Programmer's Guide to Servlets

iPlanet Web Server, Enterprise Edition

Version 6.0

816-1381-01
May 2001

Copyright © 2001 Sun Microsystems, Inc. Some preexisting portions Copyright © 2001 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.

This product includes software developed by Apache Software Foundation (http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved.

This product includes software developed by the University of California, Berkeley and its contributors. Copyright © 1990, 1993, 1994 The Regents of the University of California. All rights reserved.

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 © 2001 Sun Microsystems, Inc. Pour certaines parties préexistantes, Copyright © 2001 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

Contents

About This Book

This book discusses how to enable and install Java servlets and JavaServer Pages (JSP) in iPlanet™ Web Server, Enterprise Edition 6.0.

This book has the following chapters and appendices:

- Chapter 1, "Servlets and JavaServer Pages"

  This chapter introduces web applications, servlets, and JavaServer Pages.

- Chapter 2, "Web Applications"

  This chapter discusses the structure of web applications and how to deploy them in iPlanet Web Server 6.0. It also lists example web applications and describes the `web-apps.xml` file, which configures web applications for virtual servers in iPlanet Web Server 6.0.

- Chapter 3, "Using Servlets"

  This chapter discusses how to enable and install servlets in iPlanet Web Server 6.0.

- Chapter 4, "Using JavaServer Pages"

  This chapter discusses how to enable and install JavaServer Pages in iPlanet Web Server 6.0, including how to install the JDK.

- Chapter 5, "Debugging Servlets and JSPs"

  This chapter discusses how to debug servlets and JSPs, and includes a section on debugging using Forte for Java.

- Chapter 6, "Session Managers"

  This chapter discusses the session managers provided with iPlanet Web Server and explains how to customize session behavior to suit your own needs.

- Chapter 7, "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.

- Chapter 8, "Legacy Servlet and JSP Configuration"

  This chapter discusses how to configure the default virtual server as in iPlanet Web Server 4.*x*, and the `servlets.properties`, `rules.properties`, and `contexts.properties` files.

- Appendix A, "Servlet Settings in magnus.conf and obj.conf"

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

- Appendix B, "Converting SSJS Applications"

  This appendix discusses how to convert LiveWire applications to JSPs.

- Appendix C, "JVM Configuration"

  This appendix discusses how to manually specify JVM configuration information.

- Appendix D, "Remote Servlet Profiling"

  This appendix discusses how to enable remote profiling for servlets.

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

Chapter 1

Servlets and JavaServer Pages

iPlanet Web Server 6.0 supports servlets and JavaServer Pages (JSPs). This chapter gives a brief overview of servlets and JSPs in iPlanet Web Server 6.0.

The sections in this chapter are:

- Web Applications
- Servlets
- JavaServer Pages

iPlanet Web Server 6.0 runs with a specific version of the JRE or JDK that is operating-system dependent. For more information, see “What Does the Server Need to Run JSP?,” on page 37.

Web Applications

iPlanet Web Server 6.0 supports the Servlet 2.2 API specification, which allows servlets and JSPs to be included in web applications.

A web application is a collection of servlets, JavaServer Pages, HTML documents, and other web resources which might include image files, compressed archives, and other data. A web application may be packaged into an archive (a WAR file) or exist in an open directory structure.

For more details about web application support in iPlanet Web Server, see Chapter 2, “Web Applications.”

Although using web applications is recommended, you can still configure servlets and JSPs as in iPlanet Web Server 4.*x*. For more information, see Chapter 8, “Legacy Servlet and JSP Configuration.”

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 6.0 supports the Java Servlet Specification version 2.2 (including Web Application and WAR file support).

NOTE Servlet API version 2.2 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`

JavaServer Pages

iPlanet Web Server 6.0 supports JavaServer Pages (JSP) Specification version 1.1.

A JSP is a page, much like an HTML page, that can be viewed in a web browser. However, in addition to HTML tags, it can include a set of JSP tags and directives intermixed with Java code that extend the ability of the web page designer to incorporate dynamic content in a page. These additional features 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.

iPlanet Web Server supports precompilation of JSPs, however, and this is recommended for production servers.

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.

iPlanet Web Server 6.0 supports JSP tag libraries and standard portable tags.

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`

Chapter 2

Web Applications

iPlanet Web Server 6.0 supports the Servlet 2.2 API specification, which allows servlets and JSPs to be included in web applications.

A web application is a collection of servlets, JavaServer Pages, HTML documents, and other web resources which might include image files, compressed archives, and other data. A web application may be packaged into an archive (a WAR file) or exist in an open directory structure. For more information about web applications, see the Servlet 2.2 API specification:

`http://java.sun.com/products/servlet/index.html`

This chapter describes how web applications are supported in iPlanet Web Server, and includes the following sections:

- Web Application Structure
- Dynamic Reconfiguration
- web.xml Clarifications
- The web-apps.xml File and Virtual Servers
- Deploying a Web Application using wdeploy
- Web Application Examples

Web Application Structure

Web Applications have a directory structure, all accessible from a mapping to the application's document root (for example, `/catalog`). The document root contains JSP files, HTML files, and static files such as image files.

A special directory under the document root, `WEB-INF`, contains everything related to the application that is not in the public document tree of the application. No file contained in `WEB-INF` can be served directly to the client. The contents of `WEB-INF` include:

- `/WEB-INF/web.xml`, an XML-based deployment descriptor that specifies the web application configuration, including mappings, initialization parameters, and security constraints.
- `/WEB-INF/classes/*`, the directory for the servlet and utility classes.
- `/WEB-INF/lib/*.jar`, the directory for the JAR files containing servlets, beans, and other utility classes.

A WAR (web application archive) file contains a complete web application in compressed form. iPlanet Web Server cannot access an application in a WAR file. You must uncompress a web application (deploy it using the `wdeploy` utility) before iPlanet Web Server can serve it.

Web application configuration includes two parts:

- The `web.xml` file, which is the standard Servlets 2.2 deployment descriptor. Each web application has its own `web.xml` file. For information about `web.xml`, see the Servlet 2.2 API specification:

  `http://java.sun.com/products/servlet/index.html`

- The `web-apps.xml` file, which is specific to iPlanet Web Server. Each virtual server has its own `web-apps.xml` file, which configures all the applications running in that virtual server.

Dynamic Reconfiguration

When you make changes to a web application, you do not need to restart the server. Changes are reloaded automatically at a frequency set in the `reload-interval` attribute of the `class-loader` element in the `web-apps.xml` file.

When you make changes to the `web-xml` or `web-apps.xml` file, you also do not need to restart the server. However, you must apply the changes by clicking the Apply link and then clicking the Load Configuration Files button on the Apply Changes screen.

For more information about dynamic reconfiguration, see Chapter 1, “Basics of Server Operation” in the *NSAPI Programmer’s Guide for iPlanet Web Server*.

If you add or remove a web application, servlet, or JSP, you must restart the server.

web.xml Clarifications

This section provides clarifications for using the `web.xml` file with iPlanet Web Server 6.0.

login-config

If there is a security constraint and no `login-config` element is present, or if no `auth-method` is specified, the `auth-method` defaults to `BASIC`. For `BASIC` authentication, if no `realm-name` is specified, the `realm-name` defaults to `iWS Web Container`. If `auth-method` is not `FORM`, the `form-login-config` element is ignored even if specified.

security-constraint

if no `url-pattern` is specified, no constraints are applied. If no `http-method` is specified in a `web-resource-collection` subelement, the constraint is applied to all HTTP methods. If no `transport-guarantee` is specified in a `user-data-constraint` subelement, it defaults to `NONE`. If no `auth-constraint` subelement is present or if no `role-name` is specified, no user is allowed any access.

session-timeout

If `session-timeout` is specified in `web.xml`, this timeout value overrides any `timeOut` parameter value specified in the `session-manager` element in `web-apps.xml`. This override does not apply when you configure a single session manager that all web applications in a virtual server use. It only applies when you:

- Do not configure a session manager for all web applications in a virtual server (the default behavior) Each web application has its own session manager.
- Configure a web-application-specific session manager in addition to the virtual-server-wide one. In that case, only the web-application-specific session manager's `timeOut` value is overridden.

Note that `session-timeout` value is specified in *minutes* while `session-manager timeOut` parameters are specified in *seconds*.

For more information about session managers, see Chapter 6, "Session Managers."

The web-apps.xml File and Virtual Servers

The `web-apps.xml` file defines contexts for a set of web applications running in a virtual server. The context information includes a context path of the web application and other properties such as how it handles session management or authentication.

The `web-apps.xml` file follows the standard J2EE deployment descriptor format: a well-formed .XML document specified by a `.dtd` file. The `web-apps.xml` file contains a set of web applications defined by their context path and physical location.

Each web application may also define a number of configuration elements specific to iPlanet Web Server that customize how the application is serviced. For example, a web application can define pluggable session management or specify a java compiler. The `web-apps.xml` file allows users to define a set of global configuration elements and share (or override) them across multiple web applications.

Each `web-apps.xml` file must be referenced in the `server.xml` file, which defines virtual servers. For more information about `server.xml`, see the *NSAPI Programmer's Guide for iPlanet Web Server*.

The Default Context for a Virtual Server

When you create a new virtual server, an empty `web-apps.xml` file is created. You can use the default context that is created automatically, or you can create your own by modifying the `web-apps.xml` file.

If a `servlets.properties` file is present in the default virtual server (in the `config` directory of the server instance), an iPlanet Web Server 4.*x* servlet context is created instead of a web application context. For more information about legacy configuration, see Chapter 8, "Legacy Servlet and JSP Configuration."

Example server.xml File

The `server.xml` file contains two variables, `webapps_enable` and `webapps_file`, that are relevant to web applications.

`webapps_enable`	A `true` or `false` variable that allows you to enable or disable web applications for a virtual server. If not present in a virtual server definition, web applications are enabled by default.

`webapps_file`	The path to the `web-apps.xml` file for a virtual server.

