

Programmer's Guide to Servlets

iPlanet Web Server, FastTrack Edition

Version 4.1

8xx-xxxx-xx
July 2000

Copyright © 2000 Sun Microsystems, Inc. Some preexisting portions Copyright © 2000 Netscape Communications Corporation. All rights reserved.

Sun, Sun Microsystems, and the Sun logo, iPlanet, and the iPlanet logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Netscape and the Netscape N logo are registered trademarks of Netscape Communications Corporation in the U.S. and other countries. Other Netscape logos, product names, and service names are also trademarks of Netscape Communications Corporation, which may be registered in other countries.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions

The product described in this document is distributed under licenses restricting its use, copying, distribution, and decompilation. No part of the product or this document may be reproduced in any form by any means without prior written authorization of the Sun-Netscape Alliance and its licensors, if any.

THIS DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2000 Sun Microsystems, Inc. Pour certaines parties préexistantes, Copyright © 2000 Netscape Communication Corp. Tous droits réservés.

Sun, Sun Microsystems, et the Sun logo, iPlanet, and the iPlanet logo sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et d'autre pays. Netscape et the Netscape N logo sont des marques déposées de Netscape Communications Corporation aux Etats-Unis et d'autre pays. Les autres logos, les noms de produit, et les noms de service de Netscape sont des marques déposées de Netscape Communications Corporation dans certains autres pays.

Le produit décrit dans ce document est distribué selon des conditions de licence qui en restreignent l'utilisation, la copie, la distribution et la décompilation. Aucune partie de ce produit ni de ce document ne peut être reproduite sous quelque forme ou par quelque moyen que ce soit sans l'autorisation écrite préalable de l'Alliance Sun-Netscape et, le cas échéant, de ses bailleurs de licence.

CETTE DOCUMENTATION EST FOURNIE “EN L'ÉTAT”, ET TOUTES CONDITIONS EXPRESSES OU IMPLICITES, TOUTES REPRÉSENTATIONS ET TOUTES GARANTIES, Y COMPRIS TOUTE GARANTIE IMPLICITE D'APTITUDE À LA VENTE, OU À UN BUT PARTICULIER OU DE NON CONTREFAÇON SONT EXCLUES, EXCEPTÉ DANS LA MESURE OÙ DE TELLES EXCLUSIONS SERAIENT CONTRAIRES À LA LOI.

Contents

About This Book	7
iPlanet Web Server, FastTrack Edition Overview	7
Book Summary	7
Chapter 1 Using Servlets and JavaServer Pages	9
Servlets	10
JavaServer Pages	11
What Does the Server Need to Run Servlets and JSP?	12
Serving Servlets and JSP	14
Using the Server Manager Interface	15
Activating Servlets and JSP	15
Configuring Global Servlet Attributes	16
Registering Servlet Directories	17
Registering Individual Servlets	18
Specifying Servlet Virtual Paths	20
Specifying Servlet Contexts	23
Servlet Output	24
Configuring JRE/JDK Paths	24
Deleting Version Files	26
Configuring JVM	27
Running 0.92 JSP	28
Maximizing Servlet Performance	28
Chapter 2 Servlet and JSP Examples	31
Examples Shipped with iPlanet Web Server 4.1	31
Servlet Examples	32
A Simple Servlet Example	32
Example of a Servlet that Counts Visits	34
JSP Examples	36
JSP that Accesses the Request Object	37

JSP that Responds to a Form and Uses Java Beans	38
Creating a JSP Custom Tag Library	42
Appendix A Session Managers	53
Session Overview	53
Specifying a Session Manager	54
SimpleSessionManager	55
Parameters	55
Enabling SimpleSessionManager	56
Source Code for SimpleSessionManager	56
JdbcSessionManager	57
Parameters	57
Enabling JdbcSessionManager	58
Source Code for JDBCSessionManager	59
How Do Servlets Access Session Data?	60
Appendix B Servlet Settings in obj.conf	61
Directives for Enabling Servlets	61
Directives for Registered Servlet Directories	63
JSP	64
Appendix C Properties Files	65
servlets.properties	65
rules.properties	66
Using Regular Expressions in rules.properties	67
contexts.properties	68
isModifiedCheckAggressive	70
parameterEncoding	70
sessionExpireOnClose	71
singleClassLoader	71
Appendix D JVM Configuration	73
Appendix E Debugging Servlets and JSPs	75
Servlet Debugging	75
JSP Debugging	76
Appendix F Remote Servlet Profiling	79
Appendix G API Clarifications	81
Clarifications for Using Methods	81

HttpUtils.getRequestURL	82
HttpServletRequest.getContextPath	82
HttpServletRequest.getUserPrincipal	83
HttpServletRequest.isUserInRole	83
HttpSession.setMaxInactiveInterval	83
GenericServlet.getInitParameter and getInitParameterNames	84
ServletContext.getAttributeNames	84
ServletContext.getContext	85
ServletContext.getRealPath	85
ServletContext.getResourceAsStream	85
ServletRequest.getAttribute	86
ServletRequest.getParameter	86
RequestDispatcher.forward and include	87
Request.getInputStream and getReader	88
Other Useful Information	88
Database Connection Pooling	89
Fetching the Client Certificate	89
Index	91

About This Book

This book discusses how to enable and install Java servlets and JavaServer Pages (JSP) in iPlanet™ Web Server, FastTrack Edition 4.1.

iPlanet Web Server, FastTrack Edition Overview

iPlanet Web Server, FastTrack Edition was designed for developing and testing Java applications and deploying low traffic web sites. Because it shares a common code base with iPlanet Web Server, Enterprise Edition, applications developed for it can be run on the Enterprise Edition without changes.

Enterprise Edition features that are absent or limited in the FastTrack Edition and relevant to this manual are as follows:

- Only five concurrent threads are supported.
- Only five virtual servers are supported (Enterprise Edition supports 256).
- Multiprocess mode has been disabled. Therefore, servlets and JSPs do not run in multiprocess mode.

For a full description of the iPlanet Web Server, Enterprise Edition, see:

http://www.iplanet.com/products/infrastructure/web_servers/index.html

Book Summary

This book has the following chapters and appendices:

- Chapter 1, “Using Servlets and JavaServer Pages”

This chapter discusses how to enable and install servlets and JSPs in iPlanet Web Server 4.1. It explains how to specify settings for servlets and for the JRE and JDK by using the Server Manager interface or by editing configuration files.

- Chapter 2, “Servlet and JSP Examples”

This chapter discusses example servlets and JSP.

- Appendix A, “Session Managers”

This appendix discusses the session managers provided with iPlanet Web Server and gives an overview of a sample session manager that you can extend to customize session behavior to suit your own needs.

- Appendix B, “Servlet Settings in obj.conf”

This appendix discusses how the configuration file `obj.conf` changes depending on the settings for servlets and JSP.

- Appendix C, “Properties Files”

This appendix discusses the `servlets.properties` file, which contains configuration information for servlets, the `rules.properties` file, which defines virtual paths for servlets, and the `contexts.properties` file, which defines contexts for servlets.

- Appendix D, “JVM Configuration”

This appendix discusses how to manually specify JVM configuration information.

- Appendix E, “Debugging Servlets and JSPs”

This appendix discusses how to debug servlets and JSPs.

- Appendix F, “Remote Servlet Profiling”

This appendix discusses how to enable remote profiling for servlets.

- Appendix G, “API Clarifications”

This chapter discusses methods in the Servlets API that behave marginally differently in iPlanet Web Server than specified in the Sun Microsystems’ Servlets API documentation or where the behavior documented by Sun Microsystems is ambiguous.

NOTE

Throughout this manual, all Unix-specific descriptions apply to the Linux operating system as well, except where Linux is specifically mentioned.

Using Servlets and JavaServer Pages

iPlanet Web Server 4.1 supports servlets and JavaServer Pages (JSPs). This chapter gives a brief overview of servlets and JSPs and discusses how to enable and configure them in iPlanet Web Server 4.1.

The sections in this chapter are:

- Servlets
- JavaServer Pages
- What Does the Server Need to Run Servlets and JSP?
- Serving Servlets and JSP
- Using the Server Manager Interface
- Activating Servlets and JSP
- Configuring Global Servlet Attributes
- Registering Servlet Directories
- Registering Individual Servlets
- Specifying Servlet Virtual Paths
- Specifying Servlet Contexts
- Servlet Output
- Configuring JRE / JDK Paths
- Deleting Version Files
- Configuring JVM

- Running 0.92 JSP
- Maximizing Servlet Performance

Servlets

Java servlets are server-side Java programs that web servers can run to generate content in response to a client request in much the same way as CGI programs do. Servlets can be thought of as applets that run on the server side without a user interface. Servlets are invoked through URL invocation.

iPlanet Web Server 4.1 includes support for JavaSoft's Servlet API at the level of the 2.2.1 specification (except for Web Application and WAR file support).

NOTE	Servlet API version 2.2.1 is fully backward compatible with version 2.1, so all existing servlets will continue to work without modification or recompilation.
-------------	--

To develop servlets, use Sun Microsystems' Java Servlet API. For information about using the Java Servlet API, see the documentation provided by Sun Microsystems at:

<http://java.sun.com/products/servlet/index.html>

iPlanet Web Server 4.1 includes all the files necessary for developing Java Servlets. The `servlets.jar` file is in the iPlanet Web Server 4.1 installation directory at:

server_root/bin/https/jar

When compiling servlets, make sure the `servlets.jar` file is accessible to your Java compiler. Include the `servlets.jar` file in your CLASSPATH.

iPlanet Web Server 4.1 supports the `<SERVLET>` tag as introduced by Java Web Server. This tag allows you to embed servlet output in an HTML file. No configuration changes are necessary to enable this behavior. If SSI and servlets are both enabled, the `<SERVLET>` tag is enabled.

The `<SERVLET>` tag syntax is slightly different from that of other SSI commands; it resembles the `<APPLET>` tag syntax:

```
<servlet name=name code=classfile codebase=path iParam1=v1  
iParam2=v2>  
<param name=param1 value=v3>  
<param name=param2 value=v4>  
.  
.  
</servlet>
```

The code parameter, which specifies the .class file for the servlet, is always required. The .class extension is optional. The codebase parameter is required if the servlet is *not* defined in the `servlets.properties` file and the .class file is *not* in the same directory as the HTML file containing the `<SERVLET>` tag. The name parameter is required if the servlet is defined in the `servlets.properties` file, and must match the servlet name defined in that file.

For more information about the `servlets.properties` file, see Appendix C, “Properties Files.” For more information about SSI commands, see the *Programmer’s Guide for iPlanet Web Server*.

JavaServer Pages

iPlanet Web Server 4.1 supports JavaServer Pages (JSP) to the level of JSP API 1.1 compliance (except for Web Application and WAR file support).

NOTE	JSP API version 1.x is not backward compatible with JSP API version 0.92. To run version 0.92 JSPs, you must create legacy directories for them as described in the section “Running 0.92 JSP,” on page 28.
-------------	---

A JSP is a page, much like an HTML page, that can be viewed in a web browser. However, as well as containing HTML tags, it can include a set of JSP tags that extend the ability of the web page designer to incorporate dynamic content in a page. These tags provide functionality such as displaying property values and using simple conditionals.

One of the main benefits of JSPs is that, like HTML pages, they do not need to be compiled. The web page designer simply writes a page that uses HTML and JSP tags and puts it on their web server. The web page designer does not need to learn how to define Java classes or use Java compilers.

JSP pages can access full Java functionality in the following ways:

- by embedding Java code directly in scriptlets in the page
- by accessing Java beans
- by using server-side tags that include Java servlets

Both beans and servlets are Java classes that need to be compiled, but they can be defined and compiled by a Java programmer, who then publishes the interface to the bean or the servlet. The web page designer can access a pre-compiled bean or servlet from a JSP page.

For information about creating JSPs, see Sun Microsystem's JavaServer Pages web site at:

<http://java.sun.com/products/jsp/index.html>

For information about Java Beans, see Sun Microsystem's JavaBeans web page at:

<http://java.sun.com/beans/index.html>

What Does the Server Need to Run Servlets and JSP?

iPlanet Web Server 4.1 includes the Java Runtime Environment (JRE) but not the Java Development Kit (JDK) due to licensing restrictions. The server can run servlets using the JRE, but it needs the JDK to run JSP.

iPlanet Web Server 4.1 requires you to use the following recommended versions of JRE/JDK or later versions, with different platforms requiring different versions, as summarized in Table 1-1.

Table 1-1 Supported JRE/JDK Versions by Platform

Platform	JRE/JDK Version
Solaris Sparc	1.2.2_01
Windows NT	1.2.2_01
HPUX	1.2.2_02

Table 1-1 Supported JRE/JDK Versions by Platform

Platform	JRE/JDK Version
AIX	1.2.2_01
Compaq	1.2.2-3
Linux	1.2.2RC4
IRIX	1.2.1

Check the *iPlanet Web Server Installation Guide* and the latest release notes for updates on required JDK versions.

NOTE On Sun Solaris, the JRE included is the JRE 1.2.2 reference implementation from JavaSoft. For better performance, use the latest SunSoft production release of JDK.

JDK 1.2 (and other JDK versions) are available from Sun Microsystems at:

<http://java.sun.com/products/jdk/1.2/>

You can specify the path to the JDK in either of the following ways:

- You can specify the path during the server installation process.

When you install iPlanet Web Server 4.1, one of the dialog boxes in the installation process asks if you want to use a custom Java Development Kit (JDK), and if so, you can specify the path to it.

- You can specify it after the server is installed.

To specify the path to the JDK, switch to the Web Server Administration Server, select the Global Settings tab, and use the Configure JRE/JDK Paths page, as discussed in the section “Configuring JRE/JDK Paths,” on page 24.

Whether you specify the path to the JDK during installation or later, the path is the directory in which you installed the JDK.

Serving Servlets and JSP

iPlanet Web Server 4.1 includes an appropriate version of the Java runtime environment (JRE) for running servlets. For the server to be able to serve JSP, you must specify a path to a Java Development Kit (JDK) as discussed in the section “What Does the Server Need to Run Servlets and JSP?,” on page 12.

For the server to serve servlets and JSP, servlet activation must be enabled. (See the section “Activating Servlets and JSP,” on page 15 for details.)

When servlets are enabled, you have a choice of two ways to make a servlet accessible to clients:

- Put the servlet class file in one of the directories that has been registered with the iPlanet Web Server as a servlet directory. For more information, see “Registering Servlet Directories” on page 17.
- Define a servlet virtual path for the servlet. In this case, the servlet class can be located anywhere in the file system or even reside on a remote machine. For more information, see “Specifying Servlet Virtual Paths” on page 20.

