Building a Server-based Application with Equinox

Building server-based applications with Eclipse Equinox is relatively easy. Building a very simple application based on Java Servlets is the first step toward realizing the full potential of Equinox as a vehicle for delivering server-based applications. This tutorial—which steps the reader through the process of building a server application based on Java Servlets using Equinox—is based on a tutorial created by the by the Equinox project's Server component team. In this tutorial, you will learn how to get a servlet up and running using a handful of bundles on the Equinox framework. The use of Java Servlets is just one example of the kinds of server that can be created using Equinox.

> ## Bundle or Plug-in?
>
> The terms "bundle" and "plug-in" tend to be used interchangeably. From a purely technical perspective, they are equivalent. A bundle is a collection of artifacts (code, images, configuration, etc.) that are in somehow logically related. Bundles contain metadata, found in the *MANIFEST.MF* file, that provide both basic information (name, creator, version, etc.) and information about how the bundle interacts with its environment. The manifest lists, for example, the other bundles upon which the bundle depends; it also lists the functionality exposed by the bundles (i.e. functionality that can be used by other bundles).
>
> The term "plug-in" tends to be used in the context of Eclipse. That is, plug-ins are bundles that extend the Eclipse framework.

Steps

To build a servlet-based server application using Equinox:

1. Install and configure Eclipse for RCP/Plug-in Developers
2. Create a new "Plug-in Project"
3. Configure dependencies for the Plug-in Project
4. Create a servlet as an extension
5. Build a *Launch Configuration*, run, and test the servlet

The steps for this tutorial have been captured in a live demonstration.

[Click to view demonstration]

Install and configure Eclipse for RCP/Plug-in Developers

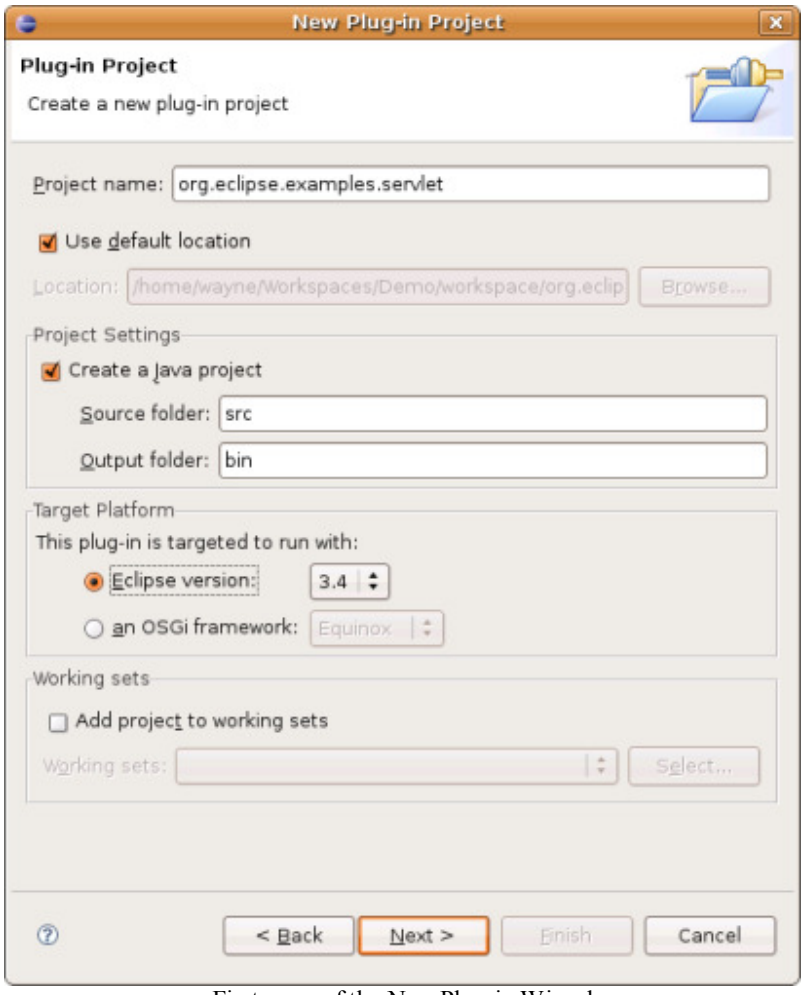
Everything you need to run this tutorial is included in the Ganymede M5 release (and later) of the *Eclipse for RCP/Plug-in Developers*, which can be downloaded from the EPP Ganymede Milestone Package Builds page.