The following `server.xml` file uses these variables:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- declare any variables to be used in the obj.conf file in the
   ATTLIST below -->
<!DOCTYPE SERVER SYSTEM "server.dtd" [
<!ATTLIST VARS
   docroot CDATA #IMPLIED
   adminusers CDATA #IMPLIED
   webapps_file CDATA #IMPLIED
   webapps_enable CDATA #IMPLIED
   accesslog CDATA #IMPLIED
   user CDATA #IMPLIED
   group CDATA #IMPLIED
   chroot CDATA #IMPLIED
   dir CDATA #IMPLIED
   nice CDATA #IMPLIED
>
]>

<SERVER legacyls="ls1">
   <VARS accesslog="/iws60/https-server.iplanet.com/logs/access"/>
   <LS id="ls1" ip="0.0.0.0" port="80" security="off"
   acceptorthreads="1">
      <CONNECTIONGROUP id="group1" matchingip="default"
      servername="acme.com" defaultvs="acme.com"/>
   </LS>
   <MIME id="mime1" file="mime.types"/>
   <ACLFILE id="acl1"
   file="/iws60/httpacl/generated.https-server.iplanet.com.acl"/>
   <VSCLASS id="defaultclass" objectfile="obj.conf"
   rootobject="default">
      <VARS docroot="/iws60/docs"/>
      <VS id="acme.com" connections="group1" mime="mime1"
      aclids="acl1">
         <VARS webapps_file="web-apps.xml" webapps_enable="on"/>
         <USERDB id="default" database="default"/>
      </VS>
   </VSCLASS>
</SERVER>
```

For more information about the `server.xml` file, see Chapter 8, “Virtual Server Configuration Files” in the *NSAPI Programmer’s Guide for iPlanet Web Server*.

Example web-apps.xml File

Note that the `!DOCTYPE` declaration must be present and of the following format:

```
<!DOCTYPE vs  PUBLIC "-//Sun Microsystems, Inc.; iPlanet//DTD
Virtual Server Web Applications 6.0//EN"
"http://developer.iplanet.com/webserver/dtds/iws-webapps_6_0.dtd">
```

If for some reason if this URL is not accessible, the DTD file is located here:

`"file:/`*server_root*`/bin/https/dtds/iws-webapps_6_0.dtd"`

On Windows NT, be sure to include the drive letter, as follows:

`"file:/`*drive*`:/`*server_root_path*`/bin/https/dtds/iws-webapps_6_0.dtd"`

The following `web-apps.xml` file configures a session manager and one web application:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- iWS 6.0 specific web application configuration. -->
<!DOCTYPE vs  PUBLIC "-//Sun Microsystems, Inc.; iPlanet//DTD
Virtual Server Web Applications 6.0//EN"
"http://developer.iplanet.com/webserver/dtds/iws-webapps_6_0.dtd">

<vs>

<!-- Define global configuration -->

<!- Configure a session manager and tracking configuration -->
<session-manager
   class='com.iplanet.server.http.session.IWSSessionManager'
   <init-param>
      <param-name>maxSessions</param-name>
      <param-value>1000</param-value>
   </init-param>
   <init-param>
      <param-name>timeOut</param-name>
      <param-value>1800</param-value>
   </init-param>
   <init-param>
      <param-name>reapInterval</param-name>
      <param-value>600</param-value>
   </init-param>
   <init-param>
      <param-name>session-data-dir</param-name>
      <param-value>/net/dotcom.com/sessions</param-value>
   </init-param>
</session-manager>
```

```
<session-tracking use-cookies="true" />

<!-- Define the web applications for this virtual server-->

<!-- catalog application -->
<web-app uri="/catalog" dir="/webapps/catalog">

<!-- Specify a tempory directory. A path returned in the
"javax.servlet.context.tempdir" property; -- defaults to
WEB-INF/tmp. -->
<tempdir dir='/var/catalog/tmp'/>

<!-- reload classes at every 5 minutes; also include mycatlog.jar
file in the classpath -->
<class-loader reload-interval='300'
classpath='/home/work/mycatalog.jar' />

</web-app>

</vs>
```

web-apps.xml Element Reference

This section shows the elements allowed in a `web-apps.xml` file. These elements are defined in the master `dtd` file.

auth-native

Configures a specific native user/group database for authentication and role mapping. If this element is not specified, authentication is enabled using the native default authentication database.

Subelements: none

Attributes:

`authdb`	The native authentication database. This database must also be defined in the `server.xml` file in the `database` attribute of a `USERDB` element, and in the `dbswitch.conf` file.

class-loader

The class loader for the virtual server or the web application. There is a default class loader for the virtual server if you don't specify one. Classes loaded by the virtual-server-level class loader are not dynamically reloaded.

You can use the `ServletContext.getAttribute` method to retrieve the class loader and the class loader's classpath or reload interval for a context. For more information, see Chapter 7, "API Clarifications."

Subelements: none

Attributes:

`classpath`	The classpath used by the class loader.
`delegate`	Specifies that the class loader for the virtual server or system is called first to load a class. Allowed values are `true` and `false`. The default is `false`.
`reload-interval`	The time interval in seconds within which the server checks for web applications being modified. The default is `30`.

description

A description of a parameter. Used within an `init-param` element. iPlanet Web Server ignores this element.

Subelements: none

Attributes: none

filter, filter-mapping

These elements implement the Filter API from the Servlet 2.3 specification. Used within a `web-app` element.

Although iPlanet Web Server 6.0 supports only the Servlet 2.2 API in the `web.xml` file, the Filter API from the Servlet 2.3 specification is available in the `web-apps.xml` file.

The `filter` and `filter-mapping` elements implement the Filter API. They are both subelements of a `web-app` element in the `web-apps.xml` file. Except for their file location, `filter` and `filter-mapping` are as described in the Servlet 2.3 specification. For more information, see:

`http://java.sun.com/products/servlet/index.html`

form-login-session

Configures form-based authentication for single sign-on across all web applications in a virtual server. If not present, the default virtual server level session manager is used.

Subelements: `session-manager`

Attributes:

`cookie-name`	The name of the cookie that tracks the session ID. The default is `iwsformloginid`.
`timeOut`	The session timeout in seconds. The default is `600` (10 minutes).

init-param

Specifies an initialization parameter for the containing element. The attributes of `init-param` depend on the object referenced by the containing element.

For example, if the containing element is `session-manager` and the session manager is `IWSSessionManager`, the attributes of `init-param` are the initialization parameters of `IWSSessionManager`.

Subelements: `param-name`, `param-value`, `description`

Attributes: variable

jsp-servlet

Configures JSP compilation behavior. For a description of the initialization parameters, see “The JSP Command-Line Compiler,” on page 43.

Subelements: `init-param`

Attributes:

`enable`	Enables JSP. Allowed values are `true` and `false`. The default is `true`.

param-name

The name of a parameter. Used within an `init-param` element.

Subelements: none

Attributes: none

param-value

The value of a parameter. Used within an `init-param` element.

Subelements: none

Attributes: none

parameter-encoding

Advises the web server on how to decode parameters from forms.

Subelements: none

Attributes:

`enc`	Allowed values are `auto` (the default), `none`, or a specific encoding such as `utf8` or `Shift_JIS`:	
	any supported java character encoding	A specific encoding, such as `utf8` or `Shift_JIS`. Set this option if you know the encoding that servlet parameters use. A complete list is available here: `http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html`
	`none`	Uses the system default encoding. Set this option if the encoding of the servlet parameter data is the same as the system default encoding.

	`auto`	(Default) Tries to figure out the proper encoding from, in order, 1) the `charset` if it is set in the `Content-Type` header, 2) the `parameterEncoding` attribute (see "ServletRequest.setAttribute," on page 73), then 3) a hidden form field defined in `form-hint-field`. 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 `none` or a specific encoding.
`form-hint-field`	The name of the hidden field in the form that specifies the encoding. The default is `j_encoding`.	

If a servlet uses the `ServletRequest.getParameter` method to retrieve values in form fields having non-UTF-8 characters, the `enc` attribute must be set to `auto` (the default). Otherwise, the values extracted by the `getParameter` method are zeros or are undefined. For more information see "ServletRequest.getParameter," on page 74.

The `enc` attribute setting applies to how a servlet processes parameters after the iPlanet Web Server receives a request. The URI that is sent to the server must consist of only the standard ASCII set, especially if the request method is GET. All other characters must be encoded.

For example, to encode a backslash, you would replace the backslash with a `%` followed by the hexadecimal number corresponding to the backslash in the ASCII character set, `5c`. Therefore, `vw\xyz` becomes `vw%5cxyz`.

For more information, read section 2.2 of the following document from the URI working group of the Internet Engineering Task Force:

`http://www.ietf.org/rfc/rfc1738.txt`

response-buffer

Configures the initial and default size of the HTTP servlet's response buffer. A servlet can reconfigure its response buffer size using the `setBufferSize` method of the `ServletResponse` object.

Subelements: none

Attributes:

`flush-timeout`	Forces the stream to flush the data if the specified number of seconds has elapsed since the last flush. If set to `0` (the default) or a negative number, the output stream doesn't force a flush unless the buffer is full.
`size`	The buffer size in bytes. The default is `8192`.

response-cookie

Tells the server to respond with a specific cookie version.

Subelements: none

Attributes:

`version`	The cookie version. The default is `0`.

role-mapping

Maps `role-name` values from `web.xml` to LDAP users, groups, or roles.

Subelements: none

Attributes:

`map-to`	Specifies whether to map `role-name` values from `web.xml` to LDAP users or groups. Values are `group` (the default) and `user`.

session-cookie

Sets parameters for the session cookie.

Subelements: none

Attributes:

`domain`	If this attribute is present, its value is tagged onto the cookie. There is no default value.
`is-secure`	If set to `true`, the server sends the `secure` attribute in the session cookie if the request came in a secure connection. The default is `false`.

session-manager

The session manager for the web application. See Chapter 6, “Session Managers” for the initialization parameters for each session manager.

If `session-timeout` is specified in `web.xml`, it overrides any session manager’s `timeOut` initialization parameter. For details, see “session-timeout,” on page 15.

Subelements: `init-param`

Attributes:

`class`	The class for the session manager.

session-tracking

Determines the method of session tracking.

Subelements: none

Attributes:

`use-cookies`	Uses cookies for session tracking if `true` (the default). Allowed values are `true` and `false`.
`use-url-rewriting`	Uses URL rewriting for session tracking if `true` (the default). Allowed values are `true` and `false`.

tempdir

A temporary directory used by the web application.

Subelements: none

Attributes:

`dir`	The temporary directory.

vs

The top-level element in the `web-apps.xml` file. Subelements other than `web-app` set defaults for all web applications.

Subelements: `auth-native`, `class-loader`, `form-login-session`, `jsp-servlet`, `parameter-encoding`, `response-buffer`, `response-cookie`, `role-mapping`, `session-manager`, `session-tracking`, `session-cookie`, `tempdir`, `web-app`

Attributes: none

web-app

The web application. A web application is packaged in a WAR file and can contain servlets, JSPs, HTML pages, class files, and other resources of an application.

The subelements of a `web-app` element override the equivalent subelements of the containing `vs` element for that web application.

Subelements: `auth-native`, `class-loader`, `filter`, `filter-mapping`, `jsp-servlet`, `parameter-encoding`, `response-buffer`, `response-cookie`, `role-mapping`, `session-manager`, `session-tracking`, `session-cookie`, `tempdir`

Attributes:

`dir`	The directory where the web application contents are located.
`uri`	The URI that clients use to access the web application. This URI can be a regular expression.

Deploying a Web Application using wdeploy

Before you can deploy a web application manually, you must make sure that the *server_root*`/bin/https/httpsadmin/bin` directory is in your path and that the `IWS_SERVER_HOME` environment variable is set to your *server_root* directory.

You can use the `wdeploy` utility at the command line to deploy a WAR file into a virtual server web application environment:

`wdeploy deploy -u` *uri_path* `-i` *instance* `-v` *vs_id* `[-d` *directory*`]` *war_file*

You can also delete a virtual server web application:

`wdeploy delete -u` *uri_path* `-i` *instance* `-v` *vs_id* `hard|soft`

You can also list the web application URIs and directories for a virtual server:

`wdeploy list -i` *instance* `-v` *vs_id*

The command parameters have the following meanings:

uri_path	The URI prefix for the web application.
instance	The server instance name.
vs_id	The virtual server ID.
directory	(optional) The directory to which the application is deployed, or from which the application is deleted. If not specified for deployment, the application is deployed to the document root directory.
`hard|soft`	Specifies whether the directory and the `web-apps.xml` entry are deleted (`hard`) or just the `web-apps.xml` entry is deleted (`soft`).
war_file	The WAR file name.

CAUTION If you deploy a web application and do not specify a *directory*, the application is deployed to the document root directory. If you then delete the application using the `hard` parameter, the document root directory will be deleted.

