# 7

# Manager Configuration

In this chapter we will go over the management tools available with Tomcat versions 3.x and 4.x. These tools allow administrators to deploy web applications, view deployed applications, and finally undeploy them. While these tasks can also be performed manually by editing Tomcat's configuration files, this method requires Tomcat to be restarted. The `manager` application, however, helps in automating them and also allows them to be performed on a running instance of Tomcat. This way, applications that are already running are left undisturbed.

## Sample Web Application

In this chapter, we will use a simple web application for testing the manager commands. This application consists of nothing more than one HTML and one JSP file.

The HTML file (`index.html`) has a form that asks for the user's name and uses HTTP `POST` to send the result to a JSP page:

```
<html>
  <head>
    <title>Hello Web Application</title>
  </head>

  <body>
    <h1>Hello Web Application</h1>
    <form action="/hello/Hello.jsp" method="POST" >
      <table width="75%">
        <tr>
          <td width="48%">What is your name?</td>
          <td width="52%">
```

```
            <input type="text" name="name" />
          </td>
        </tr>
      </table>
      <p>
        <input type="submit" name="Submit" value="Submit name" />
        <input type="reset" name="Reset" value="Reset form" />
      </p>
    </form>
  </body>
</html>
```

The JSP page (`hello.jsp`) then prints a Hello <name> message. The <name> is the name that the user entered in the `index.html` form:

```
<html>
  <head>
    <title>Hello Web Application</title>
  </head>
  <body>
    <h1>Hello Web Application</h1>
    <br/><br/>

    <%
    String name   = request.getParameter("name");
    if (name.trim().length() == 0) {
    %>
      You didn't tell me your name!<br><br><br>
    <%
    } else {
    %>
      Hello <%=name%><br><br><br>
    <%
    }
    %>
    <a href="/hello/index.html">Try again?</a>
  </body>
</html>
```

This web application will be deployed with the `/hello` context path, and therefore http://localhost:8080/hello/index.html would be the URL to access it.

The commands for building the WAR file are:

```
$ cd /path/to/hello
$ jar cvf hello.war .
```

The `/path/to/hello` is the directory where the `index.html` and `hello.jsp` files reside.

# Tomcat 3.x Administration Tool

Tomcat 3.x comes with an administration tool for performing simple administration tasks that allows us to list all web application contexts and to remove/install web applications. In this section, we will go over these and also see how permissions for the administration application are enabled. The version of Tomcat used here is 3.3.1.

## Enabling Permissions for the Admin Tool

To enable permissions for the Tomcat 3.x `admin` tool, run `tomcat.bat/tomcat.sh` with the `enableAdmin` option, as follows. Here is the command for Linux:

```
$ cd $TOMCAT_HOME/bin
$ ./tomcat.sh enableAdmin
Overriding apps-admin settings
```

This rewrites the `<Context>` definition of the `admin` application (`$TOMCAT_HOME/conf/apps-admin.xml`), changing the `trusted` attribute to `true`, and thus allowing it to administer other web applications:

```
<webapps>
  <!-- Special rules for the admin web application -->
  <Context path="/admin"
           docBase="webapps/admin"
           trusted="true">
    <SimpleRealm filename="conf/users/admin-users.xml" />
  </Context>
</webapps>
```

A web application with the `trusted` attribute switched on can access the internal objects of Tomcat and manipulate them. This feature is used by the Tomcat `admin` application to perform actions like removing and installing web applications, and examining data from user sessions. The `trusted` attribute is `false` by default for security reasons.

After enabling the `admin` application, its password should be changed from the default "`changethis`" password. The password is stored in the `$TOMCAT_HOME/conf/users/admin-users.xml` file (listed below) as cleartext:

```
<tomcat-users>
  <user name="admin" password="changethis"
        roles="tomcat_admin,tomcat,role1" />
</tomcat-users>
```

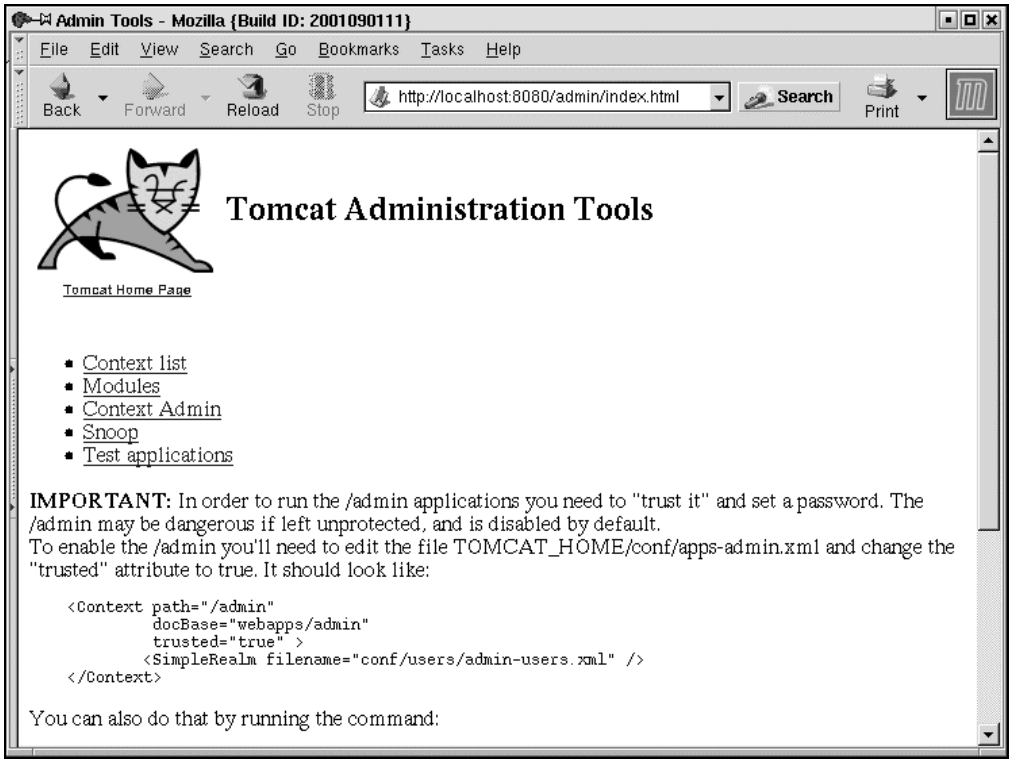
Leaving the `admin` application enabled without the password changed from this default password is a security risk.

There is no corresponding `disableAdmin` command for undoing this step – the administrator will have to edit the `$TOMCAT_HOME/conf/apps-admin.xml` file, modify the `trusted` attribute to `false` and then restart Tomcat.

## admin Application Tasks

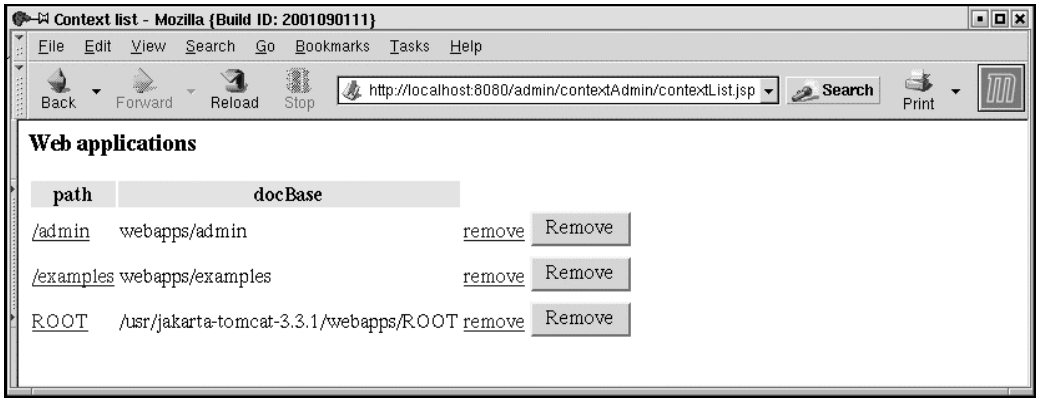
If Tomcat is running on a local machine, the `admin` application is accessible via the URL http://localhost:8080/admin/. The host name and port number (localhost, 8080) in the URL should be changed to the appropriate host and port for your installation. If we have just changed the `trusted` attribute manually, or via the `tomcat enableAdmin` command, we need to restart Tomcat before we can start using the `admin` application.

