Chapter 1: Server Setup and Configuration

Topics in This Chapter

- Downloading the JDK
- Obtaining a development server
- Configuring and testing the server
- Deploying and accessing HTML and JSP pages
- Setting up your development environment
- Deploying and accessing servlets
- Simplifying servlet and JSP deployment

Chapter 1

Before you can start learning specific servlet and JSP techniques, you need to have the right software and know how to use it. This introductory chapter explains how to obtain, configure, test, and use free versions of all the software needed to run servlets and JavaServer Pages.

1.1 Download the Java Development Kit (JDK)

You probably already have the JDK installed, but if not, installing it should be your first step. Version 2.3 of the servlet API and version 1.2 of the JSP API require the Java 2 platform (standard or enterprise edition). If you aren't using J2EE features like EJB or JNDI, I recommend that you use the standard edition, JDK 1.3 or 1.4.

For Solaris, Windows, and Linux, obtain JDK 1.3 at *http://java.sun.com/j2se/1.3/* and JDK 1.4 at *http://java.sun.com/j2se/1.4/*. For other platforms, check first whether a Java 2 implementation comes preinstalled as it does with MacOS X. If not, see Sun's list of third-party Java implementations at *http://java.sun.com/cgi-bin/java-ports.cgi*.

1.2 Download a Server for Your Desktop

Your second step is to download a server that implements the Java Servlet 2.3 and JSP 1.2 specifications for use on your desktop. In fact, I typically keep *two* servers installed on my desktop (Apache's free Tomcat server and one commercial server) and test my applications on both to keep myself from accidentally using nonportable constructs.

Regardless of the server that you will use for final deployment, you will want at least one server *on your desktop* for development. Even if the deployment server is in the office next to you connected by a lightning-fast network connection, you still don't want to use it for your development. Even a test server on your intranet that is inaccessible to customers is much less convenient for development purposes than a server right on your desktop. Running a development server on your desktop simplifies development in a number of ways, as compared to deploying to a remote server each and every time you want to test something.

1. **It is faster to test.** With a server on your desktop, there is no need to use FTP or another upload program. The harder it is for you to test changes, the less frequently you will test. Infrequent testing will let errors persist that will slow you down in the long run.
2. **It is easier to debug.** When running on your desktop, many servers display the standard output in a normal window. This is in contrast to deployment servers where the standard output is almost always either completely hidden or only available on the screen of the system administrator. So, with a desktop server, plain old `System.out.println` statements become useful tracing and debugging utilities.
3. **It is simple to restart.** During development, you will find that you need to restart the server frequently. For example, the server typically reads the *web.xml* file (see Chapter 4, "Using and Deploying Web Applications") only at startup. So, you normally have to restart the server each time you modify *web.xml*. Although some servers (e.g., ServletExec) have an interactive method of reloading *web.xml*, tasks such as clearing session data, resetting the `ServletContext`, or replacing modified class files used indirectly by servlets or JSP pages (e.g., beans or utility classes) may still necessitate restarting the server. Some older servers also need to be restarted because they implement servlet reloading unreliably. (Normally, servers instantiate the class that corresponds to a servlet only once and keep the instance in memory between requests. With *servlet reloading*, a server automatically replaces servlets that are in memory but whose class file has changed

on the disk). Besides, some deployment servers recommend completely disabling servlet reloading in order to increase performance. So, it is much more productive to develop in an environment where you can restart the server with a click of the mouse without asking for permission from other developers who might be using the server.

4. **It is more reliable to benchmark.** Although it is difficult to collect accurate timing results for short-running programs even in the best of circumstances, running benchmarks on systems that have heavy and varying system loads is notoriously unreliable.
5. **It is under your control.** As a developer, you may not be the administrator of the system on which the test or deployment server runs. You might have to ask some system administrator every time you want the server restarted. Or, the remote system may be down for a system upgrade at the most critical juncture of your development cycle. Not fun.

Now, if you can run on your desktop the same server you use for deployment, all the better. But one of the beauties of servlets and JSP is that you don't *have* to; you can develop with one server and deploy with another. Following are some of the most popular free options for desktop development servers. In all cases, the free version runs as a standalone Web server; in most cases, you have to pay for the deployment version that can be integrated with a regular Web server like Microsoft IIS, iPlanet/Netscape, or the Apache Web Server. However, the performance difference between using one of the servers as a servlet and JSP engine within a regular Web server and using it as a complete standalone Web server is not significant enough to matter during development. See *http://java.sun.com/products/servlet/industry.html* for a more complete list of servers.

- **Apache Tomcat.**
  Tomcat 4 is the official reference implementation of the servlet 2.3 and JSP 1.2 specifications. Tomcat 3 is the official reference implementation for servlets 2.2 and JSP 1.1. Both versions can be used as a standalone server during development or can be plugged into a standard Web server for use during deployment. Like all Apache products, Tomcat is entirely free and has complete source code available. Of all the servers, it also tends to be the one that is most compliant with the latest servlet and JSP specifications. However, the commercial servers tend to be better documented, easier to configure, and a bit faster. To download Tomcat, see *http://jakarta.apache.org/tomcat/*.
- **Allaire/Macromedia JRun.**
  JRun is a servlet and JSP engine that can be used in standalone mode for development or plugged into most common commercial Web servers for deployment. It is free for development purposes, but you

Update from Marty: There have been many Tomcat releases since the book was written. For the latest information on downloading and configuring Apache Tomcat, please see http://www.coreservlets.com/Apache-Tomcat-Tutorial/.

have to purchase a license before deploying with it. It is a popular choice among developers that are looking for easier administration than Tomcat. For details, see *http://www.allaire.com/products/JRun/*.

