Tomcat for beginning Web developers

Skill Level: Introductory

Sing Li (westmakaha@yahoo.com)
Author
Wrox Press

20 Oct 2005

The Apache Tomcat application server is no longer the exclusive domain of advanced Web system developers. In this tutorial, Sing Li shows beginning Web developers how to leverage their current Java™ development skills to program server-side JSPs, servlets, and Web services using Tomcat.

Section 1. Before you start

About this tutorial

This tutorial gives Java Web developers an introduction to programming JavaServer Pages (JSPs), servlets, and Web services using Tomcat, an open source application server from the Apache Foundation. The tutorial guides you through the following tasks:

- Downloading and installing your own Tomcat server.
- Coding and deploying a JSP on Tomcat.
- Coding and deploying a servlet on Tomcat.
- Coding and deploying a Web service using Tomcat and Apache Axis.

The tutorial provides an overview of JSP, servlets, and Web services but is not intended to cover those technologies in depth.

Prerequisites

You need to be familiar with the Java programming language, object-oriented design

principles, and basic TCP/IP networking concepts to get the maximum benefit from this tutorial. An understanding of the networking APIs in the JDK is ideal, although not mandatory.

To run the examples in this tutorial, you need:

- A working installation of JDK 1.5.0 or later.
- A running installation of Tomcat 5.5 or later, available at http://jakarta.apache.org/tomcat/. This tutorial includes detailed download, installation, and setup instructions for Tomcat.

To run the Web services example, you also need to install:

- Apache Ant 1.5.2 or later, available at http://ant.apache.org/
- Apache Axis 1.2.1 or later, available at http://ws.apache.org/axis/. This tutorial includes detailed installation instructions for Axis.

The recommended system configuration for running the tutorial:

- A system supporting JDK 1.5.0 with at least 512MB of memory. The instructions in the tutorial are based on a system running Microsoft Windows.
- At least 50MB of disk space to install the software and examples.

Section 2. What is Tomcat?

Tomcat is an open source server from the Apache Software Foundation. It is a Web application server, which means that it comes ready to support programming using JavaServer Pages (JSPs) and servlets.

Since early 2000, Tomcat has served as the reference implementation for the latest Java Servlet and JSP specifications. Tomcat 5.5, the latest Tomcat version as of this writing, supports the latest Java Servlet 2.4 and JavaServer Pages 2.0 standards (see Resources). Tomcat also includes a limited Web server that can serve static Web pages when executed in stand-alone mode (by default).

Because of a variety of open source libraries and extensions, Tomcat supports:

- Web services using the Apache Axis servlet
- Development frameworks, such as Apache Struts
- Templating engines, such as Apache Jakarta Velocity

- Object-relational mapping technology, such as Hibernate

This tutorial shows you how to use Tomcat to learn JSP, servlet, and Web services programming. Use of Struts, Velocity, and Hibernate with Tomcat is beyond this tutorial's scope.

In the past, because a high level of expertise was required to configure and administer Tomcat, the primary Tomcat users were advanced server-side application developers. Now -- thanks to the maturing of Tomcat's GUI installer, the ability to install the server as a system service, and stabilization of the server's features -- even beginning Web developers can take advantage of this versatile server.

Section 3. Tomcat installation and setup

Downloading Tomcat

To download the latest version of Tomcat, go to the Apache Tomcat home page (see Prerequisites), shown in Figure 1, and click the **Tomcat 5.x** link under the **Download** heading (the area outlined in red in Figure 1):

Figure 1. Apache Tomcat project home page

Page 3 of 11 Document options Print this page PDF - A4 475 KB PDF - Letter 479 KB Get Adobe® Reader® Sample code Rate this tutorial Help us improve this content Tomcat installation and setup Downloading Tomcat To download the latest version of Tomcat, go to the Apache Tomcat home page (see Prerequisites), shown in Figure 1, and click the Tomcat 5.x link under the Download heading (the area outlined in red in Figure 1): Figure 1. Apache Tomcat project home page You have a choice among the latest 5.5.x releases. Choose the binary distribution of the latest stable (nonbeta and nonalpha) release. For Windows systems, download the EXE binary for simple installation.

Installing Tomcat

The EXE binary installer does the following:

