falkner.ch1.qxd 8/21/03 4:42 PM Page 1

Chapter 1

Setting Up a Servlet and JSP Environment

Before you start developing with Servlets and JavaServer Pages, you need to understand two very important things: Why is using the technology desirable, and what is needed in order to use the technology? This chapter answers these two questions, and in the process provides an introduction to the entire book. We start with an introduction to traditional Web development. The discussion describes why Servlets and JSP were initially created and why the technologies are currently popular. The end of the discussion segues to the software needed in order to run the book's examples.

It is preferred that you follow the instructions in this chapter to ensure your coding environment most closely matches the one all of the code examples of this book have been tested against. If you are using an already established Servlet/JSP environment, make sure it has support for JavaServer Pages 2.0, Servlets 2.4, and the Java 2 Standard Edition 1.4. Examples in this book require these technologies and some features covered are not backwards-compatible.

A Quick History of Web Development

The Servlet and JSP environment extends past the need for basic Java support. Any computer running JSP or Servlets needs to also have a *container*. A container is a piece of software responsible for loading, executing, and unloading the Servlets and JSP. The reasons for this are largely related to the history of server-side Web development. A quick overview of one of the earliest and most prominent server-side dynamic content solutions, CGI, and the differences between it and Servlets is very helpful in understanding why a JSP/Servlet container is required. The exact life cycle events that are managed by a container are discussed in Chapter 2.

CGI

The Common Gateway Interface, or CGI, is commonly referred to as one of the first practical technologies for creating dynamic server-side content. With CGI a server passes a client's request to an external program. This program executes, creates some content, and sends a response to the client. When first developed, this functionality was a vast improvement over static content and greatly expanded the functionality available to a Web developer. Needless to say CGI quickly grew in popularity and became a standard method for creating dynamic Web pages. However, CGI is not perfect.

CGI was originally designed to be a standard method for a Web server to communicate with external applications. An interesting point to note is that the functionality available for generating dynamic Web pages was really a side effect of this design goal. This largely explains why CGI has maybe the worst life cycle possible. As designed, each request to a CGI resource creates a new process on the server and passes information to the process via standard input and environment variables. Figure 1-1 provides a diagram of this single-phase CGI life cycle.

While it does work, the CGI life cycle is very taxing on server resources and greatly limits the number of concurrent CGI users a server can support. In case you are unfamiliar with operating systems and processes, a good analogy to use would be starting up and shutting down a Web server each time a user makes a request. As you probably know, this is almost never the case for a real Web server. It takes a noticeable amount of time to start and stop the entire process. A better solution is to start the server process once, handle all requests, and then

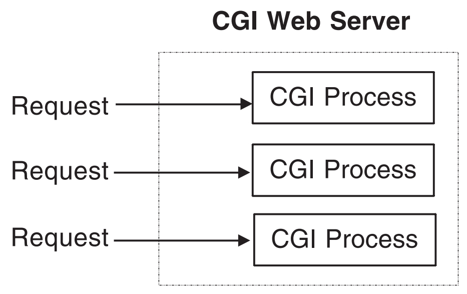


Figure 1-1 CGI Life Cycle

shut it down when there is no longer a need for a Web server. Starting and stopping a Web server is like the single-phase life cycle of CGI, and it was a very noticeable problem. CGI did not scale well and would often bring a Web server to a halt.

Even with poor performance by today's standards, CGI was a revolutionary step in the evolution of server-side programming. Developers had a cross platform method of creating dynamic content using most any of their favorite scripting and programming languages. This popularity sparked second-generation CGI implementations that attempted to counter the performance problems of original CGI, namely FastCGI. While the single phase life cycle still existed, CGI implementations improved efficiency by pooling resources for requests. This eliminated the need to create and destroy processes for every request and made CGI a much more practical solution. Figure 1-2 shows the improved implementation of CGI. Instead of one request per a process, a pool of processes is kept that continuously handle requests. If one process finishes handling a request, it is kept and used to manage the next incoming request rather than start a new process for each request.

This same pooling design can still be found in many of today's CGI implementations. Using pooling techniques, CGI is a viable solution for creating dynamic content with a Web server, but it is still not without problems. Most notable is the difficulty in sharing resources such as a common logging utility or server-side object between different requests. Solving these problems involves using creative fixes that work with the specific CGI and are custom-made for individual projects. For serious Web applications, a better solution, preferably one that addresses the problems of CGI, was required.

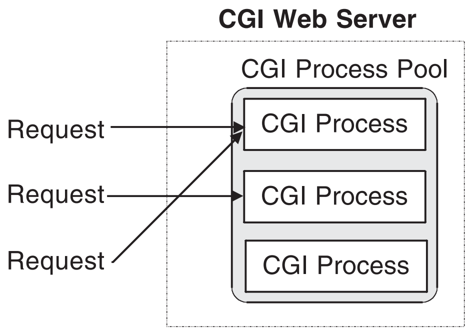


Figure 1-2 Pooled CGI Resources

Java Servlets

In the Java world, Servlets were designed to solve the problems of CGI and create robust server-side environments for Web developers. Similar to CGI, Servlets allow a request to be processed by a program and let the same program produce a dynamic response. Servlets additionally defined an efficient life cycle that included the possibility of using a single process for managing all requests. This eliminated the multi-process overhead of CGI and allowed for the main process to share resources between multiple Servlets and multiple requests. Figure 1-3 gives a diagram of a Web server with Servlet support.

The Servlet diagram is similar to that of second-generation CGI, but notice all the Servlets run from one main process, or a parent program. This is one of the keys to Servlet efficiency, and, as we will see later, the same efficiency is found with JSP. With an efficient design and Java's cross-platform support, Servlets solved the common complaints of CGI and quickly became a popular solution to dynamic server-side functionality. Servlets are still popular and are now also used as the foundation for other technologies such as JSP. Currently, Servlets and JSP combined make up the official "Web Tier" for the Java 2 Enterprise Edition, J2EE[1].

Containers

Servlet performance can be attributed directly to a Servlet *container*. A Servlet container, also called "container" or "JSP container", is the piece of software

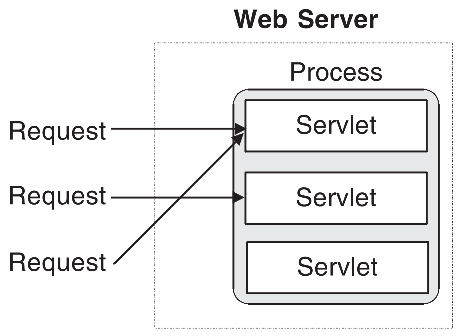


Figure 1-3 Servlet Web Server Diagram

1. This book's title, *The J2EE Technology Web Tier*, comes directly from the marketing jargon Sun uses. Logically, Web Tier means the code that interacts with the World Wide Web.

that manages the Servlet life cycle. Container software is responsible for interacting with a Web server to process a request and passing it to a Servlet for a response. The official definition of a container is described fully by both the JSP and Servlet specifications. Unlike most proprietary technologies, the JSP and Servlet specifications only define a standard for the functionality a container must implement. There is not one but many different implementations of Servlet and JSP containers from different vendors with different prices, performance, and features. This leaves a Servlet and JSP developer with many options for development software.

With containers in mind, the previous diagram of Servlets is better drawn using a container to represent the single process that creates, manages, and destroys threads running Servlets on a Web server. Note that this may or may not be a separate physical process. Figure 1-4 shows a Web server using a Servlet container.

Only Servlets are depicted in Figure 1-3, but in the next two chapters the Servlet and JSP life cycles are covered in detail, and it will be clear that a container manages JSP in exactly the same way as Servlets. For now it is safe to assume that Servlets and JSP are the same technology. What Figure 1-4 does not show is that in some cases a container also acts as the Web server rather than a module to an existing server. In these cases the Web server and container in Figure 1-3 are essentially the same thing.