- **New Atlanta's ServletExec.** ServletExec is another popular servlet and JSP engine that can be used in standalone mode for development or, for deployment, plugged into the Microsoft IIS, Apache, and iPlanet/Netscape Web servers. Version 4.0 supports servlets 2.3 and JSP 1.2. You can download and use it for free, but some of the high-performance capabilities and administration utilities are disabled until you purchase a license. The ServletExec Debugger is the configuration you would use as a standalone desktop development server. For details, see *http://www.servletexec.com/*.
- **Caucho's Resin.**
  Resin is a fast servlet and JSP engine with extensive XML support. It is free for development and noncommercial deployment purposes. For details, see *http://www.caucho.com/*.
- **LiteWebServer from Gefion Software.**
  LWS is a small standalone Web server that supports servlets and JSP. It is free for both development and deployment purposes, but a license will entitle you to increased support and the complete server source code. See *http://www.gefionsoftware.com/LiteWebServer/* for details.

1.3 Change the Port and Configure Other Server Settings

Most of the free servers listed in Section 1.2 use a nonstandard default port in order to avoid conflicts with other Web servers that may be using the standard port (80). However, if you are using the servers in standalone mode and have no other server running permanently on port 80, you will find it more convenient to use port 80. That way, you don't have to use the port number in every URL you type in your browser. There are one or two other settings that you might want to modify as well.

Changing the port or other configuration settings is a server-specific process, so you need to read your server's documentation for definitive instructions. However, I'll give a quick summary of the process for three of the most popular free servers here: Tomcat, JRun, and ServletExec.

Apache Tomcat

Tomcat Port Number

With Tomcat 4, modifying the port number involves editing *install_dir/conf/server.xml*, changing the `port` attribute of the `Connector` element from 8080 to 80, and restarting the server. Remember that this section applies to the use of Tomcat in standalone mode on your desktop system where no other server is already running permanently on port 80. On Unix/Linux, you must have system administrator privileges to start services on port 80 or other port numbers below 1024. You probably have such privileges on your desktop machine; you do not necessarily have them on deployment servers.

The original element will look something like the following:

```
<Connector
  className="org.apache.catalina.connector.http.HttpConnector"
  port="8080" ...
  ... />
```

It should change to something like the following:

```
<Connector
  className="org.apache.catalina.connector.http.HttpConnector"
  port="80" ...
  ... />
```

The easiest way to find the correct entry is to search for 8080 in *server.xml*; there should only be one noncomment occurrence. Be sure to make a backup of *server.xml* before you edit it, just in case you make a mistake that prevents the server from running. Also, remember that XML is case sensitive, so for instance, you cannot replace `port` with `Port` or `Connector` with `connector`.

With Tomcat 3, you modify the same file (*install_dir/conf/server.xml*), but you need to use slightly different `Connector` elements for different minor releases of Tomcat. With version 3.2, you replace 8080 with 80 in the following `Parameter` element.

```
<Connector ...>
  <Parameter name="port" value="8080"/>
</Connector>
```

Again, restart the server after making the change.

Update from Marty: Configuration for Tomcat has changed substantially. For the latest information on downloading and configuring Apache Tomcat, please see http://www.coreservlets.com/Apache-Tomcat-Tutorial/.

Other Tomcat Settings

Besides the port, three additional Tomcat settings are important: the JAVA_HOME variable, the DOS memory settings, and the CATALINA_HOME or TOMCAT_HOME variable.

The most critical Tomcat setting is the JAVA_HOME environment variable—failing to set it properly prevents Tomcat from handling JSP pages. This variable should list the base JDK installation directory, not the *bin* subdirectory. For example, if you are on Windows 98/Me and installed the JDK in *C:\JDK1.3*, you might put the following line in your *autoexec.bat* file.

```
set JAVA_HOME=C:\JDK1.3
```

On Windows NT/2000, you would go to the Start menu and select Settings, then Control Panel, then System, then Environment. Then, you would enter the JAVA_HOME value.

On Unix/Linux, if the JDK is installed in */usr/j2sdk1_3_1* and you use the C shell, you would put the following into your *.cshrc* file.

```
setenv JAVA_HOME /usr/j2sdk1_3_1
```

Rather than setting the JAVA_HOME environment variable globally in the operating system, some developers prefer to edit the startup script to set it there. If you prefer this strategy, edit *install_dir/bin/catalina.bat* (Tomcat 4; Windows) or *install_dir/bin/tomcat.bat* (Tomcat 3; Windows) and change the following:

```
if not "%JAVA_HOME%" == "" goto gotJavaHome
echo You must set JAVA_HOME to point at ...
goto cleanup
:gotJavaHome
```

to:

```
if not "%JAVA_HOME%" == "" goto gotJavaHome
set JAVA_HOME=C:\JDK1.3
:gotJavaHome
```

Be sure to make a backup copy of *catalina.bat* or *tomcat.bat* before making the changes. Unix/Linux users would make similar changes in *catalina.sh* or *tomcat.sh*.

If you use Windows, you may also have to change the DOS memory settings for the startup and shutdown scripts. If you get an "Out of Environment Space" error message when you start the server, you will need to right-click on *install_dir/bin/startup.bat*, select Properties, select Memory, and change the Initial Environment entry from Auto to 2816. Repeat the process for *install_dir/bin/shutdown.bat*.

In some cases, it is also helpful to set the CATALINA_HOME (Tomcat 4) or TOMCAT_HOME (Tomcat 3) environment variables. This variable identifies the Tomcat

installation directory to the server. However, if you are careful to avoid copying the server startup scripts and you use only shortcuts (called "symbolic links" on Unix/Linux) instead, you are not required to set this variable. See Section 1.6 for more information on using these shortcuts.

Please note that this section describes the use of Tomcat as a standalone server for servlet and JSP *development*. It requires a totally different configuration to deploy Tomcat as a servlet and JSP container integrated within a regular Web server. For information on the use of Tomcat for deployment, please see *http://jakarta.apache.org/tomcat/tomcat-4.0-doc/*.