Information and help on installation can be found on the Eclipse for RCP/Plug-in Developers page.

Create a new "Plug-in Project"

Start Eclipse. Create a new Plug-in Project (File > New > Project > Plug-in Project).

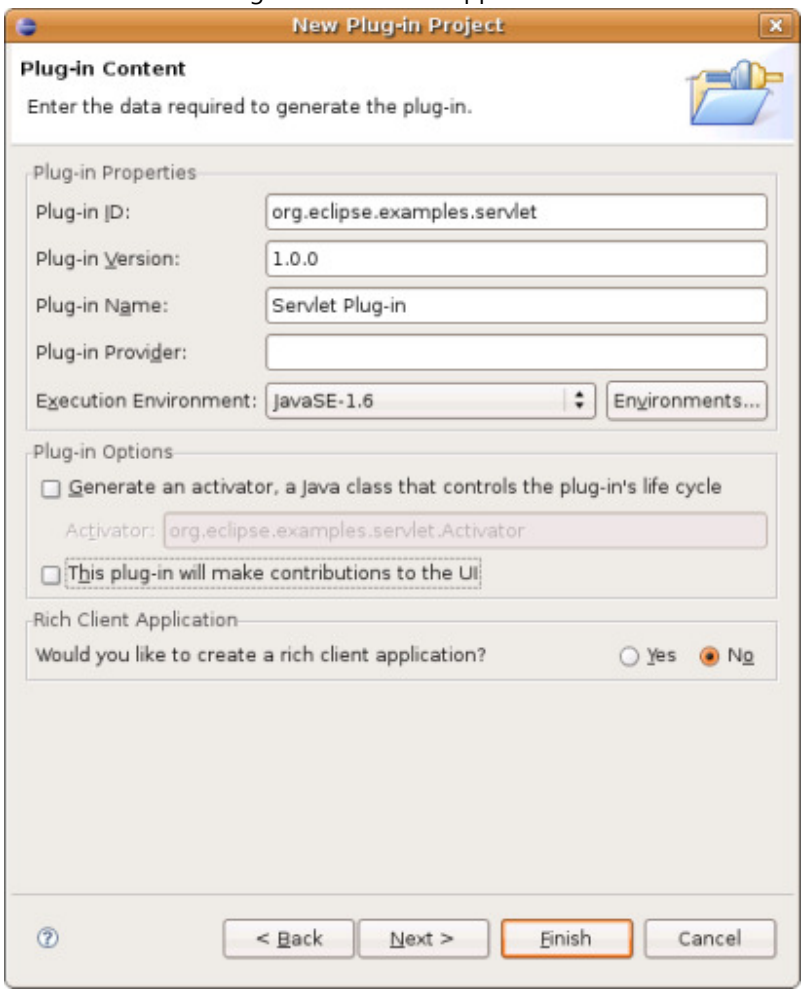
On the first page of the wizard (shown below), provide a name for the project ("org.eclipse.examples.servlet" is used here), and set the "Target Platform" to Eclipse 3.4.



First page of the New Plug-in Wizard

Click "Next" to move to the next page.

On the second page of the wizard (shown below) ensure that the two "Plug-in Options" are turned off. Turning these options off will avoid adding configuration to the Plug-in project that we will not require with this example.