This `admin` page lists all the `admin` application tasks (shown overleaf). As mentioned earlier, access to these tasks is restricted via a username/password that is specified in the `admin-user.xml` file. The screenshot overleaf shows us the `admin` home page:

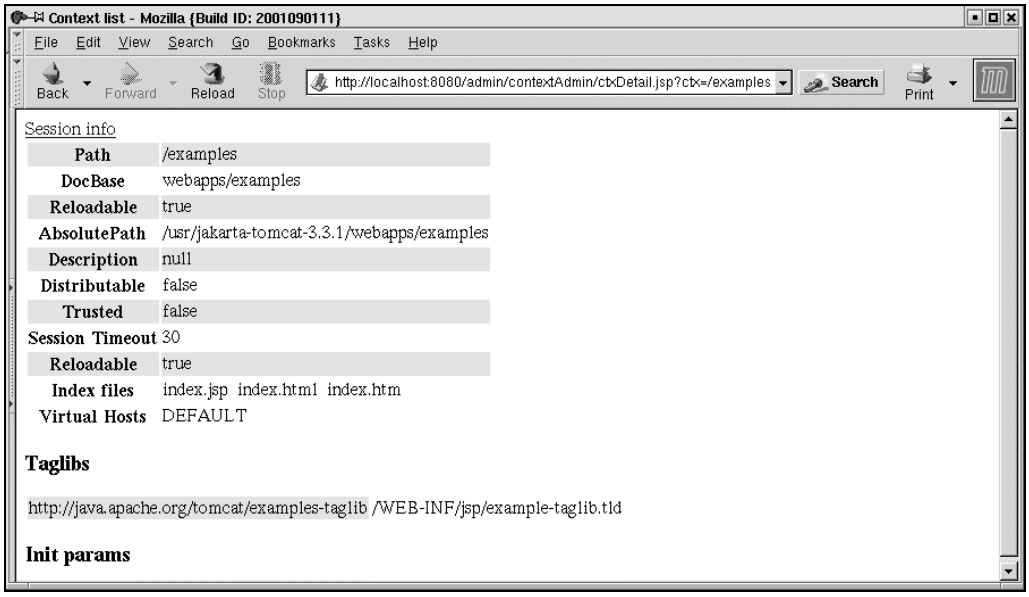



## *List All Web Application Contexts*

The first option of the `admin` application's index page is Context list. This option shows all the deployed contexts (see the screenshot below):




Clicking on any item in the path column, we can see the attributes of the context as configured in the `app-[name].xml` or `server.xml` files. The screenshot opposite shows the context attributes for the `example` web application. We also can see session information (number of active/recycled sessions, default timeout) for each application context via the Session info link:

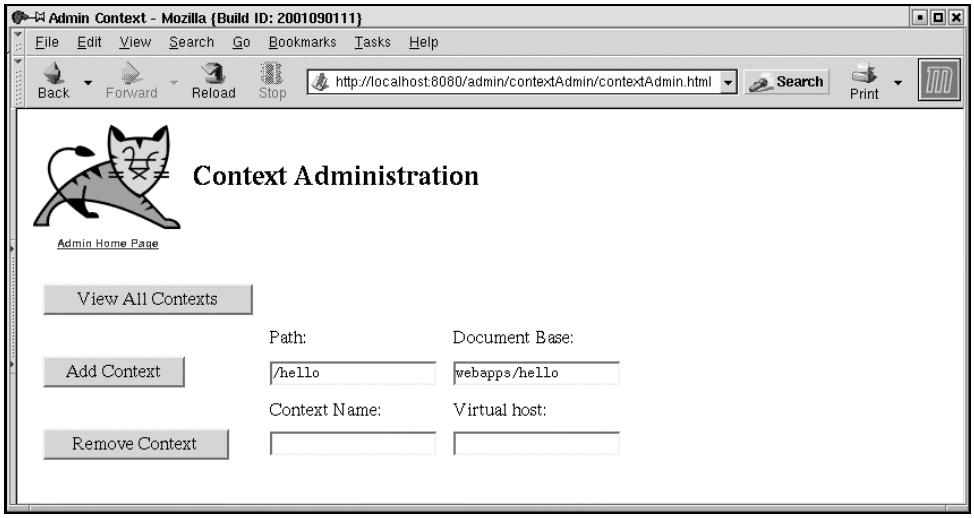



## *Add Web Application Context*

The Context Admin link on the admin home page leads us to the Context Administration page. The Add Context option here adds a web application context to a running instance of Tomcat. This is done by adding a new Context entry to Tomcat's internal runtime representation of application contexts (the `org.apache.tomcat.core.ContextManager` class). Tomcat's configuration files are not modified.

This option is useful for development purposes – Tomcat developers can temporarily add a context, test it, and then remove it. The option for removing a context is covered in the next section.

We can specify the context path (for example `/hello` for the `hello` web application mentioned earlier), and the document base. The web application is then started up without requiring Tomcat to be restarted:



The document base is relative to the Tomcat install directory – therefore, in the screenshot shown previously, the path webapps/hello refers to `$TOMCAT_HOME/webapps/hello`. This means that the expanded web application needs to be copied inside the Tomcat install directory – the recommended place is `$TOMCAT_HOME/webapps`.

We could have placed the web application directory outside the `$TOMCAT_HOME/webapps` directory, for example, under `$TOMCAT_HOME` itself. If we did this, the web application is not loaded if Tomcat is restarted. This is because Tomcat loads only those web applications that are present in `$TOMCAT_HOME/webapps`. If they lie elsewhere in the filesystem, they need to be configured via explicit `<Context>` entries in the `server.xml` file.

The options that we can specify for a new context are limited to the path and document base. The other attributes get assigned defaults – `reloadable` set to `true`, `trusted` set to `false`, and so on. If we want to set an attribute that is not the default, we have to add explicit `<Context>` entries for the application in `server.xml` and restart Tomcat.

If we add an application that already had a `<Context>` entry configured in `server.xml` (for example, if we had removed it earlier, and are adding it again), it still gets default `<Context>` attribute values – the `server.xml` is not read.

The `admin` application does not do any error checking, therefore if we do an Add Context twice, or supply an invalid document base, a success message is still shown in the browser window. Any internal Tomcat errors while adding the context get logged to the standard log files.

## *Disabling Web Applications*

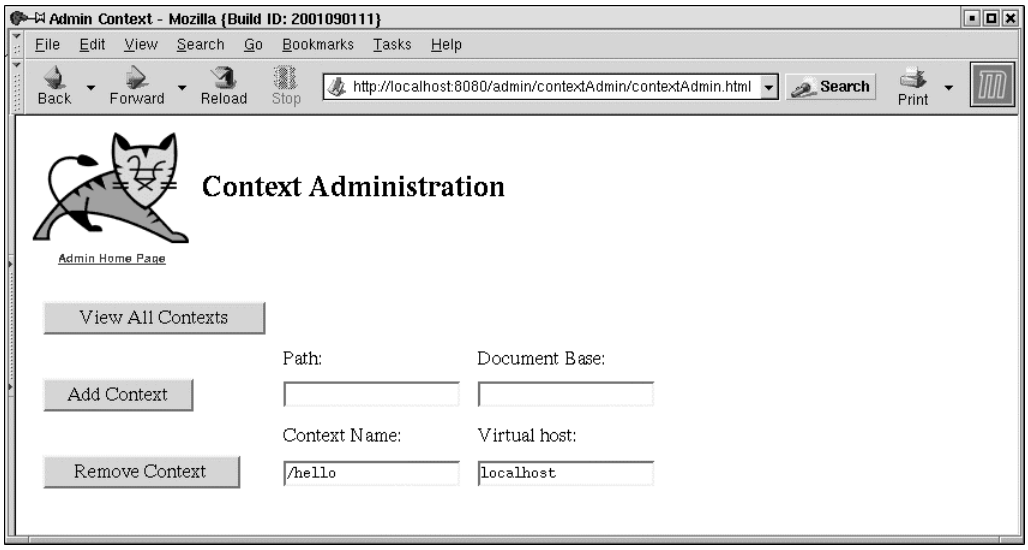
The option to disable a web application context is accessible via the View All Contexts page and the Context Administration page. This option is called Remove or Remove Context, but the name is deceptive. This option:

- Temporarily disables the web application by removing its context from the in-memory representation of the web application contexts. The internal Tomcat class that maintains the list of application contexts is `org.apache.tomcat.core.ContextManager`, and the Remove Context option deletes the context from it.
- Shuts down the web application.
- Doesn't remove the web application from the `$TOMCAT_HOME/webapps` directory, or remove the `apps-[name].xml` configuration file, if any, from the `$TOMCAT_HOME/conf` directory. Hence, if we restart Tomcat, the web application is loaded up again.

As we mentioned in the previous section, the Add and Remove Context options are useful for development purposes.

In the Context Administration page, we need to specify the name of the context (example `/hello` or just `hello` for the `hello` web application or an empty string for the `ROOT` context) to remove that context. The other parameter for this option, Virtual Host, can be left empty. This parameter is ignored by the `admin` application.

The screenshot below shows the `hello` web application being disabled:




# Tomcat 4.x Manager Application

The Tomcat `manager` application is a web application that allows us to carry out various system administration tasks related to deploying, undeploying, and managing a web application.

Administrators interact with the `manager` application over HTTP. In Tomcat 4.1, administrators can also run the `manager` application tasks via Ant scripts. Using the `manager` application through Ant allows developers to build, deploy, and test web applications easily – we show how this can be done in the *Managing Applications with Ant* section later in the chapter.

Access to the `manager` application is restricted to authorized users. This prevents unauthorized users from undeploying (or deploying) applications, or performing any other operation that they shouldn't. In the next section, we will see how this access control is configured. We will then examine the other configuration parameters for the `manager` application. Finally, we will describe all the `manager` application commands in more detail.

A summary of the tasks that the `manager` application can do is listed below. Some of these tasks are new features in Tomcat 4.1, and are not available with Tomcat 4.0:

- Deploy a new web application (Tomcat 4.1 only)
- Install a new web application
- List the currently deployed web applications, as well as the sessions that are currently active for those web applications
- Reload an existing web application
- List the available global JNDI resources (Tomcat 4.1 only)

- List the available security roles (Tomcat 4.1 only)
- Remove an installed web application
- Start a stopped application
- Stop an existing application, but do not undeploy it
- Undeploy a web application (Tomcat 4.1 only)
- Display session statistics

In the list above, we use the terms "install" and "deploy". At first glance, these look the same, however, there are differences as far as the Tomcat `manager` application is concerned.