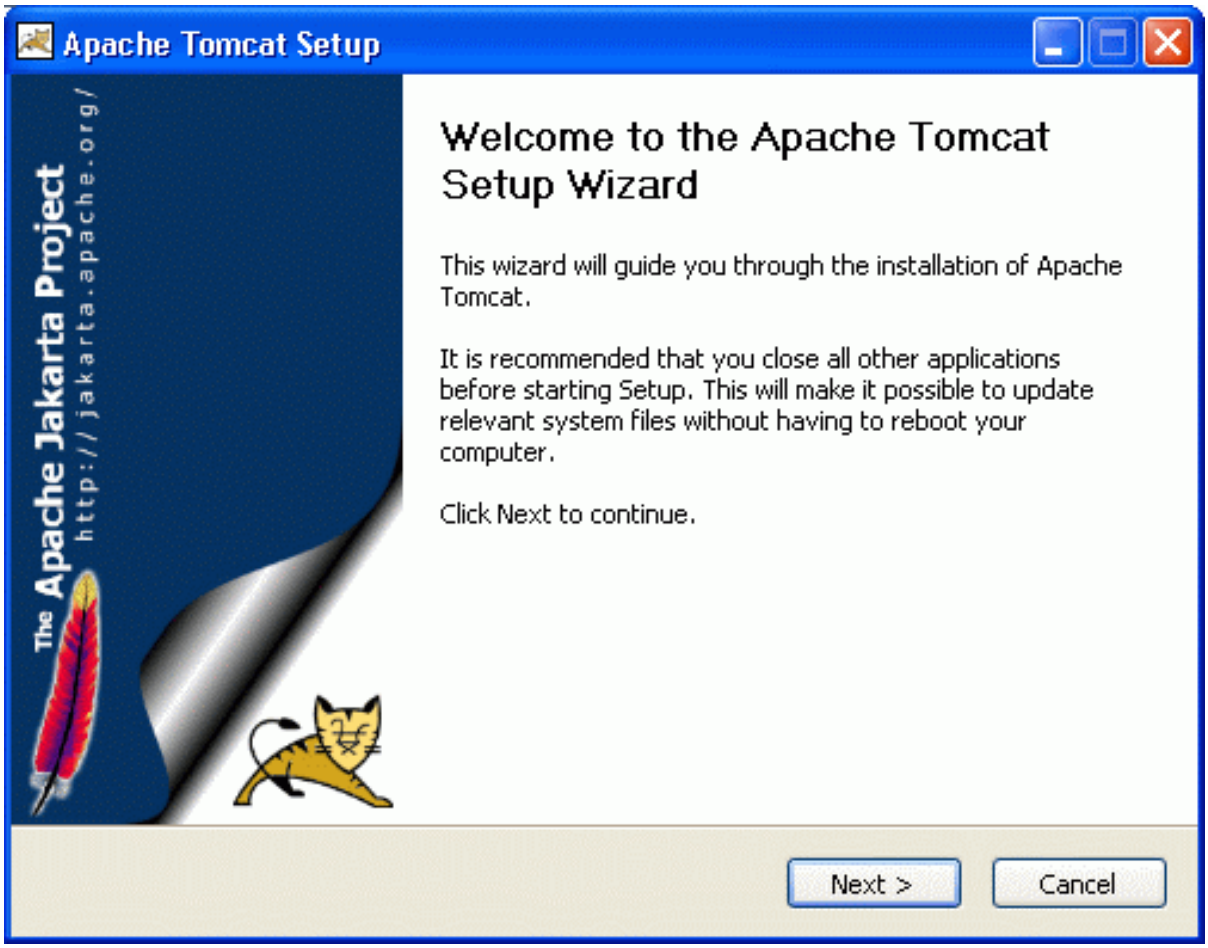
- Unpacks and installs the Tomcat server components.
- Lets you specify the TCP port that the server will use when listening for incoming requests. (A TCP port is a networking endpoint, represented by

a number, that a client application can specify when connecting to the server.)

- Configures the server to run as a system service.

Start the installation EXE. You'll see the initial splash screen, shown in Figure 2:

Figure 2. Tomcat setup wizard splash screen



The installation EXE runs a wizard-based installer with step-by-step instructions. You must have administrator privileges on the machine, because Tomcat is installed as a system service. If you are on your own PC as the default user and have installed other software successfully, you probably already have administrator privileges.

Table 1 describes the items that each screen of the setup wizard prompts for, along with the responses you should make.

Table 1. Tomcat setup wizard prompts

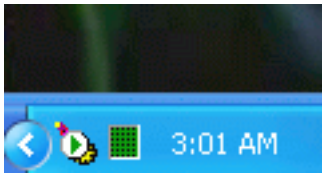
Setup wizard screen	Description
License agreement	This is the Apache License 2.0, one of the more liberal open source software licenses in existence. Read the license terms carefully. If you agree to the terms, click the **I Agree** button to proceed.

Choose components	Select the components of Tomcat to install. By default, the mandatory components are checked. If you have enough disk space, consider installing the examples. They are great for learning Web application programming.
Choose install location	Select the directory on your computer where the Tomcat server will install. If this is your first installation, the default that the wizard selects should be fine. This screen also shows how much disk space the Tomcat installation will take up and the amount of free space you have on the disk.
Configuration	This screen lets you performs basic Tomcat server configuration. You can select the TCP port that the server listens on, as well as an administrator username and password. It is recommended that you leave the TCP port at 8080. Leave the administrator username as *admin* and enter your own administrator password. Do not forget the password; you'll need it later to deploy the examples in this tutorial.
Java Virtual Machine	This screen lets you select the JVM that Tomcat runs under. Unless you have multiple JDKs installed on your machine, you can use the default. For the latest Tomcat 5.5 release, you should select JVM version 1.5.0 or later.
Completing the Apache Tomcat Setup Wizard	This is the final step of the installation. Select the **Run Apache Tomcat** checkbox. This starts the system service immediately after installation.

Note that on some versions of Windows with a firewall, you might need to give Tomcat explicit permission to listen to the TCP port for requests.

After installation, the Tomcat server will be running, with the Apache service-monitor icon appearing in the lower right-hand corner of your Windows Taskbar (the long bar on the bottom of the screen), as shown in Figure 3:

Figure 3. Service monitor showing Tomcat running



In Figure 3, the green arrow on the monitor icon indicates that the Tomcat service is running.