Second page of the New Plug-in Wizard

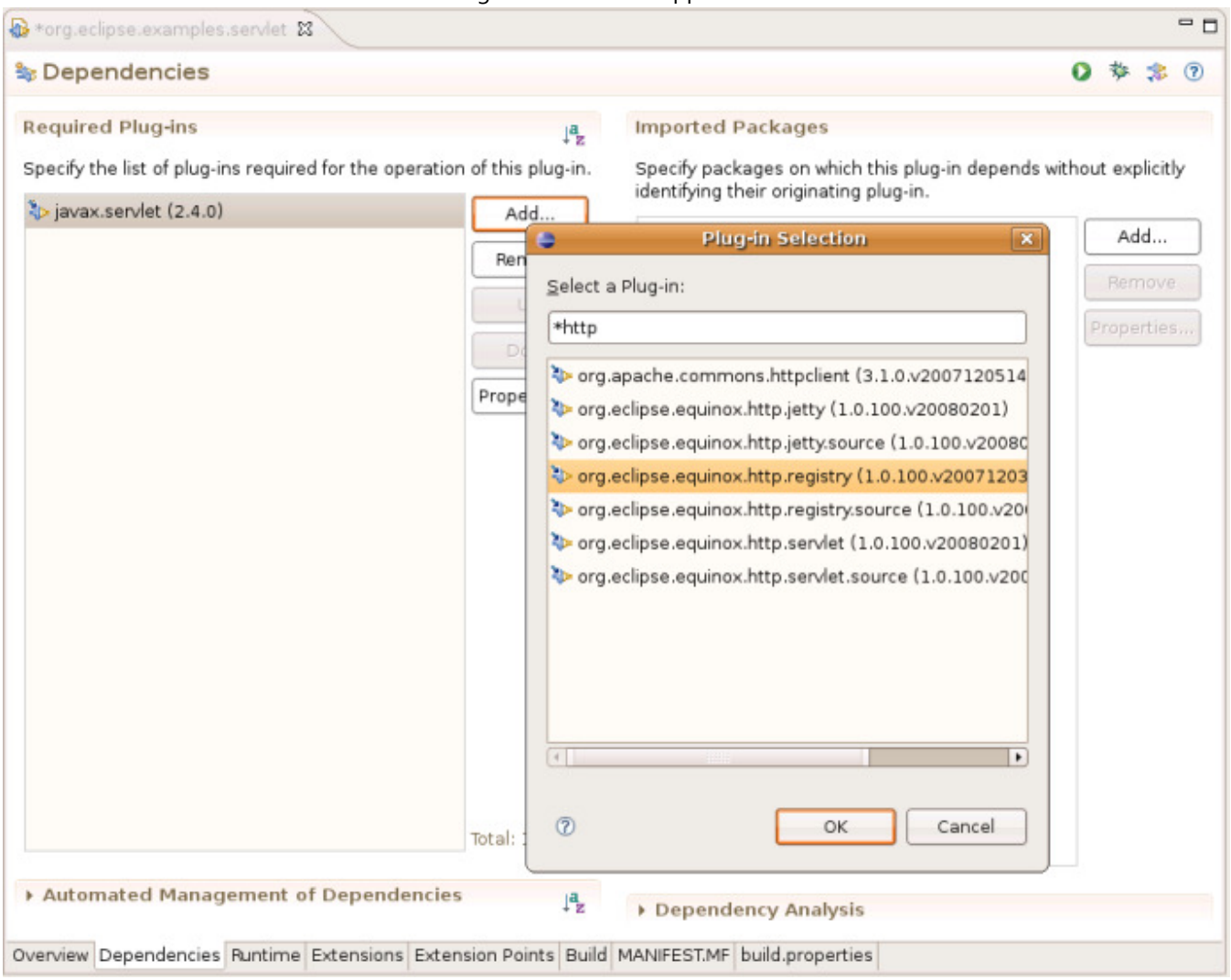
Too Many Dependencies

Turning on these options will add some dependencies to the project that we just don't need. As a general rule, it's best to keep the dependencies for a plug-in project as small as they need to be. At runtime, Equinox will attempt to resolve the dependencies for your bundle and—as part of that resolution process—will load those dependencies (as well as any dependencies they have). You could end up with a lot more bundles loaded than you really need.

In the worst case, your deployment may not contain some of the dependencies; if Equinox cannot resolve your bundle's dependencies, it will not load your bundle.

Configure dependencies for the Plug-in Project

After your new Plug-in project has been created, Eclipse will open the Manifest Editor for the project. Navigate to the "Dependencies" page in the new plug-in's Manifest Editor and add the `org.eclipse.equinox.http.registry` and `javax.servlet` bundles as "Required Plug-ins". Add dependencies by clicking the "Add" button and selecting them from the dialog box (as shown below).

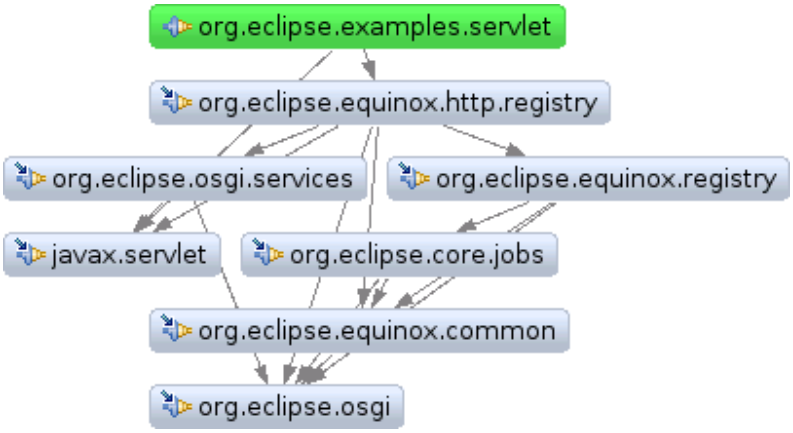


The Dependencies page of the Plug-in Project's Manifest Editor

Save the changes to the bundle's manifest.