No special steps are needed to enable JSP pages other than making sure that JSP is enabled on the iPlanet Web Server. So long as JSP is enabled, the iPlanet Web Server treats all files with a `.jsp` extension as JSPs. (Do not put JSP files in a registered servlet directory, since the iPlanet Web Server expects all files in a registered servlet directory to be servlets.) An exception is a JSP page written to the 0.92 spec, which must be placed in a legacy directory; see the section “Running 0.92 JSP” on page 28 for details.

NOTE	You cannot run JSPs in aliased directories in iPlanet Web Server. For example, if the document root is <i>server_root/docs</i> , mapping <code>http://foo.com/myjsp</code> to <code>/some/other/dir</code> instead of <i>server_root/docs/myjsp</i> does not work.
-------------	--

In detail, to enable the iPlanet Web Server to serve servlets and JSP pages, do the following steps:

1. Activating Servlets and JSP (this is the only step needed to enable JSP)
2. Configuring Global Servlet Attributes
3. Registering Servlet Directories
4. Registering Individual Servlets if Needed
5. Specifying Servlet Virtual Paths if Desired

6. Configuring JVM if Necessary

Using the Server Manager Interface

For information about using the Server Manager interface to specify settings for servlets, see the following topics in the online help. All these pages are located on the Servlets tab.

- The Enable / Disable Servlets / JSP Page
- The Servlet Directory Page
- The Legacy JSP Directory Page
- The Configure Global Servlet Attributes Page
- The Configure Servlet Attributes Page
- The Configure Servlet Virtual Path Translation Page
- The Configure JVM Attributes Page
- The Delete Version Files Page

In addition, the Configure JRE / JDK Paths page on the Global Settings tab in the Web Server Administration Server allows you to specify paths to the JRE and JDK.

Activating Servlets and JSP

To enable and disable servlets and JSP in iPlanet Web Server 4.1, use the Servlets>Enable / Disable Servlets / JSP page in the Server Manager interface.

You must enable both servlets and JSP to run JSP. Even if servlets are enabled, JSP can still be disabled. However, if you disable servlets, JSP is automatically also disabled. In this case, if you enable servlets later, you will need to re-enable JSP also if desired.

You can also define a thread pool to be used for servlets. For more information about thread pools, see “Maximizing Servlet Performance,” on page 28 and “Adding and Using Thread Pools” in Chapter 7, “Configuring Server Preferences,” in the *iPlanet Web Server Administrator’s Guide*.

NOTE	You can use only five threads at a time in the FastTrack edition of iPlanet Web Server.
-------------	---

To enable servlets programmatically, add the following lines to `obj.conf`. These directives first load the shared library containing the servlet engine, which is in `server_root/bin/https/bin/NSServletPlugin.dll` on Windows NT or `server_root/bin/https/lib/libNSServletPlugin.so` on Unix. Then they initialize the servlet engine.

```
Init fn="load-modules"
shlib="server_root/bin/https/bin/NSServletPlugin.dll"
funcs="NSServletEarlyInit,NSServletLateInit,NSServletNameTrans,
NSServletService" shlib_flags="(global|now)"
Init fn="NSServletEarlyInit" EarlyInit="yes"
Init fn="NSServletLateInit" LateInit="yes"
```

In the default object in `obj.conf`, add the following `NameTrans` directive:

```
NameTrans fn="NSServletNameTrans" name="servlet"
```

By default, regardless of whether servlets are enabled or disabled, the file `obj.conf` contains additional objects with names such as `servlet`, `jsp`, and `ServletByExt`. Do not delete these objects. If you delete them, you can no longer activate servlets through the Server Manager.

A JSP page written to the 0.92 spec must be placed in a legacy directory; see the section “Running 0.92 JSP,” on page 28 for details.

Configuring Global Servlet Attributes

You can specify the following optional servlet attributes:

- **Startup Servlets** -- servlets to be loaded when the iPlanet Web Server starts up.
- **Session Manager** -- the session manager for servlets. For more information about the session manager, see Appendix A, “Session Managers.”
- **Session Manager Args** -- the session manager arguments for the servlet engine. For more information about the session manager, see Appendix A, “Session Managers.”
- **Reload Interval** -- the time period that the server waits before re-loading servlets and JSPs if they have changed on the server. The default value is 5 seconds.

You can set these attributes interactively in the Servlets>Configure Global Servlet Attributes page in the Server Manager interface. Alternatively, you can edit the configuration file `servlets.properties` in the server's config directory.

The following code shows an example of the settings in `servlets.properties`:

```
# General properties:
servlets.startup=hello
servlets.config.reloadInterval=5
servlets.config.docRoot=C:/Netscape/Server4/docs
servlets.sessionmgr=com.netscape.server.http.session.SimpleSessionM
anager
```

Registering Servlet Directories

One of the ways to make a servlet accessible to clients is to put it into a directory that is registered with the iPlanet Web Server as a servlet directory. Servlets in registered servlet directories are dynamically loaded when needed. The server monitors the servlet files and automatically reloads them on the fly as they change.

You can register any number of servlet directories for the iPlanet Web Server. Initially, the iPlanet Web Server has a single servlet directory, which is `server_root/docs/servlet/`.

For example, if the `SimpleServlet.class` servlet is in the `servlet` subdirectory of the server's document root directory (the default servlet directory), you can invoke the servlet by pointing the web browser to:

`http://your_server/servlet/SimpleServlet`

The iPlanet Web Server expects all files in a registered servlet directory to be servlets. The server treats any files in that directory that have the `.class` extension as servlets. The iPlanet Web Server does not correctly serve other files, such as HTML files or JSPs, that reside in that directory.

The server can have multiple servlet directories. You can map servlet directories to virtual directories if desired. For example, you could specify that `http://poppy.my_domain.com/products/` invokes servlets in the directory `server_root/docs/january/products/servlets/`.

To register servlet directories and to specify their URL prefixes, use the Servlets>Servlet Directory page in the interface.

Alternatively, you can register servlet directories by adding appropriate `NameTrans` directives to the default object in the file `obj.conf`, such as:

```
NameTrans fn="pfx2dir" from="/products"  
dir="d:/netscape/server4/docs/january/products/servlets/"  
name="ServletByExt"
```

You can invoke a servlet in a subdirectory of a registered servlet directory if you include a package directive in the servlet code that corresponds to the path from the registered servlet directory. For example, suppose the servlet is in the following location, and that `server_root/docs/servlet/` is a registered servlet directory:

`server_root/docs/servlet/HelloWorld/HelloWorldServlet.class`

Include the following package directive as the first line in the Java source file:

```
package HelloWorld;
```

You can then invoke the servlet by pointing the web browser to:

`http://your_server/servlet/HelloWorld.HelloWorldServlet`

For information about reloading packaged servlets, see “`isModifiedCheckAggressive`,” on page 70.

Registering Individual Servlets

The iPlanet Web Server treats any file in a registered servlet directory as a servlet. There is no need to register individual servlets that reside in these directories unless any of the following criteria apply:

- The servlet takes input parameters that are not passed through the request URL.
- You want to set up additional virtual URLs for the servlet.
- Your servlets are packaged or in a `.jar` file. The server does not search `.class` or `.jar` files for packaged servlets.

If any of these conditions is true, register the individual servlet by using the `Servlets>Configure Servlet Attributes` page in the Server Manager interface. Alternatively, you can edit the file `servlets.properties` to add an entry for the servlet.

When registering an individual servlet, specify the following attributes:

- **Servlet Name** -- The iPlanet Web Server uses this value as a servlet identifier to internally identify the servlet. (This identifier is not part of the URL that is used to invoke the servlet, unless by coincidence the identifier is the same as the class code name.)
- **Servlet Code (class name)** -- the name of the class file. You do not need to specify the `.class` extension.
- **Servlet Classpath** -- This is the absolute pathname or URL to the directory or zip/jar file containing the servlet. The classpath can point anywhere in the file system. The servlet classpath may contain a directory, a `.jar` or `.zip` file, or a URL to a directory. (You cannot specify a URL as a classpath for a zip or jar file.)

If the servlet classpath is not a registered servlet directory, you must additionally provide a servlet virtual path for it (as discussed in “Specifying Servlet Virtual Paths,” on page 20) to make the servlet accessible to clients.

iPlanet Web Server supports the specification of multiple directories, jars, zips, and URLs in the servlet classpath.

- **Servlet Args** -- a comma delimited list of additional arguments for the servlet if required.

For example, in Figure 1-1, the **Servlets>Configure Servlet Attributes** page of the Server Manager interface shows configuration information for a servlet whose class file `buynow1A` resides in the directory

`D:/Netscape/server4/docs/servlet/buy`. This servlet is configured under the name `BuyNowServlet`. It takes additional arguments of `arg1=45`, `arg2=online`, `arg3="quick shopping"`.



Figure 1-1 Configuring attributes for an individual servlet

The following code shows an example of the configuration information for the same servlet in `servlets.properties`:

```

servlet.BuyNowServlet.classpath=D:/Netscape/server4/docs/servlet
/buy;D:/Netscape/server4/docs/myclasses
servlet.BuyNowServlet.code=BuyNow1A
servlet.BuyNowServlet.initArgs=arg1=45,arg2=online,arg3="quick
shopping
    
```

Note that you can specify multiple values as the servlet classpath if needed.

Specifying Servlet Virtual Paths

If you register a servlet individually instead of putting it in a servlet directory, you must define a servlet virtual path for it. For example, you could specify that the URL

`http://poppy.my_domain.com/plans/plan1`

invokes the servlet defined in the directory

`server_root/docs/plans/releaseA/planP2Version1A.class`

You can set up servlet virtual paths for servlets that reside anywhere in the file system, in or out of a registered servlet directory.

To specify a servlet virtual path, use the **Servlets>Configure Servlet Virtual Path Translation** page in the Server Manager interface. In this page, specify the virtual path name and the servlet name. You can alternatively manually edit the `rules.properties` configuration file to add a servlet virtual path. Only servlets for which a virtual path has been set up can use initial arguments (See “`GenericServlet.getInitParameter` and `getInitParameterNames`,” on page 84 for information about initial arguments.)

Before using a servlet virtual path, a servlet identifier (or servlet name) must be added for the servlet in the **Servlets>Configure Servlet Attributes** page of the interface (or in the `servlets.properties` configuration file).

Virtual Servlet Path Example

This example shows how to specify that the logical URL

`http://poppy.my_domain.com/plans/plan1`

invokes the servlet defined in

`server_root/docs/plans/releaseA/planP2Version1A.class`.

1. Specify the servlet identifier, class file, and class path.

In the **Servlets>Configure Servlet Attributes** page in the interface, do the following:

- In the **Servlet Name** field, enter an identifier for the servlet, such as `plan1A`. (Notice that this is not necessarily the same as the class file name).
- In the **Servlet Code** field, enter the name of the class file, which is `planP2Version1A`. Don't specify any directories. The `.class` extension is not required.
- In the **Servlet Classpath** field, enter the absolute path name for the directory, jar or zip file where the servlet class file resides, or enter a URL for a directory. In this example, you would enter `server_root/docs/servlet/plans/releaseA`. (For example: `D:/netscape/server4/docs/servlet/plans/releaseA`.)
- In the **Servlet Args** field, enter the additional arguments that the servlet needs, if any. (This example does not use extra arguments.)

Save the changes.

Figure 1-2 shows the settings in the interface.



Figure 1-2 Specifying the servlet name, code, and class path

To make this change programmatically, add the following lines to the configuration file `servlets.properties`:

```
servlet.plan1A.classpath=D:/Netscape/server4/docs/servlet/plans/releaseA/
servlet.plan1A.code=planP2Version1A
```

2. Specify the virtual path for the servlet.

In the `Servlets>Configure Servlet Virtual Path Translations` page, do the following:

- In the Virtual Path field, enter the virtual path name. Note that the server name is implied as a prefix, so in this case you would only need to enter `/plans/plan1` to specify the virtual path `http://poppy.mcom.com/plans/plan1`.
- In the Servlet field, enter the identifier for the servlet that is invoked by this virtual path. This is the servlet identifier that you specified in the `Configure Servlet Attributes` page, which in this case is `plan1A`.

Save the changes.

Figure 1-3 shows the settings in the interface.

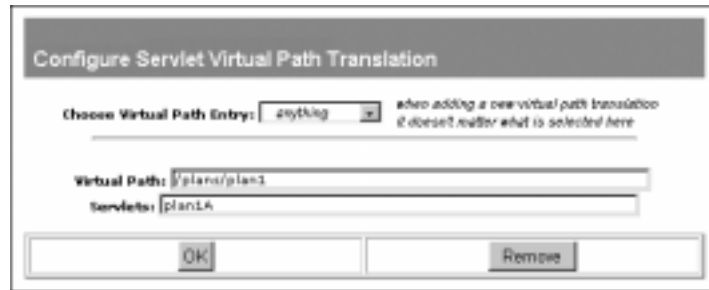


Figure 1-3 Adding a virtual path

To do this programmatically, add the following line to `rules.properties`:

```
/plans/plan1=plan1A
```

After this virtual servlet path has been established, if a client sends a request to the server for the URL `http://poppy.my_domain.com/plans/plan1`, the server sends back the results of invoking the servlet in `server_root/docs/servlet/plans/releaseA/plan2PVersion1A.class`.

Specifying Servlet Contexts

Contexts allow multiple servlets to exchange data and access each other's fields. Contexts are useful for defining virtual servers or for code isolation. You define contexts in the `servlets.properties` and `contexts.properties` files. For more information, see Appendix C, "Properties Files."

NOTE You can use only five hardware virtual servers in the FastTrack edition of iPlanet Web Server.

Servlet Output

Because servlets are external to the iPlanet Web Server, the `System.out` and `System.err` output are not sent to the web server's error log.

The servlet output is sent directly to standard output of the web server process. For example, if you start the web server from a terminal window, you should see the output in the terminal window.

On Unix, you can redirect the standard output to a file (modify your *server_root/https-server-name/start* file).

Configuring JRE/JDK Paths

When you install iPlanet Web Server 4.1, you can choose to install the Java Runtime Environment (JRE) that is shipped with the server, or you can specify a path to your own JRE or the Java Development Kit (JDK).