Given you now know a little more about containers, it should be clear that a container must be installed to run Servlets and JSP. Examples in this book require a Servlet 2.4 and JSP 2.0-compatible container. If you do not have one, do not worry. A walk-through is provided later in this chapter, explaining how to obtain

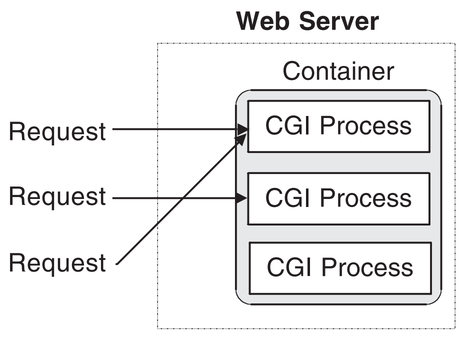


Figure 1-4 Container Process

the reference implementation JSP 2.0 and Servlet 2.4 container. If you have a previously installed container, make sure it supports the correct version of Servlets and JSP; older containers do not support some of the features of JSP 2.0 and Servlet 2.4 specifications. In this book specifically, all examples were created and tested using the reference implementation container, Tomcat. Version 5 of Tomcat is the reference implementation for both Servlets 2.4 and JSP 2.0. If you need to install a compatible container, take the time to now download and install Tomcat 5.

Getting Java Support

Servlets and JavaServer Pages, also meaning all containers, rely on the Java programming language. In the case of Servlets, the code is nothing more than a Java class that implements the appropriate interface(s). Compiling a Servlet is identical to compiling a Java class. JSP is slightly more abstract. JSP, is compiled and used exactly like a Servlet, but the process is almost always done automatically, without a JSP developer noticing. When authoring a JSP, a developer uses a combination of markup and code to make a page that usually resembles an HTML or XML document. This page is compiled into a class file by the Servlet container and automatically loaded.

The official Java platform is designed, owned, and managed by Sun Microsystems. Unlike most other programming languages, Java is not designed for you to compile to platform-specific instructions. Instead Java is compiled down to byte code that gets interpreted by the computer running your Java program. This means a Java program developed and compiled on one computer will run on any other computer with Java support; Servlets and JSP can be compiled and run on most any operating system. This includes most all the Windows, Linux, and Macintosh operating systems. This book assumes you are installing a JSP environment on one of these systems and instructions are provided for installation on each.

Downloading the Java 2 Standard Edition 1.4

Java 2 Standard Edition 1.4 (J2SE 1.4) support is required for code examples in this book. Sun Microsystems provides a free reference implementation of Java 1.4 online. Unless you are using Macintosh OS X, go to `http://java.sun.com/j2se/1.4/` and download the latest Java distribution for your computer. Macintosh OS X has proprietary support for Java 2. If you are running OS X, no additional downloads are needed.

Java only needs to be installed once on your computer and only for the specific operating system you are using. It is not required that you read all of the sections covering installation on all of the featured platforms. They exist only to give readers a guide to their specific operating system. Complete coverage of the Java 2 Standard Edition 1.4 is outside the scope of this book; however, later use of the J2SDK by this book will not require comprehensive knowledge of the tools provided by Sun. If you are looking for a detailed guide for this information, refer to *Thinking In Java, 3rd Edition*[2], by Bruce Eckel.

Installing J2SE 1.4 on Microsoft Windows

Microsoft Windows comes in many varieties, with early versions having a big distinction between a desktop computer and a computer designed to be a server. Practically speaking, a desktop computer running Windows 95 or Windows 98 is a not a good choice for installing a production environment for Servlets and JSP, but a Windows 95 or Windows 98 computer will suffice in order to try this book's examples. The Java distribution for Microsoft Windows will run on all versions of the operating system excluding Windows 3.x. In this book, the focus is on installing Java on a Windows NT, 2000, or XP computer. However, we realize that many readers have a desktop PC running Windows 95 or Windows 98 at home. An attempt will be made to help you if this is the case.

After downloading the J2SE 1.4, it must be installed on your system. The download should be an executable file that can be run by double-clicking it. Double-click on this file and follow the installation wizard through all the steps. It does not matter where you install the J2SE 1.4, but it is worth noting the location, as it is needed in a later part of this chapter.

Installation of the Java 2 Standard Development Kit 1.4 is now complete. Skip ahead to the section entitled, "Tomcat".

Installing J2SE 1.4 on Linux Distributions

Linux comes in far more varieties than Windows and operates on many more hardware architectures. A walk-through installation guide for all Linux distributions is not attempted, but this guide should work on the vast majority of distributions. Specifically this section gives a walk-through of installing the J2SE 1.4 on Red Hat Linux 7.3. It will greatly resemble installation on any Linux distribution for x86 processors, as an RPM is not used. If you downloaded the RPM or

2. A free copy of the book can be found online, http://www.mindview.net/Books/TIJ/.

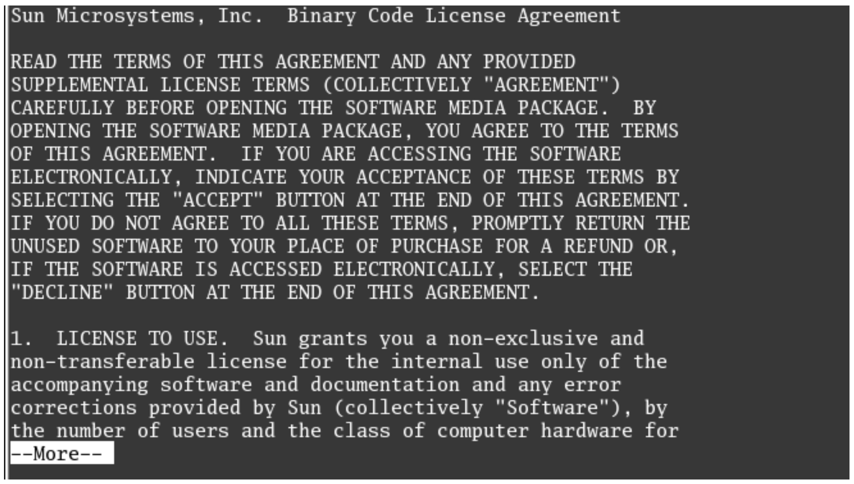
equivalent for your distribution, feel free to install it, make note of the installation directory, and skip to the next section of this chapter.

At the introduction to this section, you should have downloaded the J2SE 1.4 Linux binary installation file. The file should be named something similar to `j2sdk-1_4_0_01-linux-i586.bin` with appropriate version numbers for the latest release. Any post-1.4 release should be adequate; this guide uses version 1.4.0_01. From the command prompt make sure the file has executable permissions and execute the program. These are what the commands would be to make the file executable and then execute it; assume the download is in the `/root/download` directory and you have proper permissions.

```
chmod +x /root/download/j2sdk-1_4_0_01-linux-i586.bin
/root/download/j2sdk-1_4_0_01-linux-i586.bin
```

When the file executes, Sun's licensing terms are displayed and you have the option of agreeing to continue the installation. Figure 1-5 shows an example display of the licensing terms.

If you agree to the terms, files will automatically be unpacked to a `j2sdk1.4.0` directory created by the installation program. You can move this directory to any location you prefer, but remember the location where the J2SDK 1.4 is because it will be needed later when the environment variables are set. Installation of the standard Java development kit is now complete.



```
Sun Microsystems, Inc.  Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED
SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT")
CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE.  BY
OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS
OF THIS AGREEMENT.  IF YOU ARE ACCESSING THE SOFTWARE
ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY
SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT.
IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE
UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR,
IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE
"DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1.  LICENSE TO USE.  Sun grants you a non-exclusive and
non-transferable license for the internal use only of the
accompanying software and documentation and any error
corrections provided by Sun (collectively "Software"), by
the number of users and the class of computer hardware for
--More--
```

Figure 1-5 Sun's J2SDK 1.4 Licensing Terms

For Non-x86 Users: Compiling Java for Yourself

It is possible you are either an advanced user who dislikes non-optimized code or that you do not have an x86 microprocessor. Both of these cases are largely outside the scope of this book, but a quick pointer should get you well on your way if that is what you seek. The "official" Java Linux code is maintained at Blackdown Linux, `http://www.blackdown.org`. Visit Blackdown Linux if you need the source code to either optimize your Java distribution or compile it for a different architecture.

Tomcat

Tomcat, the reference implementation of Servlets and JSP, is part of the open source Apache Jakarta project and is freely available for download from `http://jakarta.apache.org/tomcat`[3]. Tomcat can be run on Windows, Linux, Macintosh, Solaris, and most any of the other Unix distributions. You can use Tomcat both commercially and non-commercially as specified by the Apache Software Foundation License.