Dependency Graph

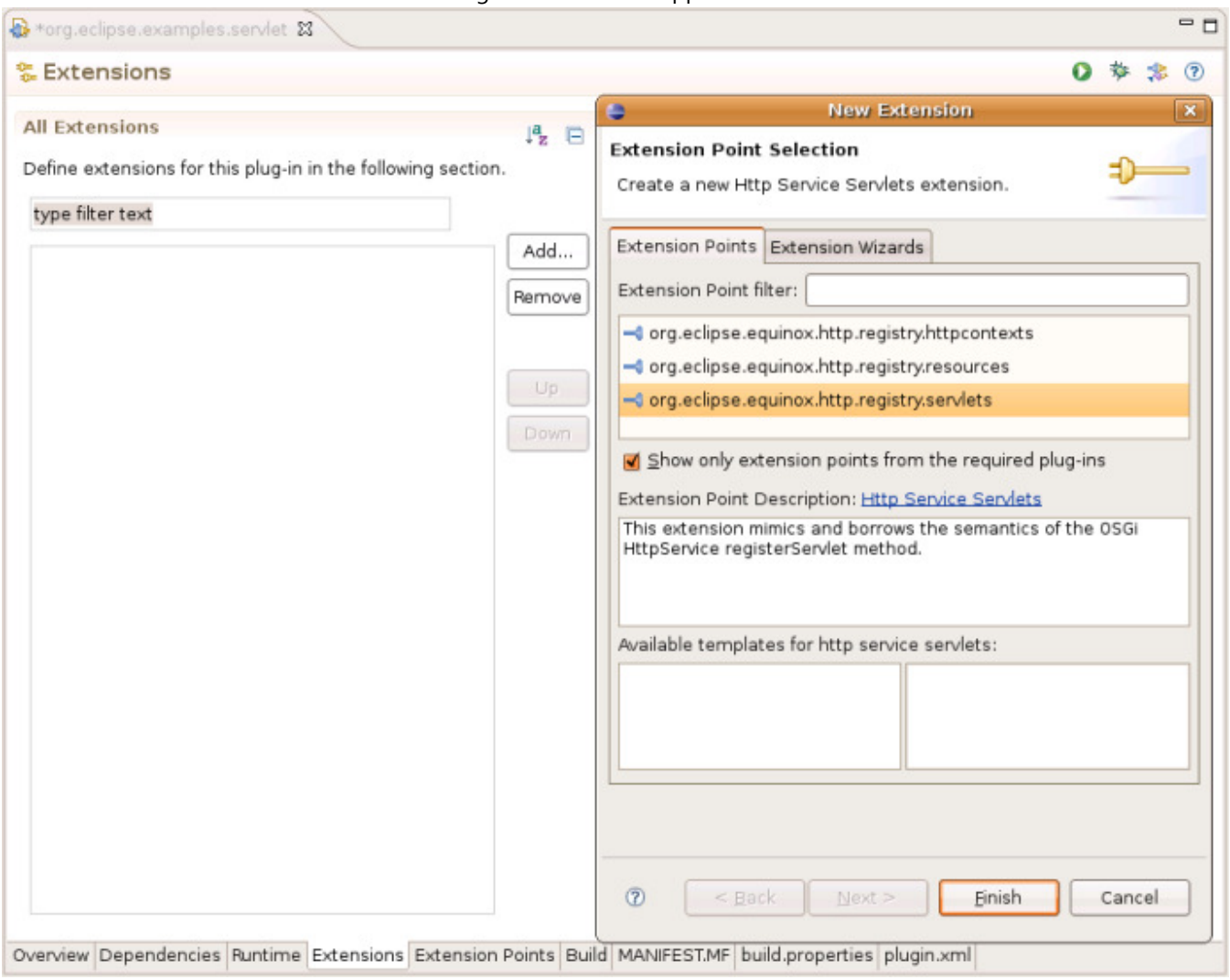
The new bundle has only two immediate dependencies, but the bundles that it depends on themselves depend on other bundles. Dependency graphs (the graph for our bundle is shown below) can get quite complex; thankfully Equinox manages the complexity for us.



The dependency graph for the new bundle

Create a servlet as an extension

Navigate to the "Extensions" page and add an extension to `org.eclipse.equinox.http.registry.servlets`. Click on the "Add" button to open the "New Extension" wizard.