Allaire/Macromedia JRun

When using JRun in standalone mode (vs. integrated with a standard Web server), there are several options that you probably want to change from their default values. All can be set from the graphical JRun Management Console and/or through the JRun installation wizard.

JRun Port Number

To change the JRun port, first start the JRun Admin Server by clicking on the appropriate icon (on Windows, go to the Start menu, then Programs, then JRun 3.x). Then, click on the JRun Management Console (JMC) button or enter the URL *http://localhost:8000/* in a browser. Log in, using a username of `admin` and the password that you specified when you installed JRun, choose JRun Default Server, then select JRun Web Server. Figure 1–1 shows the result. Next, select Web Server Port, enter 80, and press Update. See Figure 1–2. Finally, select JRun Default Server again and press the Restart Server button.

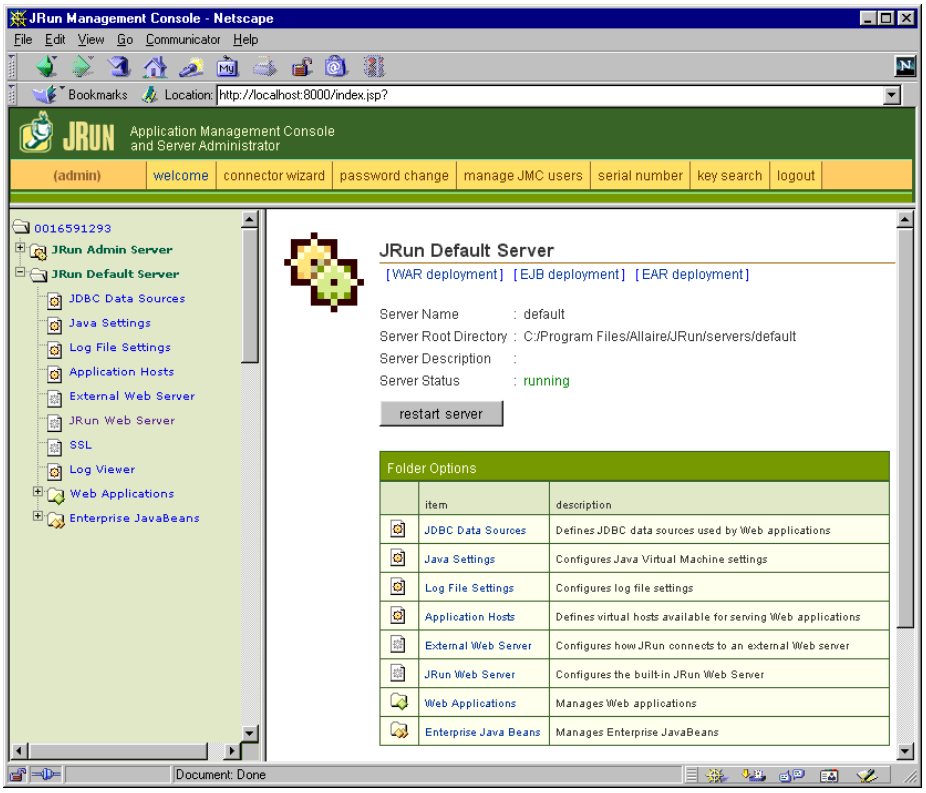

Figure 1–1 JMC configuration screen for the JRun Default Server.

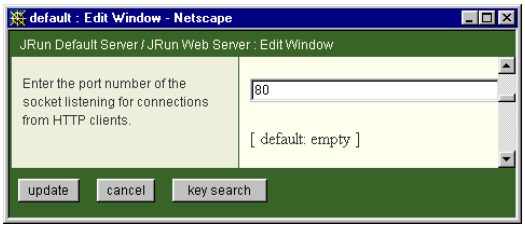

Figure 1–2 JRun Default Server port configuration window.

Other JRun Settings

When you install JRun, the installation wizard will ask you three questions that are particularly relevant to using JRun in standalone mode for development purposes. First, it will ask for a serial number. You can leave that blank; it is only required for deployment servers. Second, it will ask if you want to start JRun as a service. You

should *deselect* this option; starting JRun automatically is useful for deployment but inconvenient for development because the server icon does not appear in the taskbar, thus making it harder to restart the server. The wizard clearly states that using JRun as a service should be reserved for deployment, but since the service option is selected by default, you can easily miss it. Finally, you will be asked if you want to configure an external Web server. Decline this option; you need no separate Web server when using JRun in standalone mode.

New Atlanta ServletExec

The following settings apply to use of the ServletExec Debugger 4.0, the version of ServletExec that you would use for standalone desktop development (vs. integrated with a regular Web server for deployment).

ServletExec Port Number

To change the port number from 8080 to 80, edit *install_dir/StartSED40.bat* and add "-port 80" to the end of the line that starts the server, as below.

```
%JAVA_HOME%\bin\java ... ServletExecDebuggerMain -port 80
```

Remember that this section applies to the use of ServletExec in standalone mode on your desktop system where no other server is already running permanently on port 80. On Unix/Linux, you must have system administrator privileges to start services on port 80 or other port numbers below 1024. You probably have such privileges on your desktop machine; you do not necessarily have them on deployment servers.

Other ServletExec Settings

ServletExec shares two settings with Tomcat. The one required setting is the `JAVA_HOME` environment variable. As with Tomcat, this variable refers to the base installation directory of the JDK (not the *bin* subdirectory). For example, if the JDK is installed in *C:\JDK1.3*, you should modify the `JAVA_HOME` entry in *install_dir/StartSED40.bat* to look like the following.

```
set JAVA_HOME=C:\JDK1.3
```

Also as with Tomcat, if you use Windows, you may have to change the DOS memory settings for the startup script. If you get an "Out of Environment Space" error message when you start the server, you will need to right-click on *install_dir/bin/StartSED40.bat*, select Properties, select Memory, and change the Initial Environment entry from Auto to 2816.