The next few sections provide a walk-through for installing Tomcat on Windows, Linux distributions, and Macintosh OS X. If needed, follow the appropriate section to get Tomcat correctly set up for later examples.

Installing Jakarta Tomcat 5 on Windows

The Tomcat installation for Windows greatly resembles installing any other piece of Windows software. The process involves downloading the Tomcat installation executable file and running it to launch the Windows installation wizard. After a few simple clicks, the wizard will have set up Tomcat to work with your Windows distribution.

If you have not done so already, download the Windows executable file for the Tomcat installation. You can find it at the following URL, `http://jakarta.apache.org/builds/jakarta-tomcat/release/`. Simply follow the links to the latest Tomcat 5 Windows executable file—as of this writing, the 5.0.2 beta release of Tomcat is at `http://jakarta.apache.org/builds/jakarta-tomcat/release/v5.0.2-alpha/bin/jakarta-tomcat-5.0.2.exe`. After downloading the executable, double-click on it to start the installation wizard. A screen similar to

3. This part of the book is perhaps the most important. If for any reason you are having trouble installing Tomcat, please consult the book support site, http://www.jspbook.com, for a complete, up-to-date Tomcat 5 installation guide.

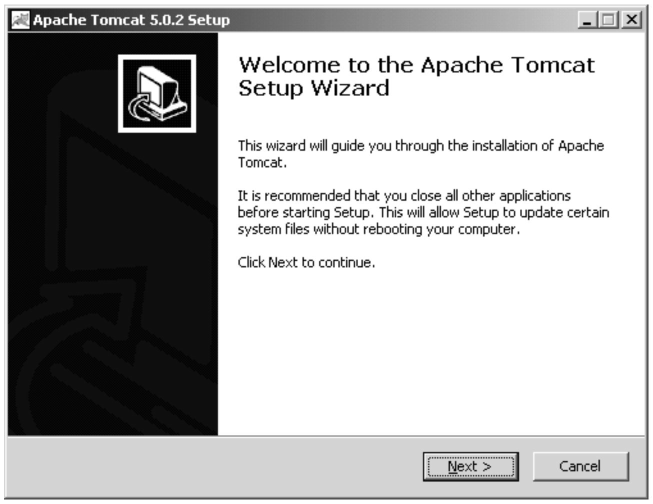
Figure 1-6 should appear notifying you that the Tomcat Installer located a J2SDK installation.

Click OK and continue on to the screen displaying the Apache Software Foundation license. Read through the terms and make sure you agree with Tomcat's licensing. If you accept the terms, the installation continues and you can choose what components you wish to install. The default ones should be fine, as shown in Figure 1-7.

Proceed with the installation wizard to the next screen, and you can choose a directory to install Tomcat. Figure 1-8 shows the corresponding installation wizard screen. Choose any directory you prefer.

After choosing an installation directory, the wizard will ask for some initial Tomcat configuration information, including a port number and administrative access information. The default options are fine, but later on in the chapter we will be changing the port Tomcat uses to be port 80 instead of 8080. You may change the port to 80 via the installation wizard now or later on in the chapter. Figure 1-9 displays the corresponding installation wizard screen.

The final piece of configuration information Tomcat requires is the location of your Java Virtual Machine, which should be wherever you installed it earlier in the chapter. By default Tomcat attempts to locate the most recent JVM for you, as shown in Figure 1-10.


Apache Tomcat 5.0.2 Setup

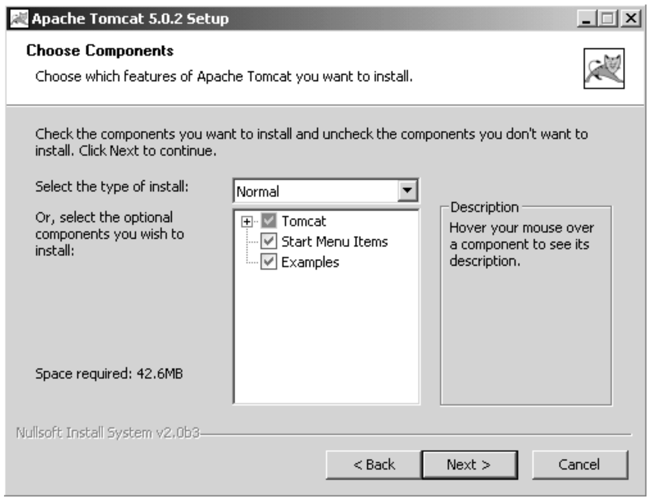
Welcome to the Apache Tomcat Setup Wizard

This wizard will guide you through the installation of Apache Tomcat.

It is recommended that you close all other applications before starting Setup. This will allow Setup to update certain system files without rebooting your computer.

Click Next to continue.

Figure 1-6 Windows Installation Wizard


Apache Tomcat 5.0.2 Setup

Choose Components

Choose which features of Apache Tomcat you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Normal

Or, select the optional components you wish to install:

- [✓] Tomcat
- [✓] Start Menu Items
- [✓] Examples

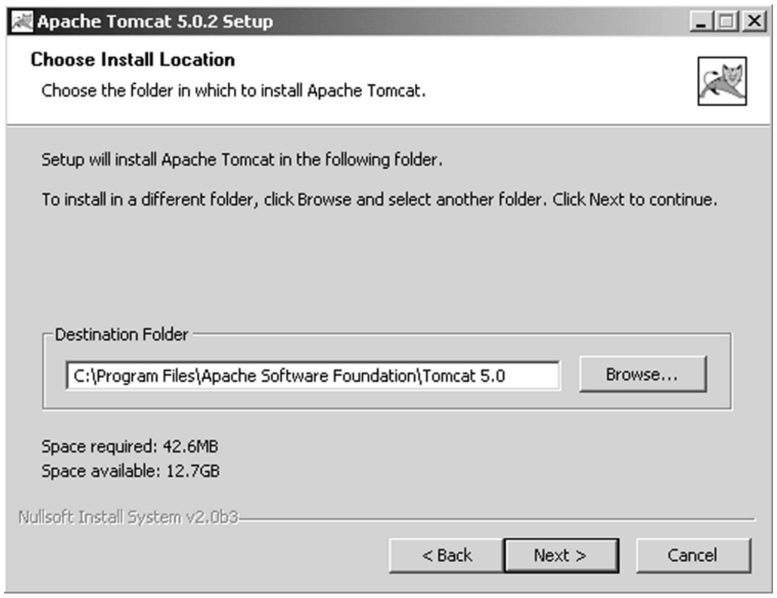
Description

Hover your mouse over a component to see its description.

Space required: 42.6MB

Nullsoft Install System v2.0b3

Figure 1-7 Installation Wizard Choosing Tomcat's Components


Apache Tomcat 5.0.2 Setup

Choose Install Location

Choose the folder in which to install Apache Tomcat.

Setup will install Apache Tomcat in the following folder.

To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

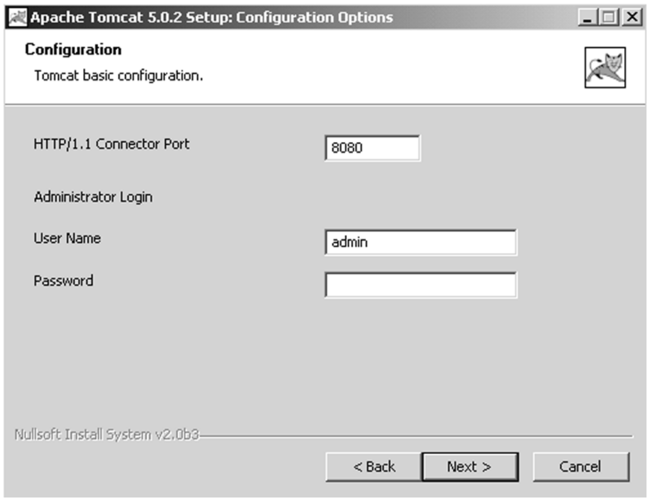
C:\Program Files\Apache Software Foundation\Tomcat 5.0

Space required: 42.6MB
Space available: 12.7GB

Nullsoft Install System v2.0b3

Figure 1-8 Installation Wizard for Installing Tomcat

falkner.ch1.qxd 8/21/03 4:42 PM Page 12


Apache Tomcat 5.0.2 Setup: Configuration Options

Configuration
Tomcat basic configuration.