Adding an extension using the Extensions page of the Plug-in Project's Manifest Editor

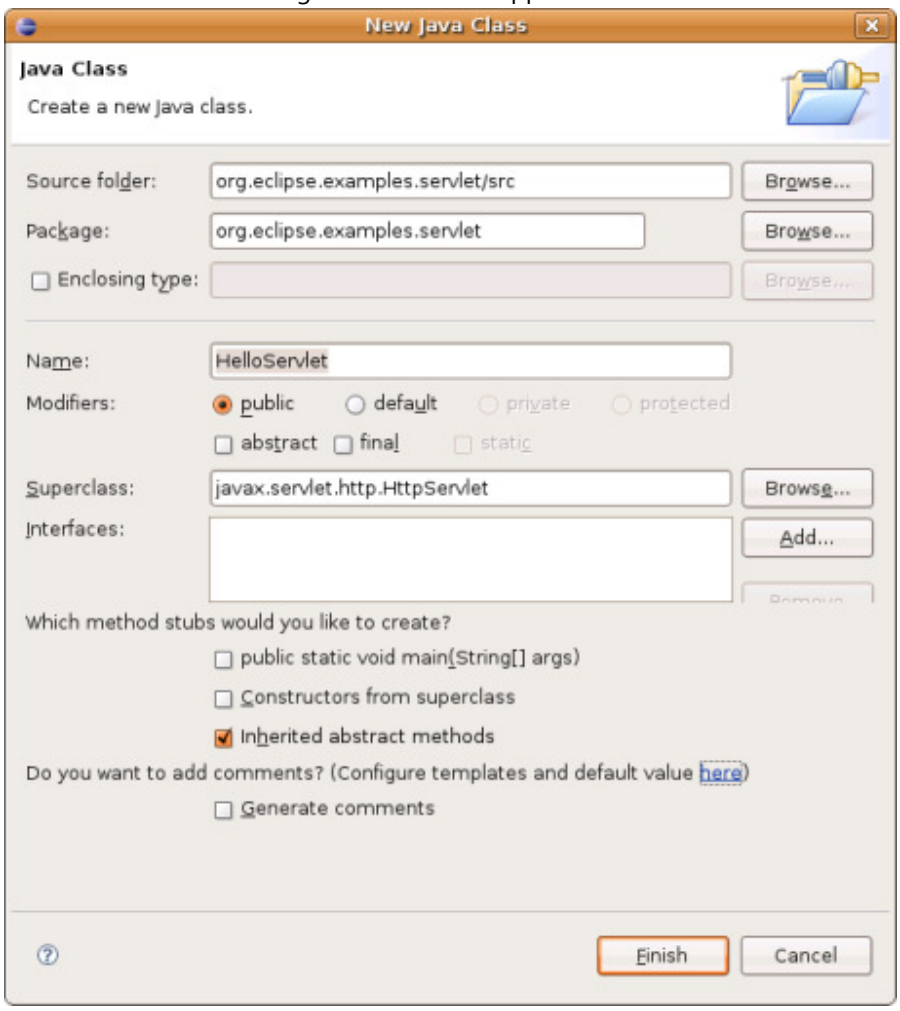
In the "Extension Details" area on the Extensions page (shown below), provide a suitable class name (`org.eclipse.examples.servlets.HelloServlet`) and alias (the alias is what you'll use to access the servlet; e.g. for "http://localhost/hello", the alias is "/hello").

Extension Details on the Extensions page

What's an Extension?

An extension is a mechanism by which bundle developers can hook into the Eclipse framework. The `org.eclipse.equinox.http.registry.servlets` extention point, provided by the `org.eclipse.equinox.http.registry` bundle, provides a hook for developers to add servlets to the framework. There are many extension points provided by the various bundles. You can use, for example, create extensions to add your own views, editors, menu entries, etc. to Eclipse.

Click on the "class*:" label to open the "New Class" wizard (shown below). Here, change the superclass to `javax.servlet.http.HttpServlet`, remove the suggested interface (`javax.servlet.Servlet` is implemented by `javax.servlet.http.HttpServlet` anyway), and click "Finish".