When you execute the `wdeploy deploy` command, two things happen:

- A web application with the given *uri_path* and *directory* gets added to the `web-apps.xml` file.

- The .WAR file gets extracted at the target *directory*.

For example:

```
wdeploy deploy -u /hello -i server.iplanet.com -v acme.com
-d /iws60/https-server.iplanet.com/acme.com/web-apps/hello
/iws60/plugins/servlets/examples/web-apps/HelloWorld/HelloWorld.war
```

This utility results in the following `web-apps.xml` entry:

```
<vs>
   <web-app uri="/hello"
   dir="/iws60/https-server.iplanet.com/acme.com/webapps/hello"/>
</vs>
```

The `/iws60/https-server.iplanet.com/acme.com/web-apps/hello` directory has the following contents:

```
colors
index.jsp
META-INF
WEB-INF/
   web.xml
   /classes/
      HelloWorldServlet.class
      HelloWorldServlet.java
      SnoopServlet.class
      SnoopServlet.java
```

Before you can run a web application that has been deployed, you must make sure that the `server.xml` file for the server instance points to the `web-apps.xml` file for your virtual server.

After you have deployed an application, you can access it from a browser as follows:

`http://`*vs_urlhost*`[:`*vs_port*`]/`*uri_path*`/[`*index_page*`]`

The parts of the URL have the following meanings:

vs_urlhost	One of the `urlhosts` values for the virtual server.
vs_port	(optional) Only needed if the virtual server uses a non-default port.
uri_path	The same one you used to deploy the application. This is also the context path.
index_page	(optional) The page in the application that end users are meant to access first.

For example:

`http://acme.com:80/hello/index.jsp`

or:

`http://acme.com/hello/`

Web Application Examples

iPlanet Web Server 6.0 comes with a set of example web applications. You can find them at the following location:

server_root/`plugins/servlets/examples/web-apps`

This directory contains the following directories:

- `HelloWorld` -- Contains a simple web application in a `HelloWorld.war` file.
- `filter-test` -- Contains an example of the Filter API feature in a `filter-test.war` file.
- `utility-taglib` -- Contains an example JSP tag library in a `utility-taglib.war` file. This tag library was created by the Jakarta project at `jakarta.apache.org`.

You can deploy any of these examples using the `wdeploy` utility.

Chapter 3

Using Servlets

This chapter discusses how to enable and configure servlets in iPlanet Web Server 6.0. The sections in this chapter are:

- What Does the Server Need to Run Servlets?
- Using the User Interface
- Enabling Servlets
- Making Servlets Available to Clients
- The <SERVLET> SHTML Tag
- Servlet Output
- Configuring JVM
- Maximizing Servlet Performance

What Does the Server Need to Run Servlets?

iPlanet Web Server 6.0 includes all the files necessary for developing Java Servlets. The `servlet.jar` file is in the iPlanet Web Server 6.0 installation directory at:

server_root`/bin/https/jar`

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

iPlanet Web Server 6.0 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 only or the JDK. For information about installing the JDK, see "What Does the Server Need to Run JSP?," on page 37.

Using the User Interface

For information about using the user interface to specify settings for servlets, see the following topics in the online help.

This page is located in the Web Server Administration Server on the Global Settings tab.

- The Configure JRE / JDK Paths Page

These pages are located in the Server Manager on the Java tab.

- The Enable / Disable Servlets / JSP Page
- The Configure JVM Attributes Page
- The Delete Version Files Page

This page is located on the Virtual Servers tab in the Class Manager. (To open the Class Manager, select the Manage Classes page on the Virtual Server Class tab in the Server Manager, select a class from the list, then select the Manage button.)

- The Java Web Apps Settings Page

Enabling Servlets

To enable servlets, select the Java tab in the Server manager, then select the Enable / Disable Servlets / JSP tab. Check the Enable Java Globally box to enable servlets for the entire server. Check the Enable Java for Class box to enable servlets for a single virtual server class. You cannot enable servlets for a class unless Java is globally enabled. By default, Java is globally enabled and enabled for each virtual server class.

Making Servlets Available to Clients

You can make servlets accessible to clients in one of these two ways:

- Include the servlets in web applications and deploy those web applications. How to do this is described in Chapter 2, “Web Applications.”
- Configure the servlets in the default virtual server. This is provided for backward compatibility with iPlanet Web Server 4.*x*. How to do this is described in Chapter 8, “Legacy Servlet and JSP Configuration.”

The <SERVLET> SHTML Tag

iPlanet Web Server 6.0 supports the `<SERVLET>` tag as introduced by Java Web Server. This tag allows you to embed servlet output in an SHTML 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=code codebase=path iParam1=v1  iParam2=v2>
<param name=param1 value=v3>
<param name=param2 value=v4>
.
.
</servlet>
```

If the servlet is part of a web application, the `code` parameter is required and other parameters are ignored. The `code` parameter must include:

- The value of the `url-pattern` element defined in the `web.xml` file for the web application. For more information about `web.xml`, see the Servlet 2.2 API specification:

  `http://java.sun.com/products/servlet/index.html`

- The value of the `uri` attribute defined in the `web-apps.xml` file for the web application. For more information about `web-apps.xml`, see Chapter 2, “Web Applications.”

For example, if you wanted to include the following in your SHTML file:

```
<servlet name=pparams code="/PrintApp/PrintParams">
</servlet>
```

you would need to include the following in your `web-apps.xml` file:

```
<web-app uri="/PrintApp"
dir="/iws60/https-server.iplanet.com/acme.com/webapps/PrintApp"/>
```

You would also need to include the following in your `web.xml` file:

```
<servlet>
   <servlet-name> pparams </servlet-name>
   <servlet-class> PrintPackage.PrintParams </servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name> pparams </servlet-name>
   <url-pattern> /PrintParams </url-pattern>
</servlet-mapping>
```

You must also include any servlet initialization parameters in the `web.xml` file.

For legacy (iPlanet Web Server 4.*x*) servlets, the `code` parameter specifies the `.class` file for the servlet and is required. 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. Legacy servlets must be configured in the default virtual server and do not require a `web.xml` file.

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

Servlet Output

When iPlanet Web Server is started in the background, which it is by default, the `System.out` and `System.err` output of servlets are not sent to the web server's error log, because servlets are external to iPlanet Web Server.

On Unix, you can modify your *server_root*/`https-`*server_id*/`start` file to run iPlanet Web Server in the foreground or to redirect servlet output. First enter the following command from the *server_root*/`https-`*server_id* directory:

```
./start -shell
```

This command puts you in the *server_root*/`bin/https/bin` directory. Then enter the following command:

```
./ns-httpd -d server_root/https-server_id/config
```

On NT, you can run iPlanet Web Server in the NT console, and thus in the foreground, by including the following line in the `magnus.conf` file:

```
Init fn="nt-console-init" stdout=console stderr=console
```

Configuring JVM

If necessary, you can configure parameters for JVM either by using the Java>Configure JVM Attributes page in the Server Manager interface, or by editing `jvm12.conf`. For more information about JVM settings, see Appendix C, "JVM Configuration."

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, you can add the JAR location to the JVM classpath.

Maximizing Servlet Performance

Consider the following guidelines for improving servlet performance:

- 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 the *iPlanet Web Server Administrator's Guide*.

- Increase the front-end thread stack size in `magnus.conf` (via the `StackSize` directive), 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 `jvm.classpath` is short (if you don't need some of the examples). You can set `jvm.include.CLASSPATH=0` so it won't inherit the `CLASSPATH` environment variable.

- Sometimes, iPlanet Web Server 6.0 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` directive in the `magnus.conf` file. For more information, see the *NSAPI Programmer's Guide for iPlanet Web Server*.

- 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 Chapter 6, "Session Managers."

NOTE When running an SSL server *without* Java, you can improve performance by enabling SmartHeap in the server's start script. However, SmartHeap is *not* compatible with Java.

Chapter 4

Using JavaServer Pages

This chapter discusses how to enable and configure JavaServer Pages (JSPs) in iPlanet Web Server 6.0. The sections in this chapter are:

- What Does the Server Need to Run JSP?
- Configuring JRE/JDK Paths
- Using the User Interface
- Enabling JSPs
- Making JSPs Available to Clients
- Deleting Cache Version Files
- The JSP Command-Line Compiler
- JSP Tag Libraries and Standard Portable Tags

What Does the Server Need to Run JSP?

iPlanet Web Server 6.0 includes the Java Runtime Environment (JRE) but not the Java Development Kit (JDK) due to licensing restrictions. The server can run servlets and precompiled JSPs using the JRE, but you need the JDK to develop new JSPs or to deploy uncompiled JSPs. For information about precompiling JSPs, see “The JSP Command-Line Compiler,” on page 43.

iPlanet Web Server 6.0 requires you to use the following recommended versions of JRE/JDK or later versions, with different platforms requiring different versions, as summarized in Table 4-1.

Table 4-1 Supported JRE/JDK Versions by Platform

Platform	JRE/JDK/JVM/JIT Version	Comments
Sun Solaris 2.6, 2.8	Solaris VM (build Solaris_JDK_1.2.2_07, native threads, sunwjit)	Comment out `-Xrs` flag in `jvm12.conf` to generate stack traces. For more information, see "Generating a Stack Trace for Debugging," on page 51. For JVMPI based profiling (such as hprof) or debugging (say attach Solaris dbx) purposes, use the reference implementation downloadable from: `http://java.sun.com/products/jdk/1.2/jre/`
Windows NT 4.0	Java version 1.2.2 Classic VM (build JDK-1.2.2_007, native threads, symcjit)	
HPUX	Java version 1.2.2.07 Classic VM (build 1.2.2.07-00/12/08-PA_RISC1.1, native threads, HP)	iPlanet also bundles a variant HotSpot VM (1.0.1fcs, mixed mode, PA2.0 build 1.2.2.07-00/12/08-PA_RISC2.0). This VM is not enabled. For further details on using this version, see: `http://www.unix.hp.com/java/infolibrary/prog_guide/java2/hotspot.html`
AIX	Java version 1.2.2 Classic VM (J2RE 1.2.2 IBM build ca122-20001206 (JIT enabled: jitc))	
Compaq Tru64	Java version 1.2.2-8 Classic VM (build J2SDK.v.1.2.2:10/31/2000-18:00, native threads, jit_122)	The Compaq version of iPlanet Web Server is available from Compaq.
RedHat Linux 6.2	Java version 1.2.2 Classic VM (build Linux_JDK_1.2.2_FCS, native threads, sunwjit)	This version of the JVM is from `blackdown.org`.

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

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 6.0, 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 39.

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

Configuring JRE/JDK Paths

When you install iPlanet Web Server 6.0, 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 JSPs that are not precompiled. 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 6.0 requires you to use version of the JDK listed in the section "What Does the Server Need to Run JSP?," on page 37.

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/6.0/`. 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`).

CAUTION Be very careful when editing these Windows NT registry entries. Incorrect values may require you to reinstall iPlanet Web Server.

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

Using the User Interface

For information about using the user interface to specify settings for JSPs, see the following topics in the online help.

This page is located in the Web Server Administration Server on the Global Settings tab.

- The Configure JRE/JDK Paths Page

These pages are located in the Server Manager on the Java tab.

- The Enable/Disable Servlets/JSP Page
- The Configure JVM Attributes Page
- The Delete Version Files Page

This page is located on the Virtual Servers tab in the Class Manager. (To open the Class Manager, select the Manage Classes page on the Virtual Server Class tab in the Server Manager, select a class from the list, then select the Manage button.)

- The Java Web Apps Settings Page

Enabling JSPs

To enable JSPs, follow these steps:

1. Enable servlets. Select the Java tab in the Server manager, then select the Enable / Disable Servlets / JSP tab. Check the Enable Java Globally box to enable servlets for the entire server. Check the Enable Java for Class box to enable servlets for a single virtual server class. You cannot enable servlets for a class unless Java is globally enabled. By default, Java is globally enabled and enabled for each virtual server class.
2. Include the `jsp-servlet` element with `enable="true"` in the `web-apps.xml` file. For more information about the `web-apps.xml` file, see Chapter 2, "Web Applications."
3. Add `tools.jar` to the JVM classpath. For more information, see Appendix C, "JVM Configuration."

Making JSPs Available to Clients

You can make JSPs accessible to clients in one of these two ways:

- Include the JSPs in web applications and deploy those web applications. How to do this is described in Chapter 2, "Web Applications."
- Configure the JSPs in the default virtual server. This is provided for backward compatibility with iPlanet Web Server 4.*x*. How to do this is described in Chapter 8, "Legacy Servlet and JSP Configuration."

Deleting Cache Version Files

The server uses the following directory to cache information for JavaServer Pages (JSP):

server_root`/https-`*server_id*`/ClassCache/`*virtual_server_id*`/`*webapp_uri*`/`

You can change the location of the JSP class cache using the `scratchdir` initialization parameter of the `jsp-servlet` element in the `web-apps.xml` file. For more information, see "Other JSP Initialization Parameters," on page 47.

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.

The 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 file, it deletes the directory structures for the corresponding caches and re-creates the version file. Next time the server serves a JSP page, it recreates the JSP class cache.

You can delete the version file simply by deleting it from the `ClassCache` directory as you would normally delete a file, or you can use the Java>Delete Version Files page in the Server Manager to delete it. 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 file before the server serves any JSPs.

The JSP Command-Line Compiler

You can precompile JSPs for faster performance; this is recommended for production servers. A command-line JSP compiler is included with iPlanet Web Server. The JSP compiler is located under *server_root*`/bin/https/bin` (make sure this directory is in your path). The `IWS_SERVER_HOME` environment variable must be set to the *server_root* directory.

The format of the `jspc` command is as follows:

`jspc [`*options*`]` *jsp_files*

The *jsp_files* can be one of the following:

files	One or more JSP files to be compiled.
`-webapp` *dir*	A directory containing a web application. All JSPs in the directory and its subdirectories are compiled. You cannot specify a WAR, JAR, or ZIP file.

The *options* for the `jspc` command are:

`-q`	Enables quiet mode (same as `-v0`). Only fatal error messages are displayed.

`-v[`*level*`]`	Enables verbose mode. The *level* is optional; the default is 2. Possible *level* values are: • `0` - fatal error messages only • `1` - error messages only • `2` - error and warning messages only • `3` - error, warning, and informational messages • `4` - error, warning, informational, and debugging messages
`-d` *dir*	Specifies the output directory for the compiled JSPs. Package directories are automatically generated based on the directories containing the uncompiled JSPs. The default top-level directory is the directory from which `jspc` is invoked.
`-dd` *dir*	Specifies the literal output directory for the compiled JSPs. Package directories are not made. The default is the directory from which `jspc` is invoked.
`-p` *name*	Specifies the name of the target package for all specified JSPs, overriding the default package generation performed by the `-d` option.
`-c` *name*	Specifies the target class name of the first JSP compiled. Subsequent JSPs are unaffected.
`-mapped`	Generates separate `write` calls for each HTML line and comments that describe the location of each line in the JSP file. By default, all adjacent `write` calls are combined and no location comments are generated.
`-die[`*code*`]`	Causes the JVM to exit and generates an error return *code* if a fatal error occurs. If the *code* is absent or unparsable it defaults to `1`.
`-uribase` *dir*	Specifies the URI directory to which compilations are relative. Applies only to explicitly declared JSP files. This is the location of each JSP file relative to the `uriroot`. If this cannot be determined, the default is `/`.

`-uriroot` *dir*	Specifies the root directory against which URI files are resolved. Applies only to explicitly declared JSP files. If this option is not specified, all parent directories of the first JSP page are searched for a `WEB-INF` subdirectory. The closest directory to the JSP page that has one is used. If none of the JSP's parent directories have a `WEB-INF` subdirectory, the directory from which `jspc` is invoked is used.
`-webinc` *file*	Creates partial servlet mappings for the `-webapp` option, which can be pasted into a `web.xml` file.
`-webxml` *file*	Creates an entire `web.xml` file for the `-webapp` option.
`-ieplugin` *class_id*	Specifies the Java plugin COM class ID for Internet Explorer. Used by the `<jsp:plugin>` tags.
`-genclass`	Generates class files in addition to Java files. The JDK `tools.jar` file must be in the JVM classpath.

The `-webinc` and `-webxml` options may not be useful for JSPs in iPlanet Web Server.

For example, this command (all on one line) compiles the JSPs in the HelloWorld web application:

```
jspc -d dir -genclass -webapp
server_root/plugins/servlets/examples/web-apps/HelloWorld
```

The compiled JSPs are written under *dir*`/_jsps/`. You can then put these class files in a .JAR file.

Additional documentation for the JSP compiler is on the Jakarta site:

`http://jakarta.apache.org/`

Jasper and iPlanet Web Server 6.0 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 more information about JVM settings, see Appendix C, "JVM Configuration."

Package Names Generated by the JSP Compiler

When a JSP is compiled, a package is created for it. The package name starts with `_jsps` and has each pathname component of the JSP prefixed with an underscore. For example, the generated package name for `/myjsps/hello.jsp` is `_jsps._myjsps`.

Because of the implicit package name associated with a generated servlet, you need an explicit `import` directive when you use a bean in a JSP, especially if the bean doesn't have a package name. For example:

```
<%@page import="MyBean" %>
<jsp:useBean id="myBean" class="MyBean" />
```

Specifying that JSPs Are Precompiled

The `jsp-servlet` element in the `web-apps.xml` file allows you to tell iPlanet Web Server that JSPs in a virtual server are precompiled. Include the following tags inside a `vs` element:

```
<jsp-servlet enable="true">
   <init-param>
      <param-name>use-precompiled</param-name>
      <param-value>true</param-value>
   </init-param>
</jsp-servlet>
```

For more information about the `web-apps.xml` file, see Chapter 2, "Web Applications."

You do not need to install the JDK to run precompiled JSPs. However, you need the JDK to develop new JSPs. For information about installing the JDK, see "What Does the Server Need to Run JSP?," on page 37.

Other JSP Initialization Parameters

You can include the following initialization parameters under the `jsp-servlet` element in your `web-apps.xml` file. The JSP compiler uses the default values for parameters that are not included in the file.

`keepgenerated`	If set to `true` (the default), keeps the generated Java files. If `false`, deletes the Java files.
`largeFile`	If set to `true`, static HTML is stored is a separate data file. This is useful when a JSP is very large. The default is `false`.
`scratchdir`	The working directory created for storing all the generated code. If this parameter is not specified, the default location is *server_root*`/https-`*server_id*`/ClassCache/`*virtual_server_id*`/`*webapp_uri*`/`.
`mappedfile`	If set to `true`, generates separate `write` calls for each HTML line and comments that describe the location of each line in the JSP file. By default, all adjacent `write` calls are combined and no location comments are generated.
`ieClassId`	The Java plugin COM class ID for Internet Explorer. Used by the `<jsp:plugin>` tags.
`use-precompiled`	If set to `true`, specifies that the JSPs in a virtual server are precompiled and do not need to be compiled at runtime. The default is `false`. If set to `true`, changes to JSPs are not automatically reloaded. You can compile JSPs using the command line JSP compiler, put the classes in a JAR file, and put the JAR file in the `WEB_INF/lib` directory of your web application.

JSP Tag Libraries and Standard Portable Tags

iPlanet Web Server supports tag libraries and standard portable tags. For more information about tag libraries, see the JSP 1.1 specification at:

`http://java.sun.com/products/jsp/download.html`

Chapter 5

Debugging Servlets and JSPs

This appendix gives guidelines for debugging servlets and JSPs in iPlanet Web Server 6.0. It includes the following sections:

- Servlet Debugging
- JSP Debugging
- Generating a Stack Trace for Debugging
- Using Forte for Java to Debug Servlets and JSPs
- JPDA Options for Debugging

Debugging servlets and JSPs requires that you edit the`jvm12.conf` file as described in this chapter. For more general information about this file, see Appendix C, “JVM Configuration.”

Servlet Debugging

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 39.

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

1. Make sure that the server is running in single-process mode. Single-process mode is the default, but you can check in the file `magnus.conf` to make sure that the `MaxProcs` parameter is not set to a value greater than 1. If you do not see a setting for `MaxProcs` in `magnus.conf`, the default value of 1 is enabled for it. For more information about single process mode versus multi-process mode, see the *iPlanet Web Server Administrator’s Guide*.

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

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

3. 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
   ```

4. 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 follows in `jvm12.conf`:

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

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

6. 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

You can debug your JSPs by following these steps:

1. Make sure that the server is running in single-process mode. Single-process mode is the default, but you can check in the file `magnus.conf` to make sure that the `MaxProcs` parameter is not set to a value greater than 1. If you do not see a setting for `MaxProcs` in `magnus.conf`, the default value of 1 is enabled for it. For more information about single process mode versus multi-process mode, see the *iPlanet Web Server Administrator's Guide*.

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

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

3. 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.enabledebug=1` makes iPlanet Web Server 6.0 generate debuggable Java servlets from the JSPs.

Generating a Stack Trace for Debugging

You can generate a Java stack trace for debugging as described here:

`http://developer.java.sun.com/developer/technicalArticles/Programming/Stacktrace/`

Comment out the `jvm.option=-Xrs` flag (for reduced signal usage) in the `jvm12.conf` file before generating the stack trace. If the `-Xrs` flag is used, the server may simply dump core and restart when you send the signal to generate the trace.

For more about the `jvm12.conf` file and using `jvm.option`, see Appendix C, “JVM Configuration.”

Using Forte for Java to Debug Servlets and JSPs

To set up iPlanet Web Server so you can use the Forte for Java debugger, follow these steps:

1. If you have not already done so, install the JDK version 1.2.*x* that iPlanet Web Server requires as described in “What Does the Server Need to Run JSP?,” on page 37. The specific version you need depends on your platform.

2. Install the JDK version 1.3, available here:

   `http://java.sun.com/j2se/1.3/`

   Although iPlanet Web Server uses the JDK version 1.2.*x*, Forte for Java requires version 1.3.

3. Install Forte for Java, Community Edition 1.0, available here:

   `http://www.sun.com/forte/ffj/ce/`

4. iPlanet strongly recommends that you also install the JPDA, available here:

   `http://java.sun.com/products/jpda/`

5. If you installed the JPDA, copy all the files from the *jpda_install*`/bin` directory to the *jdk1.2_install*`/jre/bin` directory. Also copy the `jpda.jar` file from the *jpda_install*`/lib` directory to the *jdk1.2_install*`/jre/lib/ext` directory.

   Note that the JPDA runs on JDK 1.2, not 1.3.