Verifying server operations

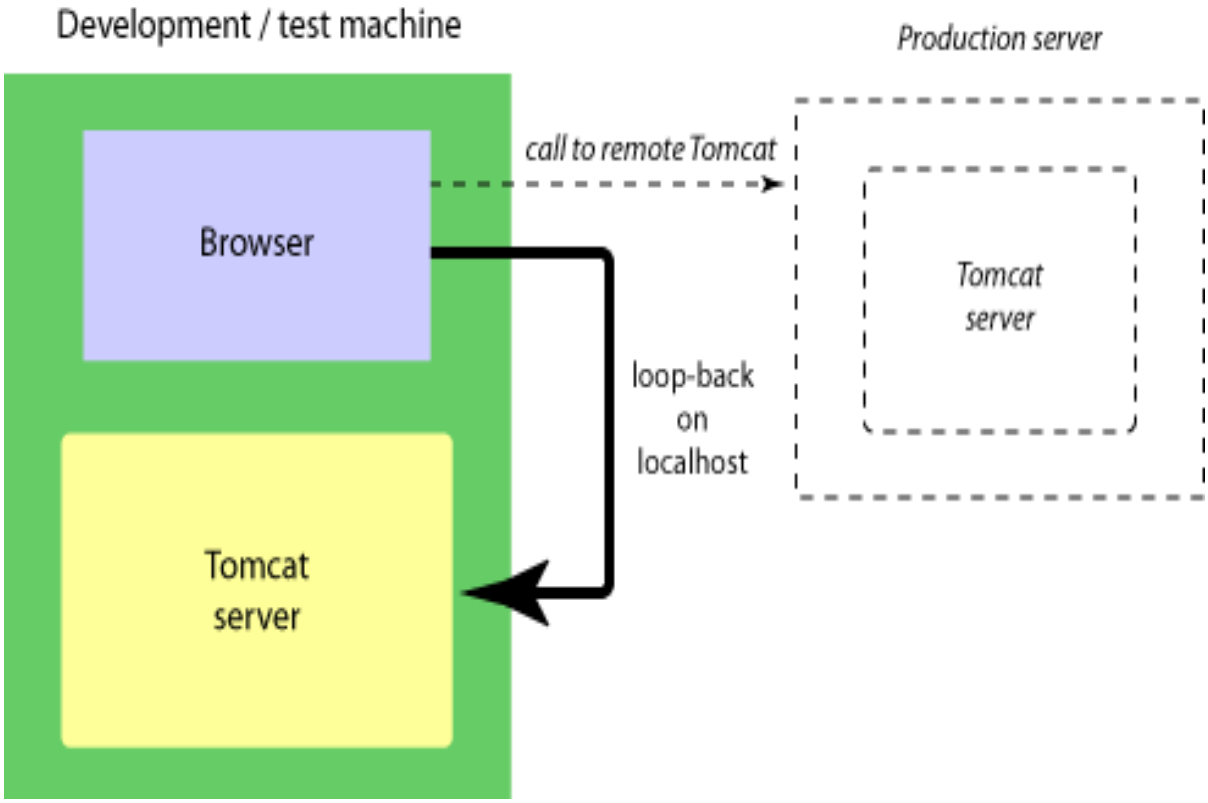
It's simple to access the running Tomcat server and verify that the installation was successful. Start a browser and point it to the address http://localhost:8080/.

The Tomcat server is listening at port 8080. (You configured this during the installation.) Figure 4 shows the welcome screen that Tomcat displays:

Figure 4. Tomcat's welcome screen

By running the Tomcat server on the same machine as your browser, you are simulating a networked environment. Figure 5 shows this loop-back network configuration:

Figure 5. Loop-back configuration for single-machine server-side development



In Figure 5, both the client (browser) and the server (Tomcat) are running on the same machine. The TCP connection between the client and the server is running in a loop-back mode. This is a common practice in Web development, enabling you to perform server-side development using a single machine. In actual production, you can change the host name of the URL from localhost to the IP address of your networked production Tomcat server (shown within dashed lines in Figure 5).

Section 4. Your first JSP application on Tomcat

Brief introduction to JSP programming

JSP is a popular scripting and templating language for creating the presentation layer of server-side Java applications. Typically, JSP is used for pages with a dynamic user interface. It can generate HTML, XML, Cascading Style Sheets (CSS), JavaScript, and virtually any client-side presentation content dynamically. The latest widely implemented JSP version is 2.0, based on Java Specification Request (JSR) 152 (see Resources).

Essential elements of the JSP language are:

- Directives

- Standard actions
- Expression Language (EL)
- Custom tag libraries
- JavaBeans

JSP has built-in capability to access JavaBeans. In production applications, JavaBeans are typically used to carry data values between the application logic (implemented using servlets and other components) and JSP. The JSP code's typical responsibility is to display the value contained in the JavaBean.

One of the more frequently used tag libraries for JSP is the JSP Standard Tag Library (JSTL). JSTL, defined in JSR 52 (see Resources), contains a large library of tags that can be used in conjunction with the EL in JSP. The latest version of JSTL (as of October 2005) is 1.1.

Unlike Java programs, JSP programs need not be precompiled. Tomcat compiles a JSP the first time it is executed and keeps a copy of the compiled binary for subsequent execution. This allows for rapid development and testing cycles.

With early versions of JSP (before 2.0), it was difficult to write general application logic without resorting to embedded Java coding. In fact, JSP versions prior to 2.0 allowed and encouraged the use of mixed Java/JSP coding. This practice generally creates code that is messy and difficult to maintain.

Beginning with JSP 2.0, with its support for EL and JSTL, embedding Java code in JSP programs is no longer necessary. It is recommended that all new JSP developers not intermix embedded Java code with JSP. This approach is often called *scriptless JSP*.

There's a lot more you can learn about JSP programming after you finish this tutorial (see Resources for more information on JSP). The rest of this section shows you how to create and run JSP applications using Tomcat, so you can start trying out your own JSP programming right away.

A simple JSP program

The example JSP program in this tutorial illustrates JSP's dynamic HTML-generation capabilities. The program prints out a message containing the current server-side time and a multiplication table. The date information changes every time you access the page. The multiplication table is generated using a programming algorithm.