1.4 Test the Server

Before trying your own servlets or JSP pages, you should make sure that the server is installed and configured properly. For Tomcat, click on *install_dir/bin/startup.bat* (Windows) or execute *install_dir/bin/startup.sh* (Unix/Linux). For JRun, go to the Start menu and select Programs, JRun 3.1, and JRun Default Server. For ServletExec, click on *install_dir/bin/StartSED40.bat*. In all three cases, enter the URL *http://localhost/* in your browser and make sure you get a regular Web page, not an error message saying that the page cannot be displayed or that the server cannot be found. Figures 1–3 through 1–5 show typical results. If you chose not to change the port number to 80 (see Section 1.3, "Change the Port and Configure Other Server Settings"), you will need to use a URL like *http://localhost:8080/* that includes the port number.

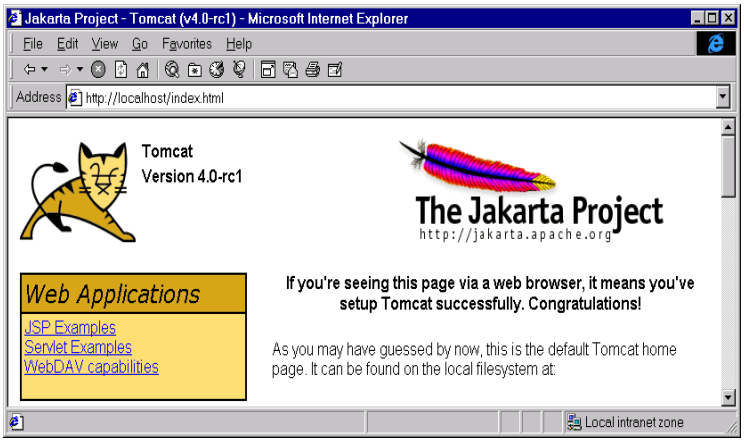

Figure 1–3 Initial home page for Tomcat 4.0.

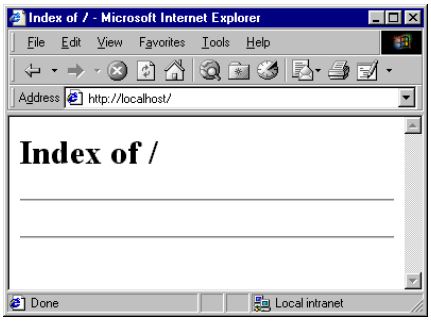

Figure 1–4 Initial home page for JRun 3.1.

Figure 1–5 Initial home page for ServletExec 4.0.

1.5 Try Some Simple HTML and JSP Pages

After you have verified that the server is running, you should make sure that you can install and access simple HTML and JSP pages. This test, if successful, shows two important things. First, successfully accessing an HTML page shows that you understand which directories should hold HTML and JSP files. Second, successfully accessing a new JSP page shows that the Java compiler (not just the Java virtual machine) is configured properly.

Eventually, you will almost certainly want to create and use your own Web applications (see Chapter 4, "Using and Deploying Web Applications"), but for initial testing I recommend that you use the default Web application. Although Web applications follow a common directory structure, the exact location of the default Web application is server specific. Check your server's documentation for definitive instructions, but I summarize the locations for Tomcat, JRun, and ServletExec in the following list. Where I list *SomeDirectory* you can use any directory name you like. (But you are never allowed to use *WEB-INF* or *META-INF* as directory names. For the default Web application, you also have to avoid a directory name that matches the URL prefix of any other Web application.)

- **Tomcat Directory**
  install_dir/webapps/ROOT
  (or *install_dir/webapps/ROOT/SomeDirectory*)
- **JRun Directory**
  install_dir/servers/default/default-app
  (or *install_dir/servers/default/default-app/SomeDirectory*)
- **ServletExec Directory**
  install_dir/public_html[1]
  (or *install_dir/public_html/SomeDirectory*)
- **Corresponding URLs**
  http://host/Hello.html
  (or *http://host/SomeDirectory/Hello.html*)
  http://host/Hello.jsp
  (or *http://host/SomeDirectory/Hello.jsp*)

For your first tests, I suggest you simply take *Hello.html* (Listing 1.1, Figure 1–6) and *Hello.jsp* (Listing 1.2, Figure 1–7) and drop them into the appropriate locations. The code for these files, as well as *all* the code from the book, is available online at *http://www.moreservlets.com*. That Web site also contains updates, additions, information on short courses, and the full text of *Core Servlets and JavaServer Pages* (in PDF). If neither the HTML file nor the JSP file works (e.g., you get File Not Found—404—errors), you likely are using the wrong directory for the files. If the HTML file works but the JSP file fails, you probably have incorrectly specified the base JDK directory (e.g., with the `JAVA_HOME` variable).

1. Note that the *public_html* directory is created automatically by ServletExec the first time you run the server. So, you will be unable to find *public_html* if you have not yet tested the server as described in Section 1.4 (Test the Server).

Listing 1.1 *Hello.html*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>HTML Test</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1>HTML Test</H1>
Hello.
</BODY>
</HTML>
```

Listing 1.2 *Hello.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>JSP Test</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1>JSP Test</H1>
Time: <%= new java.util.Date() %>
</BODY>
</HTML>
```

Figure 1–6 Result of *Hello.html*.

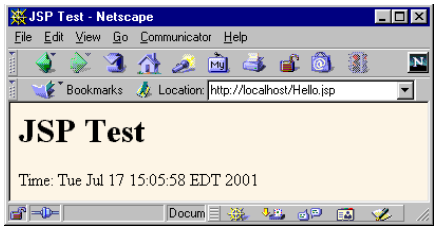

Figure 1–7 Result of *Hello.jsp*.