The New Class Wizard

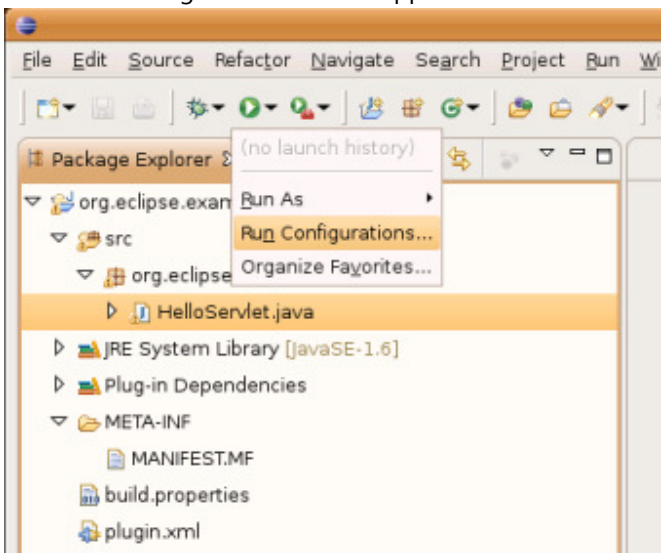
In the body of the resulting class, create a `doGet` method as shown below. The easiest way to do this is to put the cursor into the members area of the class, type some part of "doget", hit CTRL+1 for code assist, and select "`doGet(HttpServletRequest req, HttpServletResponse resp) void`" from the list.

Create a `doGet` method

Run and Test the Servlet

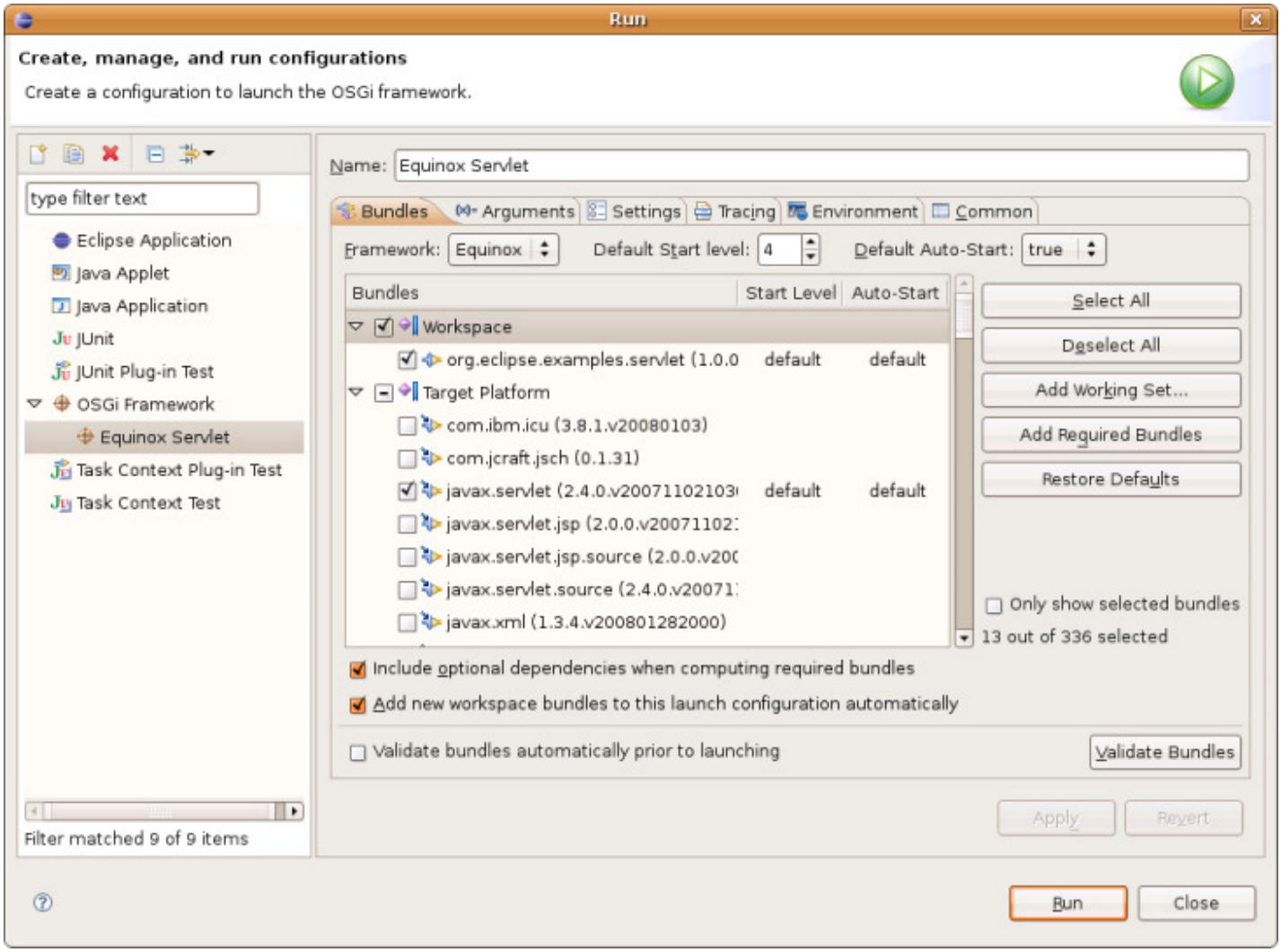
All that's left is to run it. You can start the bundle by simply right-clicking on the project in the Package Explorer and selecting "Run As... > OSGi Framework". While this will run your bundle, this default mechanism will include far more bundles than you actually require.