When we deploy a web application, it makes permanent changes to Tomcat's configuration, and hence the web application is available across Tomcat restarts. The install option however, does not make permanent changes to Tomcat's configuration. Thus the `install` command is useful for test purposes. We can build a web application, install it, and then try it out. Once it is sufficiently robust, we can run the `deploy` command to permanently place it into a Tomcat installation. In some ways, `install` and `uninstall` are similar to the Add Context and Remove Context operations in Tomcat 3.x.

Secondly, the `deploy` command allows us to deploy a web application remotely. With `install`, the web application JAR file (or the extracted web application path) must be on the same machine as that of the Tomcat instance.

The `undeploy` and `uninstall` commands undo the effects of `deploy` and `install` – we shall see these later in the section.

However, to add to our confusion, Tomcat 4.0 has a command called `install` that actually has an effect similar (though not identical) to the `deploy` command in Tomcat 4.1. The Tomcat documentation, to help matters, calls it the command to deploy applications (even though the name of the command is install). We shall shed some light on this later in the section.

Why do we need a special `manager` application to deploy and undeploy applications? We can deploy an application manually too. The ways to do this are:

- Add a `<Context>` entry in Tomcat's `server.xml` configuration file. This allows us to place the web application in a location other than the default `$CATALINA_HOME/webapps` directory.
- Copy the entire application directory into the `$CATALINA_HOME/webapps` directory. The `server.xml` file does not have to be edited in this case.
- Copy the WAR file for the application into the `$CATALINA_HOME/webapps` directory. In this option too, the `server.xml` file does not have to be edited.

However, all these ways of deploying require us to restart Tomcat. When we do this via the `manager` application, Tomcat is not restarted and hence the other running web applications are not affected. Another advantage of using the `manager` application is that we can install remotely. That is, we do not have to transfer the web application directory (or WAR file) via FTP or some other means to the host machine running Tomcat – the `deploy` command takes care of transferring the web application WAR file from our local development machine to the remote machine running the Tomcat server.

**The latest version of the 4.1.x line (Tomcat 4.1.7 Beta) at the time of writing, added a new web-based user interface on top of the `manager` application. This does not affect the existing application functionality and usage (explained below), but provides a simplified interface. We will be covering this interface later in this chapter.**

## Enabling Access To the Manager Application

Before using the `manager` application, we need to configure the server to allow us access. Access to this application is controlled via a security realm (realms were discussed in Chapter 5).

Tomcat comes configured with the memory realm by default. In this case, the usernames and their supporting information are stored in memory and are initialized at startup from an XML configuration file (`$CATALINA_HOME/conf/tomcat-users.xml`) kept on the filesystem.

We need to edit this file to add a user with a role of `manager`. In the entry below, the username and password for this role is `admin` and `secret` respectively.

In Tomcat 4.0:

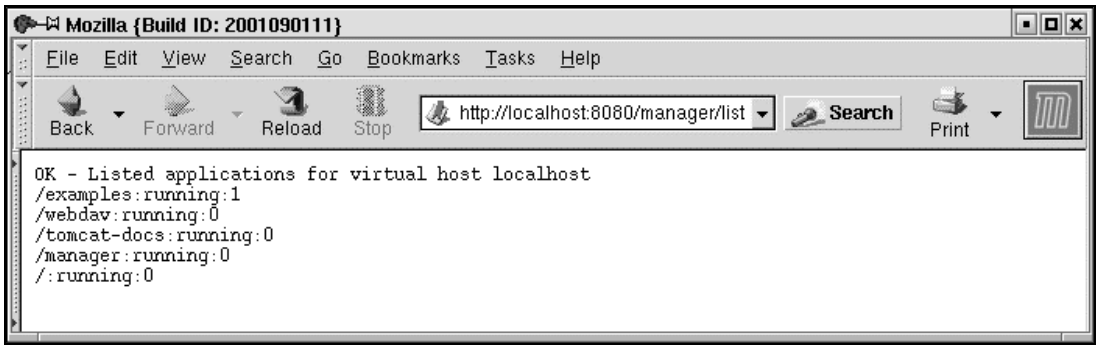
```
<tomcat-users>
  <user name="admin" password="secret" roles="manager" />
  ...
</tomcat-users>
```

And in Tomcat 4.1:

```
<tomcat-users>
  <role rolename="manager"/>
  ...
  <user username="admin" password="secret" roles="manager" />
  ...
</tomcat-users>
```

We need to restart Tomcat to make it re-read the `tomcat-users.xml` file. We are now ready to use the Tomcat `manager` application. If our setup was successful, the URL http://localhost:8080/manager/list will lead to a prompt for a username and password.

After entering the username and password we should see the applications currently deployed listed in the browser:



As we can see in the screenshot, the response for a successful command execution begins with an `'OK'` string. A missing `'OK'` is an indication of failure, and the rest of the response page gives the reasons. We will go over the possible causes of failure for each command later in the chapter. The response page is in the `text/plain` format, that is, it contains no HTML markup.

The data fields returned in a manager command response are always delimited by the ":" character. In the screenshot for the `list` command shown previously, each line has the (unique) context path of the web application, the status (running or stopped), and the number of active sessions for the application. In Tomcat 4.1, the document base for the web application is also shown.

These conventions allow for scripts to be written that take the output of the `manager` command and perform appropriate actions.

### *Note for Tomcat 4.1 On Windows*

During installation of Tomcat 4.1 on Windows, the installer asks the user for the `admin` username and password. The username and password entered at install time are used to generate entries for the `tomcat-users.xml` file. Hence, we do not have to do any configuration in this case, except if we need to add another user with `manager` privileges or need to change the `manager` password.

## Manager Application Configuration

In the previous section, we looked at `tomcat-users.xml` that defines the username and password for the manager role. The other manager application-related configuration parameters are the manager context entry and the deployment descriptor.

We do not have to make any changes here for the `manager` application to work – the settings are configured by default. We can, however, modify these to our requirements – for example, change the security constraints for the `manager` application, the authentication mechanism for users in the `manager` role, or even change the name of the role from `manager` to some other name if required.

### *Manager Application Context Entry*

In Tomcat 4.0, the `manager` context is configured in the same way as the other application contexts. The following is the default configuration for the `manager` application from the `$CATALINA_HOME/conf/server.xml` file:

```
<!-- Tomcat Manager Context -->
<Context path="/manager" docBase="manager" debug="0" privileged="true"/>
```

In Tomcat 4.1, the configuration information for the `manager` application gets picked up from the `$CATALINA_HOME/webapps/manager.xml` file. The default manager context from the configuration file is listed below:

```
<Context path="/manager" docBase="../server/webapps/manager"
         debug="0" privileged="true">

  <!-- Link to the user database we will get roles from -->
  <ResourceLink name="users" global="UserDatabase"
                type="org.apache.catalina.UserDatabase"/>
</Context>
```

In the context, we specify the context path for the `manager` application via the `path` attribute (the `manager` application can be accessed as http://host:port/manager) and the document base directory for the web application via the `docBase` attribute ("../server/webapps/manager"). The `privileged` attribute is set to `true` – this enables the application to access the container's servlets. This attribute is `false` for a normal web application deployed in Tomcat. The `<ResourceLink>` element creates a link to a global JNDI resource database from where the usernames and roles are picked up.

## *Manager Application Deployment Descriptor*

Earlier we looked at the `tomcat-users.xml` file that defined the username and password for the `manager` role. We will now see how the security constraints for this role are specified. The deployment descriptor for the Tomcat 4.0 `manager` application is `$CATALINA_HOME/webapps/manager/WEB-INF/web.xml`.

In Tomcat 4.1, the `web.xml` is in `$CATALINA_HOME/server/webapps/manager/WEB-INF`.

The `web.xml` defines, among other things, the security constraints on the `manager` application. The snippet below describes the default security constraint definition for the `manager` web application (the "`/*`" URL pattern matches the entire web application). The `<role-name>` defined below (`manager`) specifies that only users in the `manager` role can access the `manager` web application:

```
<!-- Define a Security Constraint on this Application -->
<security-constraint>
    <web-resource-collection>
      <web-resource-name>Entire Application</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
       <!-- NOTE:  This role is not present in the default users file -->
       <role-name>manager</role-name>
    </auth-constraint>
  </security-constraint>
```

The authentication mechanism for the `manager` application is also defined here. The default setting is `BASIC` authentication. Administrators could set up a more rigorous mechanism for `manager` application authentication, for example a client certificate-based mechanism (`<auth-method>` set to `CLIENT-CERT`):

```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Tomcat Manager Application</realm-name>
  </login-config>
```

The `<security-role>` lists all the roles that can log in to the `manager` application. In this case, it is restricted to only one user role – the `manager` role:

```
  <!-- Security roles referenced by this web application -->
  <security-role>
    <description>
      The role that is required to log in to the Manager Application
    </description>
    <role-name>manager</role-name>
  </security-role>
```

## Manager Application Commands

The `manager` application commands that are issued via the web browser have the following format:

```
http://{hostname}:{portnumber}/manager/{command}?{parameters}
```

In the command above the various parts are:

- `hostname`
  The host that the Tomcat instance is running on.
- `portnumber`
  The port that the Tomcat instance is running on.
- `command`
  The manager command that we wish to run. The allowed values for `command` are `deploy`, `install`, `list`, `reload`, `remove`, `resources`, `roles`, `sessions`, `start`, `stop`, and `undeploy`. We shall be looking at these in more detail later in the chapter.
- `parameters`
  The parameters passed to the commands listed above. These are command-specific, and are explained in detail along with the specific command below. Many of these parameters contain the context path to the web application (the `path` parameter) and the URL to the web application file (the `war` parameter). The context path for the `ROOT` application is an empty string. For all other web applications, the context path must be preceded by a '`/`'. The URL to the web application can be in one of the following formats:
  - `file:/absolute/path/to/a/directory`
    This specifies the absolute path to a directory where a web application is present in an unpackaged form. This entire path is then added as the context path of the web application in Tomcat's configuration.
  - `file:/absolute/path/to/a/webapp.war`
    This specifies the absolute path to a WAR file. The Tomcat documentation states that this format is not allowed for the `install` command. However, our tests with Tomcat 4.1.3 indicate that it works fine for `install` too.
  - `jar:file:/absolute/path/to/a/warfile.war!/`
    The `jar` protocol allows for specifying the URL for a WAR file. This is handled by the `java.net.JarURLConnection` that provides a URL connection to a JAR/WAR file. Here the URL specified is for a file on the local filesytem.
  - `jar:http://hostname:port/path/to/a/warfile.war!/`
    In this case, the URL points to a WAR file on a remote host that is accessible via HTTP.