6. On Windows NT, add the following line to the `magnus.conf` file to enable the NT console:

   ```
   Init fn="nt-console-init" stdout=console stderr=console
   ```

7. Edit the `jvm12.conf` file to enable remote debugging. If you *did not* install the JPDA, add the following lines:

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

   If you installed the JPDA, add the following lines:

   ```
   jvm.enableDebug=1
   jvm.compiler=NONE
   jvm.option=-classic
   jvm.option=-Xnoagent
   jvm.option=-Xrunjdwp:transport=dt_socket,server=y,suspend=n
   ```

8. Start Forte for Java, and mount the directory that contains the servlet or JSP you want to debug.

9. Start iPlanet Web Server. You will see a line similar to the following displayed in the console:

   `Listening for transport dt_socket at address:` *port_number*

   Write down this *port_number*.

10. In Forte for Java, select the Debug menu and the Attach to VM... option. Type the *port_number* in the Port: text box, then select OK.

You are now ready to debug your servlet or JSP.

JPDA Options for Debugging

A list of debugging options that you can include in the `jvm12.conf` file if the JPDA is installed is available here:

`http://java.sun.com/products/jpda/doc/conninv.html#Invocation`

Chapter 6

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
- IWSSessionManager
- MMapSessionManager (Unix Only)
- Deprecated Session Managers
- Load Balancing, Session Failover, and Session IDs

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.

iPlanet Web Server 6.0 comes with these session managers for creating and managing sessions:

- `IWSSessionManager` -- the default session manager, which can use a database or a file store for persistent sessions, and can run in single-process or multi-process mode.
- `MMapSessionManager (Unix Only)` -- a session manager for running the server in multi-process mode. Does not support distributed sessions.
- `SimpleSessionManager` -- a deprecated simple session manager. Does not support distributed sessions.
- `JdbcSessionManager` -- a deprecated session manager that stores session information in a database using the JDBC API and that supports distributed sessions.

Multi-process mode is supported only on Unix platforms. All multi-process mode features of session managers are ignored on Windows NT.

iPlanet Web Server 6.0 also allows you to develop your own session managers and load them into the server. The source code files for session manager 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/servlets/iws-apis/sessions`.

Specifying a Session Manager

By default iPlanet Web Server uses `IWSSessionManager` as the session manager for servlets. You can change the session manager in any of the following ways:

- Edit the file `web-apps.xml` in the directory *server_id*`/config`.

  Add a `session-manager` element within the `web-app` element for the servlet or JSP as in the following example:

  ```
  <session-manager
        class='com.iplanet.server.http.session.YourSesMgr'
        <init-param>
              <param-name>maxSessions</param-name>
              <param-value>1000</param-value>
        </init-param>
        <init-param>
              <param-name>timeOut</param-name>
              <param-value>1800</param-value>
  ```

```
        </init-param>
        <init-param>
                <param-name>reapInterval</param-name>
                <param-value>600</param-value>
        </init-param>
</session-manager>
```

For more information about the `web-apps.xml` file, see Chapter 2, “Web Applications.”

- Use the Legacy 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`. This will apply to the default virtual server only.

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

- Edit the file `contexts.properties` in the directory *server_id*/`config`. This will apply to the default virtual server only.

  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.iplanet.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 Chapter 8, “Legacy Servlet and JSP Configuration.”

IWSSessionManager

The `IWSSessionManager` is the default session manager.

`IWSSessionManager` works in both single process and multi-process mode. It can be used for sharing session information across multiple processes possibly running on different machines. The `MaxProcs` directive in the `magnus.conf` file determines whether the server is running in single process mode or multi-process mode. For more information, see the *NSAPI Programmer's Guide for iPlanet Web Server*.

For session persistence, `IWSSessionManager` can use a database or a distributed file system (DFS) path that is accessible from all servers in a server farm. Each session is serialized to the database or distributed file system. You can also create your own persistence mechanism.

If iPlanet Web Server is running in single-process mode, then by default, no session persistence mode is defined and therefore sessions are not persistent.

If iPlanet Web Server is running in multi-process mode, sessions are persistent by default. If a persistence mode is not defined, `IWSSessionManager` uses a DFS.

Multi-process mode is supported only on Unix platforms. All multi-process mode features of `IWSSessionManager` are ignored on Windows NT.

Parameters for IWSSessionManager

`IWSSessionManager` 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).

  If `session-timeout` is specified in `web.xml`, it overrides this `timeOut` parameter value. For details, see "session-timeout," on page 15.
- `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).

- `maxLocks` - the number of cross-process locks to use for synchronizing access to individual sessions across processes. The default value is 10. This default value is used if the value 0 is specified. This parameter is ignored in single-process mode.
- `session-data-store` - the name of the class that determines the means of session persistence. The classes supplied with iPlanet Web Server are:
    - `com.iplanet.server.http.session.JdbcStore`
    - `com.iplanet.server.http.session.FileStore`

  If you do not specify the `session-data-store` parameter, sessions are not persistent in single-process mode, and `FileStore` is the default in multi-process mode.

  The `JdbcStore` and `FileStore` classes are subclasses of the `SessionDataStore` class. You can create your own class that implements session persistence by extending `SessionDataStore`.

NOTE Prior to using `JdbcStore`, you must create the table in which the session information is stored. The name of the table is specified by the `table` parameter, and the table's four columns are specified by the `accessTimeColumn`, `timeOutColumn`, `sessionIdColumn`, and `valueColumn` parameters.

If the `session-data-store` parameter is set to the `JdbcStore` or `FileStore` class, `IWSSessionManager` takes the following additional parameter:

- `session-failover-enabled` - specifies whether sessions are reloaded from the persistent store for every request, and always forced to true in multi-process mode.

If the `session-data-store` parameter is set to the `FileStore` class, `IWSSessionManager` takes the following additional parameter:

- `session-data-dir` - the directory in which session data for all servers and web applications is kept.

  If the `session-data-dir` parameter is not specified, the following directory is used by default:

  server_root/*server_id*/`SessionData`/*virtual_server_id*/*web_app_URI*

If the `session-data-store` parameter is set to the `JdbcStore` class, `IWSSessionManager` takes the following additional parameters:

- `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`

NOTE The `JdbcStore` class does not recognize JDBC driver classes assigned to the `classpath` attribute of the `class-loader` element in `web-apps.xml`. Assign them to the `jvm.classpath` variable in the `jvm12.conf` file instead. For more information, see Appendix C, "JVM Configuration."

- `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).
- `timeOutColumn` - the name of the column that holds the session timeout in minutes (the default name is `TimeOut`). 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 IWSSessionManager

You may want to enable `IWSSessionManager` to change its default parameters. You can also enable `IWSSessionManager` for a particular context if the server is running in single process mode. To enable iPlanet Web Server to use `IWSSessionManager`, do any of the following:

- Edit the file `web-apps.xml` in the directory *server_id*/`config`.

  Add a `session-manager` element within the `web-app` element for the servlet or JSP as in the following example:

  ```
  <session-manager
          class='com.iplanet.server.http.session.IWSSessionManager'
          <init-param>
                  <param-name>maxSessions</param-name>
                  <param-value>1000</param-value>
          </init-param>
          <init-param>
                  <param-name>timeOut</param-name>
                  <param-value>1800</param-value>
          </init-param>
          <init-param>
                  <param-name>reapInterval</param-name>
                  <param-value>600</param-value>
          </init-param>
          <init-param>
                  <param-name>session-data-dir</param-name>
                  <param-value>/net/dotcom.com/sessions</param-value>
          </init-param>
  </session-manager>
  ```

  For more information about the `web-apps.xml` file, see Chapter 2, “Web Applications.”

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

  In the Session Manager field specify:

  `com.iplanet.server.http.session.IWSSessionManager`

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

  `maxSessions=20,session-data-dir=/net/dotcom.com/sessions`

Source Code for IWSSessionManager

The `IWSSessionManager` creates an `IWSHttpSession` object for each session. The source files for `IWSSessionManager.java` and `IWSHttpSession.java` are in the *server_root*/`plugins/servlets/iws-apis/sessions` directory. The source code files for `IWSSessionManager.java` and `IWSHttpSession.java` are provided so you can use them as the starting point for defining your own session managers and session objects.

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

The `JdbcStore.java` and `FileStore.java` source files and the source file for the parent class, `SessionDataStore.java`, are provided so you can modify the session persistence mechanism of `IWSSessionManager`. These files are also located in the directory *server_root*/`plugins/servlets/iws-apis/sessions`.

MMapSessionManager (Unix Only)

This is a persistent memory map (mmap) file based session manager that works in both single process and multi-process mode.

The `MaxProcs` directive in the `magnus.conf` file determines whether the server is running in single process mode or multi-process mode. For more information, see the *NSAPI Programmer's Guide for iPlanet Web Server*.

Parameters

MMapSessionManager 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.
- maxValuesPerSession - the maximum number of values or objects a session can hold. The default value is 10.
- maxValueSize - the maximum size of each value or object that can be stored in the session. The default value is 4096.
- timeOut - the amount of time in seconds after a session is last 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).

  If session-timeout is specified in web.xml, it overrides this timeOut parameter value. For details, see "session-timeout," on page 15.
- 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).
- maxLocks - the number of cross-process locks to use for synchronizing access to individual sessions across processes. The default value is 1. This default value is used if the value 0 is specified. This parameter is ignored in single-process mode.

Enabling MMapSessionManager

You may want to enable MMapSessionManager to change its default parameters. You can also enable MMapSessionManager for a particular context if the server is running in single process mode. To enable iPlanet Web Server to use MMapSessionManager, do any of the following:

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

  In the Session Manager field specify:

  ```
  com.iplanet.server.http.session.MMapSessionManager
  ```

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

```
maxSessions=20,maxValueSize=1024,timeOut=300
```

- Edit the file `servlets.properties` in the directory *server_id*/`config`. This will apply to the default virtual server only.

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

```
servlets.sessionmgr=com.iplanet.server.http.session.MMapSessionManager
servlets.sessionmgr.initArgs=maxSessions=20,maxValueSize=1024,timeOut=300
```

- Edit the file `contexts.properties` in the directory *server_id*/`config`. This will apply to the default virtual server only.