The server can run servlets using the JRE, but it needs the JDK to run JSP. The JDK is not bundled with the iPlanet Web Server, but you can download it for free from Sun Microsystems at:

<http://java.sun.com/products/jdk/1.2/>

iPlanet Web Server 4.1 requires you to use version of the JDK listed in the section “What Does the Server Need to Run Servlets and JSP?,” on page 12.

Regardless of whether you choose to install the JRE or specify a path to the JDK during installation, you can tell the iPlanet Web Server to switch to using either the JRE or JDK at any time. Switch to the Web Server Administration Server, select the Global Settings tab, and use the Configure JRE/JDK Paths page. You can also change the path to the JDK in this page.

On the Configure JRE/JDK Paths page, supply values for the following fields if you select the JDK radio button:

- **JDK Path**
Enter the path for the JDK. This is the directory where you installed the JDK.
- **JDK Runtime Libpath**
Enter the runtime library path for the JDK.

- **JDK Runtime Classpath**

The class path includes the paths to the directories and jar files needed to run the servlet engine, the servlet examples, and any other paths needed by servlets that you add. You can add new values to the existing class path, but don't delete the existing value since it includes paths that are essential for servlet operation.

Supply values for the following fields if you select the JRE radio button:

- **JRE Path**

Enter the path for the JRE. This is the directory where you installed the JRE.

- **JRE Runtime Libpath**

Enter the runtime library path for the JRE.

NOTE	If you are not sure of the JDK runtime libpath, the JDK runtime classpath, or the JRE runtime libpath, leave these fields blank to tell the server to use the default paths.
-------------	--

It is easiest to use the Configure JRE/JDK Paths page to switch between the JRE and the JDK, but you can also make the change programmatically, as follows:

- **On Unix:**

Edit the file *server_root*/https-admserv/start-jvm.

If the server is currently using the JRE, this file has a variable `NSES_JRE`. To enable the server to use a JDK, add the variable `NSES_JDK` whose value is the JDK directory. You'll also need to change the value of the `NSES_JRE` variable.

`NSES_JDK` should point to the installation directory for the JDK, while `NSES_JRE` should point to the JRE directory in the installation directory for JDK (that is, *jdk_dir*/jre).

- **On Windows NT:**

Add the path to the Java libraries to the `extrapath` setting in `magnus.conf`.

Edit the `NSES_JDK` and `NSES_JRE` variables in the registry `HKEY_LOCAL_MACHINE\SOFTWARE\Netscape\Enterprise\4.0\`. If the server is enabled to use the JDK, both these variables are needed. If the server is to use the JRE, only the `NSES_JRE` variable should be set.

NSES_JDK should point to the installation directory for the JDK, while NSES_JRE should point to the JRE directory in the installation directory for JDK (that is, *jdk_dir/jre*).

NOTE To activate changes to the JRE/JDK paths, you must restart the server from the On/Off option on the Preferences tab.

Deleting Version Files

The server uses two directories to cache information for JavaServer Pages (JSP) and servlets:

- **ClassCache**

When the server serves a JSP page, it creates a *.java* and a *.class* file associated with the JSP and stores them in the JSP class cache under the ClassCache directory.

- **SessionData**

If the server uses the *MMapSessionManager* session manager, it stores persistent session information in the SessionData directory. (For more information about session managers, see Appendix A, “Session Managers.”)

Each cache has a version file containing a version number that the server uses to determine the structure of the directories and files in the caches. You can clean out the caches by simply deleting the version file.

When the server starts up, if it does not find the version files, it deletes the directory structures for the corresponding caches and re-creates the version files. Next time the server serves a JSP page, it recreates the JSP class cache. The next time the server serves a JSP page or servlet while using *MMapSessionManager* session manager, it recreates the session data cache.

If a future upgrade of the server uses a different format for the caches, the server will check the number in the version file and clean up the caches if the version number is not correct.

You can delete the version files simply by deleting them from the ClassCache or SessionData directories as you would normally delete a file, or you can use the Servlets>Delete Version Files page in the Server Manager to delete them. After deleting one or both version files, be sure to restart the iPlanet Web Server to force it to clean up the appropriate caches and to recreate the version files before the server serves any servlets or JSPs.

Configuring JVM

If necessary, you can configure parameters for JVM either by using the Servlets>Configure JVM Attributes page in the Server Manager interface, or by editing `jvm12.conf`.

The default settings in iPlanet Web Server for JVM are suitable for running servlets. However, there may be times when you want to change the settings. For example, if a servlet or bean file uses a JAR file, add the JAR location to the Classpath variable. To enable the use of a remote profiler, set the OPTITDIR and Profiler variables.

NOTE	A few attributes on the Configure JVM Attributes page on the Servlets tab show as “Default.” Since you can use different JVMs, these default values are unknown. You cannot query a JVM to find out the actual default values; instead, refer to your JVM documentation. For example, for Sun’s JVM, if you choose Yes for the JIT Compiler option, it shows as “Default” because JIT is enabled in the JVM by default. However, if you choose No for the JIT compiler, an explicit entry, <code>jvm.compiler=NONE</code> , is added to the <code>jvm12.conf</code> file.
-------------	---

The JVM parameters you can set are:

- Option -- You can set any options allowed by the vendor’s JVM.
- Profiler -- If you are using the Optimizeit! 3.0 profiler from Intuitive Systems, enter the value `optimizeit`. For more information about this optimizer, see Appendix F, “Remote Servlet Profiling.”
- OPTITDIR -- If you are using the Optimizeit! 3.0 profiler from Intuitive Systems, enter the pathname for the directory where Optimizeit! resides, for example, `D:/App/IntuitiveSystems/OptimizeIt300`. For more information about this optimizer, see Appendix F, “Remote Servlet Profiling.”
- Minimum Heap Size -- determines the minimum heap size allocated for Java.
- Maximum Heap Size -- determines the maximum heap size allocated to Java.
- Compiler -- You can specify options to turn on and off JIT (just-in-time compiler). See your JVM documentation for details.
- Classpath -- Enter additional classpath values as needed. For example, if a JSP uses a bean that is packaged in a JAR, add the JAR path to the classpath.

The classpath must not include backslashes in directory names. If you use backslashes in the directory path when using the Web Server Administrative Server interface, the system automatically converts the backslashes to forward slashes. However, if you edit the `jvm12.conf` file, do not use backslashes in directory names.

- **Enable Class GC** -- Specifies whether or not to enable class garbage collection. The default is yes.
- **Verbose Mode** -- Determines whether the JVM logs a commentary on what it is doing, such as loading classes. The commentary appears in the error log.
- **Enable Debug** -- You can enable or disable remote debugging. The default is disabled. For more information about remote debugging, see Appendix E, "Debugging Servlets and JSPs."

Running 0.92 JSP

JSP API version 1.x is not backward compatible with version 0.92. However, the iPlanet Web Server correctly serves JSPs written to the 0.92 spec if you place them in JSP legacy directories. If a 0.92 JSP does not reside in a JSP legacy directory, it will not work.

The Legacy JSP Directory page in the Servlets tab of the Server Manager allows you to add legacy JSP directories. For information about how to use this page, see the Legacy JSP Directory page in the online help.

JSP legacy directories can contain JSPs and other files such as HTML pages.

Maximizing Servlet Performance

Consider the following guidelines for improving servlet performance:

- Use Sun's JVM on Solaris--it's up to 50% faster than JavaSoft's.
- If you edit your `obj.conf` file manually, make sure that the servlet `NameTrans` (`NameTrans fn="NSServletNameTrans" name="servlet"`) is always the first `NameTrans` directive.

This directive uses a highly optimized URI cache for loaded servlets and returns `REQ_PROCEED` if the match is found, thus eliminating the need of other `NameTrans` directives to be executed.

- Servlets defined individually (via the Configure Servlet Attributes page or `rules.properties` and `servlets.properties`) are slightly faster than dynamically loaded servlets (in servlet directories).
- The `jvm12.conf` file has a configuration parameter, `jvm.stickyAttach`. Setting the value of this parameter to 1 causes threads to remember that they are attached to the JVM, thus speeding up request processing by eliminating `AttachCurrentThread` and `DetachCurrentThread` calls. It can, however, have a side-effect: recycled threads which may be doing other processing can be suspended by the garbage collector arbitrarily.

Thread pools can be used to eliminate this side effect for other subsystems. For more information about thread pools, see “Adding and Using Thread Pools” in Chapter 7, “Configuring Server Preferences,” in the *iPlanet Web Server Administrator’s Guide*.

- Increase the front-end thread stack size in `magnus.conf` (via the `StackSize` variable), or the respective pool stack size parameter if you’re using thread pools. For more information, see the *NSAPI Programmer’s Guide for iPlanet Web Server*.
- Increase the heap size to help garbage collection: `jvm.minHeapSize` or `maxHeapSize` or the Configure JVM Attributes page.
- Ensure that your classpath is short: `jvm.classpath` (if you don’t need some of the examples). You can set `jvm.include.CLASSPATH=1`, so it won’t inherit the `CLASSPATH` environment variable.
- Sometimes, iPlanet Web Server 4.1 may run out of stack space if applications use deep recursion when a JIT compiler is enabled, especially on UNIX platforms where the default stack size is small, or in any cases where very complex JSP pages are used.

You can set the stack space using the `StackSize` parameter in the `magnus.conf` file. For more information, see the *NSAPI Programmer’s Guide for iPlanet Web Server*.

- The use of the NSAPI cache improves servlet performance in cases where the `obj.conf` configuration file has many directives. To enable the NSAPI cache, include the following line in `obj.conf`:

```
Init fn="nsapi-cache-init" enable=true
```

- The session ID generator, which is used for servlet sessions, employs cryptographically strong unique random number generation algorithms. This may present a performance problem on older, slow machines. For more information, see Appendix A, “Session Managers.”

Servlet and JSP Examples

This chapter discusses some Servlet and JSP examples in the following sections:

- Examples Shipped with iPlanet Web Server 4.1
- Servlet Examples
- JSP Examples

Examples Shipped with iPlanet Web Server 4.1

iPlanet Web Server 4.1 comes with a set of example servlets and JSP files. You can find them at the following location:

server_root/plugins/samples/servlets

This directory contains the following directories:

- `beans` -- Contains example Java Bean files for JSP 0.92.
- `beans.10` -- Contains example Java Bean files for JSP 1.x.
- `bookstore` -- Contains files for an online bookstore example. This example contains both servlets and JSPs.
- `jsp.092` -- Contains subdirectories that each contain an example for JSP 0.92. To use one of these examples, you must place it in a legacy directory; see “Running 0.92 JSP,” on page 28 for details.
- `jsp.10` -- Contains subdirectories that each contain an example for JSP 1.x.
- `make` -- Contains example makefiles for servlets. These are common makefiles containing rules that are included by all other makefiles.
- `servlets` -- Contains subdirectories that each contain Java source files and makefiles for servlet examples.

- sessions -- Contains session manager directories.

The `SimpleSession` directory contains code for `SimpleSessionManager.java`, which is the default servlet session manager, and `SimpleSession.java`, which defines session objects, the sessions managed by `SimpleSessionManager`. The source code for `SimpleSessionManager` and `SimpleSession` are provided for you to use as the starting point for defining your own session managers if desired.

The `JdbcSession` directory contains `JdbcSessionManager.java` and `JdbcSession.java`, which contain support for sessions stored in a database using JDBC.

For more information about sessions and session managers, see Appendix A, “Session Managers.”

- taglibs -- Contains an example of a custom tag library, which is described in “Creating a JSP Custom Tag Library,” on page 42.
- tools -- Contains the `SDKTools.jar` file and other utility files.

Servlet Examples

This section discusses two servlet examples as follows:

- A Simple Servlet Example -- generates a very simple page to be displayed in a web browser.
- Example of a Servlet that Counts Visits -- this servlet is used to count visits to a web page.

You can find additional examples in the directory `server_root/plugins/samples/servlets/servlets`.

These examples are simple, introductory examples. For information about using the Java Servlet API, see the documentation provided by Sun Microsystems at:

<http://java.sun.com/products/servlet/index.html>

A Simple Servlet Example

The following example code defines a very simple servlet. This is the `SimpleServlet` example in the `server_root/plugins/samples/servlets/servlets/Simple1` directory.

This servlet generates an HTML page that says “This is output from SimpleServlet.” as shown in Figure 2-1.



Figure 2-1 Output from SimpleServlet.class

This example defines the main servlet class as a subclass of `HttpServlet` and implements the `doGet` method. The code is shown below:

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

public class SimpleServlet extends HttpServlet
{
    public void doGet (
        HttpServletRequest request,
        HttpServletResponse response
    ) throws ServletException, IOException
    {
        PrintWriter out;
        String title = "Simple Servlet Output";

        // set content type and other response header fields first
        response.setContentType("text/html");

        // then write the data of the response
        out = response.getWriter();

        out.println("<HTML><HEAD><TITLE>");
        out.println(title);
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>" + title + "</H1>");
        out.println("<P>This is output from SimpleServlet.");
        out.println("</BODY></HTML>");
    }
}
```

Example of a Servlet that Counts Visits

The following example code defines a servlet that counts visits to a web page. This is the CounterServlet example in the

server_root/plugins/samples/servlets/servlets/Counter directory.

This servlet generates an HTML page that reports the number of visits for an individual user and for all users, as shown in Figure 2-2.