1.6 Set Up Your Development Environment

The server startup script automatically sets the server's `CLASSPATH` to include the standard servlet and JSP classes and the *WEB-INF/classes* directory (containing compiled servlets) of each Web application. But *you* need similar settings, or you will be unable to compile servlets in the first place. This section summarizes the configuration needed for servlet development.

Create a Development Directory

The first thing you should do is create a directory in which to place the servlets and JSP pages that you develop. This directory can be in your home directory (e.g., *~/ServletDevel* on Unix) or in a convenient general location (e.g., *C:\ServletDevel* on Windows). It should *not*, however, be in the server's installation directory.

Eventually, you will organize this development directory into different Web applications (each with a common structure—see Chapter 4). For initial testing of your environment, however, you can just put servlets either directly in the development directory (for packageless servlets) or in a subdirectory that matches the servlet package name. Many developers simply put all their code in the server's deployment directory (see Section 1.9). I strongly discourage this practice and instead recommend one of the approaches described in Section 1.8 (Establish a Simplified Deployment Method). Although developing in the deployment directory seems simpler at the beginning since it requires no copying of files, it significantly complicates matters in the long run. Mixing locations makes it hard to separate an operational version from a version you are testing, makes it difficult to test on multiple servers, and makes organization much more complicated. Besides, your desktop is almost certainly not the final deployment server, so you'll eventually have to develop a good system for deploying anyhow.

Core Warning

Don't use the server's deployment directory as your development location. Instead, keep a separate development directory.

Make Shortcuts to Start and Stop the Server

Since I find myself frequently restarting the server, I find it convenient to place shortcuts to the server startup and shutdown icons inside my main development directory. You will likely find it convenient to do the same.

For example, for Tomcat on Windows, go to *install_dir/bin*, right-click on *startup.bat*, and select Copy. Then go to your development directory, right-click in the window, and select Paste Shortcut (not just Paste). Repeat the process for *install_dir/bin/shutdown.bat*. On Unix, you would use `ln -s` to make a symbolic link to *startup.sh*, *tomcat.sh* (needed even though you don't directly invoke this file), and *shutdown.sh*.

For JRun on Windows, go to the Start menu, select Programs, select JRun 3.x, right-click on the JRun Default Server icon, and select Copy. Then go to your development directory, right-click in the window, and select Paste Shortcut. Repeat the process for the JRun Admin Server and JRun Management Console.

For the ServletExec Debugger (i.e., standalone development server), go to *install_dir*, right-click on *StartSED40.bat*, and select Copy. Then go to your development directory, right-click in the window, and select Paste Shortcut (not just Paste). There is no separate shutdown file; to stop ServletExec, just go to *http://localhost/* (see Figure 1–5) and click on the Shutdown link in the General category on the left-hand side.

Set Your CLASSPATH

Since servlets and JSP are not part of the Java 2 platform, standard edition, you have to identify the servlet classes to the compiler. The *server* already knows about the servlet classes, but the *compiler* (i.e., `javac`) you use for development probably doesn't. So, if you don't set your `CLASSPATH`, attempts to compile servlets, tag libraries, or other classes that use the servlet API will fail with error messages about unknown classes. The exact location of the servlet JAR file varies from server to server. In most cases, you can hunt around for a file called *servlet.jar*. Or, read your server's documentation to discover the location. Once you find the JAR file, add the location to your development `CLASSPATH`. Here are the locations for some common development servers:

- **Tomcat 4 Location.**
  install_dir/common/lib/servlet.jar
- **Tomcat 3 Location.**
  install_dir/lib/servlet.jar
- **JRun Location.**
  install_dir/lib/ext/servlet.jar

- **ServletExec Location.**
  install_dir/ServletExecDebugger.jar

Now, in addition to the servlet JAR file, you also need to put your development directory in the CLASSPATH. Although this is not necessary for simple packageless servlets, once you gain experience you will almost certainly use packages. Compiling a file that is in a package and that uses another class in the same package requires the CLASSPATH to include the directory that is at the top of the package hierarchy. In this case, that's the development directory I just discussed in the first subsection. Forgetting this setting is perhaps *the* most common mistake made by beginning servlet programmers.

Core Approach

Remember to add your development directory to your CLASSPATH. Otherwise, you will get "Unresolved symbol" error messages when you attempt to compile servlets that are in packages and that make use of other classes in the same package.

Finally, you should include "." (the current directory) in the CLASSPATH. Otherwise, you will only be able to compile packageless classes that are in the top-level development directory.

Here are a few representative methods of setting the CLASSPATH. They assume that your development directory is *C:\devel* (Windows) or */usr/devel* (Unix/Linux) and that you are using Tomcat 4. Replace *install_dir* with the actual base installation location of the server. Be sure to use the appropriate case for the filenames. Note that these examples represent only one approach for setting the CLASSPATH. Many Java integrated development environments have a global or project-specific setting that accomplishes the same result. But these settings are totally IDE-specific and won't be discussed here.

- **Windows 98/Me.** Put the following in your *autoexec.bat*. (Note that this all goes on one line with no spaces—it is broken here for readability.)

```
set CLASSPATH=.;
              C:\devel;
              install_dir\common\lib\servlet.jar
```

- **Windows NT/2000.** Go to the Start menu and select Settings, then Control Panel, then System, then Environment. Then, enter the CLASSPATH value from the previous bullet.