  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.iplanet.server.http.session.MMapSessionManager
context.global.sessionmgr.initArgs=maxSessions=20,maxValueSize=1024,timeOut=300
```

  You can change the global context or define a new context and assign specific servlets to it. For more information, see Chapter 8, “Legacy Servlet and JSP Configuration.”

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

Deleting SessionData Version Files

If the server uses the `MMapSessionManager` session manager, it stores persistent session information in the `SessionData` directory. This 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 file, it deletes the directory structures for the corresponding caches and re-creates the version file. The next time the server serves a servlet while using `MMapSessionManager` session manager, it recreates the session data cache.

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

Deprecated Session Managers

The `SimpleSessionManager` and the `JdbcSessionManager` are provided for backward compatibility with iPlanet Web Server 4.*x*.

NOTE The parent class of the deprecated session managers is also deprecated and is included for backward compatibility only:

`com.netscape.server.http.session.NSHttpSessionManager`

Extend the following class instead:

`com.iplanet.server.http.session.IWSHttpSessionManager`

SimpleSessionManager

The `SimpleSessionManager` works only in single process mode. Its sessions are not persistent, that is, all sessions are lost when the server is stopped.

NOTE The `SimpleSessionManager` is deprecated and is included for backward compatibility only. Use `IWSSessionManager` with no session persistence instead.

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).

  If `session-timeout` is specified in `web.xml`, it overrides this `timeOut` parameter value. For details, see “session-timeout,” on page 15.
- `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. You can also enable `SimpleSessionManager` for a particular context if the server is running in multi-process mode. To enable the iPlanet Web Server to use `SimpleSessionManager`, do any of the following:

- Use the Legacy 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`. This will apply to the default virtual server only.

  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`. This will apply to the default virtual server only.

  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 Chapter 8, "Legacy Servlet and JSP Configuration."

JdbcSessionManager

This is a persistent JDBC-based session manager that works in both single process and multi-process modes. It can be used to store sessions in a custom database, which can then be shared across multiple processes possibly running on different machines.

NOTE The `JDBCSessionManager` is deprecated and is included for backward compatibility only. Use `IWSSessionManager` with `JdbcStore` session persistence instead.

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).

  If `session-timeout` is specified in `web.xml`, it overrides this `timeOut` parameter value. For details, see "session-timeout," on page 15.

- `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`

NOTE The `JdbcStore` class does not recognize JDBC driver classes assigned to the `classpath` attribute of the `class-loader` element in `web-apps.xml`. Assign them to the `jvm.classpath` variable in the `jvm12.conf` file instead. For more information, see Appendix C, "JVM Configuration."

- `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 if the server is running in single process mode. To enable iPlanet Web Server to use `JdbcSessionManager`, do any of the following:

- Use the Legacy 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`. This will apply to the default virtual server only.

  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`. This will apply to the default virtual server only.

  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 Chapter 8, “Legacy Servlet and JSP Configuration.”

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

Load Balancing, Session Failover, and Session IDs

iPlanet Web Server 6.0 supports server farm (or cluster) configurations with off-the-shelf, session-aware front-end load balancers such as Resonate. iPlanet Web Server 6.0 implements sticky sessions by prefixing the *node_id* of the server host that generated the session to JSESSIONID, the session cookie name specified in the Servlets 2.2 specification, as follows:

```
Set-Cookie: JSESSIONID=node_id-3ad%253A39c02099%253Ad19e53e2a2;
path=/app;expires=Thu, 14-Sep-2000 01:19:30 GMT
```

This enables the front-end load balancer to forward future requests to the same host that generated the session.

If the session manager used by a cluster supports distributed sessions, an alternate server can pick up a session created by a server that crashed. The `IWSSessionManager` supports distributed sessions if its `session-data-store` parameter has a value that defines the mode of session persistence. The `JdbcSessionManager` also supports distributed sessions.

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 `IWSSessionManager.java` example file described in "Source Code for IWSSessionManager," on page 62).

NOTE iPlanet Web Server session managers generate session IDs with a maximum length of 108 ASCII characters.

Chapter 7

API Clarifications

This appendix clarifies ways in which the Servlet 2.2 API specification is implemented in iPlanet Web Server 6.0 in the following sections:

- Clarification of HttpSession Scope
- Clarifications for Using Methods
- Other Useful Information

Clarification of HttpSession Scope

The Servlet 2.2 API Specification is open to interpretation in regard to the scope of HttpSession objects.

By default, iPlanet Web Server marks the session tracking cookie's path to that of the context or application's contextPath. This results in the browser not replaying a session cookie to an application for which it is not intended. Hence, the HttpSession objects are scoped appropriately.

However, if servlet A in one context dispatches a request to servlet B in a different context, the specification is open to interpretation. By default iPlanet Web Server permits this, and you must make sure that the attributes placed in the HttpSession object are loaded by a class loader that is common to the applications involved. You can use a virtual-server-level class loader or the system class loader, although this is not generally recommended.

To allow different applications to share sessions and session attributes:

- Use a common session manager at the virtual server level.

```
<vs>
       <session-manager class="..." />
</vs>
```

- Make sure that the session attributes are loaded by the virtual-server-level class loader. Add any common beans and so on that are going to be used as session attributes to the classpath there.

  ```
  <vs>
        <class-loader classpath="/myapps/sessionattrs.jar" />
  </vs>
  ```

This configuration lets you share sessions across all applications within a virtual server or none at all. For iPlanet Web Server 6.0, there is no way that only a set of applications can share the same session data.

Clarifications for Using Methods

This section provides clarifications for using the following Servlet 2.2 API methods with iPlanet Web Server 6.0:

- HttpServlet.service
- ServletContext.getAttribute
- ServletRequest.setAttribute
- ServletRequest.getParameter
- ServletResponse.getOutputStream and getWriter
- RequestDispatcher.forward and include

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`

HttpServlet.service

```
public void service(ServletRequest req, ServletResponse res) throws
ServletException, java.io.IOException
```

This method dispatches client requests to the protected `service` method.

Clarification

Servlets may create additional threads to handle their service logic. However, the API functions exposed by these objects must be accessed by either the request handling thread or one of the threads the servlet created, but not by both simultaneously.

ServletContext.getAttribute

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

Returns the servlet container attribute with the given name, or `null` if there is no attribute by that name.

Clarification

To obtain the context class loader (a `java.lang.ClassLoader` object), use the `com.iplanet.server.http.servlet.classloader` attribute. To obtain the context class loader's classpath (a `java.lang.String` object), use the `com.iplanet.server.http.servlet.classpath` attribute.To obtain the context class loader's reload interval (a `java.lang.Integer` object), use the `com.iplanet.server.http.servlet.reload-interval` attribute.

ServletRequest.setAttribute

```
public void setAttribute(java.lang.String name, java.lang.Object o)
```

Stores an attribute in this request. Attributes are reset between requests. This method is most often used in conjunction with `RequestDispatcher`.

Clarification

You can set the `com.iplanet.server.http.servlet.parameterEncoding` attribute in the request object to allow the `getParameter` method to know the encoding of the parameters it extracts.

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, you must do one of the following, or the values extracted by the `getParameter` method are zeros or are undefined:

- Set the enc attribute of the `parameter-encoding` element to `auto` (the default) in the `web-apps.xml` file. For more information, see "parameter-encoding," on page 22.
- Set the `parameterEncoding` property to `auto` (the default) or `responseCT` in the `contexts.properties` file, as in iPlanet Web Server 4.*x*. For more information, see "parameterEncoding," on page 92.

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 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` by default), for example:

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

    - Set the `com.iplanet.server.http.servlet.parameterEncoding` attribute in the request object (see "ServletRequest.setAttribute," on page 73). The `getParameter` method uses this attribute to decode the parameters.
    - If `parameterEncoding=responseCT` in `contexts.properties` and the same servlet or JSP generates and processes the form, you can 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"%>
```

ServletResponse.getOutputStream and getWriter

```
public ServletOutputStream getOutputStream() throws
java.io.IOException
```

Returns a `ServletOutputStream` suitable for writing binary data in the response. The servlet container does not encode the binary data. Either this method or `getWriter` may be called to write the body, not both.

```
public java.io.PrintWriter getWriter() throws java.io.IOException
```

Returns a `PrintWriter` object that can send character text to the client. The character encoding used is the one specified in the `charset=` property of the `setContentType(java.lang.String)` method, which must be called before calling this method for the charset to take effect.

Either this method or `getOutputStream` may be called to write the body, not both.

Clarification

The specification recommends that when a servlet calls the `getWriter` method on a response for which the `getOutputStream` method has already been called, or the other way around, the servlet container should throw an `IllegalStateException`.

iPlanet Web Server 6.0 doesn't throw the exception, while ensuring that the writes to writer and the output stream are ordered. This implementation is more lenient but preserves correctness and permits scenarios like these:

- A servlet calls `getOutputStream` and throws an exception, and a JSP is used for the exception page. With strict compliance, this fails, because the JSP tries to call `getWriter` on the response object.
- A JSP that calls `getWriter` includes a servlet that calls `getOutputStream`.

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`), it will *not* 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 6.0, 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.

You can control the nesting depth of the `RequestDispatcher.forward` and `include` methods using the `requestDispatcherNestDepth` parameter in `magnus.conf`. For more information, see Appendix A, "Servlet Settings in magnus.conf and obj.conf."