The "`!/`" characters at the end of these URLs allow them to be used in a web browser and not cause the default MIME type action for the `.war` extension to take effect. For example, if we use the URL shown below to install a web application and leave out the "`!/`" at the end, we may be prompted to (depending on how the browser MIME settings are configured) save to disk, open the file in the browser, or open the file in an application (for example, Winzip):

http://localhost:8080/manager/install?path=/hello&war=jar:file:/path/to/hello.war!/

There are a number of problems that could occur while working with the `manager` application. The possible causes of failure are listed in the *Possible Errors* section.

## Deploying a New Application (Tomcat 4.1 Only)

The `deploy` command is new in Tomcat 4.1. It allows users to deploy a web application to a running instance of Tomcat. The effect of this command is:

- The WAR file for the web application is uploaded from the client machine to the machine that Tomcat is running on, and copied into the application base directory of the given virtual host. For example, if the virtual host name configured in `server.xml` were `localhost` itself, the WAR file would get copied under `$CATALINA_HOME/work/Standalone/localhost/manager`.
- An entry for the web application's context is added into Tomcat's runtime data structures.
- The web application gets loaded.

The two steps above make the web application available for use right away, but there is a final step in the `deploy` process:

- Tomcat writes out a `<Context>` entry for this web application into the `$CATALINA_HOME/conf/server.xml` file. Due to this, the web application gets loaded each time Tomcat is restarted. If the WAR file contains a `<Context>` element definition (`META-INF/context.xml`), this context entry takes precedence over the default entry that the `manager` application generates.

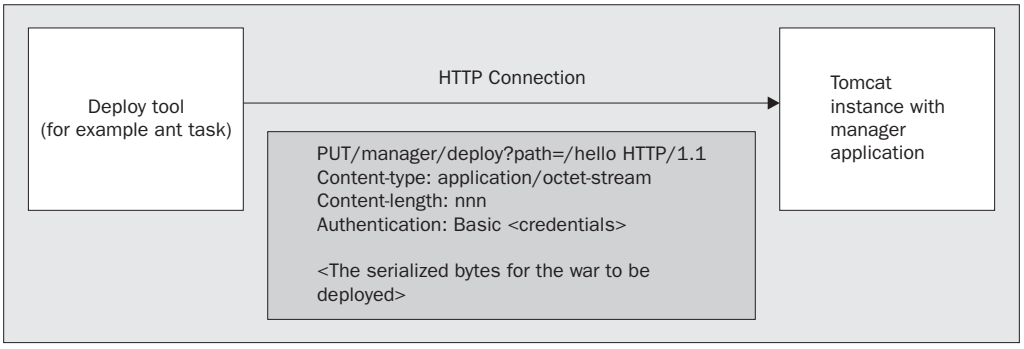
The general format for the `deploy` command is shown below:

```
http://{hostname}:{portnumber}/manager/deploy?path={context_path}
```

Here `hostname` and `portnumber` are the host and port for the Tomcat instance and `context_path` is the context path for the application. The WAR file to be deployed is passed inside the request data of the HTTP `PUT` request. Therefore, if we were to deploy our `hello` application that we saw at the beginning of the chapter at the context path `/hello`, we would have to do an HTTP `PUT` to the URL `http://{hostname}:{portnumber}/manager/deploy?path=/hello`.

Since the WAR file is passed as the request data, this command cannot be invoked via a web browser. Instead, it should be invoked from a tool (for example, within an Ant script).

Essentially, the tool that sends the `deploy` command will have to build an HTTP request that looks something like that shown below, and execute an HTTP `PUT` command to send it over to the manager servlet:



A successful `deploy` command returns a success message OK – Deployed application at context path {context_path}. If the operation failed, the error message would start with a "FAIL" string, and contain the cause for failure.

Tomcat 4.0 has an `install` command that it calls a `deploy` command. This is actually true – the behaviour of the `install` command is different between 4.0 and 4.1. We shall see this difference in the next section.

## *Installing a New Application*

The Tomcat 4.0 documentation says that the command called `install` **deploys** an application. This is true, as this command has a permanent effect in Tomcat's configuration. The effect that the Tomcat 4.0 `install` command has is:

- The web application WAR file (or extracted directory) is copied into the application base directory
- The web application is started
- Tomcat's internal runtime data structures are updated to reflect the new application context

The last two steps ensure that the new web application is available for use right away. The first step (the copying) makes the installation permanent (that is, across Tomcat restarts).

In Tomcat 4.1, the `install` option just does the last two steps – updates Tomcat's internal runtime data structures and starts the application. Thus, if Tomcat is restarted, the web application is not reloaded. This is the correct behavior for the future, as the `install` command is meant for developers to test new web applications, and, once they are happy with them, use the `deploy` command (shown above) to update Tomcat's installation.

The general format for the `install` command URL (in both Tomcat 4.0 and 4.1) is shown below:

```
http://{hostname}:{portnumber}/manager/install?path={context_path}&war={war_url}
```

To install from a WAR file, we use the command shown below. Here, `file:/path/to/hello.war` is the URL for the local filesystem location of `hello.war`:

http://localhost:8080/manager/install?path=/hello&war=jar:file:/path/to/hello.war!/

We can also install the extracted web application from a filesystem path (`/path/to/hello`). We need to enter the following command in the browser to install the `hello` application:

http://localhost:8080/manager/install?path=/hello&war=file:/path/to/hello

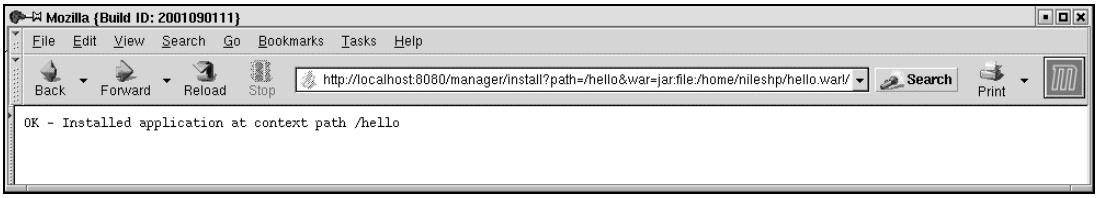
The command assumes that the `hello` application is extracted into `/path/to/hello`:

And the Windows version:

http://localhost:8080/manager/install?path=/hello&war=file:/C:\path\to\hello.war

If this succeeds, a message "OK – Installed application at context path /hello" will be shown in the browser window. If the operation fails, an appropriate error message is displayed.

The screenshot opposite shows the `hello` application WAR file (`hello.war`) being installed with the context path `/hello`:




If we try to rerun this command we get the "Application already exists" error message as shown below. The context path for a web application is unique – if we wish to update an already installed application, we would need to either reload it, or remove and install it again. These options are covered later in the chapter:


Tomcat 4.1 introduces two new options in the `install` command – unfortunately, these are not listed in the current Tomcat documentation:

```
http://{hostname}:{portnumber}/manager/install?config={config_url}

http://{hostname}:{portnumber}/manager/install?config={config_url}&war={war_url}
```

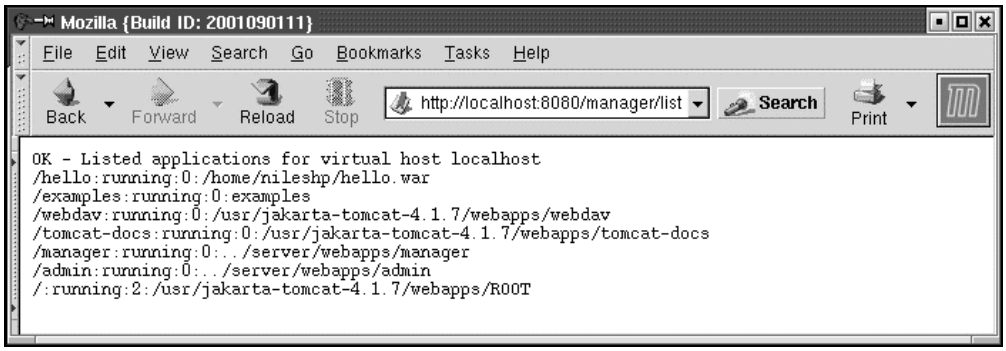
The `config_url` is the URL for a context configuration file. This file contains the `<Context>` element entry for the web application. The document base in the context is used to point to the location of the WAR file or to the directory where the web application is extracted. The second version of the command allows us to pass the URL to the WAR file (`war_url`). This overrides the document base specified in the context configuration file.

## List Installed and Deployed Applications

The format for the `list` command URL that lists all deployed and installed applications is shown below:

```
http://{hostname}:{portnumber}/manager/list
```

The screenshot below shows the `list` command being run:



As we can see, the first line of the response message contains a string indicating success ("OK – Listed applications of virtual host {hostname}").