- **Unix/Linux (C shell).** Put the following in your *.cshrc*. (Again, in the real file it goes on a single line without spaces.)
```
setenv CLASSPATH .:
                 /usr/devel:
                 install_dir/common/lib/servlet.jar
```

Bookmark or Install the Servlet and JSP API Documentation

Just as no serious programmer should develop general-purpose Java applications without access to the JDK 1.3 or 1.4 API documentation (in Javadoc format), no serious programmer should develop servlets or JSP pages without access to the API for classes in the `javax.servlet` packages. Here is a summary of where to find the API:

- ***http://java.sun.com/products/jsp/download.html***
  This site lets you download the Javadoc files for either the servlet 2.3 and JSP 1.2 API or for the servlet 2.2 and JSP 1.1 API. You will probably find this API so useful that it will be worth having a local copy instead of browsing it online. However, some servers bundle this documentation, so check before downloading.
- ***http://java.sun.com/products/servlet/2.3/javadoc/***
  This site lets you browse the servlet 2.3 API online.
- ***http://java.sun.com/products/servlet/2.2/javadoc/***
  This site lets you browse the servlet 2.2 and JSP 1.1 API online.
- ***http://java.sun.com/j2ee/j2sdkee/techdocs/api/***
  This address lets you browse the complete API for the Java 2 Platform, Enterprise Edition (J2EE), which includes the servlet 2.2 and JSP 1.1 packages.

Update from Marty: Here are newer links to bookmark:
- Servlet 2.4 API: http://tomcat.apache.org/tomcat-5.0-doc/servletapi/
- JSP 2.0 API: http://tomcat.apache.org/tomcat-5.0-doc/jspapi/
- Java 5 API: http://java.sun.com/j2se/1.5.0/docs/api/

1.7 Compile and Test Some Simple Servlets

OK, so your environment is all set. At least you *think* it is. It would be nice to confirm that hypothesis. Following are three tests that help verify this.

Test 1: A Servlet That Does Not Use Packages

The first servlet to try is a basic one: no packages, no utility (helper) classes, just simple HTML output. Rather than writing your own test servlet, you can just grab *HelloServlet.java* (Listing 1.3) from the book's source code archive at *http://www.moreservlets.com*. If you get compilation errors, go back and check your `CLASSPATH` settings (Section 1.6)—you most likely erred in listing the location of the JAR file that contains the servlet classes (e.g., *servlet.jar*). Once you compile *HelloServlet.java*, put *HelloServlet.class* in the appropriate location (usually the *WEB-INF/classes* directory of your server's default Web application). Check your server's documentation for this location, or see the following list for a summary of the locations used by Tomcat, JRun, and ServletExec. Then, access the servlet with the URL *http://localhost/servlet/HelloServlet* (or *http://localhost:8080/servlet/HelloServlet* if you chose not to change the port number as described in Section 1.3). You should get something similar to Figure 1–8. If this URL fails but the test of the server itself (Section 1.4) succeeded, you probably put the class file in the wrong directory.

- **Tomcat Directory.**
  install_dir/webapps/ROOT/WEB-INF/classes
- **JRun Directory.**
  install_dir/servers/default/default-app/WEB-INF/classes
- **ServletExec Directory.**
  install_dir/Servlets
- **Corresponding URL.**
  http://host/servlet/HelloServlet

Listing 1.3 *HelloServlet.java*

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Simple servlet used to test server. */

public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
```

Listing 1.3 *HelloServlet.java (continued)*

```
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>Hello</H1>\n" +
                "</BODY></HTML>");
  }
}
```

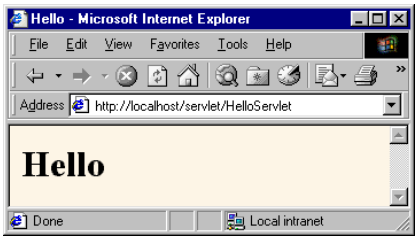

Figure 1–8 Result of `HelloServlet`.

Test 2: A Servlet That Uses Packages

The second servlet to try is one that uses packages but no utility classes. Again, rather than writing your own test, you can grab *HelloServlet2.java* (Listing 1.4) from the book's source code archive at *http://www.moreservlets.com*. Since this servlet is in the `moreservlets` package, it should go in the *moreservlets* directory, both during development and when deployed to the server. If you get compilation errors, go back and check your `CLASSPATH` settings (Section 1.6)—you most likely forgot to include "." (the current directory). Once you compile *HelloServlet2.java*, put *HelloServlet2.class* in the *moreservlets* subdirectory of whatever directory the server uses for servlets that are not in custom Web applications (usually the *WEB-INF/classes* directory of the default Web application). Check your server's documentation for this location, or see the following list for a summary of the locations for Tomcat, JRun, and ServletExec. For now, you can simply copy the class file from the development directory to the deployment directory, but Section 1.8 (Establish a Simplified Deployment Method) will provide some options for simplifying the process.

Once you have placed the servlet in the proper directory, access it with the URL *http://localhost/servlet/moreservlets.HelloServlet2*. You should get something similar

to Figure 1–9. If this test fails, you probably either typed the URL wrong (e.g., used a slash instead of a dot after the package name) or put *HelloServlet2.class* in the wrong location (e.g., directly in the server's *WEB-INF/classes* directory instead of in the *moreservlets* subdirectory).

- **Tomcat Directory.**
  install_dir/webapps/ROOT/WEB-INF/classes/moreservlets
- **JRun Directory.**
  install_dir/servers/default/default-app/WEB-INF/classes/moreservlets
- **ServletExec Directory.**
  install_dir/Servlets/moreservlets
- **Corresponding URL.**
  http://host/servlet/moreservlets.HelloServlet2

Listing 1.4 *moreservlets/HelloServlet2.java*

```
package moreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Simple servlet used to test the use of packages. */

public class HelloServlet2 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello (2)</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>Hello (2)</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Figure 1–9 Result of `HelloServlet2`.