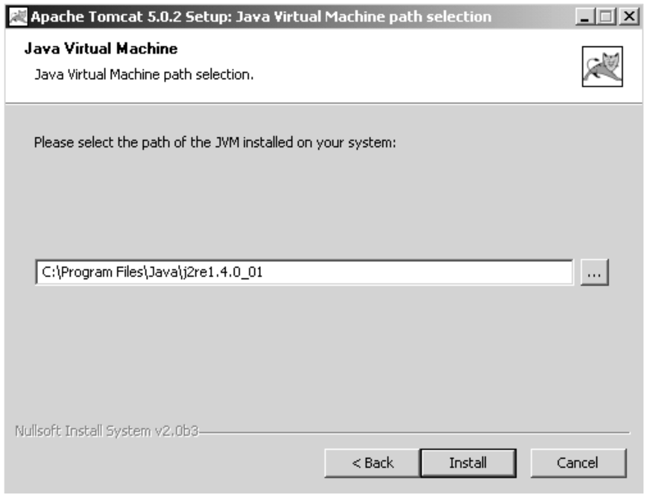
HTTP/1.1 Connector Port: 8080

Administrator Login

User Name: admin

Password:

Figure 1-9 Installation Wizard for Configuring Tomcat


Apache Tomcat 5.0.2 Setup: Java Virtual Machine path selection

Java Virtual Machine
Java Virtual Machine path selection.

Please select the path of the JVM installed on your system:

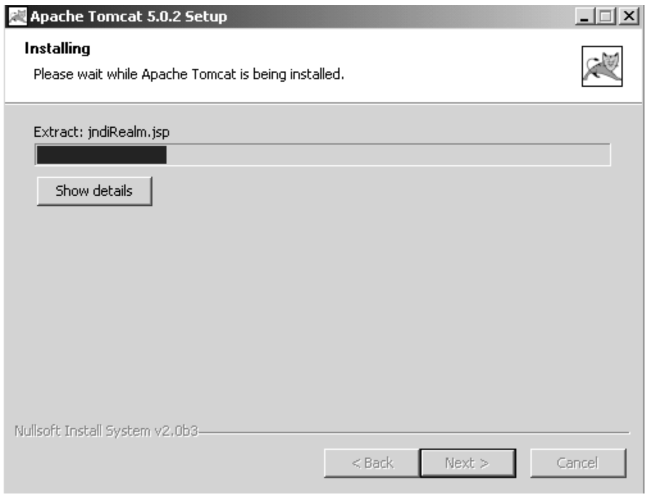
C:\Program Files\Java\j2re1.4.0_01

Figure 1-10 Installation Wizard Showing the Location of the Current JVM

The default JVM found by the installation wizard should be fine, but you can change the location to be any JVM you desire (note a Java 1.4-compatible JVM is required to execute this book's examples). Finally, the installation wizard will automatically install all the Tomcat files. Figure 1-11 shows the corresponding installation wizard screen.

To complete the installation and run Tomcat, start the server by double-clicking on the Tomcat icon or by executing the startup script `startup.bat` found in the `TOMCAT_HOME/bin` directory. Check the service is running by browsing to your local computer on port 8080 (or 80 if you changed Tomcat's port via the installation wizard), `http://127.0.0.1:8080`. A Tomcat welcome page should be displayed[4] as shown in Figure 1-12.

Installation of Tomcat is complete, and you are now ready to run book examples. Tomcat comes bundled with documentation and examples of both Servlets and JSP. Feel free to examine the bundled examples, but this book will not cover them further. Before continuing, there are two important scripts to be aware of.



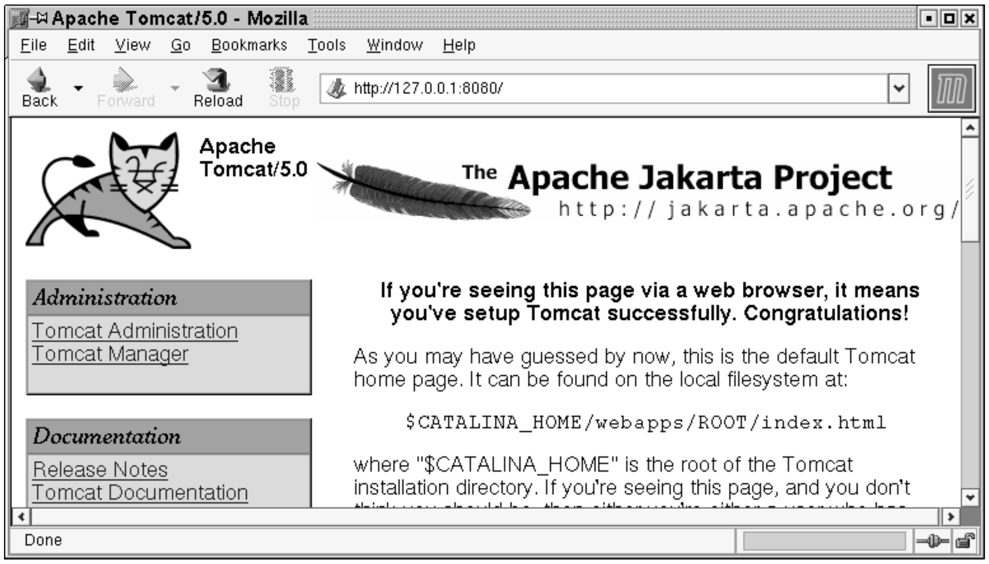
Installing

Please wait while Apache Tomcat is being installed.

Extract: jndiRealm.jsp

Figure 1-11 Installation Wizard for Installing Tomcat Files

4. Port 8080 should not be previously in use, but if it is, you will not be able to see the Tomcat welcome page. If you are sure you have correctly installed Tomcat and think a port bind is causing problems, consult the later section in this chapter which deals with changing the port Tomcat runs on.



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

`$CATALINA_HOME/webapps/ROOT/index.html`

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't

Figure 1-12 Tomcat Welcome Page after Completing the Installation

- **startup.bat:** The `startup.bat` script in the `TOMCAT_HOME/bin` directory is used to start the Tomcat container and Web server. Servlet and JSP code examples rely on Tomcat, and you must have Tomcat turned on before testing them.
- **shutdown.bat:** The `shutdown.bat` script in the `TOMCAT_HOME/bin` directory is used to terminate the Tomcat container and Web server.

It is important to be aware of these two scripts because from time to time an example will require that Tomcat be restarted. When this is asked, it is assumed Tomcat is currently running and implies that the `shutdown.bat` script and then `startup.bat` be executed to reload the server. Alternatively you can also use an automated utility like Ant, which is discussed later in this chapter, to manage reloading, compiling, and other repetitive aspects of developing Servlets and JSP.

Windows 9X/ME "out of environment space"

Just because you are using Windows 9x, or Millennium, does not mean you will have a problem, but be aware that a common problem does exist. If when executing `startup.bat` or `shutdown.bat`, an "out of environment space" error occurs, do the following. Right-click on the `startup.bat` and `shutdown.bat` files. Click on Properties then on the Memory tab. For the Initial environment field, enter 4096. After you click apply, Windows will create shortcuts in the directory which you can use to start and stop the container. Use these shortcuts to start and stop Tomcat.

What is happening is that the batch file is not allocating enough memory to execute its commands. The fix is to simply add more memory. The Tomcat developers are aware of this problem and are trying to incorporate fixes into `startup.bat` and `shutdown.bat`.

Installing Jakarta Tomcat 5 on Linux and Macintosh OS X

The Tomcat project is continuously expanding the different types of installation packages in which Tomcat releases are available. If you notice a Tomcat installation package that matches special installation software for your specific Linux distribution, feel free to download and use the appropriate file. For this walk-through, installation will be of the compiled Java code packaged in a tarball[5]. Download the Tomcat tarball `jakarta-tomcat-5.0.tar.gz"` from `http://jakarta.apache.org/builds/jakarta-tomcat/release/`. Installation is as easy as decompressing the Tomcat binaries and setting two environment variables.