When you run something in Eclipse, a "launch configuration" is required. In many cases, Eclipse will automatically create a launch configuration for you, so many users do not even realize that they exist. Creating a launch configuration is relatively easy. From the "Run" drop down ( ), select "Open Run Dialog"



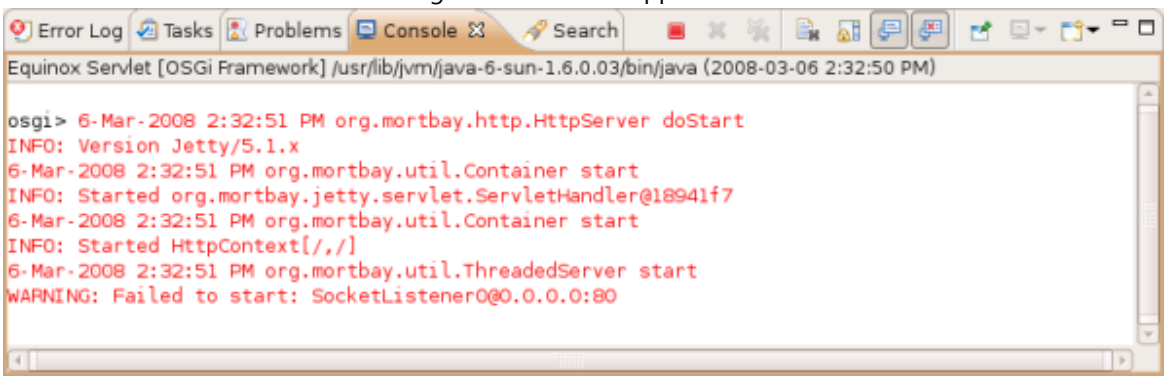
Create a launch configuration.

In the Run Configurations dialog, highlight "OSGi Framework" and click the "New Launch Configuration" button ( ). Provide a meaningful name for the launch configuration ("Equinox Servlet"). On the "Bundles", ensure that your bundle (`org.eclipse.examples.servlet`) has a check mark next to it. Remove the check from the "Target Platform" branch (which should be checked by default) and put a check next to the `org.eclipse.equinox.http.jetty` bundle. Then click the "Add Required Bundles" button. This will put checks next to all the bundles that are required to run (a total of 13 bundles should have check marks next to them). Click the "Run" button.



Create a launch configuration

Clicking run will bring the Console View into focus; the console shows the status of the HTTP service. The Equinox console can be used to view the status of the runtime. For a list of commands understood by the console, type "?" (typing any unrecognized command has the same effect).