You can find the example JSP program, called index.jsp, in the step1 subdirectory of the code distribution (see Download). Listing 1 shows index.jsp:

Listing 1. Sample JSP Program: index.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<jsp:useBean id="timeNow" class="java.util.Date" />
```

```
<html>
<head>
<title>developerWorks Tomcat Tutorial</title>
<link rel=stylesheet type="text/css" href="specials.css">
</head>
<body>
<table width="600">
  <tr>
    <td class="mainHead" colspan="9">
      <h1>Today is
         <fmt:formatDate value="${timeNow}"
         type="date" dateStyle="long" /></h1>
    </td>
  </tr>
  <tr>
    <c:forEach var="i" begin="1" end="9" step="1">
      <th>${i}x</th>
    </c:forEach>
  </tr>
    <c:forEach var="row" begin="1" end="9" step="1">
      <tr>
      <c:forEach var="col" begin="1" end="9" step="1">
        <td><c:out value="${row * col}" /></td>
      </c:forEach>
      </tr>
    </c:forEach>
</table>
</body>
</html>
```

Some JSP techniques to note in this program include:

- The `<%@taglib>` directive, used to include the core and formatting components of JSTL, and associating their tags with the namespace prefix of `c:` and `fmt:` respectively.
- The `<jsp:useBean>` standard action to instantiate an instance of the `java.util.Date` class as a JavaBean, representing the current time.
- Use of EL in the `${timeNow}` expression, representing the JavaBean instance of `java.util.Date`, to print the current month and day.
- Use of the date-formatting library tag in JSTL to format the date value.
- Use of the `<c:forEach>` tag from JSTL to create the loop that prints out the multiplication table.
- The mix of static HTML content with JSP-generated dynamic content.

Preparing to run a JSP application on Tomcat

You need to complete some packaging work before you can run the index.jsp program on Tomcat. In general, you need to follow these steps:

1. Create your JSP application. If you are working with only one page, call it index.jsp, as you've done for the example program.
2. Create a deployment descriptor -- a web.xml file -- and place it in the WEB-INF directory.

3. Copy the JSTL libraries to the WEB-INF/lib directory.

4. Bundle all the code, using the JAR tool from the JDK, into a Web application archive (WAR) file for deployment.

5. Use the Tomcat Web Application Manager to deploy and run the WAR file.

The WAR file is a standard Java Enterprise Edition (Java EE) deployment unit. It is a JAR file in a very specific format, with a .war filename extension. In this WAR file, you must have a deployment descriptor file called web.xml, which contains instructions telling the server how to deploy the content of the WAR.

n the case of the example program, the web.xml file (see Listing 2) is trivial, because the application contains only one JSP page:

Listing 2. The web.xml deployment descriptor

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <description>
   developerWorks Beginning Tomcat Tutorial
    </description>
    <display-name>IBM developerWorks Beginning Tomcat Tutorial
    </display-name>
</web-app>
```

The web.xml file in Listing 2 simply provides a description and display name to Tomcat, which will be used later by the Tomcat Application Manager.

To create the WAR file, run the makewar.bat batch file in the code distribution (see Download). This batch file simply calls the JAR utility to create a JAR file. Listing 3 shows the contents of makewar.bat:

Listing 3. makewar.bat

```
jar cvf step1.war .
```

In you're on a Linux system, you can enter `jar cvf step1.war .` at the console to create the WAR file manually.

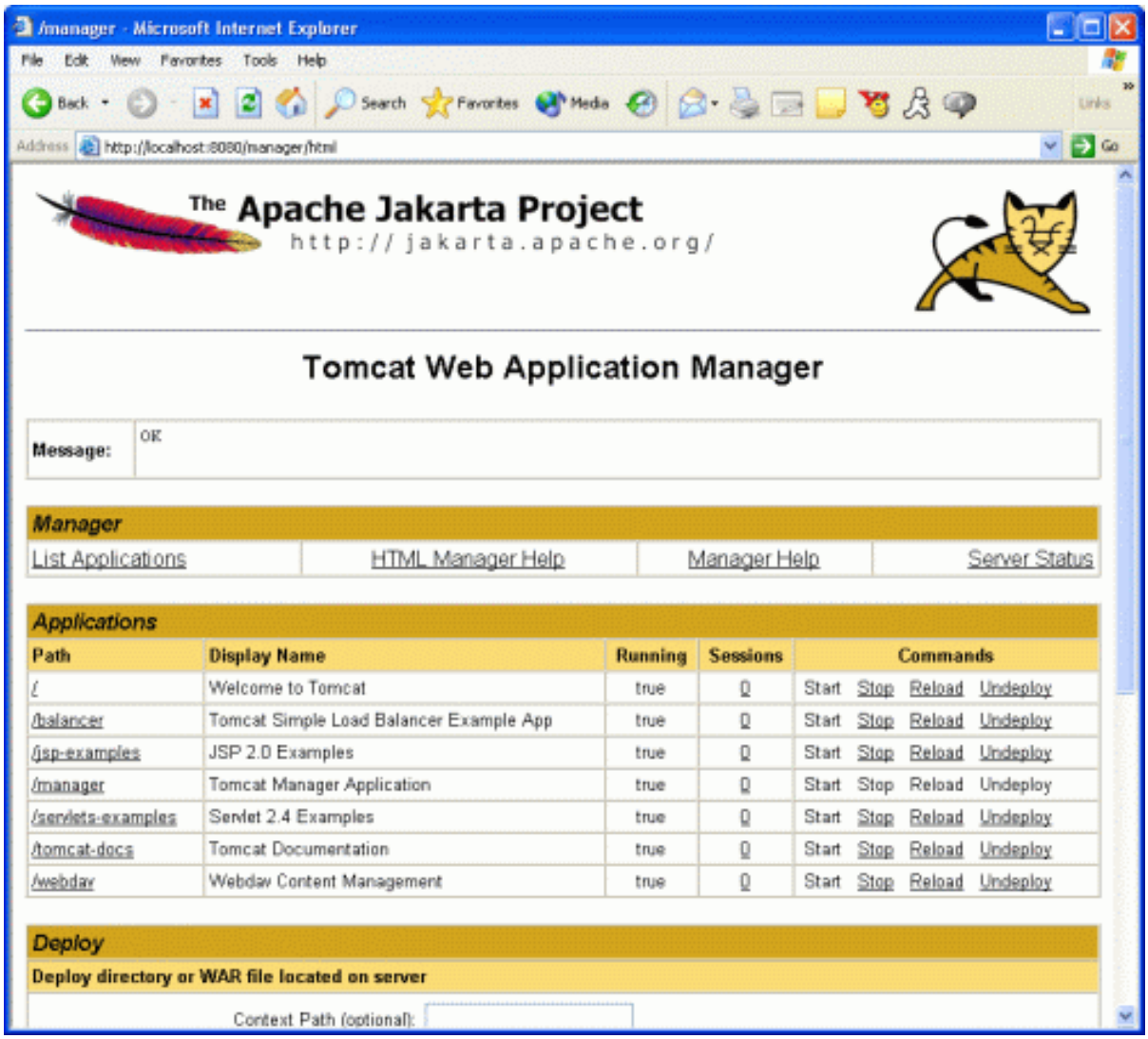
Deploying an application with Tomcat Web Application Manager