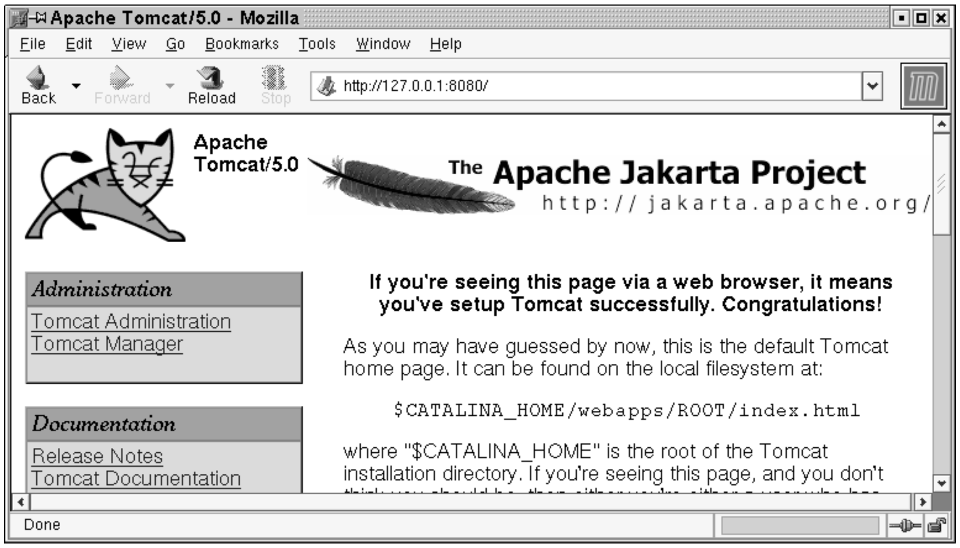
Decompress the tarball by using the tar and gzip compression utilities from the command prompt. Here is what the command would look like if you had placed the download in the `/usr` directory.

```
gunzip -c /usr/jakarta-tomcat-5.0.tar.gz | tar xvf -
```

The Tomcat binaries will be available in the `/usr/jakarta-tomcat-5.0` directory or a similarly named directory that matches the version of Tomcat you downloaded. For the rest of this walk-through, it is assumed this is the directory you are also using. If not, change the directory name in examples to match the location where you uncompressed Tomcat.

Before starting Tomcat, two environment variables need to be set: `JAVA_HOME` and `TOMCAT_HOME`. The `JAVA_HOME` variable corresponds to the directory of the J2SDK 1.4 installed earlier in this chapter. The `TOMCAT_HOME` variable corresponds to the directory into which you just uncompressed Tomcat. Set and export the

5. A tarball is a commonly used Unix term for a file compressed using gzip and tar compression. The de facto standard for distributing Unix and Linux programs is by means of a tarball because the compression is often significantly better than only ZIP.


Apache Tomcat/5.0 - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://127.0.0.1:8080/
Done

Apache
Tomcat/5.0

The Apache Jakarta Project
http://jakarta.apache.org/

Administration
Tomcat Administration
Tomcat Manager

Documentation
Release Notes
Tomcat Documentation

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

`$CATALINA_HOME/webapps/ROOT/index.html`

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't

Figure 1-13 Tomcat Welcome Page Appears after Starting Tomcat

two environment variables. The commands would be similar to the following if the J2SDK is installed in the `/usr/java/jdk1.4` directory and Tomcat 5 is installed in the `/usr/jakarta-tomcat-5.0` directory; replace each variable accordingly to match your specific case:

```
JAVA_HOME=/usr/java/jdk1.4
TOMCAT_HOME=/usr/jakarta-tomcat-5.0
export JAVA_HOME TOMCAT_HOME
```

Tomcat is now ready to run. Start the server by executing the startup script `startup.sh` found in the `/usr/jakarta-tomcat-5.0/bin` directory. Check the service is running by browsing to your local computer on port 8080, `http://127.0.0.1:8080`. A Tomcat welcome page should be displayed[6] as shown in Figure 1-13.

Installation of Tomcat is complete, and you are now ready to run book examples. Tomcat comes bundled with documentation and examples of both Servlets and JSP. Feel free to examine the bundled examples, but this book will

6. Port 8080 should not be previously in use, but if it is, you will not be able to see the Tomcat welcome page. If you are sure you have correctly installed Tomcat and think a port bind is causing problems, consult the later section in this chapter, which deals with changing the port Tomcat runs on.

not cover them further. Before continuing on, there are two important scripts to be aware of.

- **startup.sh:** The `startup.sh` script in the `TOMCAT_HOME/bin` directory is used to start the Tomcat container and Web server. Servlet and JSP code examples rely on Tomcat, and you must have Tomcat turned on before testing them.
- **shutdown.sh:** The `shutdown.sh` script in the `TOMCAT_HOME/bin` directory is used to terminate the Tomcat container and Web server.

It is important to be aware of these two scripts because from time to time an example will require that Tomcat be restarted. When this is asked, it is assumed Tomcat is currently running and implies that the `shutdown.sh` script and then `startup.sh` be executed to reload the server. Also be aware that the environment variables previously set will not persist between different terminal sessions and need to either be included in a startup script or set for each session.

Alternatively, you can also automate the redundant process of compiling book examples and reloading Tomcat by using a build utility. Use of the Jakarta Ant build utility is strongly encouraged with this book and is covered later in this chapter.

Configuring Tomcat

Tomcat is a robust JSP and Servlet container. It not only provides complete support for the JSP and Servlet specifications, but it can also act as a standalone Web server. By default, this is exactly what Tomcat does, and the default configuration is all we need for this book's examples. Full instructions on configuring Tomcat are outside the scope of this book. There are many different Servlet and JSP containers available, and it is not very practical to devote a large part of this book to Tomcat-specific information. There is only one important aspect of configuring Tomcat that needs to be discussed. If you would like to learn more about using Tomcat to its full potential, some good resources are listed at the end of this section.

Switching Tomcat to Port 80, the Default HTTP Port

For all practical purposes it does not matter what port you run Tomcat on. Ports 8080, 80, 1234, and 9999 all work the same. However, any port besides 80, which is being used for HTTP, comes with some slight annoyances that it would be nice to avoid. Specifically, the default port for HTTP is port 80. Recall the previously used URL for Tomcat's welcome page, `http://127.0.0.1:8080/index.html`. The `localhost` address, `127.0.0.1`, is universal, but the `8080` is required because Tomcat's HTTP connector is not listening on port `80`; the default configuration for Tomcat is for it to listen on port `8080`. The `8080` is slightly annoying, especially when needing to add the `8080` to all local absolute links in your Web application.

In order to simplify the book examples and to avoid confusion, we will now configure Tomcat to use port 80, which is the default HTTP port. This is done by editing Tomcat's configuration file, `/conf/server.xml`. Open this file with your favorite text editor and do a search for '8080'; you should find the following entry:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector className="org.apache.coyote.tomcat5.CoyoteConnector"
           port="8080" minProcessors="5" maxProcessors="75"
           enableLookups="true" redirectPort="8443"
           acceptCount="10" debug="0"
           connectionTimeout="20000"
          useURIValidationHack="false" />
```

The entry is responsible for configuring Tomcat's HTTP connector and the port it listens on. Change the entry to use port 80 by replacing 8080 with 80:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector className="org.apache.coyote.tomcat5.CoyoteConnector"
           port="80" minProcessors="5" maxProcessors="75"
           enableLookups="true" redirectPort="8443"
           acceptCount="10" debug="0"
           connectionTimeout="20000"
          useURIValidationHack="false" />
```

Tomcat will then listen for HTTP requests on port 80, which is assumed when no port is specified. Shut down and restart Tomcat so that it uses the new port. Both `http://127.0.0.1` and `http://127.0.01:80` will now display the Tomcat welcome page. Likewise, all subsequent requests, which do not specify a port, will be directed to Tomcat. Make sure to restart Tomcat before testing out the changes[7].

7. If another service, such as Apache or IIS, is running on port 80, Tomcat will not be able to use the port. Either choose a different port, configure Tomcat to work with the service, terminate the conflicting service, or change the services default port. Additional help with configuring Tomcat to use an existing Web server is outside the scope of this book.