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[] certs;
    certs = request.getAttribute("javax.servlet.request.X509Certificate");
    if (certs != null) {
        clientCert = certs[0];
        if (clientCert != null) {
            // Get the Distinguised Name for the user.
            java.security.Principal userDN = clientCert.getSubjectDN();
            ...
        }
    }
}
```

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

Chapter 8

Legacy Servlet and JSP Configuration

This chapter describes legacy configuration procedures and files, which are provided for backward compatibility with iPlanet Web Server 4.*x*, in these sections:

- The Default Virtual Server
- Enabling Servlets and JSP
- Making JSPs Available to Clients
- Configuring Servlets in the Default Virtual Server
- Using the User Interface
- Legacy Configuration Files
- Legacy Examples
- Maximizing Legacy Servlet Performance

NOTE Legacy servlets and JSPs, as described in this chapter, are deprecated. Creating and deploying web applications as described in Chapter 2, "Web Applications," is recommended.

The Default Virtual Server

The default virtual server is the only virtual server in which legacy applications can be run. When you first install iPlanet Web Server, the default virtual server is the only virtual server that exists. For more information about how the default virtual server is defined, see Chapter 8, "Virtual Server Configuration Files," in the *NSAPI Programmer's Guide for iPlanet Web Server*.

Enabling Servlets and JSP

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

By default, regardless of whether servlets are enabled or disabled, the file `obj.conf` contains 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.

Making JSPs Available to Clients

No special steps are needed to make JSP pages available to clients other than making sure that servlets and JSP are 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.)

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

Configuring Servlets in the Default Virtual Server

There are three ways to make a servlet accessible to clients in the default virtual server. The second two ways are provided for backward compatibility with iPlanet Web Server 4.*x*.

- Include the servlets in web applications and deploy those web applications, as with any other virtual server. How to do this is described in Chapter 2, “Web Applications.” This is new in iPlanet Web Server 6.0.
- 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 81.
- 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 84.

In detail, to serve servlets in the default virtual server as in iPlanet Web Server 4.*x*, do the following steps:

1. Configuring Global Servlet Attributes
2. Registering Servlet Directories
3. Registering Individual Servlets if Needed
4. Specifying Servlet Virtual Paths if Desired
5. Specifying Servlet Contexts if Desired

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 Chapter 6, "Session Managers."
- Session Manager Args -- the session manager arguments for the servlet engine. For more information about the session manager, see Chapter 6, "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 Legacy Servlets>Configure Global Servlet Attributes page in the Server Manager interface. Alternatively, you can edit the configuration files `servlets.properties` and `contexts.properties` in the server's `config` directory.

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 iPlanet Web Server. Initially, 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`

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 Legacy Servlets>Servlet Directory page in the Server Manager 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 92.

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 Legacy 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 84) 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.

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 Legacy 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.

Before using a servlet virtual path, a servlet identifier (or servlet name) must be added for the servlet in the Legacy 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 Legacy 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.

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

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

2. Specify the virtual path for the servlet.

In the Legacy 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.

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.

Using the User Interface

For information about using the user interface to specify settings for legacy servlets and JSPs, see the following topics in the online help.

This page is located in the Web Server Administration Server on the Global Settings tab.

- The Configure JRE/JDK Paths Page

These pages are located in the Server Manager on the Java tab.

- The Enable/Disable Servlets/JSP Page
- The Configure JVM Attributes Page
- The Delete Version Files Page

These pages are located in the Server Manager on the Legacy Servlets tab.

- The Configure Global Servlet Attributes Page
- The Configure Servlet Attributes Page
- The Configure Servlet Virtual Path Translation Page
- The Configure Servlet Directory Page

Legacy Configuration Files

This section discusses the purpose and use of the following files:

- `servlets.properties`
- `rules.properties`
- `contexts.properties`

All of these files reside in the directory *server_id*/`config`.

NOTE In iPlanet Web Server 6.0, a `web-apps.xml` file exists for each virtual server, allowing you to configure separate web applications for each virtual server. The `servlets.properties`, `rules.properties`, and `contexts.properties` files apply only to the default virtual server for the server instance; they exist for backward compatibility. Using the `web-apps.xml` file for all web application configuration is recommended. For more information about the `web-apps.xml` file, see Chapter 2, "Web Applications."

servlets.properties

The `servlets.properties` file defines global servlet settings and the list of servlets in the system, for the default virtual server only.

An example of a global servlet setting is which servlet to run when the iPlanet Web Server starts up. 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 Legacy Servlets>Configure Servlet Attributes page in the Server Manager interface or by editing `servlets.properties` directly. Whenever you make a change in the Legacy 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 default document root,
# needed so ServletContext.getRealPath will work
servlets.config.docRoot=d:/Netscape/Server4/docs
# 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 the default virtual server only. 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 Legacy 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 peformance 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 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 the description of the `servlet-mapping` element in the `web.xml` file, which is described in the Servlet 2.2 API specification:

`http://java.sun.com/products/servlet/index.html`

For a way to direct files of a specific type to a specific servlet in the `obj.conf` file, see “Init Directives in magnus.conf,” on page 95.

contexts.properties

The `contexts.properties` file defines contexts, which allow multiple servlets to exchange data and access each other’s fields, for the default virtual server only. 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 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 instatiated 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>.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>.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>.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 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=contex_uri_base - additional URI prefix which
#                servces as a context base
# context.<context_name>.authdb=name - authentication database
# context.<context_name>.classpath=name - global classpath for this context
# context.<context_name>.signleClassLoader=(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.<context_name>.isModifiedCheckAggressive=(true|false) - determines
#                whether to be aggressively checking dependencies for the servlet
#                loader to reload modified servlets
#
# <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.respondCookieVersion=0
# context.global.tempDir=/tmp
# context.global.reloadInterval=5
# context.global.bufferSize=4096
# context.global.docRoot=/foo/bar
# context.global.inputStreamLengthCheck=true
# context.global.outputStreamFlushTimer=0
# context.global.uri=/
# context.global.authdb=default
# context.global.classpath=
# context.global.singleClassLoader=false
# context.global.contentTypeIgnoreFromSSI=true
# context.global.parameterEncoding=utf8
# context.global.isModifiedCheckAggressive=false
#
################################# Contexts #############################
context.global.initArgs=docRoot=C:/iPlanet/Servers
context.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:

`none`	Uses the system default encoding. Set this option if the encoding of the servlet parameter data is the same as the system default encoding.
`auto`	(Default) Tries to figure out the proper encoding from, in order, 1) the `charset` if it is set in the `Content-Type` header, 2) the `parameterEncoding` attribute (see "ServletRequest.setAttribute," on page 73), then 3) a hidden form field, such as `j_encoding`. 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 `none` or a specific encoding.
any supported java character encoding	A specific encoding, such as `utf8` or `Shift_JIS`. Set this option if you know the encoding that servlet parameters use. A complete list is available here: `http://java.sun.com/j2se/1.3/docs/guide/intl/encodin g.doc.html`

`responseCT`	Tries to figure out the proper encoding from the response content type if it is available. Otherwise, the system default encoding is used.

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) or `responseCT` 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 74.

The `parameterEncoding` property setting applies to how a servlet processes parameters after the iPlanet Web Server receives a request. The URI that is sent to the server must consist of only the standard ASCII set, especially if the request method is GET. All other characters must be encoded.

For example, to encode a backslash, you would replace the backslash with a `%` followed by the hexadecimal number corresponding to the backslash in the ASCII character set, `5c`. Therefore, `vw\xyz` becomes `vw%5cxyz`. For more information about characters that must be encoded, see "parameter-encoding," on page 22.

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`.

Servlet reloading (which occurs when `.class` or `.jsp` files are changed) does not work when `singleClassLoader=true`.

If `singleClassLoader=true`, `RequestDispatcher.forward` calls to JSPs do not work unless you add the `ClassCache` directory to the `context.global.classpath` line in `contexts.properties`. For example:

```
context.global.singleClassLoader=true
context.global.classpath=server_root/https-server_id/ClassCache
```

Legacy Examples

iPlanet Web Server 6.0 comes with a set of legacy example servlets and JSP files. You can find them at the following location:

server_root`/plugins/servlets/examples/legacy`

The `legacy` subdirectory contains the following directories:

- `beans.10` -- Contains example Java Bean files for JSP 1.*x*.
- `jsp.10` -- Contains subdirectories that each contain an example for JSP 1.*x*.
- `jsp.10/hangman` -- Contains a JSP application that was an SSJS application in iPlanet Web Server 4.*x*. For more information about converting SSJS applications to JSPs, see Appendix B, "Converting SSJS Applications."
- `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.
- `tools` -- Contains the `SDKTools.jar` file and other utility files.

Maximizing Legacy Servlet Performance

Consider the following guidelines for improving servlet performance:

- 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.
- In the default virtual server, 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).

Appendix A

Servlet Settings in magnus.conf and obj.conf

The iPlanet Web Server 6.0 Administration Server automatically modifies the `magnus.conf` and `obj.conf` files 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 these files appropriately.

However, in case you are interested in the settings that affect servlets, this appendix describes them in the following sections:

- Init Directives in magnus.conf
- Objects in obj.conf
- Directives for Registered Servlet Directories
- JSP Directives

Init Directives in magnus.conf

The following directives in the `Init` section of `magnus.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 parameter:

```
shlib="server_root/bin/https/lib/libNSServletPlugin.so"
```

`NSServletEarlyInit` takes an optional parameter, `cache_dir`, which 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 the following optional parameters:

`CatchSignals`	Specifies whether or not Java thread dumps are logged. The value is `yes` or `no`.
`requestDispatcherNestDepth`	Controls the nesting depth of the `RequestDispatcher.forward` and `include` methods. The default depth is `10`. This parameter also affects nesting levels of JSP `forward` and `include` invocations. You cannot assign an arbitrary value for this depth unless the `StackSize` `magnus.conf` directive is set to an appropriate value. Setting this parameter to an arbitrarily high value may result in stack overflow and a server crash.

Objects in obj.conf

NOTE The information in this section applies to legacy (iPlanet Web Server 4.*x*) servlets and is deprecated.

`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 `wasp` 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="wasp"
</Object>
```

For another way to direct files of a specific type to a specific servlet, see the description of the `servlet-mapping` element in the `web.xml` file, which is described in the Servlet 2.2 API specification:

`http://java.sun.com/products/servlet/index.html`

For a way to direct files of a specific type to a specific servlet as in iPlanet Web Server 4.*x*, see "Using Regular Expressions in rules.properties," on page 89.

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.

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

NOTE The information in this section applies to legacy (iPlanet Web Server 4.*x*) servlets and is deprecated.

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 Directives

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

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

The following directive in `obj.conf` handles the processing of requests for files of type `magnus-internal/jsp` (that is, JSP files). It is necessary for legacy JSPs only.

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

Appendix B

Converting SSJS Applications

This appendix contains information about converting Server-Side JavaScript applications to JSPs. It has the following sections:

- Differences Between JavaScript and Java
- JavaScript to Java Class Conversions
- Conversion Steps
- Example Conversion

Differences Between JavaScript and Java

Before you can convert SSJS applications to JSPs, you must understand the differences between JavaScript and Java. JavaScript and Java are similar in some ways but fundamentally different in others.

JavaScript lacks Java’s static typing and strong type checking. JavaScript supports a runtime system based on a small number of data types representing numeric, Boolean, and string values. Java has a compile-time system of classes built by declarations.

JavaScript also supports functions without any special declarative requirements. Functions can be properties of objects, executing as loosely typed methods. In Java, methods are defined in and belong to classes, and they are strongly typed.

JavaScript is a very free-form language compared to Java. In Java, you must declare all variables, classes, and methods. You must declare methods as public, private, or protected. Variables, parameters, and method return types are explicitly typed.

Java is a class-based programming language designed for fast execution and type safety. Type safety means, for instance, that you can't cast a Java integer into an object reference or access private memory by corrupting Java bytecodes. Java's class-based model means that programs consist exclusively of classes and their methods. Java's class inheritance and strong typing generally require tightly coupled object hierarchies. These requirements make Java programming more complex than JavaScript authoring.

Table B-1 JavaScript and Java compared

JavaScript	Java
Interpreted (not compiled) by client.	Compiled bytecodes downloaded from server, executed on client.
A scripting language that supports objects.	An object-oriented language.
No distinction between types of objects. Inheritance is through the prototype mechanism, and properties and functions can be added to any object dynamically.	Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically.
Functions may be inside classes but do not have to be.	Every method must be inside a class.
Code integrated with, and embedded in, HTML.	Code distinct from HTML, although JSPs can contain HTML tags.
Loosely cast: variable and function types not declared (dynamic typing).	Tightly cast: variable and method types must be declared (static typing).
Cannot automatically write to hard disk.	Cannot automatically write to hard disk.
Semicolons at the ends of statements are optional.	All statements must end with a semicolon (;).

JavaScript to Java Class Conversions

The objects used in an SSJS application must be converted to classes in Java. Table B-2 can help you convert these objects.

Table B-2 JavaScript and Java basic classes

SSJS Class	Java Class	Comments
`Client`	`Session`	A Java session is not as automatic as an SSJS client.
`Project`	`Context`	
`Request`	`Request`	
`Server`	`System` or `Context`	Methods are split between two Java classes.

SSJS includes a special library to handle database connectivity. Since JSPs are Java, all database connectivity is handled through JDBC. JDBC database drivers are available directly from the database vendors. It is a good idea to use pure Java (type 4) drivers whenever they are available. Table B-3 lists the database connectivity classes.

Table B-3 JavaScript and Java database classes

SSJS Class	Java Class	Comments
`Connection`	`Connection`	
`Cursor`	`ResultSet`	There is no `Cursor` object; you can use methods in the `ResultSet` class to move the cursor.
`DbPool`	`PooledConnection`	This class is in the JDBC optional package.
`Stproc`	`CallableStatement`	
`ResultSet`	`ResultSet`	

Conversion Steps

When converting applications, follow these steps:

1. Go over the structure of the application to see if there are helper classes that provide functionality between pages. Convert these helper classes first so that they are ready to use in your pages. (The `hangman` example contains a helper class, called `hangman.js` in the original JavaScript application and called `JavaHangManUtil.java` in the JSP application shipped with iPlanet Web Server 6.0.) Write helpers as `.java` files and compile them into `.class` files before using them.

2. Define the methods within classes. The methods (functions in JavaScript) must declare what they return and who can access them. For example, in JavaScript you can write:

   `function InitAnswer(str) {` *function_code* `}`

   In Java, the same method must be declared as follows:

   `public static String InitAnswer(String str) {` *method_code* `}`

   This method passes in a `String` and returns a `String`. The public keyword means that any other class can call this method. The static keyword means that the virtual machine can run this method without creating an object of the class containing the method.