To run the application on Tomcat, you need to deploy the WAR file first. Use the Tomcat Web Application Manager (Manager for short) utility for this. You can access Manager by pointing your browser to http://localhost:8080/manager/html.

Tomcat will ask you for a username and password. Enter the administrator user

name and password that you supplied during setup. Once you have logged in to the server, you will see the Manager's display. It shows all the applications that are currently loaded and running on the Tomcat server, as shown in Figure 6:

Figure 6. The Tomcat Web Application Manager utility



To deploy step1.war, scroll down to the bottom of the Manager page. Click the **Browse** button next to the **WAR file to deploy** section. Use the browser to select the step1.war file, then click the Deploy button. This action sends the WAR file to the Tomcat server and starts it.

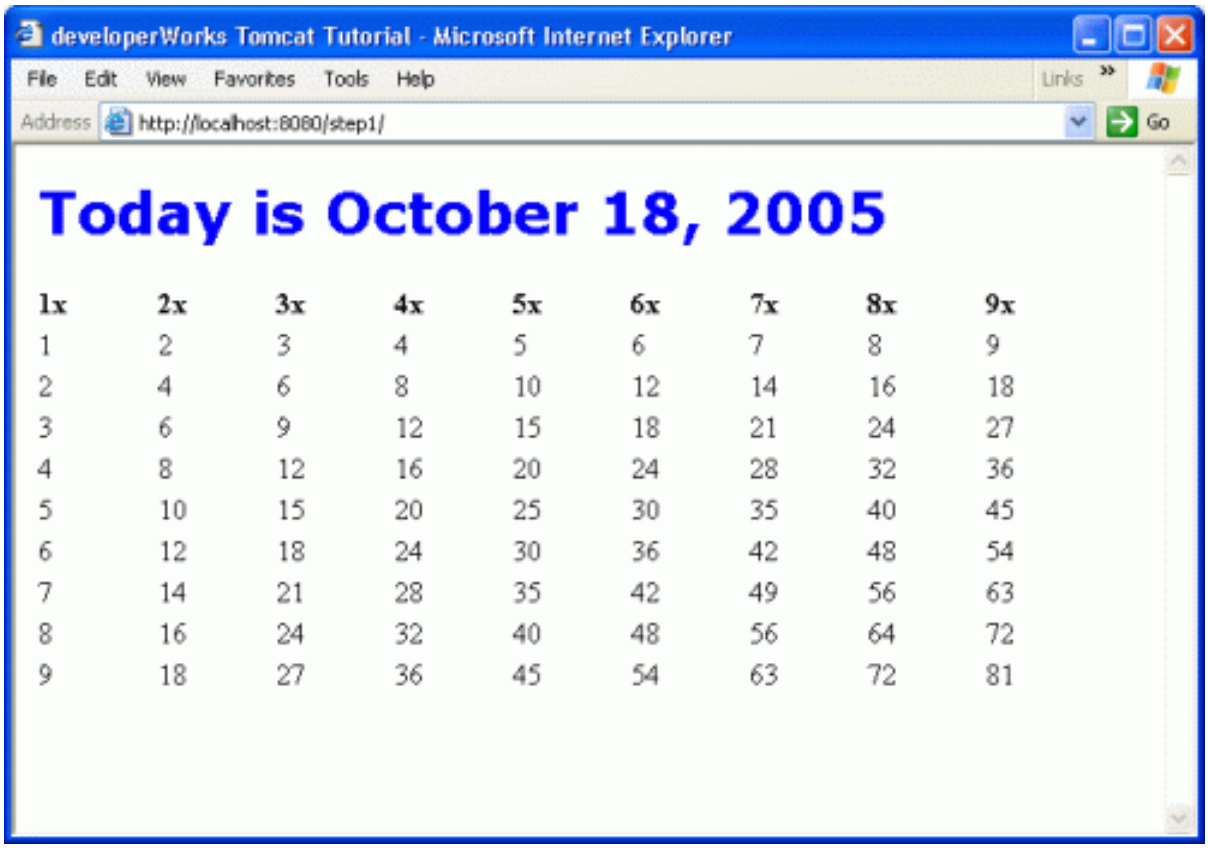
You should now see the step1.war application running on Manager's list of running applications. Note also that this list uses the display name you set in web.xml to identify the application.

Finally, you can see the JSP application running by pointing a browser to http://localhost:8080/step1/.

By default, Tomcat uses the name of the WAR file to provide a context for the application. The context is specified as part of the address you used to access the

application. In this case, the context is step1. Tomcat looks for a file called index.jsp in the application's root directory to execute if it exists. Figure 7 shows the running JSP application:

Figure 7. A running JSP application on Tomcat



1x	2x	3x	4x	5x	6x	7x	8x	9x
1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

If you make any modifications to the JSP code (as you learn JSP), you can perform the following steps to run the new code:

1. Use `makewar` to create a new WAR file.
2. Click **undeploy** in Manager to undeploy the old step1.WAR.
3. Use Manager to deploy the new WAR file.

