Eclipse platform, see "Getting started with the Eclipse Platform."
- Visit IBM developerWorks' Eclipse project resources to learn more about Eclipse.
- Stay current with developerWorks technical events and webcasts.
- Check out upcoming conferences, trade shows, webcasts, and other Events around the world that are of interest to IBM open source developers.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- To listen to interesting interviews and discussions for software developers, be sure to check out developerWorks podcasts.

Get products and technologies

- Download the Eclipse Platform and get started with Eclipse now.
- See the latest Eclipse technology downloads at IBM alphaWorks.

- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- The Eclipse Platform newsgroups should be your first stop to discuss questions regarding Eclipse. (Selecting this link will launch your default Usenet news reader application and open eclipse.platform.)
- The Equinox newsgroups contain a plethora of information about Eclipse's OSGi implementation. (Selecting this link will launch your default Usenet news reader application and open eclipse.technology.equinox.)
- The Eclipse newsgroups has many resources for people interested in using and extending Eclipse.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

Chris Aniszczyk is a software engineer at IBM Lotus focusing on OSGi related development. He is an open source enthusiast at heart, and he works on the Gentoo Linux distribution and is a committer on a few Eclipse projects (PDE, ECF, EMFT). He's always available to discuss open source and Eclipse over a frosty beverage.