3. Convert the client object of the application to a session bean. In Server-Side JavaScript, data is easier to use in the client object. In a Java Session object, the programmer must get data from the session and then put it back in if there are changes. This work can be done in a wrapper bean. The bean handling functionality of JSPs is very strong. You can specify the scope to be `session` and it is saved in the session automatically.

4. Convert the HTML files to JSP files. This mainly consists of finding the `<SERVER>` and `</SERVER>` tags and replacing them with the `<%` and `%>` JSP tags. Then go through and change the syntax to correct JSP or Java syntax.

5. It is sometimes necessary to add special lines to the file. For example, the line

   ```
   <%@ page import="HangManUtil,HangBean" %>
   ```

   imports helper classes, and the line

   ```
   <jsp:useBean id="client" scope="session" class="HangBean"/>
   ```

   uses a session bean.

Example Conversion

The `hangman` example, which was a Server-Side JavaScript example in the 4.1 version of iPlanet Web Server, have been converted to a JavaServer Pages example in the 6.0 version. You can find the converted example under the following directory:

server_root`/plugins/servlets/examples/legacy/jsp.10/hangman`

Appendix C

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).

NOTE	A few attributes on the Configure JVM Attributes page on the Java 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, `jvm.compiler=NONE`, is added to the `jvm12.conf` file.

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`

The jvm12.conf File

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=NONE
```

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.enabledebug=1
jvm.classpath=/usr/java/tools.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.

A list of debugging options that you can include in the `jvm12.conf` file if the JPDA is installed is available here:

`http://java.sun.com/products/jpda/doc/conninv.html#Invocation`

For more information about debugging servlets and JSPs and the `jvm12.conf` parameters required for debugging, see Chapter 5, “Debugging Servlets and JSPs.”

Using JVM Environment Variables

If you define JVM environment variables using the `jvm.option` setting in the `jvm12.conf` file or the Administration Server, servlets do not recognize them. Just use the plain *name*=*value* format for environment variables.

When you are running a stand-alone command line java program, use the following option to pass system properties into the java program:

```
java -Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB myprogram
```

In the `myprogram.java` file, the following line retrieves the system property set above:

```
System.out.println("org.omg.CORBA.ORBClass="+System.getProperty("org.omg.CORBA.ORBClass"));
```

If you want to do the same thing for servlets in iPlanet Web Server, you need to put the following line in the `jvm12.conf` file:

```
org.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
```

Do not use `jvm.option` settings such as the following:

```
jvm.option=-Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
```

or

```
jvm.option=org.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
```

In the servlet or JSP, you can use the following line to retrieve the system property set above:

```
out.println("org.omg.CORBA.ORBClass="+System.getProperty("org.omg.CORBA.ORBClass"));
```

jvm12.conf Parameter Reference

The JVM parameters you can set in the `jvm12.conf` file are listed in Table C-1.

Table C-1 jvm12.conf settings

Setting	Allowed Values	Default Value	Description
`jvm.minHeapSize`		`1048576` (1 MB)	The minimum heap size allocated to Java. For Solaris, change this value to `3145278` (3 MB). For HPUX, change this value to `4194304` (4 MB). For all other operating systems, 1 MB is ideal.
`jvm.maxHeapSize`		`16777216` (16 MB)	The maximum heap size allocated to Java.
`jvm.enableClassGC`	`0` (off), `1` (on)	`0`	Enables or disables class garbage collection. Use instead of `-Xnoclassgc`.

Table C-1 jvm12.conf settings

Setting	Allowed Values	Default Value	Description
`jvm.verboseMode`	`0` (off), `1` (on)	`0`	Enables or disables JVM verbose mode. If on, the JVM logs a commentary on what it is doing, such as loading classes. The commentary appears in the error log.
`jvm.enableDebug`	`0` (off), `1` (on)	`0`	Enables or disables JVM remote debugging. Use instead of `-Xdebug`. For more information about remote debugging, see Chapter 5, "Debugging Servlets and JSPs."
`jvm.printErrors`	`0` (off), `1` (logs to log file), `2` (logs to `stderr`)	`0`	Enables or disables reporting of errors through `vfprintf`.
`jvm.option`			Allows you to set vendor JVM options. The following options are ignored, because there are equivalents to these settings: `-D`, `-Xnoclassgc`, `-Xdebug`, `-Xms`, `-Xmx`, `-verbose`
`jvm.profiler`			Specifies the profiler. If you use the `optimizeit` profiler from Intuitive Systems, you must also set the `OPTIDIR` setting. For more information about this optimizer, see Appendix D, "Remote Servlet Profiling."
`jvm.disableThreadRecycling`	`0` (off), `1` (on)	`0`	Enables or disables thread recycling. If on, the server always creates a global scope thread to execute servlets. Otherwise a global scope thread is created only when the request handling thread is not in the global scope.

Table C-1 jvm12.conf settings

Setting	Allowed Values	Default Value	Description
`jvm.serializeAttach`	`0` (off), `1` (on)	`0`	If on, threads that attach to the JVM are serialized. By default (if off), threads can attach to the JVM in parallel.
`jvm.stickyAttach`	`0` (off), `1` (on)	`0`	Setting the value of this parameter to 1 causes threads to remember that they are attached to the JVM. This speeds 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 the *iPlanet Web Server Administrator's Guide.*
`jvm.trace`		5	Determines the trace level. For servlet and JSP debugging, the recommended level is 7. Level 5 displays servlet engine messages. Level 6 displays servlet and JSP engine messages. Level 7 displays these and other exceptions in the browser.
`jvm.allowExit`	`0` (off), `1` (on)	`0`	Enables or disables exit from the process.

Table C-1 jvm12.conf settings

Setting	Allowed Values	Default Value	Description
`java.compiler`			Specifies the Java compiler. See your JVM documentation for options that turn the JIT (just in time) compiler on and off. This should be set to `NONE` when `jvm.enableDebug` is on.
`OPTITDIR`	A path	*	Specifies the path to the profiler if the profiler is `optimizeit`.
`nes.jsp.enabledebug`	`0` (off), `1` (on)	`1`	Enables or disables verbose JSP compilation tracing.
			For more information about remote debugging, see Chapter 5, “Debugging Servlets and JSPs.”
`jvm.include.CLASSPATH`	`0` (off), `1` (on)	`1`	Specifies whether to include the `CLASSPATH` environment variable value in the `jvm.classpath` setting.
`nes.jsp.forkjavac`	`0` (off), `1` (on)	`0`	If on, Java compilation of JSPs runs in a separate process.
`jvm.serializeFirstRequest`	`0` (off), `1` (on)	`1` for Linux, AIX, and Compaq (DEC); `0` for other platforms	If on, ensures that only one request thread loads and constructs a servlet object. Once a servlet is loaded and initialized, new requests to the same servlet happen in parallel. This setting must be on for Linux, AIX, and Compaq (DEC).

Table C-1 jvm12.conf settings

Setting	Allowed Values	Default Value	Description
`jvm.classpath`	A path with forward slashes only		Specifies the path(s) to JAR files dependent on the JVM. 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. If you are using a session manager that uses a JDBC driver, be sure to include the driver's JAR file in this classpath.

* *N*`:/App/IntuitiveSystems/OptimizeIt30D`, where *N* is the drive on which OptimizeIt is installed.

Appendix D

Remote Servlet Profiling

You can use a profiler to perform remote profiling on the iPlanet Web Server to discover bottlenecks in server-side performance. This appendix describes two profilers:

- The Optimizeit! Profiler
- The HPROF Profiler

The Optimizeit! Profiler

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 6.0.

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

```
jvm.enableClassGC=0
jvm.option=-Xrunoii
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.

The HPROF Profiler

HPROF is a simple profiler agent shipped with the Java 2 SDK. It is a dynamically linked library that interacts with the JVMPI and writes out profiling information either to a file or to a socket in ASCII or binary format. This information can be further processed by a profiler front-end tool such as HAT.

HPROF can present CPU usage, heap allocation statistics, and monitor contention profiles. In addition, it can also report complete heap dumps and states of all the monitors and threads in the Java virtual machine. For more details on the HPROF profiler, see the JDK documentation at:

`http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/jvmpi.html#hprof`

To use HPROF profiling on Unix, follow these steps:

1. To enable HPROF profiling, edit the `jvm12.conf` file as shown here:

```
jvm.printErrors=2
jvm.profiler=hprof
jvm.option=-Xrunhprof:options
#jvm.option=-Xrs must be commented out
java.compiler=NONE
```

Suggested options for using iPlanet Web Server with HPROF are:

`jvm.option=-Xrunhprof:file=/tmp/hprof.txt,heap=all,format=a`

or:

`jvm.option=-Xrunhprof:file=/tmp/hprof.txt,cpu=samples,format=a`

The syntax of HPROF is as follows:

`-Xrunhprof[:help]|[:`*option*`=`*value*`,`*option2*`=`*value2*`, ...]`

Using `help` lists options that can be passed to HPROF. The output is as follows:

```
Hprof usage: -Xrunhprof[:help]|[:<option>=<value>, ...]

Option Name and Value   Description             Default
---------------------   -----------             -------
heap=dump|sites|all     heap profiling          all
cpu=samples|old         CPU usage               off
format=a|b              ascii or binary output  a
file=<file>             write data to file      java.hprof
                                                (.txt for ascii)
net=<host>:<port>       send data over a socket write to file
depth=<size>            stack trace depth       4
cutoff=<value>          output cutoff point     0.0001
lineno=y|n              line number in traces?  y
thread=y|n              thread in traces?       n
doe=y|n                 dump on exit?           y
```

2. You must also change a line in the iPlanet Web Server start script. The start script file is *server_root*/`https-`*server_id*/`start`. Change the following line:

   ```
   PRODUCT_BIN=uxwdog
   ```

   to this:

   ```
   PRODUCT_BIN=ns-httpd
   ```

3. Start the server by running the start script. Since the server runs in the foreground (the change in step 2), the command prompt returns only after the server has been stopped.

4. In another window or terminal, find the process ID of the server process.

   ```
   % ps -ef | grep ns-httpd
   ```

   This command lists two `ns-httpd` processes. Look at the PPID (parent process ID) column and identify which of the two processes is the parent process and which is the child process. Note the PID (process ID) of the child process ID.

5. Send a SIGQUIT signal (signal 3) to the child process:

   `% kill -QUIT` *child_PID*

This displays the following ASCII menu in the window from where the start script was invoked:

```
% start
iPlanet-WebServer-Enterprise/6.0
[LS ls1] http://test, port 9000 ready to accept requests
Default selection: alloc and heap dump
startup: server started successfully
SIGQUIT
A SIGQUIT has been received. Do you want to:
[ 0 ]  continue program
[ 1 ]  check & print one deadlock
[ 2 ]  check & print all deadlocks
[ 3 ]  dump thread stacks
[ 4 ]  dump lock registry
[ 5 ]  heap inspection
[ 6 ]  hprof dump
[ 7 ]  terminate program
Type number corresponding to selected action:
```

6. Select menu option 6 by typing the number 6 and then typing the Enter key.

   This causes HPROF data to be saved in the file specified in the `jvm.option` line in `jvm12.conf`, for example `/tmp/hprof.txt`.

7. Type `0` and press Enter to continue the program (the Web Server).
8. To capture more HPROF profiles, repeat steps 6 and 7.
9. To stop the Web Server, run the stop script from another window.

   ```
   % ./stop
   ```

10. Undo the changes in steps 1 and 2 to return your Web Server to its original configuration.

Index