Section 5. Breaking into servlets development with Tomcat

Brief introduction to servlets

Servlets are server-side Java code that is executed under the control of a *servlet container*, such as the Tomcat server. Servlets, like JSPs, accept incoming requests, perform processing or transformation, and then generate outgoing responses. Because servlets are actual Java code, you have the power and flexibility of the Java programming language at your disposal to create server-side logic.

All servlets implement the `javax.servlet.Servlet` interface, either directly or indirectly through a helper class that implements the interface. Servlets can also use the APIs provided by the container, exposing container services. For example, a servlet can obtain a database connection from the container to access a relational database.

Servlets are typically used to implement Web application logic. Servlets can fetch and process data, and then pass the data to a JSP for presentation (for example, dynamic generation of the user interface). Servlets are also used frequently to process data submitted through a Web-based form.

Tomcat 5.5 implements Servlet 2.4, the latest finalized servlet standard specified in JSR 154 (see Resources).

This section shows how to use Tomcat for learning servlets programming. See Resources for more information on servlets.

Servlet example to generate menu specials

The example in this section displays today's special menu items for a fictitious restaurant. A servlet is responsible for fetching the data, and a JSP is responsible for dynamically generating the HTML to display the specials. The JSP in this example is called showspecials.jsp, and the servlet is in the `com.ibm.dw.tutorial.tomcat.SpecialsServlet` Java class.

This example illustrates a typical pattern in Web applications:

1. A servlet accepts an incoming request from the user.
2. The servlet performs processing, based on the incoming request.
3. The servlet dispatches the request, with data attached as attributes, to a JSP.
4. The JSP generates a dynamic response to present the data.

Listing 4 shows the code for `SpecialsServlet`:

Listing 4. The SpecialsServlet servlet

```
package com.ibm.dw.tutorial.tomcat;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.IOException;
```

```
public class SpecialsServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
       throws ServletException, IOException {
         ServletContext context = getServletContext();
         request.setAttribute("specials", getSpecials());
        context.getRequestDispatcher("/showspecials.jsp")
            .forward(request, response);
        }
    protected void doPost(HttpServletRequest request,
       HttpServletResponse response)
             throws ServletException, IOException {
              doGet(request, response);
        }
    private List getSpecials() {
         List retval = new Vector();
         retval.add(new Special("Coq au Vin", 15));
         retval.add(new Special("Pad Thai", 10));
         retval.add(new Special("Lobster Thermador", 10));
         retval.add(new Special("Baked Alaska", 8));
         return retval;
    }
    public class Special {
         int price;
         String menuItem;
         public Special(String item, int inPrice)  {
              menuItem = item;
              price = inPrice;
         }
         public int getPrice() {
              return price;
         }
         public String getMenuItem() {
              return menuItem;
         }
     }
    }
```

The servlet code in Listing 4:

1. Declares an inner class called `Special` to hold a menu special.
2. Creates a list of specials in a method called `getSpecials()`.
3. In the `doGet() method`, which Tomcat calls to handle an incoming HTTP request, attaches the list of specials as an attribute called `specials` to the `request`.
4. Forwards the request to showspecials.jsp for presentation.

JSP to show today's special

Listing 5 displays the showspecials.jsp code:

Listing 5. The showspecials.jsp code

```
                    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<jsp:useBean id="timeNow" class="java.util.Date" />
<%@ page session="true"%>
<html>
<head>
<title>developerWorks Tomcat Tutorial</title>
<link rel=stylesheet type="text/css" href="specials.css">
</head>
<body>
```

```
<table width="600">
     <tr>
          <td class="mainHead" colspan="2">
          Today's specials for
          <fmt:formatDate value="${timeNow}" type="date"
               dateStyle="long" />
          </td>
     </tr>
     <tr>
      <th>Specialty</th>
      <th>Price</th>
     </tr>
     <c:forEach var="special" items="${specials}">
          <tr>
            <td>${special.menuItem}</td>
            <td>\$${special.price}</td>
          </tr>
     </c:forEach>
</table>
</body>
</html>
```

You can see that the same techniques used Listing 1 for the step1 example in the preceding section (Your first JSP application on Tomcat) are used here:

- The JSTL date formatting tag formats a `java.util.Date` JavaBean instance.
- The `<c:forEach>` JSTL loop tag iterates through the `List` attribute of `specials` (attached to the `request` object by `SpecialServlet`).
- EL expressions display the value of the specials.

Deploying the servlet to Tomcat

Servlets live in a Web application, in the same way as JSPs. You need to package the application into a WAR file before deploying both JSPs and servlets to Tomcat. Place the classes for a servlet under the WEB-INF/classes subdirectory for the application.

The Web descriptor for this example is slightly different from the one in Listing 2 in the preceding section (Your first JSP application on Tomcat). It must tell Tomcat about the servlet and how to map it to incoming requests. Listing 6 shows the web.xml file for this example:

Listing 6. web.xml file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
     version="2.4">

     <description>developerWorks Beginning Tomcat Tutorial
      </description>
     <display-name>
          IBM developerWorks Beginning Tomcat Tutorial Step 2
     </display-name>

     <servlet>
          <servlet-name>Specials</servlet-name>
          <servlet-class>
               com.ibm.dw.tutorial.tomcat.SpecialsServlet
```

```
            </servlet-class>
        </servlet>
        <servlet-mapping>
            <servlet-name>Specials</servlet-name>
            <url-pattern>/showspecials.cgi</url-pattern>
        </servlet-mapping>