```
Thank you for coming back. You have visited this page 1 times This page was accessed 1 times total
```

Figure 2-2 Output from CounterServlet.class

This example defines the main servlet class as a subclass of `HttpServlet` and implements the `doGet` method, as the `SimpleServlet` example did, but it also defines a thread, tracks total hits by reading from and writing to a file, and tracks hits from individual users using a cookie. The code is shown below:

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

public class CounterServlet extends HttpServlet
{
    private File _counterFile = new File ("/tmp/CounterServlet.dat");
    private CounterWriterThread _cntWrtThread = new CounterWriterThread ();
    private int _cnt = 0;

    private boolean _fTerminating = false;

    public void init (ServletConfig config)
        throws ServletException
    {
        super.init (config);

        readCounter ();
        _cntWrtThread.start ();
    }

    public class CounterWriterThread
        extends Thread
    {
        public void run ()
        {
            while (!_fTerminating)
            {
                writeCounter ();
                try {
                    sleep (1000);
                }
                catch (Exception ie)
                {
                }
            }
        }
    }
}
```

```

    }
}

private void writeCounter ()
{
    DataOutputStream dos = null;

    try {
        dos = new DataOutputStream (new FileOutputStream (_counterFile));
        dos.writeInt (_cnt);
    }
    catch (Exception e)
    {
    }
    finally{
        try {
            if (dos != null)
                dos.close ();
        }
        catch (Exception ioe)
        {
        }
    }
}

private void readCounter ()
{
    DataInputStream dis = null;

    try {
        dis = new DataInputStream (new FileInputStream (_counterFile));
        _cnt = dis.readInt ();
    }
    catch (Exception e)
    {
    }
    finally{
        try {
            if (dis != null)
                dis.close ();
        }
        catch (Exception ioe)
        {
        }
    }

}

public void doGet (
    HttpServletRequest    request,
    HttpServletResponse    response
)
{
    throws ServletException, IOException

    {
        PrintWriter    out;

        // set content type and other response header fields first
        response.setContentType("text/html");

        // then write the data of the response
        out = response.getWriter ();

        _cnt++;
    }
}

```

```

Cookie cookies[] = request.getCookies();
Integer nof = new Integer (0);

for(int i = 0; i < cookies.length; i++ )
{
    if (cookies[i].getName ().equals ("CounterServletCookie"))
    {
        String nofS = cookies[i].getValue ();
        try {
            nof = Integer.valueOf (nofS);
        }
        catch (Exception nfe)
        {
        }
        break;
    }
}

nof = new Integer (nof.intValue () + 1);
Cookie c = new Cookie ("CounterServletCookie", nof.toString ());

c.setMaxAge (3600 * 24 * 365);
c.setPath  ("/");

response.addCookie (c);

out.println("<HTML><BODY><CENTER>");

if (nof.intValue () > 1)
    out.println ("Thank you for coming back. You have visited this page <B>"
        + nof + "</b> times");

out.println("This page was accessed <B>" + _cnt + "</B> times total");
out.println("</CENTER></BODY></HTML>");
}
}

```

JSP Examples

This section presents the following JSP examples:

- JSP that Accesses the Request Object. This example is self-contained -- it uses no external beans or Java classes.
- JSP that Responds to a Form and Uses Java Beans.
- Creating a JSP Custom Tag Library.

You can find additional examples in the directory
server_root/plugins/samples/servlets/jsp.10.

These examples are simple, introductory examples. For information about creating JSPs, see Sun Microsystem's JavaServer Pages web page at:

<http://java.sun.com/products/jsp/index.html>

JSP that Accesses the Request Object

JavaServer Pages (JSPs) contain both standard HTML tags and JSP tags.

All JSP pages can implicitly access the request object, which contains information about the request that invoked the page, such as the requested URI, the query string, the content type, and so on. The request object has properties such as `requestURI`, `queryString`, and `contentType`.

This example displays information about the current request. It gets all its data from the request object, which is automatically passed to the JSP. This is the `snoop.jsp` example in the `server_root/plugins/samples/servlets/jsp.10/snp` directory.

Figure 2-3 shows an example of the output page generated by this JSP.

Request Information

```
JSP Request Method: GET
Request URI: /jsp.10/snp/snoop.jsp
Request Protocol: HTTP/1.0
Servlet path: /jsp.10/snp/snoop.jsp
Path info: null
Path translated: null
Query string: null
Content length: -1
Content type: null
Server name: nsep-dpro-nt-95.jukebox
Server port: 777
Remote user: null
Remote address: 206.222.236.200
Remote host: 206.222.236.200
Authorization scheme: null
```

The browser you are using is Mozilla/4.7 [en]C-AOLNSCP (WinNT; U)

Figure 2-3 Output page generated by `snoop.jsp`

The source code for `snoop.jsp` is:

```
<html>
<body bgcolor="white">
<h1> Request Information </h1>
<font size="4">

<%@ page session="false" %>
```

```

JSP Request Method: <%= request.getMethod() %>
<br>
Request URI: <%= request.getRequestURI() %>
<br>
Request Protocol: <%= request.getProtocol() %>
<br>

Servlet path: <%= request.getServletPath() %>
<br>
Path info: <%= request.getPathInfo() %>
<br>
Path translated: <%= request.getPathTranslated() %>
<br>
Query string: <%= request.getQueryString() %>
<br>
Content length: <%= request.getContentLength() %>
<br>
Content type: <%= request.getContentType() %>
<br>
Server name: <%= request.getServerName() %>
<br>
Server port: <%= request.getServerPort() %>
<br>
Remote user: <%= request.getRemoteUser() %>
<br>
Remote address: <%= request.getRemoteAddr() %>
<br>
Remote host: <%= request.getRemoteHost() %>
<br>
Authorization scheme: <%= request.getAuthType() %>
<br>
The browser you are using is <%= request.getHeader("User-Agent") %>
<br>
</font>
</body>
</html>

```

JSP that Responds to a Form and Uses Java Beans

This example discusses a simple JSP that accesses data on Java beans to respond to a form. This is the example in the *server_root/plugins/samples/servlets/jsp.10/checkbox* directory.

This example presents a web page, `check.html`, that displays a form asking the user to select their favorite fruits. The action of the form is `checkresult.jsp`. This JSP file gets information about the fruits from a Java bean. (Note that Java beans were originally designed for use with visual tool builders, and they have some overhead that can make them slow when used to retrieve data to display in web pages.)

The discussion of this example has the following sections:

- The Form
- The Output Page Generated by the JSP File
- Accessing Input Parameters
- Using Externally Defined Java Beans
- Source Code for the JSP File

The Form

The form in the page has the following elements:

- Four checkboxes named Apples, Grapes, Oranges, and Melons
- A Submit button

The form's method is POST and the action is `checkresult.jsp`. (It also works if the form's method is GET.)

```
<FORM TYPE=POST ACTION=checkresult.jsp>
```

Figure 2-4 shows an example of the form.



Check all Favorite fruits:

☒ Apples

☒ Grapes

☐ Oranges

☐ Melons

Submit

Figure 2-4 This form invokes a JSP as its action

The Output Page Generated by the JSP File

The JSP file `checkresult.jsp` responds to the form. It uses a request and then a bean to access the parameters received from the form.

The output page generated by `checkresult.jsp` displays the fruits that were selected. The JSP file gets information about the fruits from Java Beans.

This JSP file demonstrates the following features:

- Accessing Input Parameters
- Using Externally Defined Java Beans

Figure 2-5 shows an example of the output from `checkresult.jsp`:

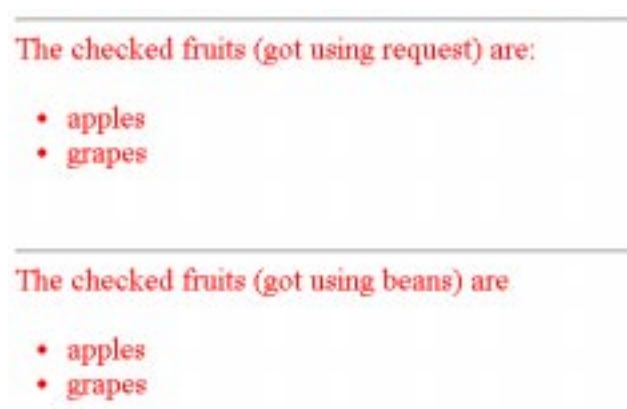


Figure 2-5 A JSP page generated in response to a form submission

Accessing Input Parameters

JSP pages can extract input parameters when invoked by a URL with a query string, such as when they are invoked as a form action for a form that uses the GET method. The `request.getParameterValues` method retrieves an object that has attributes for each parameter in the query string.

For example, if the following URL is used to invoke a JSP:

`http://my_domain.com/fruits/checkresult.jsp?Apples=on&Oranges=on`

The request object has properties `Apples` and `Oranges`.

Using Externally Defined Java Beans

Some bean objects, including the request object, are always available implicitly to a JSP page. Other objects, such as user-defined objects, are not automatically available to the page, in which case you have to include a `<useBean>` tag to tell the page which object to use.

The JSP tag `<useBean>` creates an instance of an externally defined Java Bean for use within the JSP page. For example, the following code creates an instance of the Java Bean object `checkbox.CheckTest`, which is defined in `checktest.html`:

```
<jsp:useBean id="foo" scope="page" class="checkbox.CheckTest" />
```

In this case, the bean instance exists for the duration of the page.

For more information about defining Java Beans, see:

<http://java.sun.com/beans/index.html>

Source Code for the JSP File

Here is the source code for the JSP file `checkresult.jsp`:

```
<html>

<body bgcolor="white">
<font size=5 color="red">
<%! String[] fruits; %>
<jsp:useBean id="foo" scope="page" class="checkbox.CheckTest" />

<jsp:setProperty name="foo" property="fruit" param="fruit" />
<hr>
The checked fruits (got using request) are: <br>
<%
    fruits = request.getParameterValues("fruit");
%>
<ul>
<%
    if (fruits != null) {
        for (int i = 0; i < fruits.length; i++) {
<li>
<%
            out.println (fruits[i]);
        }
    } else out.println ("none selected");
%>
</ul>
<br>
```

```
<hr>
```

The checked fruits (got using beans) are


```
<%
    fruits = foo.getFruit();
%>
<ul>
<%
    if (!fruits[0].equals("1")) {
        for (int i = 0; i < fruits.length; i++) {
%>
<li>
<%
            out.println (fruits[i]);
        }
    } else out.println ("none selected");
%>
</ul>
</font>
</body>
</html>
```

Creating a JSP Custom Tag Library

iPlanet Web Server 4.1 supports custom JSP tags. This section explains how to create a custom tag library using a working example. The example includes the following directories and files under the document root directory:

```
/dtds/
    web-jsptaglib_1_1.dtd

/jsps/
    test-tags.jar
    test-tags.jsp
```

To create the `test-tags.jar` file, you must create a work area in which you build the tag library and its associated handler classes. This work area contains the following directories and files:

```
workarea/taglibs/
  ./META-INF:
  taglib.tld
  ./examples:
  ExampleTagBase.class
  ExampleTagBase.java
  FooTag.class
  FooTag.java
  FooTagExtraInfo.class
  FooTagExtraInfo.java
  LogTag.class
  LogTag.java
```

Both sets of example files are provided with iPlanet Web Server 4.1 in the following directory:

`server_root/plugins/samples/servlets/taglibs`

Step 1: Create the TLD File

You first need to write a tag library definition (TLD) file outlining the custom tags and their associated handler classes. This TLD file is an XML 1.0 file.

The TLD file must reference an associated DTD file, which has to be named `web-jsptaglib_1_1.dtd`. This file is available at the iPlanet Web Server website for download. You must make it accessible via a URL in the TLD file (for example `http://server:port/dtds/web-jsptaglib_1_1.dtd`). An incorrect URL for the DTD file or a corrupt DTD file results in failures in opening your JSPs that contain custom tags.

The tag library must be named `taglib.tld` and must reside under the `META-INF` subdirectory in the `taglib.jar` file you will create in step 4.

Here is an example of a `taglib.tld` file:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://server:port/dtds/web-jsptaglib_1_1.dtd">
```

```

<!-- a tag library descriptor -->

<taglib>
  <!-- after this the default space is
        "http://java.sun.com/j2ee/dtds/jsptaglibrary_1_2.dtd"
  -->

  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <prefix>simple</prefix>
  <urn></urn>
  <info>
    A simple tag library for the examples
  </info>

  <!-- A simple Tag -->
  <!-- foo tag -->
  <tag>
    <name>foo</name>
    <tagclass>examples.FooTag</tagclass>
    <teiclass>examples.FooTagExtraInfo</teiclass>
    <bodycontent>JSP</bodycontent>
    <info>
      Perform a server side action; uses 3 mandatory attributes
    </info>

    <attribute>
      <name>att1</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>att2</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>att3</name>
      <required>true</required>
    </attribute>
  </tag>

  <!-- Another simple tag -->
  <!-- log tag -->
  <tag>
    <name>log</name>
    <tagclass>examples.LogTag</tagclass>
    <bodycontent>TAGDEPENDENT</bodycontent>
    <info>
      Perform a server side action; Log the message.
    </info>
    <attribute>
      <name>toBrowser</name>
      <required>false</required>

```

```

        </attribute>
    </tag>

</taglib>

```

If you do not include the DOCTYPE in your taglib.tld file, the JSP compiler throws an exception:

```

Unable to open taglibrary /jsps/test-tags.jar :
com.sun.xml.tree.TextNode

```

Step 2: Create the Tag Handler Classes

Here is the ExampleTagBase.java file:

```

package examples;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public abstract class ExampleTagBase implements Tag {

    public void setParent(Tag parent) {
        this.parent = parent;
    }

    public void setBodyContent(BodyContent bodyOut) {
        this.bodyOut = bodyOut;
    }

    public void setPageContext(PageContext pageContext) {
        this.pageContext = pageContext;
    }

    public Tag getParent() {
        return this.parent;
    }

    public int doStartTag() throws JspException {
        return SKIP_BODY;
    }

    public int doEndTag() throws JspException {
        return EVAL_PAGE;
    }

    // Default implementations for BodyTag methods as well
    // just in case a tag decides to implement BodyTag.

```

```

    public void doInitBody() throws JspException {
    }

    public int doAfterBody() throws JspException {
        return SKIP_BODY;
    }

    public void release() {
        bodyOut = null;
        pageContext = null;
        parent = null;
    }
    protected BodyContent bodyOut;
    protected PageContext pageContext;
    protected Tag parent;
}

```

Here is the FooTag.java file:

```

package examples;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.util.Hashtable;
import java.io.Writer;
import java.io.IOException;

/**
 * Example1: the simplest tag
 * Collect attributes and call into some actions
 *
 * <foo att1="..." att2="...." att3="...." />
 */

public class FooTag
    extends ExampleTagBase
    implements BodyTag
{
    private String atts[] = new String[3];
    int i = 0;

    private final void setAtt(int index, String value) {
        atts[index] = value;
    }

    public void setAtt1(String value) {
        setAtt(0, value);
    }
}

```

```

    public void setAtt2(String value) {
        setAtt(1, value);
    }

    public void setAtt3(String value) {
        setAtt(2, value);
    }

    /**
     * Process start tag
     *
     * @return EVAL_BODY_INCLUDE
     */
    public int doStartTag() {
        return EVAL_BODY_TAG;
    }

    public void doInitBody() throws JspException {
        pageContext.setAttribute("member", atts[i]);
        i++;
    }

    public int doAfterBody() throws JspException {
        try {
            if (i == 3) {
                bodyOut.writeOut(bodyOut.getEnclosingWriter());
                return SKIP_BODY;
            } else
                pageContext.setAttribute("member", atts[i]);
            i++;
            return EVAL_BODY_TAG;
        } catch (IOException ex) {
            throw new JspException(ex.toString());
        }
    }
}

```

Here is the FooTagExtraInfo.java file:

```

package examples;

import javax.servlet.jsp.tagext.*;

public class FooTagExtraInfo extends TagExtraInfo {
    public VariableInfo[] getVariableInfo(TagData data) {
        return new VariableInfo[]
        {
            new VariableInfo("member",

```

```

        "String",
        true,
        VariableInfo.NESTED)
    };
}
}

```

Here is the LogTag.java file:

```

package examples;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

import java.io.IOException;

/**
 * Log the contents of the body. Could be used to handle errors etc.
 */
public class LogTag
    extends ExampleTagBase
    implements BodyTag
{
    boolean toBrowser = false;

    public void setToBrowser(String value) {
        if (value == null)
            toBrowser = false;
        else if (value.equalsIgnoreCase("true"))
            toBrowser = true;
        else
            toBrowser = false;
    }

    public int doStartTag() {
        return EVAL_BODY_TAG;
    }

    public int doAfterBody() throws JspException {
        try {
            String s = bodyOut.getString();
            System.err.println(s);
            if (toBrowser)
                bodyOut.writeOut(bodyOut.getEnclosingWriter());
            return SKIP_BODY;
        } catch (IOException ex) {

```



```

        throw new JspException(ex.toString());
    }
}

```

Step 3: Compile the Tag Handling Classes

Change to your work directory for the java classes (for example `workarea/taglibs/examples`) and compile the classes, setting the appropriate classpaths. The following command must be all on one line:

```

/usr/java1.2/bin/javac -classpath
java_home/jre/lib/rt.jar:server_root/bin/https/jar/servlets.jar:
server_root/bin/https/jar/jspengine.jar *.java

```

Step 4: Create the JAR File

From your work directory (for example `workarea/taglibs`), create a JAR file that contains your `taglib.tld` file and the class files. Use the following command:

```
jar cvf ../test-tags.jar .
```

You should see the following output on the screen:

```

adding: META-INF/ (in=0) (out=0) (stored 0%)
adding: META-INF/taglib.tld (in=1459) (out=581) (deflated 60%)
adding: examples/ (in=0) (out=0) (stored 0%)
adding: examples/domake (in=235) (out=135) (deflated 42%)
adding: examples/ExampleTagBase.java (in=1123) (out=395)
(deflated 64%)
adding: examples/FooTag.java (in=1455) (out=571) (deflated 60%)
adding: examples/FooTagExtraInfo.java (in=426) (out=191)
(deflated 55%)
adding: examples/LogTag.java (in=994) (out=450) (deflated 54%)
adding: examples/FooTagExtraInfo.class (in=539) (out=320)
(deflated 40%)
adding: examples/ExampleTagBase.class (in=1221) (out=548)
(deflated 55%)
adding: examples/FooTag.class (in=1553) (out=784) (deflated 49%)
adding: examples/LogTag.class (in=1249) (out=702) (deflated 43%)

```

When this is complete, you should see the `test-tags.jar` file in the workarea directory. Type the following command to verify this:

```
ls -ltr test-tags.jar
```

You should see something like the following output on the screen:

```
-rw-r--r--  1 joe      staff      7009 May 18 02:37 test-tags.jar
```

Copy the file to your JSP directory:

```
cp test-tags.jar jsp_location/jsps
```

Step 5: Add the JAR File to the Classpaths

Edit the `jvm.classpath` option in the `server_root/https-server_id/config/jvm12.conf` file to include the path to the custom tag library file. The following example must be all on one line in the file:

```
jvm.classpath=java_home/jre/lib/rt.jar:java_home/lib/tools.jar:jsp_location/jsps/test-tags.jar
```

In addition, make sure that your custom tag library file is added to the server classpath along with the regular Java runtime and tools.

Also make sure that the `web-jsptaglib_1_1.dtd` file is accessible at the URL you specified in the `taglib.tld` file. Failure to locate this DTD URL results in errors like this in your log files:

```
Unable to open taglibrary /jsps/test-tags.jar : Unable to open the tag library descriptor: Stream closed.
```

Step 6: Create the JSP File

This section shows an example JSP file that uses the custom tags. A custom prefix, `tt`, indicates which tags are handed off to the custom tag library (the `foo` and `log` tags).

```
<%@ page language="java" %>
<%@ taglib prefix="tt" uri="/jsps/test-tags.jar" %>

<title>Testing taglibs </title>

<h1>Testing Jsp taglibs! </h1>

<p><em><tt:foo att1="hello" att2="world" att3="tags" /></em><br>
<p><em><tt:log toBrowser="true" /> Hello Taglibs!</em><br>
```

Step 7: Test the Custom Tags

Type the following URL in your browser:

`http://server:port/jsps/test-tags.jsp`

Your browser should show the following:

Testing Jsp taglibs!

Hello Taglibs!

Session Managers

Session objects maintain state and user identity across multiple page requests over the normally stateless HTTP protocol. A session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session either by using cookies or by rewriting URLs. Servlets can access the session objects to retrieve state information about the session.

This appendix has the following sections:

- Session Overview
- Specifying a Session Manager
- SimpleSessionManager
- JdbcSessionManager
- How Do Servlets Access Session Data?

Session Overview

An HTTP session represents the server's view of the session. The server considers a session new under these conditions:

- The client does not yet know about the session.
- The session has not yet begun.

A session manager automatically creates new session objects whenever a new session starts. In some circumstances, clients do not join the session, for example, if the session manager uses cookies and the client does not accept cookies.

NOTE The session ID generator, which is used for servlet sessions, employs cryptographically strong unique random number generation algorithms. This may present a performance problem on older, slow machines. The Session Manager API allows you to redefine the random ID generation method and customize it to your particular needs (see the `SimpleSessionManager.java` example file described in “Source Code for `SimpleSessionManager`,” on page 56).

iPlanet Web Server 4.1 comes with these session managers for creating and managing sessions:

- `SimpleSessionManager` -- the default session manager.
- `JdbcSessionManager` -- a session manager that stores session information in a database using the JDBC API.

iPlanet Web Server 4.1 also allows you to develop your own session managers and load them into the server. The build includes the source code for `SimpleSessionManager` and the session object it manages, `SimpleSession`. The source code files for these classes are provided as a starting point for you to define your own session managers if desired. These Java files are in the directory `server_root/plugins/samples/servlets/sessions/SimpleSession`.

The build also includes the source code for `JdbcSessionManager` and the session object it manages, `JdbcSession`. These Java files are in the directory `server_root/plugins/samples/servlets/sessions/JdbcSession`.

Specifying a Session Manager

By default iPlanet Web Server uses `SimpleSessionManager` as the session manager for servlets.

You can change the session manager in any of the following ways:

- Use the Servlets>Configure Global Servlet Attributes page in the Server Manager interface.

In the Session Manager field, specify the session manager, and, if appropriate, specify parameters for the session manager in the Session Manager Args field.

- Edit the file `servlets.properties` in the directory `server_id/config`.

Add a line specifying a value for `servlets.sessionmgr` and, if appropriate, also add a line specifying the parameters for the session manager. For example:

```
servlets.sessionmgr=com.netscape.server.http.session.YourSesMgr
servlets.sessionmgr.initArgs=maxSessions=20,timeOut=300,reapInterval=150
```

- Edit the file `contexts.properties` in the directory `server_id/config`.

Add a line specifying a value for `context.context_name.sessionmgr` and, if appropriate, also add a line specifying the parameters for the session manager. For example:

```
context.global.sessionmgr=com.netscape.server.http.session.YourSesMgr
context.global.sessionmgr.initArgs=maxSessions=20,timeOut=300
```

You can change the global context or define a new context and assign specific servlets to it. For more information, see Appendix C, “Properties Files.”

SimpleSessionManager

The `SimpleSessionManager` is loaded by default when a `SessionManager` is not specified in the `servlets.properties` or `contexts.properties` configuration file. These sessions are not persistent, that is, all sessions are lost when the server is stopped.

Parameters

The `SimpleSessionManager` class takes the following parameters:

- `maxSessions` - the maximum number of sessions maintained by the session manager at any given time. The session manager refuses to create any more new sessions if there are already `maxSessions` number of sessions present at that time. The default value is 1000.
- `timeOut` - the amount of time in seconds after a session is accessed by the client before the session manager destroys it. Those sessions that haven't been accessed for at least `timeOut` seconds are destroyed by the reaper method. The default value is 1800 (30 minutes).
- `reapInterval` - the amount of time in seconds that the `SessionReaper` thread sleeps before calling the reaper method again. The default value is 600 (10 minutes).

Enabling SimpleSessionManager

You may want to enable SimpleSessionManager to change its default parameters. To enable the iPlanet Web Server to use SimpleSessionManager, do any of the following:

- Use the Servlets>Configure Global Servlet Attributes page in the Server Manager interface.

In the Session Manager field specify:

```
com.netscape.server.http.session.SimpleSessionManager
```

You can also specify parameters for the session manager in the Session Manager Args field, for example:

```
maxSessions=20,timeOut=300,reapInterval=150
```

- Edit the file `servlets.properties` in the directory `server_id/config`.

Add a line specifying a value for `servlets.sessionmgr` and a line specifying the parameters for the session manager:

```
servlets.sessionmgr=com.netscape.server.http.session.SimpleSessionManager
servlets.sessionmgr.initArgs=maxSessions=20,timeOut=300,reapInterval=150
```

- Edit the file `contexts.properties` in the directory `server_id/config`.

Add a line specifying a value for `context.context_name.sessionmgr` and a line specifying the parameters for the session manager:

```
context.global.sessionmgr=com.netscape.server.http.session.SimpleSessionManager
context.global.sessionmgr.initArgs=maxSessions=20,timeOut=300,reapInterval=150
```

You can change the global context or define a new context and assign specific servlets to it. For more information, see Appendix C, “Properties Files.”

Source Code for SimpleSessionManager

The SimpleSessionManager creates a SimpleSession object for each session. The source files for SimpleSessionManager.java and SimpleSession.java are in the `server_root/plugins/samples/servlets/sessions/SimpleSession` directory.

The source code files for SimpleSessionManager.java and SimpleSession.java are provided so you can use them as the starting point for defining your own session managers and session objects. These files are very well commented.

`SimpleSessionManager` extends `NSHttpSessionManager`. The class file for `NSHttpSessionManager` is in the JAR file `NSServletLayer.jar` in the directory `server_root/bin/https/jar`. The `SimpleSessionManager` implements all the methods in `NSHttpSessionManager` that need to be implemented, so you can use `SimpleSessionManager` as an example of how to extend `NSHttpSessionManager`. When compiling your subclass of `SimpleSessionManager` or `NSHttpSessionManager`, be sure that the JAR file `NSServletLayer.jar` is in your compiler's classpath.

JdbcSessionManager

This is a persistent JDBC-based session manager. It can be used to store sessions in a custom database.

This sample JDBC session manager is not written, tested, or intended for production use. It is provided so that you can customize its behavior to suit your own needs.

`JdbcSessionManager` has been tested with a standard JDBC-ODBC driver against Microsoft SQL Server 7.0SP1. You must set up the ODBC source, database, and table for the session manager to use. It is recommended that the Session ID column be indexed for higher lookup performance.

Parameters

`JdbcSessionManager` takes the following parameters:

- `timeOut` - the amount of time in seconds after a session is accessed by the client before the session manager destroys it. Those sessions that haven't been accessed for at least `timeOut` seconds are destroyed by the `reaper` method. The default value is 1800 (30 minutes).
- `provider` - the JDBC driver (the default is `sun.jdbc.odbc.JdbcOdbcDriver`). For more information about the JDBC API, see the following web site:
<http://java.sun.com/products/jdbc/index.html>
- `url` - the data source (the default is `jdbc:odbc:LocalServer`).
- `table` - name of the SQL table that store sessions (the default is `sessions`).
- `username` - the login username for the database.
- `password` - the login password for the database.

- `reaperActive` - tells the session manager whether to run session reaper to remove expired sessions from the database when true, which is the default value. It is recommended that only one server in the cluster be running the reaper.
- `accessTimeColumn` - the name of the column that holds the last access time in minutes (the default name is `AccessTime`). The SQL type is `NUMERIC(9)`.
- `sessionIdColumn` - the name of the column that holds the session ID (the default name is `SessionID`). The SQL type is `VARCHAR(100)`.
- `valueColumn` - the name of the column that holds the session object (the default name is `Value`). The SQL type is `VARBINARY(4096)`. This column must be large enough to accommodate all your session data.

Each type of operation on the database that handles session information (looking up, inserting, updating, and deleting) is performed by a corresponding dedicated connection. Each of these connections has a precompiled SQL statement for higher performance. The following parameters allow you to customize the number of dedicated connections that perform each of the operations.

- `lookupPool` - the number of connections that perform lookup operations (the default is 4 connections).
- `insertPool` - the number of connections that perform insert operations (the default is 4 connections).
- `updatePool` - the number of connections that perform update operations (the default is 4 connections).
- `deletePool` - the number of connections that perform delete operations (the default is 2 connections).

Enabling JdbcSessionManager

You may want to enable `JdbcSessionManager` to change its default parameters. You can also enable `JdbcSessionManager` for a particular context. To enable iPlanet Web Server to use `JdbcSessionManager`, do any of the following:

- Use the [Servlets>Configure Global Servlet Attributes](#) page in the [Server Manager](#) interface.

In the [Session Manager](#) field specify:

```
com.netscape.server.http.session.JdbcSessionManager
```

You can also specify parameters for the session manager in the Session Manager Args field, for example:

```
timeOut=1200,username=mysession,password=mypassword
```

- Edit the file `servlets.properties` in the directory `server_id/config`.

Add a line specifying a value for `servlets.sessionmgr` and a line specifying the parameters for the session manager:

```
servlets.sessionmgr=com.netscape.server.http.session.JdbcSessionManager
servlets.sessionmgr.initArgs=timeOut=1200,username=mysession,password=mypassword
```

- Edit the file `contexts.properties` in the directory `server_id/config`.

Add a line specifying a value for `context.context_name.sessionmgr` and a line specifying the parameters for the session manager:

```
context.global.sessionmgr=com.netscape.server.http.session.JdbcSessionManager
context.global.sessionmgr.initArgs=timeOut=1200,username=mysession,password=mypassword
```

You can change the global context or define a new context and assign specific servlets to it. For more information, see Appendix C, “Properties Files.”

This session manager can only store objects that implement `java.io.Serializable`.

Source Code for JdbcSessionManager

The `JdbcSessionManager` creates a `JdbcSession` object for each session. The source files `JdbcSessionManager.java` and `JdbcSession.java` are in the directory `server_root/plugins/samples/servlets/sessions/JdbcSession`.

The source code files, `JdbcSessionManager.java` and `JdbcSession.java`, are provided so you can use them as the starting point for defining your own session managers and session objects. These files are very well commented.

`JdbcSessionManager` extends `NSHttpSessionManager`. The class file for `NSHttpSessionManager` is in the JAR file `NSServletLayer.jar` in the directory `server_root/bin/https/jar`. The `JdbcSessionManager` implements all the methods in `NSHttpSessionManager` that need to be implemented, so you can use `JdbcSessionManager` as an example of how to extend `NSHttpSessionManager`. When compiling your subclass of `JdbcSessionManager` or `NSHttpSessionManager`, be sure that the JAR file `NSServletLayer.jar` is in your compiler's classpath.

How Do Servlets Access Session Data?

To access the state information stored in a session object, your servlet can create a new session as follows:

```
// request is an HttpServletRequest that is passed to the servlet
SessionClass session = request.getSession(true);
```

The servlet can call any of the public methods in `javax.servlet.http.HttpSession` on the session object. These methods include (among others):

```
getCreationTime
getId
getLastAccessedTime
getMaxInactiveInterval
getValue
```

For more information about the classes `HttpServletRequest` and `HttpSession`, see Appendix G, “API Clarifications,” and the Servlets API Javadoc published by Sun Microsystems at:

<http://java.sun.com/products/servlet/2.2/javadoc/index.html>

Servlet Settings in obj.conf

The iPlanet Web Server 4.1 Administration Server automatically modifies the file `obj.conf` in the `config` directory to load the servlet engine if servlets are enabled. Whenever you make changes to servlet settings by using the Server Manager interface, the system automatically updates `obj.conf` appropriately.

However, in case you are interested in the settings that affect servlets, this appendix describes the directives in `obj.conf` and value settings in `mime.types` that are relevant to servlets.

Directives for Enabling Servlets

The following directives in the `init` section of `obj.conf` load and initialize the servlet engine to enable servlets (for Windows NT):

```
Init fn="load-modules"  
shlib="server_root/bin/https/bin/NSServletPlugin.dll"  
funcs="NSServletEarlyInit,NSServletLateInit,NSServletNameTrans,  
NSServletService" shlib_flags="(global|now)"  
Init fn="NSServletEarlyInit" EarlyInit=yes  
Init fn="NSServletLateInit" LateInit=yes
```

for Unix, the directives are the same except for the following line:

```
Init fn="load-modules"  
shlib="server_root/bin/https/lib/libNSServletPlugin.so"
```

`NSServletEarlyInit` takes an optional parameter `cache_dir` that specifies the location of a temporary cache directory for JSP classes. By default, the directory is named `ClassCache` and goes under your server root directory.

`NSServletLateInit` takes an optional parameter `CatchSignals` that specifies whether or not Java thread dumps are logged. The value is `yes` or `no`.

`NSServletService` takes two optional parameters, `servlet="servlet_name"` and `context="context_name"`. These parameters allow you to define objects in `obj.conf` that generate responses for specific servlets or contexts. You can use one or both parameters in a directive. The servlet or context must be defined in the `servlets.properties` or `contexts.properties` file. You can define an object that pertains to a particular servlet, a particular servlet context, or both. For example, you can direct files of a specific type (in this case, `*.foo`) to a specific servlet (in this case, the `waspl` servlet):

```
<Object name="default">
  NameTrans fn=assign-name name=foo from=*.foo
  ...other directives...
</Object>

<Object name="foo">
  ObjectType fn=force-type type=magnus-internal/servlet
  Service fn="NSServletService" servlet="waspl"
</Object>
```

For another way to direct files of a specific type to a specific servlet, see “Using Regular Expressions in `rules.properties`” on page 67.

For an example of the basic use of `NSServletService`, see the discussion of Service examples in Chapter 2, “Syntax and Use of `Obj.conf`” in the *NSAPI Programmer’s Guide for iPlanet Web Server*.

When servlets are enabled, the following directive appears in the default object:

```
NameTrans fn="NSServletNameTrans" name="servlet"
```

This directive is used for servlet virtual path translations and for the URI cache. Do not delete this line when servlets are enabled.

Also, `obj.conf` always has the following object, which you should not delete:

```
<Object name="servlet">
ObjectType fn=force-type type=text/html
Service fn="NSServletService"
</Object>
```

If you delete this object, you can no longer use the Server Manager interface to enable servlets and modify servlet settings.

For more information, see the *NSAPI Programmer's Guide for iPlanet Web Server*.

Directives for Registered Servlet Directories

For each registered servlet directory, the default object in `obj.conf` has a `NameTrans` directive that assigns the name `ServletByExt` to all requests to access that directory. For example:

```
NameTrans fn="pfx2dir" from="/servlet"
dir="D:/Netscape/Server4/docs/servlet" name="ServletByExt"
```

A separate object named `ServletByExt` has instructions for processing requests for servlets:

```
<Object name="ServletByExt">
ObjectType fn="force-type" type="magnus-internal/servlet"
Service type="magnus-internal/servlet" fn="NSServletService"
</Object>
```

Do not delete this object, even if no servlet directories are currently registered. If this object is deleted, you can no longer use the Server Manager interface to register servlet directories.

JSP

The following line in `mime.types` sets the type for files with the extension `.jsp`:

```
type=magnus-internal/jsp exts=jsp
```

When JSP is enabled, the following directive in `obj.conf` handles the processing of requests for files of type `magnus-internal/jsp` (that is, JSP files):

```
Service fn="NSServletService" type="magnus-internal/jsp"
```


Properties Files

This appendix discusses the purpose and use of the files `servlets.properties`, `rules.properties`, and `contexts.properties`, which reside in the directory `server_id/config`.

`servlets.properties`

The `servlets.properties` file defines global servlet settings and the list of servlets in the system.

Examples of global servlet settings are which servlet to run when the iPlanet Web Server starts up, the reload interval for servlets, and so on. The `servlets.properties` file also specifies configuration information for individual servlets. Configuration information includes the class name, the classpath, and any input arguments required by the servlet.

If you want to specify a virtual path translation for a servlet, the servlet must be configured in the `servlets.properties` file.

You can specify configuration information for servlets either by using the `Servlets>Configure Servlet Attributes` page in the Server Manager interface or by editing `servlets.properties` directly. Whenever you make a change in the `Servlets>Configure Servlet Attributes` page in the Server Manager interface, the system automatically updates `servlets.properties`.

When specifying attributes for a servlet, you specify a name parameter for the servlet. This name does not have to be the name of the class file for the servlet; it is an internal identifier for the servlet. You specify the name of the class file as the value of the code parameter.

Here is a sample `servlets.properties` file:

```
# Servlets Properties
# servlets to be loaded at startup
servlets.startup= hello
# the reload interval for dynamically-loaded servlets and JSPs
# (default is 10 seconds)
servlets.config.reloadInterval=5
# the default document root,
# needed so ServletContext.getRealPath will work
servlets.config.docRoot=d:/Netscape/Server4/docs
# the session manager
servlets.sessionmgr=com.netscape.server.http.session.SimpleSessionManager
# tracker servlet
servlet.tracker.code=MyTrackerServlet
servlet.tracker.classpath=d:/Netscape/Server4/docs/servlet
# demo1 servlet
servlet.demo1.code=Demo1Servlet
servlet.demo1.classpath=d:/Netscape/Server4/docs/demos
servlet.demo1.initArgs=a1=0,b1=3456
servlet.demo1.context=context1
```

rules.properties

The `rules.properties` file defines servlet virtual path translations. For example, you could set up a mapping so that the URL pointing to `/mytest2` invokes the servlet named `demo1` in the `servlets.properties` file. You can specify virtual paths for your servlets either by setting parameters in the `Servlets>Configure Servlet Virtual Path Translation` page of the Server Manager interface or by specifying the paths in the `rules.properties` file.

Note that the name associated with the servlet in `servlets.properties` is used in the file `rules.properties` -- the class name of the servlet does not show up in `rules.properties`. For example, the following lines in `servlets.properties` associate the servlet name `demo1` with the servlet class file `Demo1Servlet.class` in the directory `d:/Netscape/Server4/docs/demos`.

```
# in servlets.properties
# demo1 servlet
servlet.demo1.code=Demo1Servlet
servlet.demo1.classpath=d:/Netscape/Server4/docs/demos
```

The following line in `rules.properties` defines a servlet virtual path translation such that the URL `http://server_id/mytest2` invokes the servlet at `d:/Netscape/Server4/docs/demos/Demo1Servlet.class`.

```
/mytest2=demo1
```

Here is an example of `rules.properties`.

```
# Servlet rules properties
#
# This file specifies the translation rules for invoking servlets.
# The syntax is:
#   <virtual-path>=<servlet-name>
#   or
#   @regular_expression=<servlet-name> (use double back-slashes)
#
# where <virtual-path> is the virtual path used to invoke the servlet,
# and <servlet-name> is the name of the servlet. Surrounding white space
# is ignored. The ordering of the rules is not important, as the longest
# match will always be used first. Use of regular expression can lead to
# a heavy performance penalty
#
##### rules #####
/mytest1=tracker
/mytest2=demo1
```

Using Regular Expressions in `rules.properties`

iPlanet Web Server supports regular expressions in the `rules.properties` file to run a given servlet when the incoming URL matches with a regular expression. However, the example given in the file header is incorrect:

```
# Example:
#
# /simple=SimpleServlet\n
# @.*\\.foo=wasp
```

The `\\` characters are supposed to escape the `(.)` extension. The intent is to run the example `wasp` servlet whenever there is a request for URLs such as `/my/xxx.foo`. However, iPlanet Web Server 4.1 replaces `/` or `\\` characters with `/`, subsequently changing the whole semantics. To get around this, specify the regular expression without the `/` or `\\` characters:

```
@.*[.]foo$=wasp
```

For another way to direct files of a specific type to a specific servlet, see “Directives for Enabling Servlets” on page 61.

contexts.properties

The `contexts.properties` file defines contexts, which allow multiple servlets to exchange data and access each other’s fields. Contexts are useful for defining virtual servers or for code isolation. If no contexts are defined, the default global context is used for all servlets.

NOTE You can use only five hardware virtual servers in the FastTrack edition of iPlanet Web Server.

NOTE All JSPs belong to the default global context. You cannot define custom contexts for JSPs. However, you can change the properties of the global context to affect JSPs.

If the context for a servlet is not defined, the servlet belongs to the global context. You can use the same servlet in multiple contexts.

Only the name of a context is required. Any other unspecified properties are inherited from the global context. You can also change the properties of the global context. The comments in the `contexts.properties` file list the default property values of the global context.

Here is an example of `contexts.properties`.

```
# @(#)contexts.properties (autogenerated)
#
# Contexts Properties:
#
# context.<context_name>.sessionmgr=session manager (some session managers
#               (like MMapSessionManager) can only be instantiated once
#               within the server
# context.<context_name>.sessionmgr.initArgs=list of (name, value) pairs which
#               will represent parameters specific to the session manager
# context.<context_name>.initArgs=list of (name, value) pairs which will be added
#               to this context’s attributes
# context.<context_name>.realPathFromRequest=(true|false) tells the server
#               whether to calculate getRealPath based on docRoot of the
#               context or try to go through normal NSAPI steps
# context.<context_name>.respondCookieVersion=(cookie version) tells the server
#               whether to respond with specific cookie version
# context.<context_name>.sessionExpireOnClose(true|false) tells the server to
#               mark session cookies as directed to expire when the user quits
```

```

#           the browser
# context.<context_name>.includeTransparency=(true|false) tells the server whether
#           to try to honor setting headers from the included servlet
# context.<context_name>.tempDir=path (forward slashes only) - sets up Servlet API
#           2.2 property for the temporary directory
# context.<context_name>.reloadInterval=seconds - time interval within which the
#           server checks for jsp and servlet files being modified (global
#           context only)
# context.<context_name>.javaBufferSize=bytes (deprecated)
# context.<context_name>.bufferSize=bytes - initial http output stream buffer size
# context.<context_name>.docRoot=path (forward slashes only) - this context
#           document root when not specified - web server's document root will be
#           used (default)
# context.<context_name>.inputStreamLengthCheck=(true|false) - makes
#           ServletInputStream to stop reading data, when Content-Length bytes
#           are read
# context.<context_name>.outputStreamFlushTimer=(seconds|0) - forces the stream to
#           flush the data if certain time elapsed since the last flush; 0 -
#           ignore it
# context.<context_name>.uri=context_uri_base - additional URI prefix which
#           services as a context base
# context.<context_name>.host=hostname
# context.<context_name>.ip=ip
# context.<context_name>.authdb=name - authentication database
# context.<context_name>.classpath=name - global classpath for this context
# context.<context_name>.singleClassLoader=(true|false) - tells the servlet engine
#           whether to use a single class loader for all servlets in the context
# context.<context_name>.serverName=name - server instance name
# context.<context_name>.contentTypeIgnoreFromSSI=(true|false) - ignore
#           setContentType when invoked from SSI
# context.<context_name>.parameterEncoding=(utf8,none,auto) - advises the web
#           server on how to decode parameters from forms
#
# <context_name>="global" is reserved for the global context. Every new context
#           will inherit initial settings of the global context
#
# Context properties:
# context.global.sessionmgr=com.netscape.server.http.session.SimpleSessionManager
# context.global.sessionmgr.initArgs=
# context.global.initArgs=initial=0
# context.global.realPathFromRequest=false
# context.global.respondCookieVersion=0
# context.global.sessionExpireOnClose=false
# context.global.includeTransparency=true
# context.global.tempDir=/tmp
# context.global.reloadInterval=5
# context.global.javaBufferSize=0
# context.global.bufferSize=4096
# context.global.docRoot=/foo/bar
# context.global.inputStreamLengthCheck=true
# context.global.outputStreamFlushTimer=0
# context.global.uri=/
# context.global.host=
# context.global.ip=
# context.global.authdb=default
# context.global.classpath=
# context.global.singleClassLoader=false
# context.global.contentTypeIgnoreFromSSI=true
# context.global.parameterEncoding=utf8
#
##### Contexts #####
context1.name=context1

```

The following sections explain a few of the context properties in more detail.

isModifiedCheckAggressive

When you modify a packaged servlet, the new version is not reloaded automatically unless you have done one of the following:

- Set the `isModifiedCheckAggressive` property to `true`, for example:
`context.global.isModifiedCheckAggressive=true`
- Used the `.class` extension when invoking the servlet, for example:
`http://your_server/servlet/HelloWorld.HelloWorldServlet.class`

parameterEncoding

The `context.global.parameterEncoding` property allows you determine the character encoding of servlet parameters. It has the following options:

<code>utf8</code>	Uses UTF-8 encoding, which includes 7-bit ASCII. Set this option if servlet parameters use only UTF-8 characters.
<code>none</code>	Uses the system default encoding. Set this option if the encoding of the servlet parameter data is the same as the system default encoding.
<code>auto</code>	(Default) Tries to figure out the proper encoding from the <code>charset</code> if it is set in the <code>Content-Type</code> header. Otherwise, the system default encoding is used. Set this option to prevent misinterpretation of non-ASCII characters in servlet parameters.

When this property is set to `auto`, the server has to convert native characters into a java byte array before transforming them into the requested encoding. Therefore, performance is slightly better for `utf8` or `none`.

If a servlet uses the `ServletRequest.getParameter` method to retrieve values in form fields having non-UTF-8 characters, the `parameterEncoding` property must be set to `auto` (the default) in the `contexts.properties` file. Otherwise, the values extracted by the `getParameter` method are zeros or are undefined. For more information see “`ServletRequest.getParameter`,” on page 86.

sessionExpireOnClose

If you set `context.global.sessionExpireOnClose=true`, the server does *not* send the expiration time in the Set-Cookie header for the session cookie. If you set this property to `false`, the expiration time is sent. Neither setting causes a specific performance impact.

Regardless of whether the server sends the expiration time in the session cookie, you set the actual session expiration time in these ways:

- Set the session manager timeout. For more information about session managers, see Appendix A, “Session Managers.”
- Use the `HttpSession.setMaxInactiveInterval` method on the session that you get by calling `HttpServletRequest.getSession(false)`. This is optional, and it overrides the session manager timeout. If there is no session, you can create one by calling `getSession(true)`.

singleClassLoader

By default, the `singleClassLoader` property is `false`, which means that each servlet is loaded in a different class loader, even if the servlets share a context. This makes it difficult for two servlets to access each other’s static class members. To load all servlets within a context in the same class loader, set the `singleClassLoader` property to `true`.

contexts.properties

JVM Configuration

The Java Virtual Machine (JVM) works by default without any additional configuration if properly set up.

However, if you need to specify settings for the JVM, such as additional classpath information, you can configure the JVM properties for iPlanet Web Server via the Administrator interface. You can add as many other properties as you want to (up to 64).

You can also configure JVM parameters by editing the `jvm12.conf` configuration file, which resides under the server's `config` directory.

For example, to disable JIT, you can add the following line to `jvm12.conf`:

```
java.compiler=DISABLED
```

Here is an example `jvm12.conf` file. The `jvm.classpath` value must be all on one line in the actual file.

```
[JVMConfig]
#jvm.minHeapSize=1048576
#jvm.maxHeapSize=16777216
#jvm.enableClassGC=0
#jvm.verboseMode=1
#jvm.enableDebug=1
#jvm.printErrors=0
#jvm.option=-Xrunoii
#jvm.profiler=optimizeit
#jvm.disableThreadRecycling=0
#jvm.serializeAttach=0
#jvm.stickyAttach=0
#jvm.trace=5
#java.compiler=NONE
#OPTITDIR=D:/App/IntuitiveSystems/OptimizeIt30D
#jvm.serializeFirstRequest=0
#jvm.include.CLASSPATH=1
```

```
#nes.jsp.forkjavac=0
#nes.jsp.enableddebug=1
#jvm.exitOnAbort=0
jvm.classpath=server_root/plugins/samples/servlets/beans.10/
SDKBeans10.jar;server_root/plugins/samples/servlets/beans/
SDKBeans.jar;server_root/bin/https/jar/xml4j_1_1_9.jar;server_root/
bin/https/jar/Bugbase.jar;server_root/bin/https/jar/Calljsac.jar
```

Generally you should use plain property options (like *name=value*) for the JDK1.2 configuration and *jvm.option=options* for JVM-vendor dependent configurations. There can be multiple occurrences of *jvm.option* parameters.

The *jvm12.conf* file has a configuration parameter, *jvm.stickyAttach*. Setting the value of this parameter to 1 causes threads to remember that they are attached to the JVM, thus speeding up request processing by eliminating *AttachCurrentThread* and *DetachCurrentThread* calls. It can, however, have a side-effect: recycled threads which may be doing other processing can be suspended by the garbage collector arbitrarily.

Thread pools can be used to eliminate this side effect for other subsystems. For more information about thread pools, see “Adding and Using Thread Pools” in Chapter 7, “Configuring Server Preferences,” in the *iPlanet Web Server Administrator’s Guide*.

NOTE	You can use only five threads at a time in the FastTrack edition of iPlanet Web Server.
-------------	---

For information about JVM, see *The Java Virtual Machine Specification* from Sun at: <http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html>

Debugging Servlets and JSPs

This appendix gives guidelines for debugging servlets and JSPs in iPlanet Web Server 4.1.

Servlet Debugging

iPlanet Web Server 4.1 ships with the Java Runtime Environment (JRE) but not the Java Development Kit (JDK) due to licensing restrictions. However, during installation, you can select an option that tells the server to use a JDK if there is one installed elsewhere on your system.

If the server has been instructed to use a JDK, you can do remote servlet debugging. If the server is using the JRE, you need to switch it to using the JDK before you can do remote debugging. For information on instructing the server to use the JDK or the JRE, see the section “Configuring JRE/JDK Paths,” on page 24.

Assuming that the server is using the JDK, you can enable remote debugging by following these steps:

1. Set the following parameters in `jvm12.conf` as appropriate:

```
jvm.enableDebug=1  
java.compiler=NONE
```

2. To send exceptions to the client in addition to the log file, set the following parameter. If the client is a browser, exceptions are displayed in the browser.

```
jvm.trace=7
```

3. On some platforms, you may be required to specify the bootclasspath. For example, for Solaris platforms, if Java 1.2 is in /java, you set it as:

```
jvm.option=-Xbootclasspath:/java/lib/tools.jar:/java/jre/lib/rt.jar
```

4. Start the server manually and record the password for remote debugging (this is displayed on the console).
5. Start the Java debugger:

```
jdb -host your_host -password the_password
```

You should be able to debug your Java classes now using the jdb command.

JSP Debugging

iPlanet Web Server 4.1 uses a public domain JSP compiler developed as part of Apache Software Foundation's Jakarta project. The Jakarta project develops a servlet engine called Tomcat, which includes a JSP compiler called Jasper. The version of the JSP compiler is taken from the Tomcat 3.0 Milestone release. Subsequent versions of iPlanet Web Server will use later versions of the Jasper JSP compiler. For more information, see the following web site:

<http://jakarta.apache.org/>

iPlanet Web Server 4.1 uses a native servlet engine, but uses the Jasper JSP compiler for compiling a JSP page into a servlet. Jasper and iPlanet Web Server 4.1 are not tightly integrated, so you might need to edit the JVM Classpath (in the Configure JVM Attributes page of the Server Manager or in the `jvm12.conf` file) when deploying JSPs using Tag Libraries, beans, and so on.

For information about how to enable JSPs, see "Activating Servlets and JSP," on page 15.

You can debug your JSPs by following these steps:

1. Set the following parameters in `jvm12.conf` as appropriate:

```
java.compiler=NONE
jvm.trace=6
nes.jsp.enableddebug=1
```

2. To send exceptions to the client in addition to the log file, set the following parameter. If the client is a browser, exceptions are displayed in the browser.

`jvm.trace=7`

Setting `java.compiler=NONE` includes line numbers of the Java source code in the verbose output of the log files. Setting `jvm.trace=6` or `jvm.trace=7` enables verbose output from the JSP compiler and the servlet engine. Setting `nes.jsp.enableddebug=1` makes iPlanet Web Server 4.1 generate debuggable Java servlets from the JSPs.

Remote Servlet Profiling

You can use Optimizeit! 3.0 from Intuitive Systems to perform remote profiling on the iPlanet Web Server to discover bottlenecks in server-side performance.

You can purchase Optimizeit! from Intuitive Systems at:

<http://www.optimizeit.com/index.html>

Once Optimizeit! is installed using the following instructions, it becomes integrated into iPlanet Web Server 4.1.

To enable remote profiling, make the following modifications in the `jvm12.conf` files as appropriate:

```
jvm.enableClassGC=0
jvm.option=-Xrunoii # this is only required for JDK1.2
jvm.profiler=optimizeit
java.compiler=NONE
OPTITDIR=optimizeit_root_dir/OptimizeIt30D
```

When the server starts up with this configuration, you can attach the profiler (for further details see the Optimizeit! documentation).

Also, update the `PATH` and `NSES_CLASSPATH` system variables to include the profiler's own jar files and dll files.

NOTE	If any of the configuration options are missing or incorrect, the profiler may experience problems that affect the performance of the iPlanet Web Server.
-------------	---

API Clarifications

This appendix clarifies ways in which the Servlet 2.2 API specification is implemented in iPlanet Web Server 4.1 in the following sections:

- Clarifications for Using Methods
- Other Useful Information

Clarifications for Using Methods

This section provides clarifications for using the following Servlet 2.2 API methods with iPlanet Web Server 4.1:

- `HttpUtils.getRequestURL`
- `HttpServletRequest.getContextPath`
- `HttpServletRequest.getUserPrincipal`
- `HttpServletRequest.isUserInRole`
- `HttpSession.setMaxInactiveInterval`
- `GenericServlet.getInitParameter` and `getInitParameterNames`
- `ServletContext.getAttributeNames`
- `ServletContext.getContext`
- `ServletContext.getRealPath`
- `ServletContext.getResourceAsStream`
- `ServletRequest.getAttribute`
- `ServletRequest.getParameter`

- `RequestDispatcher.forward` and `include`
- `Request.getInputStream` and `getReader`

For the official documentation for the methods discussed here (and for all servlet API methods) see the Servlets API Javadoc published by Sun Microsystems at:

<http://java.sun.com/products/servlet/2.2/javadoc/index.html>

HttpUtils.getRequestURL

```
public static StringBuffer getRequestURL(HttpServletRequest request);
```

This method reconstructs the URL used by the client to make the given request on the server. This method accounts for difference in scheme (such as http, https) and ports, but does not attempt to include query parameters.

This method returns a `StringBuffer` instead of a `String` so that the URL can be modified efficiently by the servlet.

Clarification

To determine the server name part of the requested URL, iPlanet Web Server first tries to use the “Host” header and then looks at the value of `ServerName` in `magnus.conf`. By default, the server name is the machine name, but this value is editable during iPlanet Web Server 4.1 installation. If the server name has been changed, `HttpUtils.getRequestURL` might not return the host name that is needed to reconstruct the request.

For example, suppose the request is `http://abc/index.html`. However, the server name has been changed to `xyz`. In this case, `HttpUtils.getRequestURL` might return `http://xyz/index.html`, which is not the original URL that was requested.

HttpServletRequest.getContextPath

```
public java.lang.String getContextPath(java.lang.String path)
```

Returns the portion of the request URI that indicates the context of the request.

Clarification

This method always returns an empty string.

HttpServletRequest.getUserPrincipal

```
public java.security.Principal getUserPrincipal()
```

Returns a `java.security.Principal` object containing the name of the current authenticated user. If the user has not been authenticated, the method returns `null`.

Clarification

For this method to work, you must set up access control in iPlanet Web Server in the usual manner.

HttpServletRequest.isUserInRole

```
public boolean isUserInRole(java.lang.String role)
```

Returns a boolean indicating whether the authenticated user is included in the specified logical “role.” If the user has not been authenticated, the method returns `false`.

Clarification

This method maps roles to the group names in the underlying user or group database.

HttpSession.setMaxInactiveInterval

```
public void setMaxInactiveInterval(int interval);
```

Sets the amount of time that a session can be inactive before the servlet engine is allowed to expire it.

Clarification

It is not possible to set the maximum inactive interval so that the session never times out. The session always has a timeout value.

If you pass a negative or zero value, the session expires immediately.

GenericServlet.getInitParameter and getInitParameterNames

```
public String getInitParameter(String name);
```

This method returns a String containing the value of the servlet's named initialization parameter, or null if this parameter does not exist.

```
public Enumeration getInitParameterNames();
```

This method returns an enumeration of String objects containing the names of the initialization parameters for the calling servlet. If the calling servlet has no initialization parameters, `getInitParameterNames` returns an empty enumeration.

Clarification

For servlets running on iPlanet Web Server 4.1, the methods `getInitParameter` and `getInitParameterNames` for the class `ServletConfig` only work for servlets that are invoked through virtual path translations. The same restriction applies to the convenience methods of the same names in the class `GenericServlet`, which invoke the corresponding methods on `ServletConfig`.

For information about setting virtual path translations, see the section “Specifying Servlet Virtual Paths,” on page 20.

These methods do not work if the servlet is invoked by a client request that specifies a servlet in a registered servlet directory rather than using a virtual path translation to access the servlet.

ServletContext.getAttributeNames

```
public java.util.Enumeration getAttributeNames()
```

This method returns an enumeration containing the attribute names available within the servlet's context.

Clarification

If you are using `MMapSessions`, iPlanet Web Server truncates names retrieved by `ServletContext.getAttributeNames` to 128 characters.

ServletContext.getContext

```
public ServletContext getContext(String uripath);
```

Returns the servlet context object that contains servlets and resources for a particular URI path, or null if a context cannot be provided for the path.

Clarification

This method only works if both the following conditions are true:

- The servlet whose context is being obtained (that is, the servlet pointed to by `uripath`) has been configured either through the Servlets>Configure Servlet attributes property of the Server Manager interface or by editing `servlets.properties`.
- The servlet whose context is being obtained has been loaded.

iPlanet Web Server 4.1 does not load a servlet specified by a URI when `getContext` is called from another servlet to get the context of an unloaded servlet.

ServletContext.getRealPath

```
public java.lang.String getRealPath(java.lang.String path)
```

Converts the URI path to the servlet into the real path to the servlet.

Clarification

This method does not work if the path is mapped. In this case, use `ServletRequest.getRealPath` instead.

ServletContext.getResourceAsStream

```
public java.io.InputStream getResourceAsStream(java.lang.String path)
```

Returns the resource located at the named path as an `InputStream` object.

Clarification

This method does not work if the path is mapped.

ServletRequest.getAttribute

```
public java.lang.Object getAttribute(java.lang.String name)
```

Returns the value of the named attribute as an object, or returns null if no attribute of the given name exists.

Clarification

`ServletRequest.getAttribute` returns a CGI variable if it exists. However, the `getAttributeNames` method does not show these variables within its enumeration.

NOTE	The <code>HttpServletRequest.getCgiVariable</code> method is not supported.
-------------	---

ServletRequest.getParameter

```
public java.lang.String getParameter(java.lang.String name)
```

Retrieves the value associated with a parameter name.

Clarification

When your form fields contain non-UTF-8 characters, the `parameterEncoding` property must be set to `auto` (the default) in the `contexts.properties` file. Otherwise, the values extracted by the `getParameter` method are zeros or are undefined. For more information, see “`parameterEncoding`,” on page 70.

Because the original encoding used to enter data into form fields is lost when the data is URL-encoded, you must do the following:

- Always set the response content type when sending a form to a client. This ensures that the entire form gets safely to the client.
- When sending form data to a server that uses a different locale than the form fields, you must tell the server the charset before you call the `getParameter` method, as follows:
 - If the same servlet or JSP generates and processes the form, set the response content type. For servlets, explicitly set it as in this example:

```
res.setContentType("text/plain; charset=Shift_JIS");
```

For JSPs, set the response content type using a page directive, for example:

```
<%@ page contentType="text/html; charset=gb2312"%>
```

- If the servlet or JSP that generated the form is different than the one processing the form, use a hidden field in the form called `j_encoding`, for example:

```
<input type="hidden" name="j_encoding" value="US_ASCII">
```

RequestDispatcher.forward and include

```
public void forward(ServletRequest request, ServletResponse
response) throws ServletException, IOException;
```

Used for forwarding a request from this servlet to another resource on the web server. This method is useful when one servlet does preliminary processing of a request and wants to let another object generate the response.

The request object passed to the target object will have its request URL path and other path parameters adjusted to reflect the target URL path of the target object.

You cannot use this method if a `ServletOutputStream` object or `PrintWriter` object has been obtained from the response. In that case, the method throws an `IllegalStateException`.