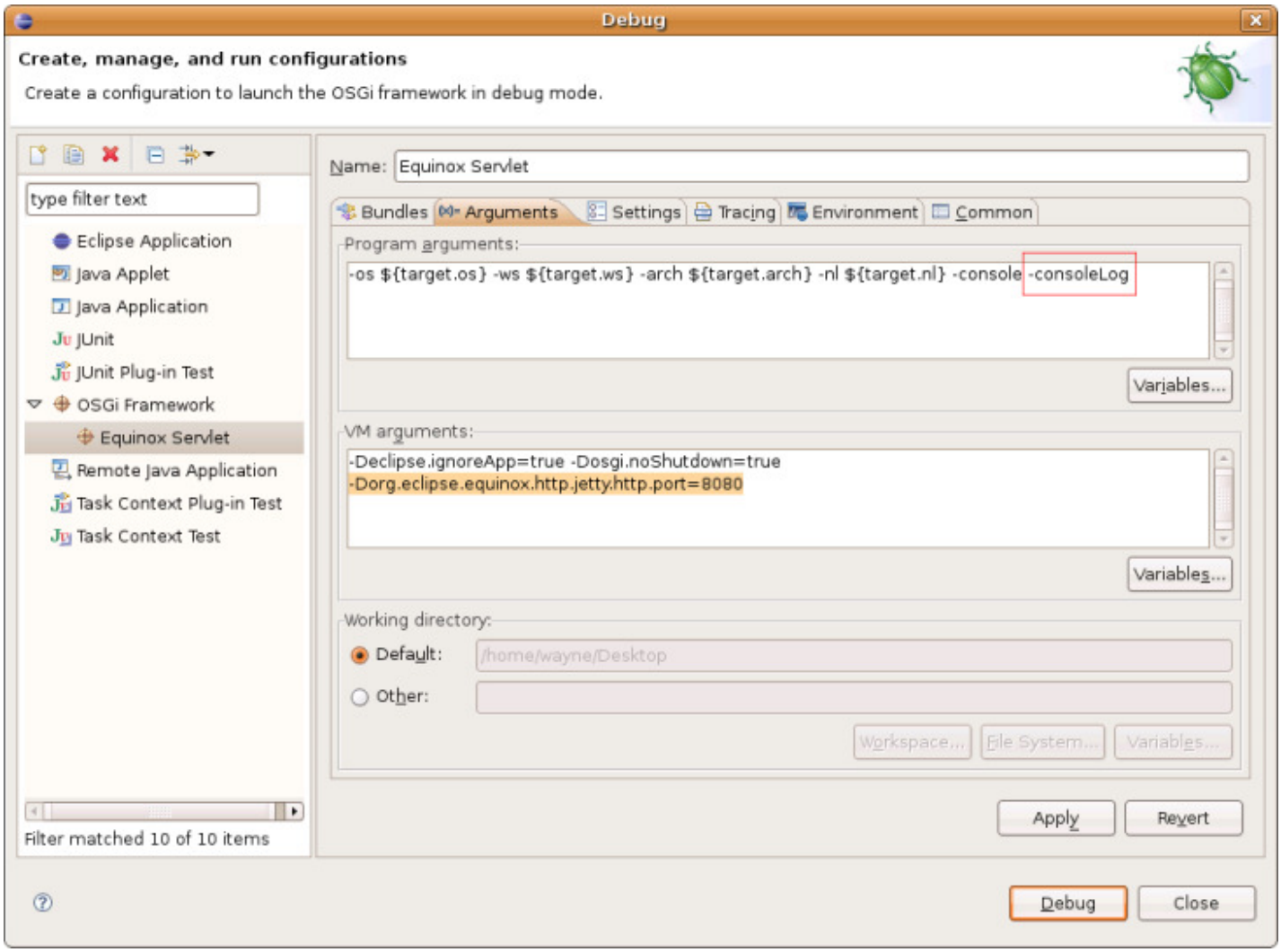
The Equinox console

You can shutdown the framework by typing "exit" at the prompt, or by clicking the "Stop" button ( ). You can relaunch the configuration using the "Run" ( ) or debug ( ) button/drop-down.

The astute reader will have noticed that the last message in the console states that the HTTP server failed to start. This is due to a UNIX and Linux restriction: non-root users cannot use ports below 1024 and—without explicit configuration—the HTTP server starts on the default port, 80. At this point, Windows users should be up-and-running and able to point their favorite web browser to "http://localhost/hello", resulting in both delight and thunderous applause from onlookers. UNIX/Linux (and most likely Mac OSX) users have a little more work to do.

Diagnosing the problem can be a challenge if you don't know where the log is. When you run an Equinox application, the logs end up in *.*log* files in the *[workspace]/.metadata/.plugins/org.eclipse.pde.core/Equinox Servlet* directory. You can also add the `-consoleLog` switch in the "Program Arguments" of your launch configuration which will cause logging information to be dumped onto the Equinox console. An entry in the log of the form `org.mortbay.util.MultiException[java.net.BindException: Permission denied]` indicates that the HTTP Server does not have permission to bind to the port.

The port that the HTTP server binds to can be configured by adding `-Dorg.eclipse.equinox.http.jetty.http.port=8080` to the "VM Arguments" section on the "Arguments" page.



Configure the port for the HTTP Server

Once running, you can access the servlet by including the port number (8080) in the URL: "http://localhost:8080/hello".

You might also find `org.mortbay.util.MultiException[java.net.BindException: Address already in use]` in the log which indicates that the port is already in use. In this case, either shutdown the process that's currently holding the port, or choose a different port.