</web-app>
```

This deployment descriptor includes a `<servlet>` element that tells Tomcat about the servlet's implementation class. The `<servlet-mapping>` element tells the server that requests destined for showspecials.cgi should be passed to this servlet.

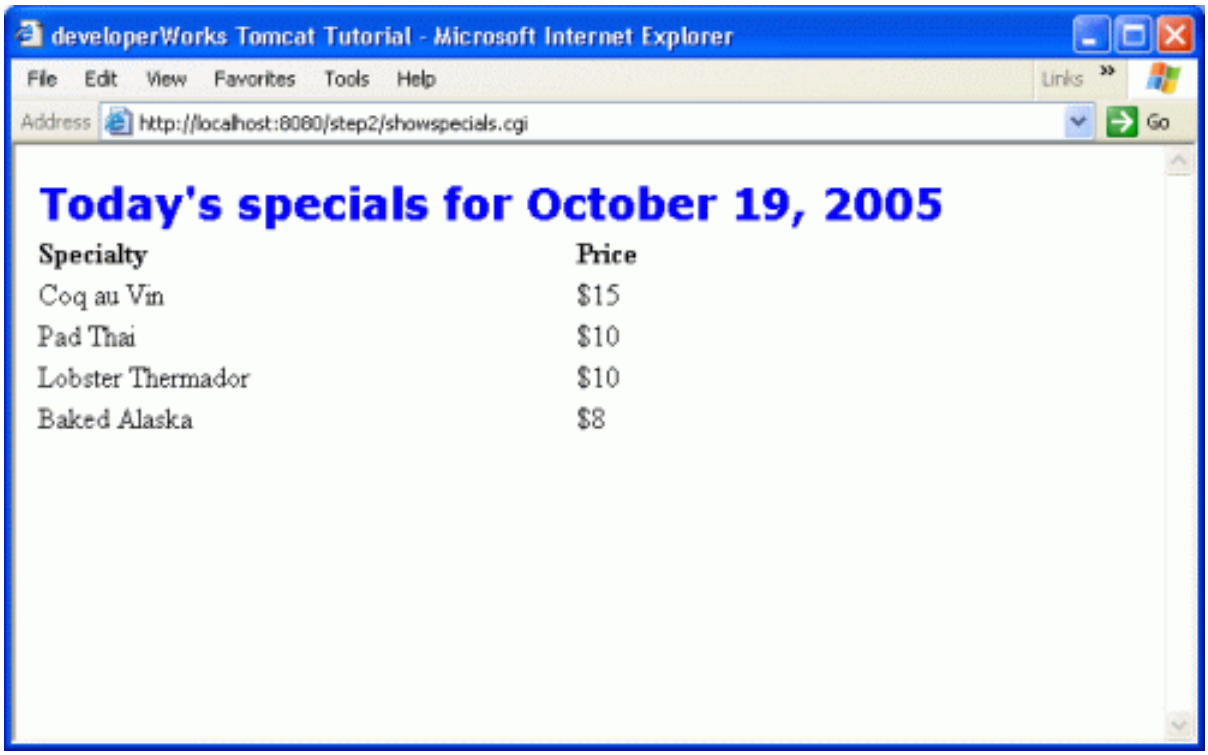
You must compile the servlet before you can create the WAR file. Because servlets are Java code, they must be compiled before they can be deployed. You can use the compile.bat batch file in the code distribution (see Download) for this purpose. However, you need to modify it to point to your own Tomcat directory, because servlet-api.jar from Tomcat's common/lib directory contains some interfaces and helper classes that are required to compile the servlet.

After you've compiled the application successfully, you can make the WAR file by executing makewar.bat. The WAR file is called step2.war this time. Once the WAR file is created, you can deploy it using the Tomcat Manager.

You can access the Web application by pointing a browser to http://localhost:8080/step2/showspecials.cgi.

Tomcat now maps the URL path of showspecials.cgi to the servlet code. This was specified in web.xml by a `<servlet-mapping>` element. Figure 8 shows the output of the application, displaying the menu specials:

Figure 8. SpecialsServlet running on Tomcat



If you make changes to the servlet code, you can run the new code with these steps:

1. Recompile the code using compile.bat.
2. Create the step2.war using makewar.bat.
3. Undeploy any old step2.war using Tomcat Manager.
4. Deploy the new step2.war.

Section 6. Discovering Web services development with Tomcat

Brief introduction to Web services

Web services are server-side code components that can expose their functionality for access over a TCP/IP network using the standard HTTP protocol. This exposure lets Web service users, called consumers, consume Web services over most network connections -- and even through firewalls.

A Web service processes incoming requests and generates responses. This is exactly what a servlet does, so it is quite natural to implement Web services using servlets.

Web services are becoming increasingly popular because they can be effectively used for business-to-business or business-to-consumer interfaces. They let requests be sent, and responses received, through the Internet. Any user who can access your Web site can also access your Web service(s). For example, both eBay and Amazon.com offer Web services that their partners and users can consume.

Web services depend on passing XML-based messages between the consumer and the service. The messages are packaged and sent according to the Simple Object Access Protocol (SOAP).

Apache Axis is a Web services development kit that can be used as an add-on to Tomcat. The next section shows you how to create a simple Web service, using Apache Axis, and deploy it on your Tomcat server. See Resources for articles and tutorials that can help you learn more about Web services programming.

Adding Axis to Tomcat

Axis can run as a servlet on Tomcat. If you haven't already done so, download the

latest version of Axis (see Prerequisites). Unarchive the Axis distribution.

Copy all the files under the webapps/axis directory of the Axis distribution to the step3/axis directory of this article's code distribution (see Download).

You can use the makewar.bat batch file, found in the step3/axis directory, to create an axis.war file that can be deployed to Tomcat as a Web application.

Before Axis will run properly on Tomcat, you might need to download some additional JAR files over the Internet and place them into the WEB-INF/lib directory of the step3 application. If you're using Axis 1.2.1, you need to download the following:

- **activation.jar** from http://java.sun.com/products/javabeans/glasgow/jaf.html
- **xmlsec-1.2.1.jar** from http://xml.apache.org/security/
- **mail.jar** from http://java.sun.com/products/javamail/

If you are not using version 1.2.1 of Axis, the above list might differ slightly. See the documentation accompanying the Axis distribution for more information.

A simple Web service to publish today's specials

Continuing the restaurant menu-special theme, the Web service you create in this section's example provides the consumer with a list of menu items on special for today.

The code for the Web service is contained in the ShowSpecials.jws file, shown in Listing 7:

Listing 7. ShowSpecials.jws