```
public void include(ServletRequest request, ServletResponse
response) throws ServletException, IOException;
```

Used for including the content generated by another server resource in the body of a response. In essence, this method enables programmatic server-side includes. The request object passed to the target object reflects the request URL path and path info of the calling request. The response object only has access to the calling servlet's `ServletOutputStream` object or `PrintWriter` object.

An included servlet cannot set headers. If the included servlet calls a method that needs to set headers (such as cookies), the method is not guaranteed to work. As a servlet developer, you must ensure that any methods that might need direct access to headers are properly resolved. To ensure that a session works correctly, start the session outside the included servlet, even if you use session tracking.

Clarification

In iPlanet Web Server 4.1, the `dispatcher.forward` method may or may not throw an `IllegalStateException` when either `Writer` or `OutputStream` have been obtained. This behavior follows the 2.2 draft and is needed for JSP error page handling. It throws the exception only if the actual data has been flushed out and sent to the client. Otherwise, the data pending in the buffer is simply discarded.

The `forward` and `include` methods may throw a `ServletException` if the target URI is identified as an unsafe URI (that is, it includes insecure path characters such as `//`, `/.//`, `/../` and `/.`, `/..` (and also `./` for NT) at the end of the URI.

Request.getInputStream and getReader

There are two ways for a servlet to read the raw data posted by a client:

- by obtaining the `InputStream` through the `request.getInputStream` method, an older method.
- by obtaining a `BufferedReader` through the `request.getReader` method, a method in use since 2.0.

Clarification

A servlet hangs if it attempts to use an `InputStream` to read more data than is physically available. (To find how much data is available, use `request.getContentLength`.) However, if the servlet reads data using a `BufferedReader` returned from a call to `getReader`, the allowed content length is automatically taken into account.

You can also set the `inputStreamLengthCheck` parameter to `true` in the `contexts.properties` file to prevent this problem.

Other Useful Information

This section contains information about the following topics:

- Database Connection Pooling
- Fetching the Client Certificate

Database Connection Pooling

Database connection pooling enhances the performance of servlet or JSP database interactions. There are several JDBC 2.0 compatible drivers that support connection pooling, for example Oracle 8i update and CloudScape 3.0.

Fetching the Client Certificate

When you enable SSL and require client certificate authorization, your servlets have access to the client certificate as shown in the following example:

```
if (request.isSecure()) {
    java.security.cert.X509Certificate cert;
    cert =
req.getAttribute("javax.servlet.request.X509Certificate");
    if (cert != null) {
        // Get the Distinguished Name for the user
        java.security.Principal userDn = cert.getSubjectDN();
    }
}
```

The userDn is the fully qualified Distinguished Name for the user.

Index

A

- about this book 7
- accessing
 - JSP 14
 - request object in JSP 37
 - servlets 14
- accessTimeColumn
 - parameter for JdbcSessionManager 58
- activating
 - JSP 15
 - servlets 15
- API
 - clarifications 81
- API reference
 - JavaBeans 12
 - JSP 12
 - servlets 10

B

- beans 12
 - example of accessing from JSP 38
 - examples directory 31
- beans.10
 - examples directory 31
- bookstore
 - examples directory 31

C

- cache directories 26
- cache_dir
 - optional parameter to NSServletEarlyInit 62
- CatchSignals
 - optional parameter to NSServletLateInit 62
- CGI variable
 - returning in a servlet 86
- clarifications
 - of API 81
- ClassCache 26
- classpath
 - for JDK 25
 - for JVM 27
 - for servlets 19
 - JVM parameter 27
- client certificate, fetching 89
- compiler
 - JVM parameter 27
- compiling
 - servlets 10
- configuring
 - global servlet attributes 16
 - individual servlets 18
 - JRE/JDK paths 24
 - JVM 27, 73
- connection pooling, database 89
- context
 - optional parameter to NSServletService 62
- contexts 23, 68
- contexts.properties 62, 68

- cookies method 87
- custom tag library, JSP 42

D

- database connection pooling 89
- debugging
 - enabling 28
 - JSPs 76
 - servlets remotely 75
- deletePool
 - parameter for JdbcSessionManager 58
- deleting
 - version files 26
- directives
 - for enabling servlets 61
- directories
 - for servlets 17
- doGet method 33, 34

E

- enable class GC
 - JVM parameter 28
- enable debug
 - JVM parameter 28
- enabling
 - JdbcSessionManager 58
 - JDK or JRE 24
 - JSP 14
 - servlets 15
 - session managers 54
 - SimpleSessionManager 56
- examples
 - form that invokes JSP 39
 - JSP 36
 - JSP accessing beans 38
 - JSP custom tag library 42
 - location in the build 31
 - servlet that parses input parameters 34
 - servlets 32

- shipped in the build 31
- simple servlet 32
- virtual servlet path 21
- exceptions, sending to the client 75, 77

F

- file extensions
 - .class 17
 - .jsp 14, 64
- forms
 - example of invoking JSP 39
- forward 87

G

- garbage collection
 - enabling 28
- GenericServlet.getInitParameter 84
- getAttribute 86
- getAttributeNames 86
- getCgiVariable 86
- getContext 85
- getContextPath 82
- getInitParameter 84
- getInitParameterNames 84
- getParameter 70, 86
- getRealPath 85
- getResourceAsStream 85
- getSession 71
- getUserPrincipal 83
- global servlet attributes
 - configuring 16

H

- HttpServlet 33, 34
- HttpServletRequest

- more info 60
- HttpServletRequest.getCgiVariable 86
- HttpServletRequest.getContextPath 82
- HttpServletRequest.getSession 71
- HttpServletRequest.getUserPrincipal 83
- HttpServletRequest.isUserInRole 83
- HttpSession
 - more info 60
- HttpSession.setMaxInactiveInterval 71, 83
- HttpUtils.getRequestURL 82

I

- include 87
- input parameters
 - accessing in JSP 40
- insertPool
 - parameter for JdbcSessionManager 58
- installing
 - JRE or JDK 12
 - servlets 14
- Intuitive Systems
 - web site 79
- isModifiedCheckAggressive context property 70
- isUserInRole 83

J

- jars
 - classpath 27
- Java Development Kit
 - see JDK
- Java Runtime Environment
 - see JRE
- Java Servlet API 10
- Java Virtual Machine
 - see JVM
- Java Virtual Machine Specification 74
- JavaBeans 12
 - specifying classpath 27

- JavaServer Pages
 - see JSP
- JDBC driver 57, 89
- JdbcSession
 - source code 59
- JdbcSessionManager 57
 - enabling 58
 - source code 59
- JDK 12
 - downloading 13
 - enabling 24
 - installing 12
 - setting path 24
 - versions 24
- JIT 27
- JRE 12
 - enabling 24
 - installing 12
 - setting path 24
- JSP 11
 - accessing beans example 38
 - accessing input parameters 40
 - accessing Java 12
 - accessing request object 37
 - activating 15
 - API reference 12
 - cache directory 26
 - custom tag library 42
 - debugging 76
 - enabling 14
 - example of invoking from forms 39
 - examples 36
 - serving 14
 - specifying classpath for beans 27
 - using 9
 - using Server Manager interface 15
- JSP tags
 - useBean 41
- JSP.092
 - examples directory 31
- JSP.10
 - examples directory 31
- just-in-time compiler 27
- JVM
 - catching thread dumps 62
 - configuration 73

- configuring 27
- more info 74
- specification 74

JVM parameters

- classpath 27
- compiler 27
- enable class GC 28
- enable debug 28
- maximum heap size 27
- minimum heap size 27
- option 27
- OPTITDIR 27
- profiler 27
- verbose mode 28

jvm12.conf 27, 73

L

lookupPool

- parameter for JdbcSessionManager 58

M

magnus-internal/jsp 64

make

- examples directory 31

maximum heap size

- JVM parameter 27

maxSessions

- parameter for SimpleSessionManager 55

minimum heap size

- JVM parameter 27

MMapSessionManager 26

multiple servlet directories 17

N

NSES_JDK 25

NSES_JRE 25

NSHttpSessionManager 57, 59

NSServletEarlyInit 61

NSServletLateInit 61

NSServletLayer.jar 57, 59

NSServletService 61, 62

O

obj.conf 61

ODBC driver 57

Optimizeit!

- purchasing 79

option

- JVM parameter 27

OPTITDIR

- JVM parameter 27

Output from servlets 24

P

parameterEncoding context property 70

password

- parameter for JdbcSessionManager 57

path

- to JRE or JDK 13, 24

path translations

- specifying 20

persistent session manger 57

pooling of database connections 89

preface 7

profiler

- JVM parameter 27

profiling

- servlets remotely 79

provider

- parameter for JdbcSessionManager 57

R

- reaper method
 - SimpleSessionManager 55
- reaperActive
 - parameter for JdbcSessionManager 58
- reapInterval
 - parameter for SimpleSessionManager 55
- registered servlet directories 17
- registering
 - individual servlets 18
 - servlet directories 17
- reload interval 16
- reloading
 - servlets 16
- remote profiling 79
- remote servlet debugging 75
- request object
 - accessing in JSP 37
- Request.getInputStream 88
- Request.getReader 88
- RequestDispatcher.forward 87
- RequestDispatcher.include 87
- rules.properties 66

S

- Server Manager interface
 - for managing servlets and JSP 15
- serving
 - servlets and JSP 14
- servlet
 - optional parameter to NSServletService 62
- Servlet Args 19
- Servlet Classpath 19
- Servlet Code (class name) 19
- servlet directories 17
 - default directory 17
- Servlet Name 19
- ServletByExt 16
- ServletContext.getAttributeNames 84
- ServletContext.getContext 85

- ServletContext.getRealPath 85
- ServletContext.getResourceAsStream 85
- ServletRequest.getAttribute 86
- ServletRequest.getAttributeNames 86
- ServletRequest.getParameter 70, 86
- ServletRequest.getRealPath 85
- servlets 10
 - accessing from clients 14
 - accessing session data 60
 - activating 15
 - API clarifications 81
 - API reference 10
 - cache directories 26
 - compiling 10
 - configuring global attributes 16
 - configuring individual servlets 18
 - debugging remotely 75
 - directing files of a specific type to a specific servlet 62, 67
 - example of accessing 17
 - examples 31, 32
 - non-ASCII parameters 70
 - output 24
 - packaged 18
 - reloading automatically 70
 - parsing input parameters 34
 - reloading 16, 70
 - remote profiling 79
 - serving 14
 - session managers 53
 - sessions 53
 - specifying virtual paths 20
 - using 9
 - using a single class loader 71
 - using Server Manager interface 15
 - virtual path translation 14
- Servlets API Javadoc 60, 82
- servlets.jar 10
- servlets.properties 62, 65
- session data
 - accessing 60
- Session Manager 16
- Session Manager Args 16
- session managers 53
 - JdbcSessionManager 57

- persistent 57
 - SimpleSessionManager 55
 - specifying 54
 - timeout, sending in cookie 71
- SessionData 26
- sessionExpireOnClose context property 71
- sessionIdColumn
 - parameter for JdbcSessionManager 58
- sessions 53
 - accessing from servlets 60
 - examples directory 32
 - overview 53
- setMaxInactiveInterval 71, 83
- simple servlet example 32
- SimpleSession
 - source code 56
- SimpleSessionManager 55
 - enabling 56
 - source code 56
- singleClassLoader servlet property 71
- snoop.jsp 37
- source code
 - JdbcSession 59
 - JdbcSessionManager 59
 - SimpleSession 56
 - SimpleSessionManager 56
- specifying
 - JDK or JRE 13
 - servlet directories 17
 - session managers 54
 - virtual servlet paths 20
- Startup Servlets 16
- static class members, sharing 71

T

- table
 - parameter for JdbcSessionManager 57
- tag library, custom JSP 42
- taglibs
 - examples directory 32
- thread pools 15, 29, 74

- timeOut
 - parameter for JdbcSessionManager 57
 - parameter for SimpleSessionManager 55
- tools
 - examples directory 32

U

- unsafe URIs 88
- updatePool
 - parameter for JdbcSessionManager 58
- url
 - parameter for JdbcSessionManager 57
- useBean
 - JSP tag 41
- username
 - parameter for JdbcSessionManager 57
- using
 - servlets and JSP 9

V

- valueColumn
 - parameter for JdbcSessionManager 58
- verbose mode
 - JVM parameter 28
- version files 26
 - deleting 26
- virtual paths
 - example 21
 - specifying 20