Tomcat User's Guide

The Tomcat's User's Guide is the official documentation for Tomcat. This should be the first place you look for help when configuring and using Tomcat. The Tomcat User's Guide can be found online at the Jakarta Tomcat Web site, `http://jakarta.apache.org/tomcat`.

Tomcat User Mailing List

The Tomcat user mailing list is the best place to find community support for configuring and using Tomcat. This mailing list consists of most all the Tomcat developers and many of the current users of Tomcat. By posting a question on this mailing list, you can try to solicit information from the current Tomcat experts. You can subscribe to this mailing list by following the instructions on the Jakarta Mailing List page, `http://jakarta.apache.org/site/mail.html`.

Be warned the Tomcat user mailing list generates a lot of traffic. A hundred or more emails a day is not uncommon; however, it is only an issue if you do not wish to deal with that volume of email. There are many benevolent people, which is obvious on the Tomcat user mailing list because of the unbelievable amount of questions they answer. An alternative to using the Tomcat user mailing list is to consult the online archives, which can be found using the same link.

Book Support Site

For the most current information and updates of this chapter's walk-through guides, use the book support site, `http://www.jspbook.com`. Along with the latest versions of the walk-through guides, this site also contains a Frequently Asked Question (FAQ) section for questions relating to this book. Many of these questions answer problems relating to Tomcat configuration issues for various software environments. Additionally, the book support site is intended to provide a place that can deal with any unexpected issues occurring after publication.

Web Applications

A "Web Application", not the commonly used "Web application" meaning any Web-savvy program, is the proper term for a complete Servlet and/or JSP project. Anytime you develop a JSP or Servlet, it is part of a larger Web Application. Each Web Application has its own unique configuration files and resources for use.

These files and resources are defined by the JSP specification and Servlet specification and managed by your container. In summary they consist of:

- **Configuration:** Each Web Application includes a configuration file, `web.xml`. This configuration file customizes the resources of a Web Application in an efficient and structured fashion. Web Applications keep `web.xml` private from outside visitors and also provide a place for privately storing other custom configuration information.
- **Static Files and JSP:** A Web Application's primary purpose is to serve content on the World Wide Web. This content includes dynamic resources such as Servlets and JSP, but it also includes static resources such as HTML pages. A Web Application automatically manages JSP and static resources deployed in it.
- **Class Files and Packages:** A Web Application also loads and manages custom Java code. For application-specific class files such as Servlets, a special location is designated from which a container can load and manage compiled code. Web Applications define a similar location for including Java Archive, JAR, files that contain packaged resources.

By the end of the chapter, most of these configuration files and resources will have been introduced and discussed, but an in-depth analysis of them cannot be attempted without understanding more about JSP and Servlets. As the book progresses, all of the preceding will be fully defined and explored. But before discussing any part of a Web Application, one must be created.

Making a Web Application from scratch requires two things. First, a directory to hold all of the files for the Web Application is needed. The directory can be located anywhere on your local computer. For simplicity, create a directory named `jspbook` under the `/webapps` directory of your Tomcat installation. The `webapps` folder is Tomcat's default location for storing Web Applications. To make this a Web Application, you also need to add a `web.xml` configuration file. To do this, go to the jspbook directory and create a subdirectory called `WEB-INF`. Inside the `/WEB-INF` directory create a file called `web.xml`. Save Listing 1-1 as the contents of `web.xml`.

Listing 1-1 web.xml Skeleton File

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4">
</web-app>
```

Do not worry about the details of this for now; this will all be explained later.

Next, the container needs to be informed that a new Web Application exists[8]. With Tomcat this can be done in one of two ways: by adding an entry in the `server.xml` file located in the `/conf` directory of a Tomcat installation, or by placing an XML file containing the configuration into Tomcat's `/webapps` directory. Placing the file in the `/webapps` directory is the preferred option; it makes upgrading Tomcat versions easier as changes to `server.xml` do not need to be transferred. To configure a Web application, Tomcat requires a `Context` tag. Create a file called `jspbook.xml` in `TOMCAT_HOME/webapps` directory with Listing 1-2 as its content.

Listing 1-2 Simple Web Application Deployment File for Tomcat

```
<Context path="/jspbook" docBase="jspbook" debug="0"/>
```

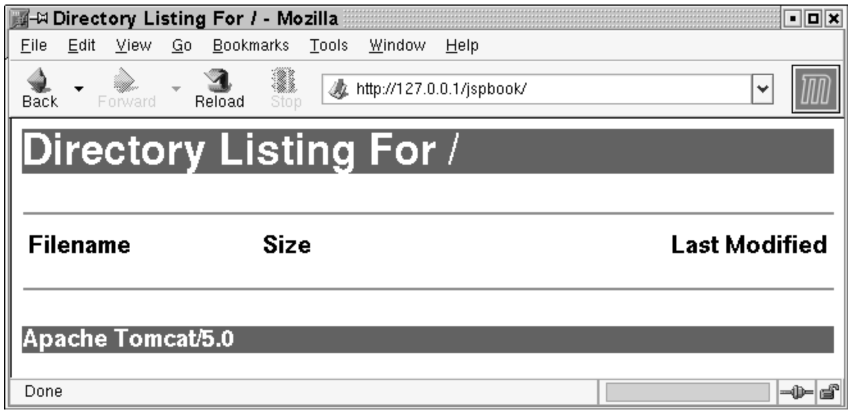
Restart Tomcat to reflect the changes made in the `server.xml` configuration file and to load the new Web Application. You can now browse to the newly created jspbook Web Application by using the following URL, `http://127.0.0.1/jspbook/`. Because nothing has been placed in this Web Application, an empty directory is displayed as shown in Figure 1-14.

The jspbook Web Application is used for examples throughout the rest of the book. Until you know a little more about JSP and Servlets, it will be difficult to create dynamic content using this Web Application; however, nothing stops you from using static resources such as an HTML file. Try creating a simple HTML page that welcomes a visitor. Copy and name the following HTML code Listing 1-3 as `index.html` and place it in the `/webapps/jspbook` directory of the Tomcat installation.

Listing 1-3 index.html