The rest of the response has information on all deployed and installed applications: one web application per line of response. Each line has the (unique) context path of the web application, the status (running or stopped), and the number of active sessions for the application. In Tomcat 4.1, the path for the web application is also shown.

## *Reload Existing Application*

An existing application can be reloaded by accessing the `manager` application via the URL given below:

```
http://{hostname}:{portnumber}/manager/reload?path={context_path}
```

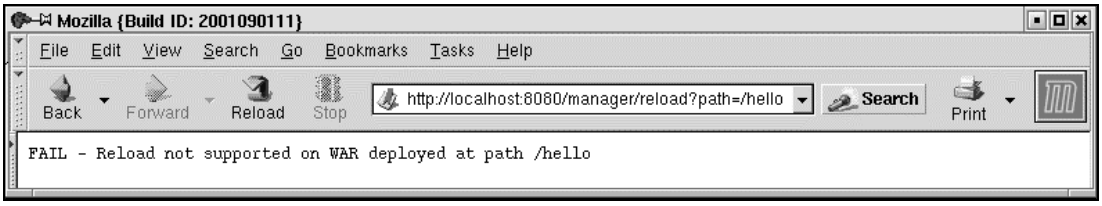
This causes the existing application to shut down and then restart. The application's deployment descriptor (`web.xml`) is not reread (at least not in the current version of Tomcat), even though the Tomcat documentation states that it is. This is a known bug, and it is expected that a future version of Tomcat will fix it. The workaround is to stop and then start the application again. The `server.xml` configuration file is also not reread, but this is by design.

The `reload` command is useful when we have a web application that has not been configured to be reloadable. A web application's `<Context>` entry in the `server.xml` file has a `reloadable` attribute. When this attribute is set to `true`, Tomcat monitors all its classes in `/WEB-INF/classes` and `/WEB-INF/lib` and reloads the web application if a change is detected. This causes a performance hit in production environments, as the class loader keeps comparing the date-time stamps for servlets in memory with those on disk. To avoid this, we can use the `reload` command to make Tomcat reload the web application when we change any of its classes.

The standard Java class loader is designed to load a Java class just once – so how does the `reloadable` attribute work? Tomcat implements its own custom class loader that is used to reload the classes in `/WEB-INF/classes` and `/WEB-INF/lib` if required. Chapter 9 discusses this topic in more detail.

> **The current version of Tomcat (4.1.3 Beta) supports reloading only if a web application has been installed from an unpacked directory. It does not support reloading if the web application has been installed from a WAR file.**

The screenshot below shows the error message we get when we try to reload the `hello` application that we had installed earlier from a WAR file:



> **There seems to be a bug with Tomcat 4.1.8 (Beta) with the `reload` command. However, it is not a serious one. The message displayed on the browser doesn't show the `context_path` (`/hello` in this case) as in the above screenshot. All other commands give the correct messages.**

The workaround with a WAR file is to either restart Tomcat or `remove` and then `install` the application again.

A successful execution of the `reload` command returns an "OK – Reloaded application at context path {context_path}" message, where {context_path} is the context path for the application.

## *List Available JNDI Resources (Tomcat 4.1 Only)*

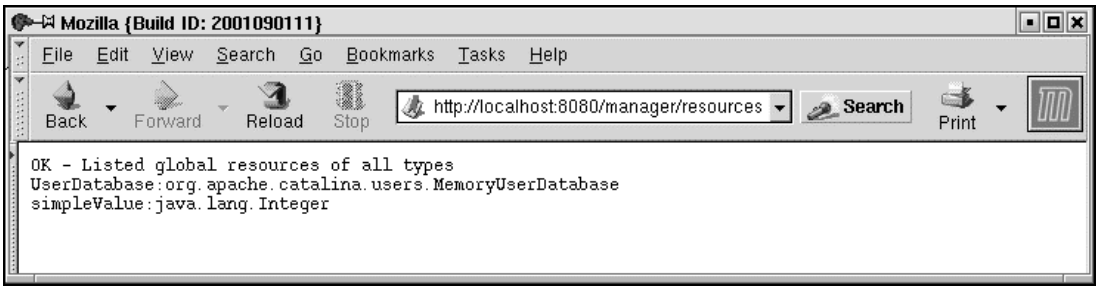
The general format of the URL for listing available JNDI resources is:

```
http://{hostname}:{portnumber}/manager/resources[?type={jndi_type}]
```

In the URL above, the `type` argument is optional. When it is not specified, all the available JNDI resources are listed. Otherwise, JNDI resources corresponding to the specified type alone are listed. The `type` field needs to be a fully qualified Java class name. For example, for JDBC data sources, the type needs to be specified as `javax.sql.DataSource`:

http://localhost:8080/manager/resources?type=javax.sql.DataSource

The response to this contains a success string ("OK – Listed global resources of all types" or "OK – Listed global resources of type {jndi_type}") followed by information about the resources – one per line. Each line contains the global resource name and the global resource type. The global resource name is the name of the JNDI resource as specified in the `global` attribute of the `<ResourceLink>` element in Tomcat's configuration. The global resource type is the fully qualified Java class name of this JNDI resource:

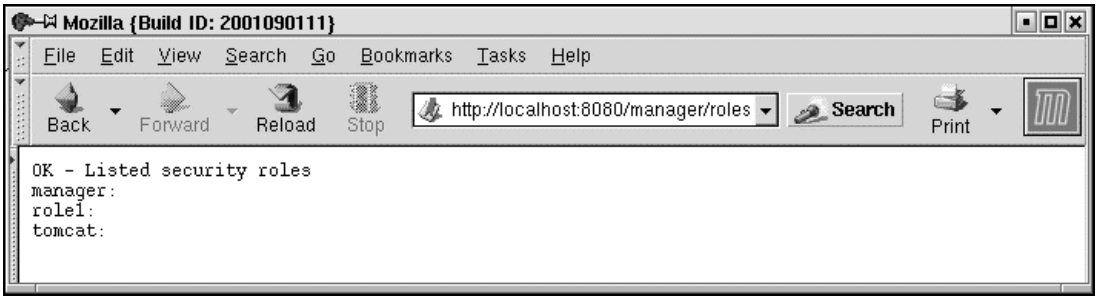



## *List the Available Security Roles (Tomcat 4.1 Only)*

The URL for listing all security role names is:

```
http://{hostname}:{portnumber}/manager/roles
```

On successful execution, the output of this command is an "OK – Listed security roles" message, followed by the security role name and a (optional) description. There is one security role listed per line, and the fields are ":" separated as before:



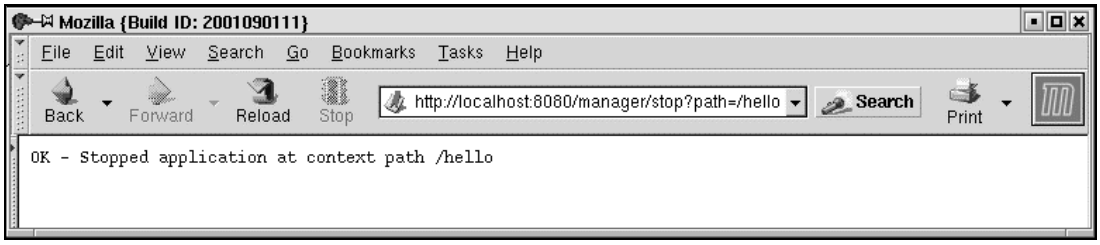

The security roles listed by this command are those that are defined in the user database. The `manager` application's configuration defines the user database resource that should be looked up for the roles in its `<ResourceLink>` section.

## Stop an Existing Application

We can use the `manager` application to stop a running application. The URL below shows how this can be done:

```
http://{hostname}:{portnumber}/manager/stop?path={context_path}
```

This command sends a signal to the web application to stop. This application is no longer available to users, though it still remains deployed. If we run the `list` command again, the state of the application would be shown as "stopped":




If the application stops successfully, the message "OK – Stopped application at context path {context_path}" is displayed. If the operation fails, a FAIL message with appropriate error information is shown.

The application can be restarted using the `start` command that is shown next.

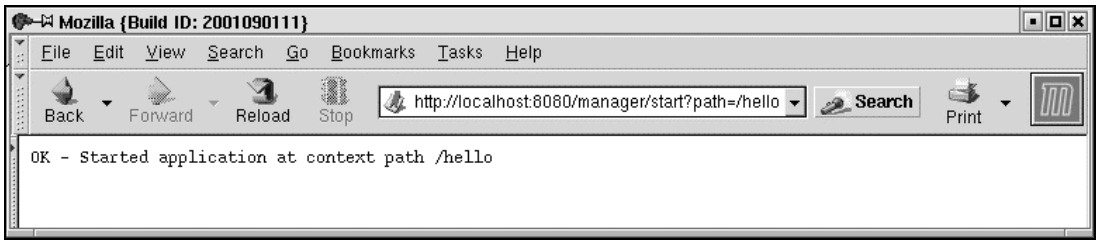
## Start a Stopped Application

We can use the `manager` application to start a stopped application. The URL below shows how this can be done:

```
http://{hostname}:{portnumber}/manager/start?path={context_path}
```

Here `{context_path}` is the context path for the web application (empty string for the `ROOT` application).

If the application starts successfully, the message "OK – Started application at context path {context_path}" is displayed:




If the operation fails, a FAIL message with appropriate error information is shown.