Test 3: A Servlet That Uses Packages and Utilities

The final servlet you should test to verify the configuration of your server and development environment is one that uses both packages and utility classes. Listing 1.5 presents *HelloServlet3.java*, a servlet that uses the `ServletUtilities` class (Listing 1.6) to simplify the generation of the `DOCTYPE` (specifies the HTML version—useful when using HTML validators) and `HEAD` (specifies the title) portions of the HTML page. Those two parts of the page are useful (technically required, in fact), but are tedious to generate with servlet `println` statements. Again, the source code can be found at *http://www.moreservlets.com*.

Since both the servlet and the utility class are in the `moreservlets` package, they should go in the *moreservlets* directory. If you get compilation errors, go back and check your `CLASSPATH` settings (Section 1.6)—you most likely forgot to include the top-level development directory. I've said it before, but I'll say it again: your `CLASSPATH` must include the top-level directory of your package hierarchy before you can compile a packaged class that makes use of another class from the same package. This requirement is not particular to servlets; it is the way packages work on the Java platform in general. Nevertheless, many servlet developers are unaware of this fact, and it is one of the (perhaps *the*) most common errors beginning developers encounter.

Core Warning

Your `CLASSPATH` must include your top-level development directory. Otherwise, you cannot compile servlets that are in packages and that also use classes from the same package.

Once you compile *HelloServlet3.java* (which will automatically cause *ServletUtilities.java* to be compiled), put *HelloServlet3.class* and *ServletUtilities.class* in the *moreservlets* subdirectory of whatever directory the server uses for servlets that are not in custom Web applications (usually the *WEB-INF/classes* directory of the default Web application). Check your server's documentation for this location, or see the following list for a summary of the locations used by Tomcat, JRun, and ServletExec. Then, access the servlet with the URL *http://localhost/servlet/moreservlets.HelloServlet3*. You should get something similar to Figure 1–10.

- **Tomcat Directory.**
  install_dir/webapps/ROOT/WEB-INF/classes/moreservlets
- **JRun Directory.**
  install_dir/servers/default/default-app/WEB-INF/classes/moreservlets
- **ServletExec Directory.**
  install_dir/Servlets/moreservlets
- **Corresponding URL.**
  http://host/servlet/moreservlets.HelloServlet3

Listing 1.5 *moreservlets/HelloServlet3.java*

```
package moreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Simple servlet used to test the use of packages
 *  and utilities from the same package.
 */

public class HelloServlet3 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Hello (3)";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>" + title + "</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Listing 1.6 *moreservlets/ServletUtilities.java*

```
package moreservlets;

import javax.servlet.*;
import javax.servlet.http.*;

/** Some simple time savers. Note that most are static methods. */

public class ServletUtilities {
  public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">";

  public static String headWithTitle(String title) {
    return(DOCTYPE + "\n" +
           "<HTML>\n" +
           "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
  }

  ...
}
```

Figure 1–10 Result of `HelloServlet3`.

1.8 Establish a Simplified Deployment Method

OK, so you have a development directory. You can compile servlets with or without packages. You know which directory the servlet classes belong in. You know the URL that should be used to access them (at least the default URL; in Section 5.3, "Assigning Names and Custom URLs," you'll see how to customize that address). But how do you move the class files from the development direc-

tory? Copying each one by hand every time is tedious and error prone. Once you start using Web applications (see Chapter 4), copying individual files becomes even more cumbersome.

There are several options to simplify the process. Here are a few of the most popular ones. If you are just beginning with servlets and JSP, you probably want to start with the first option and use it until you become comfortable with the development process. Note that I do *not* list the option of putting your code directly in the server's deployment directory. Although this is one of the most common choices among beginners, it scales so poorly to advanced tasks that I recommend you steer clear of it from the start.

1. Copy to a shortcut or symbolic link.
2. Use the `-d` option of `javac`.
3. Let your IDE take care of deployment.
4. Use `ant` or a similar tool.

Details on these four options are given in the following subsections.

Copy to a Shortcut or Symbolic Link

On Windows, go to the server's default Web application, right-click on the *classes* directory, and select Copy. Then go to your development directory, right-click, and select Paste Shortcut (not just Paste). Now, whenever you compile a packageless servlet, just drag the class files onto the shortcut. When you develop in packages, use the right mouse to drag the entire directory (e.g., the *moreservlets* directory) onto the shortcut, release the mouse, and select Copy. On Unix/Linux, you can use symbolic links (created with `ln -s`) in a manner similar to that for Windows shortcuts.

An advantage of this approach is that it is simple. So, it is good for beginners who want to concentrate on learning servlets and JSP, not deployment tools. Another advantage is that a variation applies once you start using your own Web applications (see Chapter 4). Just make a shortcut to the main Web application directory (one level up from the top of the default Web application), and copy the entire Web application each time by using the right mouse to drag the directory that contains your Web application onto this shortcut and selecting Copy.

One disadvantage of this approach is that it requires repeated copying if you use multiple servers. For example, I keep at least two different servers on my development system and regularly test my code with both servers. A second disadvantage is that this approach copies both the Java source code files and the class files to the server, whereas only the class files are needed. This may not matter much on your desktop server, but when you get to the "real" deployment server, you won't want to include the source code files.

Use the -d Option of javac

By default, the Java compiler (`javac`) places class files in the same directory as the source code files that they came from. However, `javac` has an option (`-d`) that lets you designate a different location for the class files. You need only specify the top-level directory for class files—`javac` will automatically put packaged classes in subdirectories that match the package names. So, for example, with Tomcat I could compile the `HelloServlet2` servlet (Listing 1.4, Section 1.7) as follows (line break added only for clarity; omit it in real life).