```
<html>
  <head>
    <title>Welcome!</title>
  </head>
  <body>
  Welcome to the example Web Application for
  <i>Servlets and JSP, the J2EE Web Tier</i>.
  </body>
</html>
```

8. The following step is not strictly necessary as Tomcat automatically treats directories within its `/webapps` directory as a webapp. This support does not include subdirectories of webapp.


Directory Listing For / - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://127.0.0.1/jspbook/
Done

Directory Listing For /

Filename	Size	Last Modified

Apache Tomcat/5.0

Figure 1-14 Empty Directory Listing for jspbook Web Application

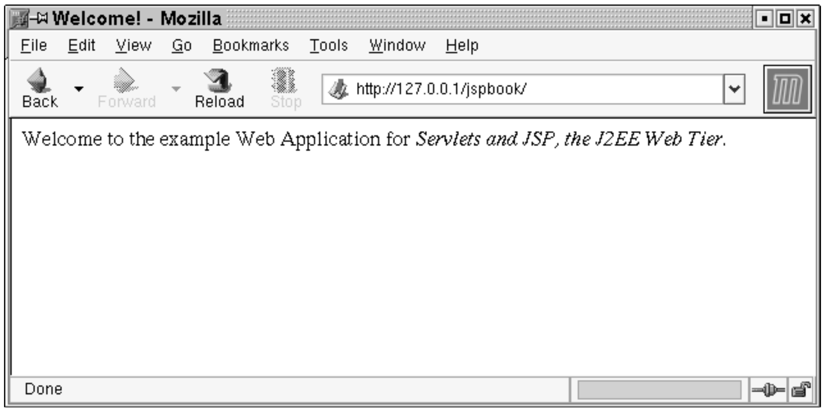
Refresh the Web browser you used to previously view the empty Web Application directory. It now displays a little welcome message, as shown in Figure 1-15, instead of the empty directory listing, as shown in Figure 1-14.

Any static content placed in the Web Application will be made available for users to see. The `index.html` page happens to be the default page Tomcat displays for a Web Application, and that is why it appeared by default when the empty directory was refreshed. Notice that the URL automatically changed to `http://127.0.0.1/jspbook/index.html`[9]. This behavior is not standard among all JSP containers, but the behavior can always be configured on a per Web Application basis. Web Application configuration involves using the Web Application Deployment Descriptor file `web.xml`.

/WEB-INF and web.xml

The Servlet specification defines a configuration file called a deployment descriptor that contains meta-data for a Web Application and that is used by the container when loading a Web Application. This file is always located in the `/WEB-INF` directory of a Web Application and must be named `web.xml`. When a container loads a Web Application, it checks this file. As noted, `web.xml` contains application meta-data, such as default pages to show, Servlets to load, and

9. Microsoft's Internet Explorer is notorious for improperly handling this. If the URL does not change, do not worry. As long as the correct page is shown, everything is fine.


Welcome! - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back
Forward
Reload
Stop
http://127.0.0.1/jspbook/
Done

Welcome to the example Web Application for *Servlets and JSP, the J2EE Web Tier*.

Figure 1-15 index.html Rendered by Web Browser

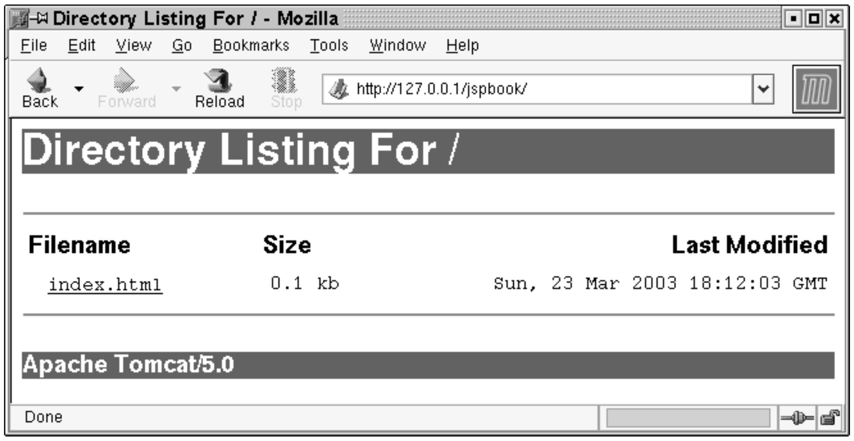
security restrictions to place on files. The Servlet specification also defines that the entire `/WEB-INF` directory of any Web Application must be kept hidden from users of the application. In other words, a user cannot browse to `http://127.0.0.1/jspbook/WEB-INF`. Try it—any configuration information for your Web Application is unattainable by a client unless you specifically create something to expose it.

You already have the skeleton of a Web Application configuration file. Further configuration information can be placed in between the starting and ending `web-app` tags. Try it out by adding Listing 1-4 to define the default page that your Web Application displays.

Listing 1-4 web.xml welcome File Configuration

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4">
  <welcome-file-list>
    <welcome-file>welcome.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Save the changes and restart your container so the Web Application reflects them. Use a Web browser to visit `http://127.0.0.1/jspbook` again. This time the `index.html` page is not shown by default; a directory listing is shown instead (see Figure 1-16), which includes `index.html`.


Directory Listing For / - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://127.0.0.1/jspbook/
Done

Directory Listing For /

Filename	Size	Last Modified
index.html	0.1 kb	Sun, 23 Mar 2003 18:12:03 GMT

Apache Tomcat/5.0

Figure 1-16 Directory Listing

The preceding outcome is the result of the new elements added to `web.xml`. The `welcome-file-list` element defines a list of welcome files in descending preference. The name of each welcome file is defined as the body of the `welcome-file` element. In the meta-data just added, one specific file was listed as default page: `welcome.html`. If file exists, the container would show it instead of the empty directory. If you would like to experiment, try changing the name of `index.html` to `weclome.html` and it will appear as the default page for the Web Application.

The rest of `web.xml` configuration is left for discussion in later sections of the book. Both JSP and Servlets have many configuration elements that can be added to `web.xml`. This book covers most all of them in examples, but a complete reference can always be found in the Servlet 2.4 schema.

Java Classes and Source Files

Throughout the book examples will be taking advantage of existing Java APIs as well as creating new Java classes. One of the most common mistakes new Servlet and JSP developers make is where they place Java source code and compiled Java classes. The intuitive approach is to place these files in the same directory as that in which static content is placed for the Web Application. While it may seem logical at first, this approach has two problems. The first is that any individual browsing to your Web site will, by default, be able to access any files not in the `/WEB-INF` directory. This means your code is freely available for people to download and use or possibly abuse. The second problem is that the container will ignore the code when it is loading the Web Application. The second issue is more problematic because the Web Application's Servlets and JSP will not be able to import and use the custom code.

The correct place to put custom code is in the `/WEB-INF/classes` directory of a Web Application. Code that is placed in this directory is loaded by the container when needed and can be imported for use by Servlets and JSP in the same Web Application. Should the code be part of a package, create a directory structure that matches the package names. An example of this would be if a class were created for the `com.foo.example` package. The appropriate place to put the compiled Java class is the `/WEB-INF/classes/com/foo/example` directory of a Web Application. For now, create the `/WEB-INF/classes` directory so that it is ready for later code examples.

Java Archive (JAR) Files

A Java Archive file is a convenient method for consolidating a set, usually a package, of Java class files into one compressed, portable file. The Java specifications define the exact rules of a JAR file, but an easy way to think of them is nothing more than a set of Java classes ZIP-compressed together. JAR files are a very popular method of distributing Java code and are commonly used by Web Applications. As defined in the Servlet specification, any JAR file placed in the `/WEB-INF/lib` directory of a Web Application is made available for use by code in the same Web Application. Later examples in this book will give some concrete examples of using this functionality. The complete specifications for creating and using a basic JAR are not in the scope of this book. You can find a complete guide to creating JAR files in the J2SE documentation included with your J2SE download. The task is not hard but is outside the scope of this text.

Web Application Resource (WAR) Files

Web Application Resource files are functionally similar to JAR files, but are used to consolidate an entire Web Application into one compressed and portable file. WAR files are officially defined by the Servlet specification. The compression used for a WAR file is also ZIP; using existing JAR utilities in the Standard Java Development Kit can easily make WAR files. WAR files are a great solution for packaging a Web Application and they are commonly used to bundle documentation with examples for JSP and Servlet-related software.

Most of the examples found in later chapters deal with expanding and enhancing existing Web Applications. WAR files are only helpful to either start a Web Application or to package a finished one. For this reason WAR files will not be appearing often in later chapters. WAR files are mentioned here for reasons of completeness and because the online book support site packages all of this book's examples into a WAR. If you would like to skip retyping code examples from the book, feel free to download the example WAR at `http://www.jspbook.com/jspbook.war`.

If you would ever like to make a WAR file, simply ZIP compress an entire working Web Application. For example, at any time you can create a WAR of your progress through this book by ZIP-compressing all of the contents in the `jspbook` directory. Using the ZIP utility on Linux, the following commands would be used:

```
zip -r jspbook.war jspbook
```

The `zip` utility would then compress the entire application into one file named `jspbook.war`. In order to deploy this file with Tomcat, simply place `jspbook.war` in the `/webapps` directory and restart Tomcat.

Note that WAR files are only as portable as you make them. Several times in upcoming chapters notes will be made about this point. For example, if you access a file using the file's absolute path name, there is no guarantee a similar file will be found on all servers with containers using your WAR. However, if a custom file is packaged in a WAR, the file can always be accessed using the Servlet API.

Ant

The Jakarta Ant project is popular with Java developers and for good reason. Ant is a cross-platform build utility that is easy to use and configure. Ant uses an XML configuration file typically called `build.xml`. If you are unfamiliar with build utilities, do not worry. This book does not rely on such knowledge. Ant is treated purely as a complimentary tool. If you have used Ant before, you should appreciate this, or if you have not used Ant, hopefully you will be glad we introduced the tool to you. At any time you can opt to completely ignore Ant and simply compile `.java` files by hand and reload Tomcat using the appropriate scripts. At no point will this book tell you to use Ant to do this. Instead, an example will dictate that Tomcat, the Web Application, or both need to be reloaded, or that code needs to be compiled. In these cases it is implied you can use Ant if you so desire.

What Does Ant Do?

Ant performs "tasks", any number of them, and in any order, possibly dependent on previously accomplished tasks. There are many default tasks that Ant defines; Ant also allows for Java developers to code custom tasks. In use, Ant takes a simple build file, which defines tasks to be done, and does them. For this book we will make a build file that turns off Tomcat, compiles code, then turns Tomcat back on. The tasks are all simple but after trying several of the code examples in this book, you might be glad Ant consolidates the redundant steps.

Installing Ant

Install Ant by downloading the latest distribution from `http://jakarta.apache.org/ant`. This book uses Ant 1.5, but any subsequent release should work fine. Binary distributions of Ant are available in either ZIP or tarball form. Download the format you are most familiar with and save the compressed file in a convenient location. Unpack the distribution and installation of Ant complete. Because the files are Java binaries, there is no need to run an installer or compile a platform-specific distribution.

Using Ant

Ant is designed to be simple to use. The `/bin` directory of the Ant distribution contains an executable file named `ant` that runs Ant. You can execute Ant at any time by running `$ANT_HOME/bin/ant`; replace `$ANT_HOME` with the appropriate directory of the Ant distribution. By default, Ant looks to the current directory and tries to use a file named `build.xml` as its build file. Listing 1-5, which is included in the example WAR, is a build file for use with this book.

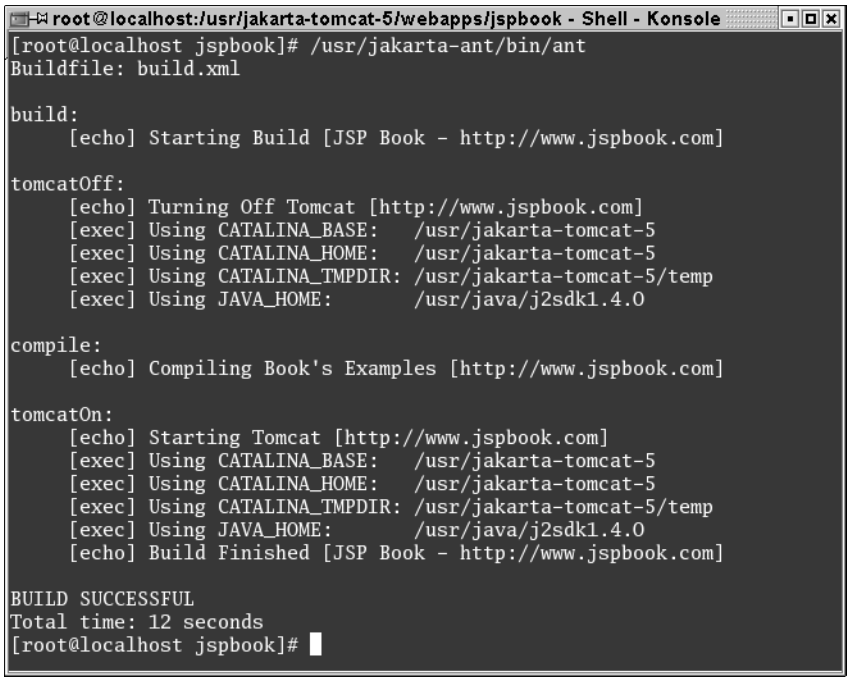
Listing 1-5 Ant Build File for Use with This Book

```
<?xml version="1.0" ?>
<project name="jspbook" default="build" basedir=".">
  <target name="build">
    <echo>Starting Build [JSP Book - http://www.jspbook.com]</echo>
    <!-- Turn Tomcat Off -->
    <antcall target="tomcatOff"/>
    <!-- Compile Everything -->
    <antcall target="compile"/>
    <!-- Turn Tomcat On -->
    <antcall target="tomcatOn"/>
    <echo>Build Finished [JSP Book - http://www.jspbook.com]</echo>
  </target>

  <target name="tomcatOff">
    <echo>Turning Off Tomcat [http://www.jspbook.com]</echo>
    <exec executable="bash" os="Windows">
      <arg value="../../bin/shutdown.bat"/>
    </exec>
    <exec executable="bash" os="Linux">
      <arg value="../../bin/shutdown.sh"/>
    </exec>
  </target>

  <target name="tomcatOn">
    <echo>Starting Tomcat [http://www.jspbook.com]</echo
    <exec executable="bash" os="Windows">
      <arg value="../../bin/startup.bat"/>
    </exec>
    <exec executable="bash" os="Linux">
      <arg value="../../bin/startup.sh"/>
    </exec>
  </target>

  <target name="compile">
    <echo>Compiling Book's Examples [http://www.jspbook.com]</echo>
    <javac
      srcdir="WEB-INF/classes"
      extdirs="WEB-INF/lib:../../common/lib"
      classpath="../../common/lib/servlet.jar"
      deprecation="yes"
      verbose="no">
      <include name="com/jspbook/**"/>
    </javac>
  </target>
</project>
```

Save the preceding code as `build.xml` in the base directory of the jspbook Web Application. You do not need to care about the entries in the build file; however, they should be intuitive. If you are interested, you can find further documentation for the Ant tasks online at `http://jakarta.apache.org/ant/manual/index.html`.

Use of the build file is simple. From shell, or command prompt in Windows, switch to the directory of the jspbook Web Application. For instance, if Tomcat was installed in `/usr/jakarta-tomcat/`[10], then the desired directory would be `/usr/jakarta-tomcat/webapps/jspbook`. From there, execute Ant. Assuming

10. These are UNIX file systems. In Windows, the starting '/' is replaced by the drive, such as `C:\`, and subsequent '/' are replaced with '\'. For instance, `C:\jakarta-tomcat\webapps\jspbook`.


root@localhost:/usr/jakarta-tomcat-5/webapps/jspbook - Shell - Konsole

```
[root@localhost jspbook]# /usr/jakarta-ant/bin/ant
Buildfile: build.xml

build:
     [echo] Starting Build [JSP Book - http://www.jspbook.com]

tomcatOff:
     [echo] Turning Off Tomcat [http://www.jspbook.com]
     [exec] Using CATALINA_BASE:   /usr/jakarta-tomcat-5
     [exec] Using CATALINA_HOME:   /usr/jakarta-tomcat-5
     [exec] Using CATALINA_TMPDIR: /usr/jakarta-tomcat-5/temp
     [exec] Using JAVA_HOME:       /usr/java/j2sdk1.4.0

compile:
     [echo] Compiling Book's Examples [http://www.jspbook.com]

tomcatOn:
     [echo] Starting Tomcat [http://www.jspbook.com]
     [exec] Using CATALINA_BASE:   /usr/jakarta-tomcat-5
     [exec] Using CATALINA_HOME:   /usr/jakarta-tomcat-5
     [exec] Using CATALINA_TMPDIR: /usr/jakarta-tomcat-5/temp
     [exec] Using JAVA_HOME:       /usr/java/j2sdk1.4.0
     [echo] Build Finished [JSP Book - http://www.jspbook.com]

BUILD SUCCESSFUL
Total time: 12 seconds
[root@localhost jspbook]# 
```

Figure 1-17 Ant Execution of the Book's Build File

Ant is in your classpath, you can simply type `ant`, or a full path can be used such as `/usr/jakarta-ant-1.5/bin/ant`. Ant automatically looks for `build.xml` and will execute the specified tasks. Execution should look similar to Figure 1-17.

Note the build file turns off Tomcat, compiles all the book's code, and turns Tomcat back on. Should an example request any of these tasks, simply run Ant and the job is finished.

Summary

The first and most important step to developing with Servlets and JSP is setting up the appropriate environment. This chapter focused on providing a detailed explanation of the requirements for a Servlet and JSP environment and provided a walk-through for installing one on a majority of the popular operating systems. By doing this, you will be ready for the code examples of later chapters and can better learn through hands-on experience developing Servlets and JSP.

A Web Application is the term given to a complete collection of static content, JSP, Servlets, custom code, and configuration information for all of the previously mentioned. This chapter also established a Web Application for use with examples to come. As you go through the book, the jspbook Web Application will be expanded and enhanced to demonstrate the many aspects of developing with Servlets and JSP.

Chapter 2 discusses Servlets and JSP at the lowest possible level by introducing and explaining the basics of the Servlet API.