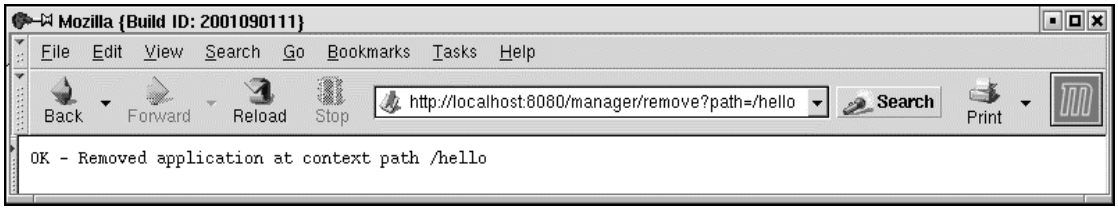
## Remove an Installed Application

The format of the `remove` command URL is listed below:

```
http://{hostname}:{portnumber}/manager/remove?path={context_path}
```

This command is the opposite of the `install` command – it signals the web application to shut down gracefully, and then makes the application context available for reuse. This is done by removing the context entry from Tomcat's runtime data structures.

We had seen earlier that the Tomcat 4.0 `install` command behaves like a `deploy` command, as it copies the web application over to `$CATALINA_HOME/webapps`. Does the `remove` command undo this and remove the web application directory and/or the WAR file? It should, but due to a bug in Tomcat, it does not. We need to manually remove the extracted web application directory:




The screenshot above shows the web application running at context path `/hello` being removed. Its context entry is deleted from Tomcat's internal runtime data structures, hence any attempt to access http://localhost:8080/hello will now fail.

If the application is removed successfully, the message "OK – Removed application at context path {context_path}" is displayed.

## Undeploy a Web Application (Tomcat 4.1 Only)

This command should be used with care – it deletes the web application directory that was created when the application was deployed. If we do not want the web application to be removed permanently, but only removed for the current Tomcat lifetime, we would use the `remove` command.

This command first signals the application to shut down (if it is still running) and then deletes the web application directory and the application WAR file. It then removes the `<Context>` entry for the web application from `$CATALINA_HOME/conf/server.xml`. Tomcat 4.1.3 Beta has a bug due to which the `<Context>` entry does not get removed – users need to manually edit `server.xml`.

In short, the `undeploy` command does the opposite of the `deploy` command that we described earlier in the chapter. However, the `undeploy` command works only on applications installed in the application base directory of the virtual host (the location where the `deploy` command put the web application WAR files) and so cannot be used for applications that were not deployed using the `deploy` command.

The URL for the `undeploy` command is listed below:

```
http://localhost:8080/manager/undeploy?path={context_path}
```

If the application undeploys successfully, the message "OK – Undeployed application at context path {context_path}" is displayed. If the operation fails, a "FAIL" message with appropriate error information is shown.

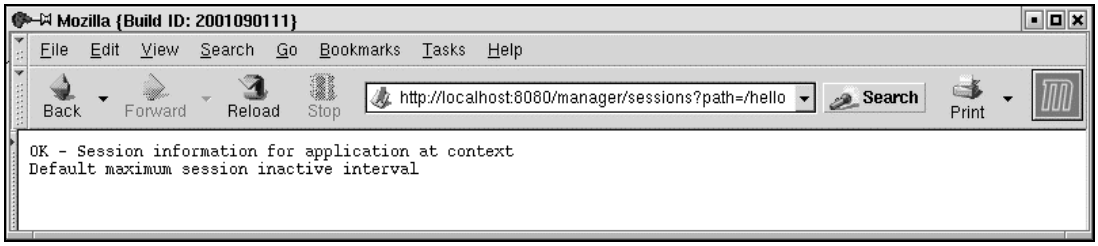
### *Display Session Statistics*

We can use the `manager` application to get statistics about a particular web application. The statistics shown are the default session timeout and the number of current active sessions.

The URL for accessing this information is:

```
http://localhost:8080/manager/sessions?path={context_path}
```

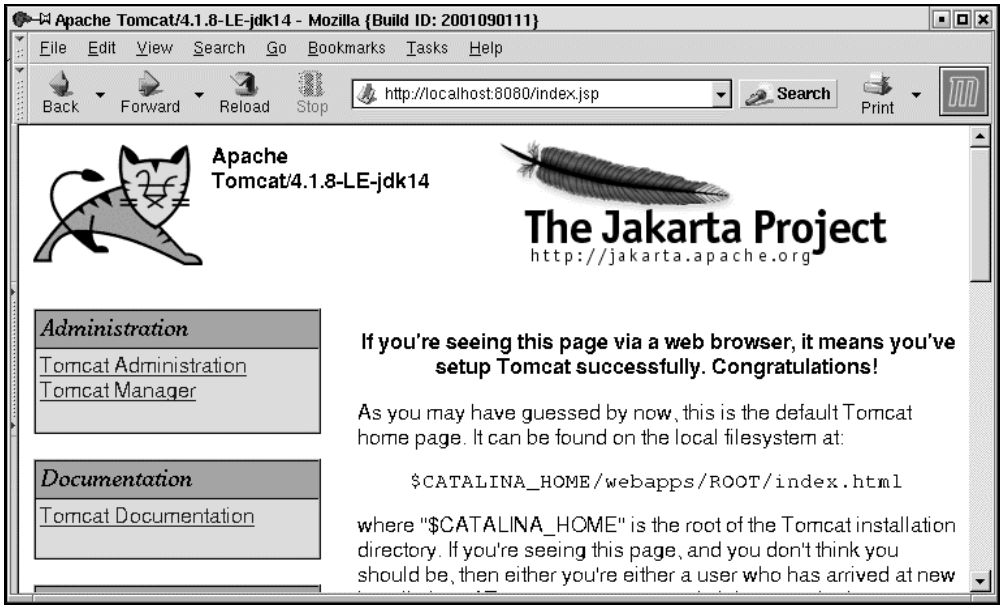
For example, we can check the statistics for the `hello` application using the command http://localhost:8080/manager/sessions?path=/hello:



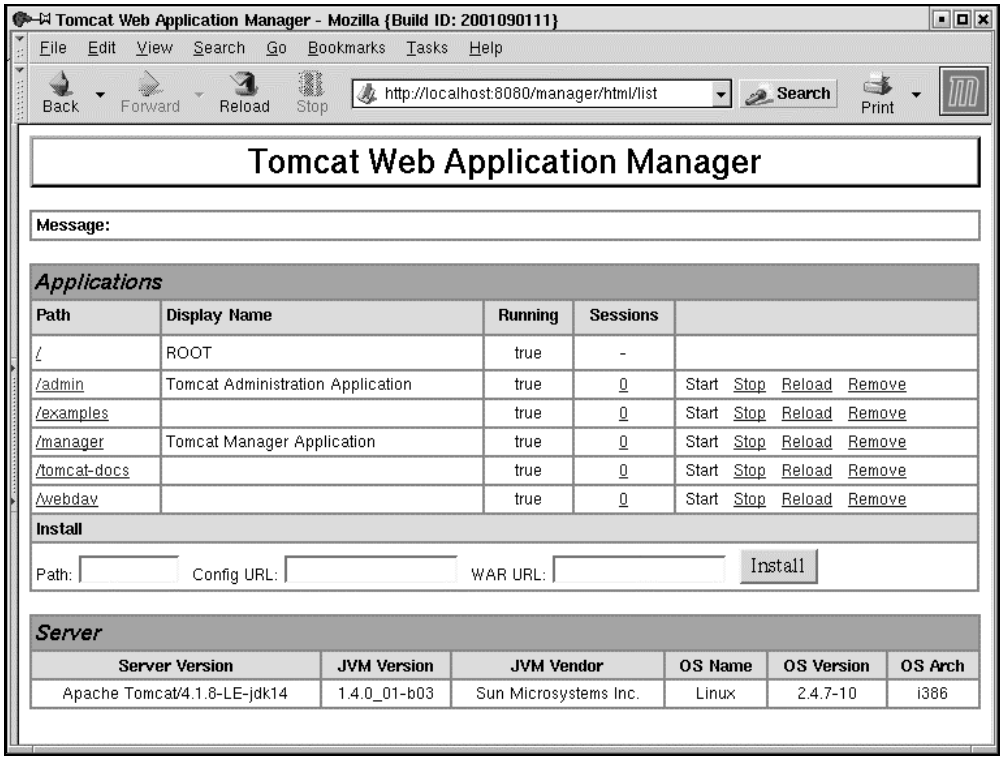

## Tomcat Web Application Manager (4.1.7 Beta Only)

As we discussed earlier, the latest version of Tomcat (4.1.7 Beta) introduces a new web interface for the `manager` application. This interface allows us to start, stop, remove, reload, and install web applications without having to type the command URL. It does not allow for the entire `manager` application task though – we cannot deploy/undeploy or view security roles or JNDI resources.

To access the web application manager access http://localhost:8080/ and click on the Tomcat Manager link on the left-hand side on the Tomcat home page as shown in the screenshot opposite:




You will then be prompted for a username and password. This will then lead you to the `manager` application home page as shown in the screenshot below:



We can then start, stop, reload, and remove web applications by clicking on the relevant links provided at the end of each application.

In Tomcat 4.0 and 4.1.3, the manager tasks were handled by a servlet called the `ManagerServlet` (`org.apache.catalina.servlets.ManagerServlet`). In Tomcat 4.1.7, the new interface is `HTMLManagerServlet`. This class extends the `ManagerServlet` and internally invokes the same commands (`install`, `uninstall`, and so on) that we discussed earlier in the chapter.