```
javac -d install_dir/webapps/ROOT/WEB-INF/classes
        HelloServlet2.java
```

You could even make a Windows batch file or Unix shell script or alias that makes a command like `servletc` expand to `javac -d install_dir/.../classes`. See *http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javac.html* for more details on `-d` and other `javac` options.

An advantage of this approach is that it requires no manual copying of class files. Furthermore, the exact same command can be used for classes in different packages since `javac` automatically puts the class files in a subdirectory matching the package.

The main disadvantage is that this approach applies only to Java class files; it won't work for deploying HTML and JSP pages, much less entire Web applications.

Let Your IDE Take Care of Deployment

Most servlet- and JSP-savvy development environments (e.g., IBM WebSphere Studio, Macromedia JRun Studio, Borland JBuilder) have options that let you tell the IDE where to deploy class files for your project. Then, when you tell the IDE to build the project, the class files are automatically deployed to the proper location (package-specific subdirectories and all).

An advantage of this approach, at least in some IDEs, is that it can deploy HTML and JSP pages and even entire Web applications, not just Java class files. A disadvantage is that it is an IDE-specific technique and thus is not portable across systems.

Use ant or a Similar Tool

Developed by the Apache foundation, `ant` is a tool similar to the Unix `make` utility. However, `ant` is written in the Java programming language (and thus is portable) and is touted to be both simpler to use and more powerful than `make`. Many servlet and JSP developers use `ant` for compiling and deploying. The use of `ant` is especially popular among Tomcat users and with those developing Web applications (see Chapter 4).

For general information on using `ant`, see *http://jakarta.apache.org/ant/manual/*. See *http://jakarta.apache.org/tomcat/tomcat-4.0-doc/appdev/processes.html* for specific guidance on using `ant` with Tomcat.

The main advantage of this approach is flexibility: `ant` is powerful enough to handle everything from compiling the Java source code to copying files to producing WAR files (Section 4.3). The disadvantage of `ant` is the overhead of learning to use it; there is more of a learning curve with `ant` than with the other techniques in this section.

1.9 Deployment Directories for Default Web Application: Summary

The following subsections summarize the way to deploy and access HTML files, JSP pages, servlets, and utility classes in Tomcat, JRun, and ServletExec. The summary assumes that you are deploying files in the default Web application, have changed the port number to 80 (see Section 1.3), and are accessing servlets through the default URL (i.e., *http://host/servlet/ServletName*). Later chapters explain how to deploy user-defined Web applications and how to customize the URLs. But you'll probably want to start with the defaults just to confirm that everything is working properly. The Appendix (Server Organization and Structure) gives a unified summary of the directories used by Tomcat, JRun, and ServletExec for both the default Web application and custom Web applications.

If you are using a server on your desktop, you can use *localhost* for the *host* portion of each of the URLs in this section.

Tomcat

HTML and JSP Pages

- **Main Location.**
  install_dir/webapps/ROOT
- **Corresponding URLs.**
  http://host/SomeFile.html
  http://host/SomeFile.jsp
- **More Specific Location (Arbitrary Subdirectory).**
  install_dir/webapps/ROOT/SomeDirectory
- **Corresponding URLs.**
  http://host/SomeDirectory/SomeFile.html
  http://host/SomeDirectory/SomeFile.jsp

Individual Servlet and Utility Class Files

- **Main Location (Classes without Package).**
  install_dir/webapps/ROOT/WEB-INF/classes
- **Corresponding URL (Servlets).**
  http://host/servlet/ServletName
- **More Specific Location (Classes in Packages).**
  install_dir/webapps/ROOT/WEB-INF/classes/packageName
- **Corresponding URL (Servlets in Packages).**
  http://host/servlet/packageName.ServletName

Servlet and Utility Class Files Bundled in JAR Files

- **Location.**
  install_dir/webapps/ROOT/WEB-INF/lib
- **Corresponding URLs (Servlets).**
  http://host/servlet/ServletName
  http://host/servlet/packageName.ServletName

JRun

HTML and JSP Pages

- **Main Location.**
  install_dir/servers/default/default-app
- **Corresponding URLs.**
  http://host/SomeFile.html
  http://host/SomeFile.jsp
- **More Specific Location (Arbitrary Subdirectory).**
  install_dir/servers/default/default-app/SomeDirectory
- **Corresponding URLs.**
  http://host/SomeDirectory/SomeFile.html
  http://host/SomeDirectory/SomeFile.jsp

Individual Servlet and Utility Class Files

- **Main Location (Classes without Package).**
  install_dir/servers/default/default-app/WEB-INF/classes
- **Corresponding URL (Servlets).**
  http://host/servlet/ServletName

- **More Specific Location (Classes in Packages).**
  install_dir/servers/default/default-app/WEB-INF/classes/packageName
- **Corresponding URL (Servlets in Packages).**
  http://host/servlet/packageName.ServletName

Servlet and Utility Class Files Bundled in JAR Files

- **Location.**
  install_dir/servers/default/default-app/WEB-INF/lib
- **Corresponding URLs (Servlets).**
  http://host/servlet/ServletName
  http://host/servlet/packageName.ServletName

ServletExec

HTML and JSP Pages

- **Main Location.**
  install_dir/public_html
- **Corresponding URLs.**
  http://host/SomeFile.html
  http://host/SomeFile.jsp
- **More Specific Location (Arbitrary Subdirectory).**
  install_dir/public_html/SomeDirectory
- **Corresponding URLs.**
  http://host/SomeDirectory/SomeFile.html
  http://host/SomeDirectory/SomeFile.jsp

Individual Servlet and Utility Class Files

- **Main Location (Classes without Package).**
  install_dir/Servlets
- **Corresponding URL (Servlets).**
  http://host/servlet/ServletName
- **More Specific Location (Classes in Packages).**
  install_dir/Servlets/packageName
- **Corresponding URL (Servlets in Packages).**
  http://host/servlet/packageName.ServletName

Servlet and Utility Class Files Bundled in JAR Files

- **Location.**
  install_dir/Servlets
- **Corresponding URLs (Servlets).**
  http://host/servlet/ServletName
  http://host/servlet/packageName.ServletName