```
                    public class ShowSpecials {
     public String [] getMenuItems() {
          return new String []{
          "Coq au Vin", "Pad Thai",
          "Lobster Thermador", "Baked Alaska" };
       }
     public int  [] getPrices() {
      return  new int [] { 15, 10, 10, 8 };
     }
     }
```

The code in Listing 7 has two public methods. The `getMenuItems()` method returns the menu items that are on special, and the `getPrices()` method retrieves their price.

Axis supports an instant-deployment mode using .jws (Java Web services) files. In this mode, all you need to do is place a Java source file with a .jws extension in the axis directory. Any public methods of this class will be exposed through the Web service. This example uses the instant-deployment mode. The `getMenuItems()` and `getPrices()` methods are automatically exposed by Axis as Web service

methods when it parses and compiles the .jws file. The parsing and compilation occur only when you first access the Web service.

Deploying the Axis servlet to Tomcat

Because Axis runs as a servlet, the procedure for deploying the Web service to Tomcat is the same as in the earlier examples for Web application deployment.

In the step3/axis directory, run makewar.bat to create axis.jar. Use the Tomcat Manager to deploy the axis.war file to your Tomcat server.

Once deployed, your Web service is available at the URL http://localhost:8080/axis/ShowSpecials.jws. However, you will not see much if you go to this URL using a browser. Web services must be consumed using a Web service consumer application.

Testing the Web service with a consumer application

Coding Web service consumer applications is out of this tutorial's scope. Instead, you'll use Apache Ant (see Prerequisites) to compile a client application from existing code provided in the code distribution (see Download).

Because Web service requests are transported through standard Web protocols, the client application can be created on any operating system using any programming language. Many Web services toolkits automatically generate Web service consumers. This is possible because a tool can automatically discover how to access a Web service.

The description of how to access a Web service is supplied in an XML file, coded in the Web Service Definition Language (WSDL). Most Web service containers (such as Axis) can generate this WSDL file from a Java class or interface. For example, to see the generated WSDL for your Web service, try using a browser to access the URL http://localhost:8080/axis/ShowSpecials.jws?wsdl. The complex-looking XML document you see is generated from your .jws code. This WSDL document can be easily processed by a tool, and the client consumer code to invoke the Web service can be generated automatically.

The client for this example does not use WSDL or automated code generation. It is hard-coded to call the ShowSpecials.jws Web service. This keeps the coding of the client relatively simple.

You need Ant installed to compile and run the Web service client (see Prerequisites) because the code needs many Axis library JAR files to compile and run successfully.

Before compiling the client, find the build.xml file in the step3/client directory and edit the file to point to your Axis installation directory.

To compile the client, run Ant with the following command:

```
ant compile
```

To start the client, consuming the Web service, run Ant with the following command:

```
ant run
```

Typical output from a successful run of the Web service client, showing the specials menu items and price, looks like this:

```
Buildfile: build.xml

run:
     [java] Specials today:
     [java]     Coq au Vin                      $15
     [java]     Pad Thai                        $10
     [java]     Lobster Thermador               $10
     [java]     Baked Alaska                    $8

BUILD SUCCESSFUL
Total time: 7 seconds
```

Section 7. Summary

The Tomcat server is a great platform for learning JSPs, servlets, and Web services programming. In this tutorial, you have learned how to:

- Download and install your own Tomcat server.
- Deploy and undeploy applications using Tomcat Manager.
- Create a JSP-based application and execute it on the Tomcat server.
- Create a Web application with a servlet and execute it on Tomcat.
- Create a Web service and execute it on Tomcat.

Now that you've had some hands-on experience with these tools to develop and run Web applications, you're ready to explore JSP, servlet, and Web services programming further on your own.

Resources

Learn

- Tomcat Web site: Tomcat documentation, how-tos, and mailing lists.
- Java Community Process: You can find the Java Specification Requests for JavaServer Pages, Java Servlets, and the JSP Standard Tag Library here.
- "Introduction to JavaServer Pages technology" (Noel J. Bergman, developerWorks, August 2001): Learn JSP programming with this tutorial.
- "Introduction to Java Servlet technology" (Roy Miller, developerWorks, December 2004): This tutorial uses Eclipse's Tomcat plug-in to make servlet development on Tomcat even easier.
- New to SOA and Web services: Get a handle on the basics of Web services technology.
- SOA and Web services zone: Find articles and tutorials about every aspect of Web services programming.
- Java technology zone: Find more resources for Java developers.

Get products and technologies

- Tomcat downloads page: Find the latest Tomcat download.
- Apache Axis: Download the Axis Web services engine.
- Apache Ant: You need to install Ant, a versatile build tool for Java projects, to build the Web service client in this tutorial.

About the author

Sing Li
Sing Li is a consultant and freelance writer. He has contributed to *Beginning JavaServer Pages* , *Professional Apache Tomcat 5* , *Pro JSP - Third Edition* , *Early Adopter JXTA*, *Professional Jini*, *Beginning J2ME: From Novice to Professional, Third Edition* , and numerous other books. He is also a regular contributor to technical magazines and an active evangelist of the VON and P2P evolutions. You can reach Sing at westmakaha@yahoo.com.