The Applications table has five columns:

- **Path**
  This lists the web application path. The path name links to the URL for the web application.
- **Display Name**
  This is picked up from the `<display-name>` element in the application's deployment descriptor (`web.xml`).
- **Running**
  The running status for the application – `true` if the application is running and `false` otherwise.
- **Sessions**
  The number of active session for the web application. Clicking on the link for the number of sessions, we get to the session statistics for that particular web application. This internally invokes the `sessions` command that we saw earlier (repeated below).

```
http://{hostname}:{portnumber}/manager/sessions?path={context_path}
```

- Finally, we have the links to the `start`, `stop`, `reload`, and `remove` commands for the web application. We saw these commands earlier, but using the web application manager saves us from the effort of typing a command URL for performing these tasks.

## Installing a Web Application

We can also install a new web application using the manager tool. In this section, we will install the `hello` web application. Enter the context path for the web application; in this case it will be `/hello`. This allows us to access it at the URL http://<host>:<port>/hello.

Next, the config URL – this is an optional field that allows us to specify the URL to a context file. The context file contains a `<Context>` element entry for the web application. This way we can have the web application context configured with attributes (such as `reloadable`, `privileged`, and so on) that are different from the defaults assigned during an install.

> **Due to a bug in the web application manager, the config URL gets passed as an empty string to the `install` command (instead of being passed as a `null` string). This causes the manager web application to report an error. We therefore have to specify the config URL.**

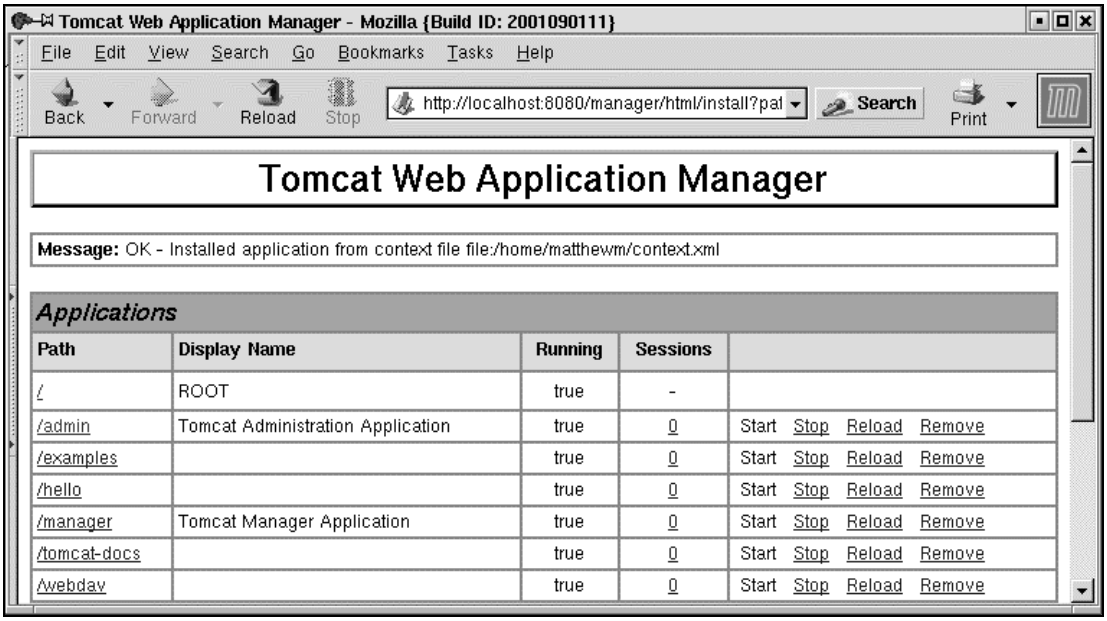
A sample config context file could have the following contents:

```
<Context path="/hello" docBase="/path/to/hello" debug="0"
         reloadable="true" crossContext="true">
</Context>
```

Save this in a file (say `context.xml`) and specify the URL to this file in the context URL field (for example, file:/path/to/context.xml on Unix or file:/c:\path\to\context.xml on Windows). The `docBase` attribute in the `<Context>` element is optional – if we specify one, it overrides the web application WAR file URL (the next attribute).

Finally, a URL to the WAR file or an extracted web application directory should be given. The format for this is the same as discussed in the *Manager Application Commands* section. For example, we could specify this as file://home/wrox/hello.war.

After successfully executing the `install` command by clicking on the Install button, Tomcat adds another row for our application in the list of applications installed/deployed:




## Possible Errors

There are a number of things that could go wrong while working with the `manager` application. The possible causes of failure are listed below:

- **Application already exists at path {context_path}**
  A web application already exists at the path specified. The context path for each web application must be unique. We would get this error if there is another application with the same context path – this can be the same application (that is, we tried to deploy twice) or a different one. To fix this, we have to either `undeploy`/`remove` the previous application, or choose a different context path.

- **Encountered exception**
  An exception occurred while trying to start the web application. The Tomcat log files will have error messages relating to the specific error. Typical causes of error are missing classes/JAR files while loading the application and invalid commands in the application's `web.xml` file.

- **Invalid context path specified**
  The context path must start with a '/'. The exception to this is when the ROOT web application (that is, at context path '/' itself) is being deployed, in which case the context path must be a zero-length string.
- **No context path specified**
  The context path is mandatory.
- **Document base does not exist or is not a readable directory**
  The value specified for the WAR file path/URL in the war parameter is incorrect. This parameter must point to an expanded web application or an actual WAR file.
- **No context exists for path {context_path}**
  The context path is invalid – there is no web application deployed corresponding to it.
- **Reload not supported on WAR deployed at path {context_path}**
  The web application had been installed from a WAR file, instead of from an unpacked directory. The current version of Tomcat does not support this.
- **No global JNDI resources**
  No JNDI global resources were configured for this Tomcat instance.
- **Cannot resolve user database reference**
  There was an error looking up the appropriate user database. For example, in the case of the roles stored in a JNDI realm, a JNDI error would result in such a message. Tomcat's log files would have more error information.
- **No user database is available**
  The <ResourceLink> element has not been configured properly in the manager.xml configuration file. See the *Manager Application Configuration* section above for more information.

## Managing Applications with Ant (Tomcat 4.1 Only)

Tomcat 4.1 allows for these administration commands to be run from an Ant script (Ant installation is covered in Chapter 3 with extra information in Chapter 17). This is convenient for development purposes as the Ant build file could be used to compile, deploy, and even start a web application. The steps for doing this once Ant is installed are:

- Copy the $CATALINA_HOME/server/lib/catalina-ant.jar file into Ant's library directory ($ANT_HOME/lib). This JAR file contains the Tomcat management task definitions for Ant.
- Add $ANT_HOME/bin to your PATH.
- Add a user with the manager role to Tomcat's user database if such a user does not exist.
- Now add <taskdef> elements to your custom build.xml script that call the Tomcat manager commands.

A sample build.xml is shown opposite. This builds and deploys the hello web application that we discussed at the beginning of the chapter. Here is the build.xml script:

```
<project name="HelloApplication" default="compile" basedir=".">

  <!-- Configure the directory into which the web application is built -->
  <property name="src" value="."/>
  <property name="build"    value="${basedir}/build"/>

  <!-- Configure the context path for this application -->
  <property name="path"     value="/hello"/>
```

The `<project>` tag has attributes for the name of the project and the default target. The default target in this case is called `compile`. Running Ant with no options will invoke the tasks associated with this default target. The `basedir` attribute is the base directory for all path calculations in the Ant build script. This is set to "." (the current directory) and therefore all the paths for the build process are taken to be relative to the directory we run `ant` from. We then define properties for the build, such as the location of the source directory and the target directory where the compiled `.class` files will go. The properties below specify the access URL and username/password for the `manager` application. We will later see how we can pass the password from the command line too:

```
  <!-- Configure properties to access the Manager application -->
  <property name="url"      value="http://localhost:8080/manager"/>
  <property name="username" value="myusername"/>
  <property name="password" value="mypassword"/>
```

The task definitions for the `manager` application are now specified. Ant allows for custom tasks that extend its functionality. Tomcat implements the custom tasks shown below for executing the `manager` application commands. For example, `org.apache.catalina.ant.DeployTask` executes the `deploy` command against the `manager` application:

```
  <!-- Configure the custom Ant tasks for the Manager application -->
  <taskdef name="deploy"
           classname="org.apache.catalina.ant.DeployTask"/>
  <taskdef name="install"
           classname="org.apache.catalina.ant.InstallTask"/>
  <taskdef name="list"
           classname="org.apache.catalina.ant.ListTask"/>
  <taskdef name="reload"
           classname="org.apache.catalina.ant.ReloadTask"/>
  <taskdef name="remove"
           classname="org.apache.catalina.ant.RemoveTask"/>
  <taskdef name="resources"
           classname="org.apache.catalina.ant.ResourcesTask"/>
  <taskdef name="roles"
           classname="org.apache.catalina.ant.RolesTask"/>
  <taskdef name="start"
           classname="org.apache.catalina.ant.StartTask"/>
  <taskdef name="stop"
           classname="org.apache.catalina.ant.StopTask"/>
  <taskdef name="undeploy"
           classname="org.apache.catalina.ant.UndeployTask"/>
```

Next is the Ant target that does initializations – in this case, create the build directory:

```
<target name="init">
  <!-- Create the build directory structure used by compile -->
  <mkdir dir="${build}"/>
  <mkdir dir="${build}/hello"/>
  <mkdir dir="${build}/hello/Web-INF"/>
  <mkdir dir="${build}/hello/Web-INF/classes"/>
</target>
```

The default `compile` target is shown below. This has Ant instructions to compile all the Java files into class files. Our `hello` application doesn't have any class files, so nothing will be done, but any serious web application will contain Java files. Notice how the `compile` task depends on the `init` task – this ensures that the initializations steps are performed before Ant compiles the Java files:

```
<!-- Executable Targets -->
<target name="compile" description="Compile web application"
        depends="init">
  <javac srcdir="${src}" destdir="${build}"/>
</target>
```

The `build` target builds the application WAR file. It has instructions to move the files to the correct directory format for a web application and build the WAR file:

```
<target name="build" description="Build web application"
        depends="compile">
  <copy file="index.html" toDir="${build}/hello"/>
  <copy file="Hello.jsp" toDir="${build}/hello"/>
  <jar destfile="${build}/hello.war" basedir="${build}/hello"/>
</target>
```

Finally, we have the manager tasks for listing all web applications, and installing/uninstalling and deploying/undeploying web applications:

```
<target name="list" description="List all web applications">
  <list url="${url}" username="${username}" password="${password}"/>
</target>

<target name="install" description="Install web application"
        depends="build">
  <install url="${url}" username="${username}" password="${password}"
           path="${path}" war="file:${build}/hello"/>
</target>

<target name="reload" description="Reload web application"
        depends="build">
  <reload  url="${url}" username="${username}" password="${password}"
           path="${path}"/>
</target>

<target name="remove" description="Remove web application">
  <remove url="${url}" username="${username}" password="${password}"
           path="${path}"/>
</target>

<target name="deploy" description="Deploy web application"
        depends="build">
  <deploy url="${url}" username="${username}" password="${password}"
```

```
            path="${path}" war="file:${build}/hello.war"/>
  </target>

  <target name="undeploy" description="Undeploy web application">
    <undeploy url="${url}" username="${username}" password="${password}"
            path="${path}"/>
  </target>

</project>
```

Before using the Ant script, we must add the `$CATALINA_HOME/server/lib/catalina-ant.jar` to the classpath and the Ant install directory to our system path (we used Ant 1.5 for our testing):

```
$ CLASSPATH=$CLASSPATH:$CATALINA_HOME/server/lib/catalina-ant.jar
$ PATH=$PATH:/path/to/ant1.5/bin
$ export CLASSPATH PATH
```

The `password` property in the Ant script contains the password for the user with manager privileges. This is useful for development environments where we don't want to specify the password each time we build and deploy.

This value can be overridden from the command line, or even omitted from the build file altogether and passed only from the command line. This avoids the security risk of putting the password in a cleartext file:

```
$ ant -Dpassword=secret list
```

The ability to run the manager commands from within Ant files allows for a very integrated develop-deploy-test cycle for web application development. For instance, after developing the HTML pages, servlets, JSP pages, and other Java classes for the web application, the developer would need to compile all the Java code:

```
$ ant build
```

The `build` target in our `build.xml` file compiles all the Java code and puts the class files into the appropriate location (the `/WEB-INF/classes` directory). It then builds the deployable JAR file. Developers may need to fix compilation errors, if any, and then rerun the `ant` command.

Next, they can use the `install` target to install the web application in the Tomcat instance specified in the Ant build file. The `install` target uses the Tomcat `manager` application's `install` command to install the web application:

```
$ ant install
```

This installed application can then be tested, and errors ironed out. During each iteration, developers would fix bugs, recompile, and then restart the web application:

```
$ ant compile reload
```

Once the application is stable, the developers can remove it from the Tomcat installation:

```
$ ant remove
```

This web application can then be packaged as a WAR file and distributed to end users. They can then deploy the application in the production Tomcat installation:

```
$ ant deploy
```

The `reload`, `remove`, and `deploy` Ant targets shown above invoke the `reload`, `remove`, and `deploy` commands of the Tomcat application manager.

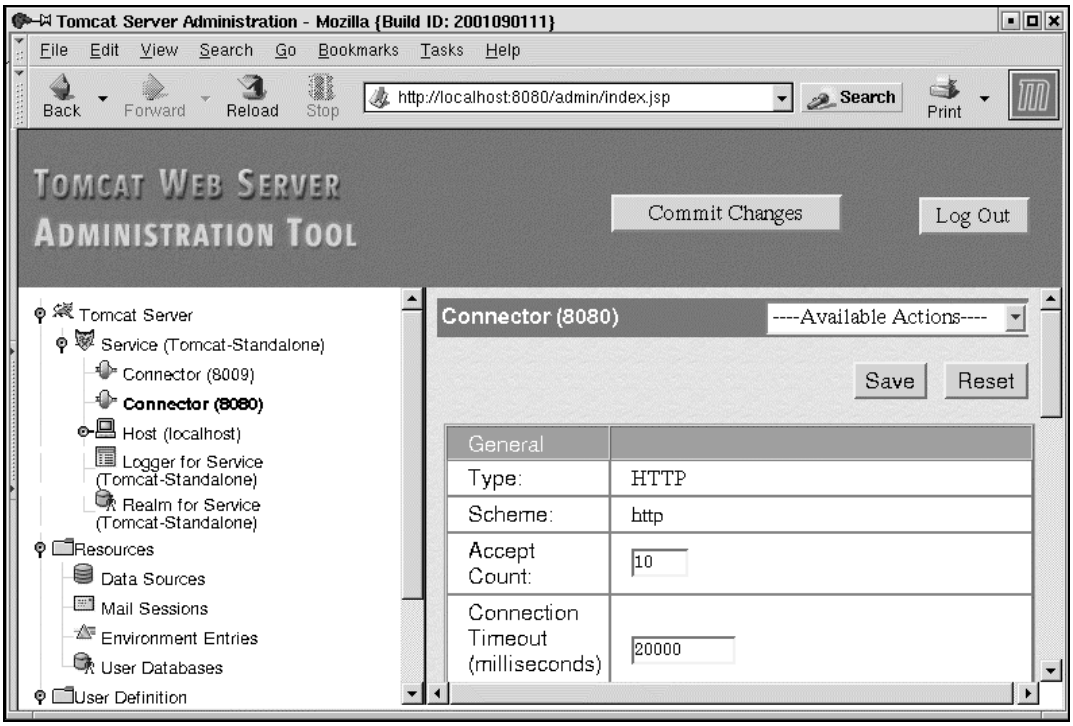
# Tomcat Administration Tool (Tomcat 4.1 Only)

Tomcat 4.1 introduces a new web-based administration tool. This allows administrators to visually edit the configuration parameters of Tomcat 4.1. These parameters are defined in Tomcat's configuration files (`server.xml`, `tomcat-users.xml`, and so on). Administrators of earlier versions had to manually edit the XML configuration files – a process that was error-prone and painful. For a more detailed description of these parameters and the configuration files, please see Chapter 5.

The `admin` tool can be accessed via the following URL:

**`http://{hostname}:{portname}/admin/index.jsp`**

As before, access to this application is restricted to users with `admin` roles. The screenshot below shows the administration tool:



## Admin Application Configuration

Access to the Tomcat 4.1 `admin` application is restricted in a manner similar to the `manager` application. The following is the memory realm user database file (`$CATLINA_HOME/conf/tomcat-users.xml`) that defines the `admin` role and sets the username and password:

```
<tomcat-users>
  <role rolename="admin"/>
  <role rolename="manager"/>

  ...
  <user username="admin" password="admin" roles="admin,manager"/>

</tomcat-users>
```

The configuration for the `admin` application is specified in the `$CATALINA_HOME/webapps/admin.xml` configuration file. The default configuration is shown below:

```
<Context path="/admin" docBase="../server/webapps/admin"
         debug="0" privileged="true">

  <Logger className="org.apache.catalina.logger.FileLogger"
          prefix="localhost_admin_log." suffix=".txt"
          timestamp="true"/>

</Context>
```

The other configuration file for the `admin` application is the deployment descriptor (`$CATALINA_HOME/server/webapps/admin/WEB-INF/web.xml`). This defines, among other things, the security constraints for the `admin` application and also the authentication mechanism for `admin` users.

To illustrate, the following extract from the `web.xml` file shows that only users with the `admin` role can access the `admin` application:

```
<security-constraint>
    <display-name>Tomcat Server Configuration Security Constraint</display-name>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <!-- Define the context-relative URL(s) to be protected -->
      <url-pattern>*.jsp</url-pattern>
      <url-pattern>*.do</url-pattern>
      <url-pattern>*.html</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <!-- Anyone with one of the listed roles may access this area -->
      <role-name>admin</role-name>
    </auth-constraint>
  </security-constraint>
```

# The Future

The `manager` application documentation states that there are plans to add a web service-based interface for Tomcat `manager` applications. Once this is done, the management tasks can be easily integrated with third-party applications, or triggered from non-Java/non-web-based client programs. However, the proposed feature list for Tomcat 5.0 (the next release) does not include this.

The proposed 5.0 features do mention adding JMX (Java Management Extensions) support to the Coyote connector. The Coyote connector provides both a native HTTP stack for Tomcat (the Coyote HTTP/1.1 connector) as well as allowing Apache and other web servers to tie in to Tomcat (the Coyote JK2 connector). JMX is a new Java technology from Sun for managing and monitoring applications. However, it is still unclear what the JMX support in Tomcat 5.0 will look like.

# Summary

In this chapter we saw the administration capabilities of the Tomcat 4.x `manager` application and the Tomcat 3.x admin commands.

Securing access to the `manager` application is important. Someone who gains unauthorized access to it can deploy malicious applications, or cause a Denial of Service (DoS) by shutting down running applications. Administrators concerned about security could configure the network to disallow access to the `manager` application URL from hosts outside the local network, or, if they are paranoid, even disable the `manager` web application altogether.