Sun Java System Web Server 6.1 SP9 Administrator's Configuration File Reference

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–5703

080731@20490

Contents

Preface

This guide discusses the purpose and use of the configuration files for Sun Java System Web Server 6.1, including `server.xml`, `magnus.conf`, and `mime.types`, and provides comprehensive lists of the elements and directives in these configuration files.

This preface contains information about the following topics:

Who Should Use This Guide

The intended audience for this guide is one who administers and maintains the Sun Java System Web Server.

This guide assumes you are familiar with the following topics:

- J2SE specification
- HTTP
- HTML
- XML
- Java programming
- Java APIs as defined in servlet, JSP, and JDBC specifications
- Relational database concepts

Using the Documentation

The Sun Java System Web Server manuals are available as online files in PDF and HTML formats at `http://docs.sun.com/app/docs/coll/1308.5`.

The following table lists the tasks and concepts described in the Sun Java System Web Server manuals.

TABLE P–1 Sun Java System Web Server Documentation Roadmap

For Information About	See the Following
Late-breaking information about the software and documentation	*Release Notes*
Getting started with Sun Java System Web Server, including hands-on exercises that introduce server basics and features (recommended for first-time users)	*Getting Started Guide*
Performing installation and migration tasks: ■ Installing Sun Java System Web Server and its various components, supported platforms, and environments ■ Migrating from Sun Java System Web Server 4.1 or 6.0 to Sun Java System Web Server 6.1	*Installation and Migration Guide*
Performing the following administration tasks: ■ Using the Administration and command-line interfaces ■ Configuring server preferences ■ Using server instances ■ Monitoring and logging server activity ■ Using certificates and public key cryptography to secure the server ■ Configuring access control to secure the server ■ Using Java™ 2 Platform, Standard Edition (J2SE platform) security features ■ Deploying applications ■ Managing virtual servers ■ Defining server workload and sizing the system to meet performance needs ■ Searching the contents and attributes of server documents, and creating a text search interface ■ Configuring the server for content compression ■ Configuring the server for web publishing and content authoring using WebDAV	*Administrator's Guide*

TABLE P–1 Sun Java System Web Server Documentation Roadmap *(Continued)*

For Information About	See the Following
Using programming technologies and APIs to do the following: ■ Extend and modify Sun Java System Web Server ■ Dynamically generate content in response to client requests ■ Modify the content of the server	*Programmer's Guide*
Creating custom Netscape Server Application Programmer's Interface (NSAPI) plug-ins	*NSAPI Programmer's Guide*
Implementing servlets and JavaServer Pages™ (JSP) technology in Sun Java System Web Server	*Programmer's Guide to Web Applications*
Editing configuration files	*Administrator's Configuration File Reference Guide*
Tuning Sun Java System Web Server to optimize performance	*Performance Tuning, Sizing, and Scaling Guide*

How This Guide Is Organized

This guide has the following chapters:

- Chapter 1, Basics of Server Operation

  Introduces the major configuration files that control the Sun Java System Web Server and describes how to activate and edit them.
- Chapter 2, Server Configuration Elements in server.xml

  Describes the `server.xml` file, which controls most aspects of server operation.
- Chapter 3, Syntax and Use of magnus.conf

  Describes the directives you can set in the `magnus.conf` file to configure the Sun Java System Web Server during initialization.
- Chapter 4, Predefined SAFs in obj.conf

  describes the predefined SAFs used in the obj.conf file.
- Chapter 5, MIME Types

  Describes the MIME types file, which maps file extensions to file types.
- Chapter 6, Other Server Configuration Files

  Lists other important configuration files and provides a quick reference of their contents.
- Appendix A, Configuration Changes Between iPlanet Web Server 4.1 and Sun Java System Web Server 6.1

  Describes the changes in configuration files between the 4.x and 6.1 versions of Sun Java System Web Server.

- Appendix B, Configuration Changes Between iPlanet Web Server 6.0 and Sun Java System Web Server 6.1

  Describes the changes in configuration files between the 6.0 and 6.1 versions of Sun Java System Web Server.
- Appendix C, Time Formats

  Describes the format strings used for dates and times in the server log.
- Appendix D, Alphabetical List of Server Configuration Elements

  Provides an alphabetical list for easy lookup of elements in `server.xml` and directives in `magnus.conf`.
- Appendix E, Alphabetical List of Predefined SAFs

  Provides an alphabetical list for easy lookup of directives in `obj.conf`.

Documentation Conventions

This section describes the types of conventions used throughout this guide:

- **File and directory paths** are given in UNIX® format (with forward slashes separating directory names). For Windows versions, the directory paths are the same, except that backslashes are used to separate directories.
- **URLs** are given in the format:

  http://*server.domain*/*path*/*file*.html

  In these URLs, server is the server name where applications are run; domain is your Internet domain name; path is the server's directory structure; and file is an individual filename. Italic items in URLs are placeholders.
- **Font conventions** include:
  - The `monospace` font is used for sample code and code listings, API and language elements (such as function names and class names), file names, pathnames, directory names, and HTML tags.
  - *Italic* type is used for code variables.
  - *Italic* type is also used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
  - **Bold** type is used as either a paragraph lead-in or to indicate words used in the literal sense.

  Installation root directories are indicated by *install_dir* in this document.

  By default, the location of *install_dir* on UNIX-based platforms is:

  ```
  /opt/SUNWwbsvr/
  ```

On Windows, it is:

```
C:\Sun\WebServer6.1
```

Product Support

If you have problems with your system, contact customer support using one of the following mechanisms:

- The online support web site at:

  http://www.sun.com/training/
- The telephone dispatch number associated with your maintenance contract

Please have the following information available prior to contacting support. This helps to ensure that our support staff can best assist you in resolving problems:

- Description of the problem, including the situation where the problem occurs and its impact on your operation
- Machine type, operating system version, and product version, including any patches and other software that might be affecting the problem
- Detailed steps on the methods you have used to reproduce the problem
- Any error logs or core dumps

◆ ◆ ◆ CHAPTER 1

Basics of Server Operation

The configuration and behavior of Sun Java™ System Web Server is determined by a set of configuration files. When you use the Administration interface, you change the settings in these configuration files. You can also manually edit these files.

This chapter has the following sections:

Configuration Files

The configuration and operation of the Sun Java System Web Server is controlled by configuration files. The configuration files reside in the directory *instance_dir*/`config`. This directory contains various configuration files for controlling different components. The exact number and names of configuration files depends on which components have been enabled or loaded into the server.

However, this directory always contains four configuration files that are essential for the server to operate. These files are:

- "server.xml" on page 22 -- contains most of the server configuration.
- "magnus.conf" on page 22 -- contains global server initialization information.
- "obj.conf" on page 22 -- contains instructions for handling HTTP requests from clients.
- "mime.types" on page 23 -- contains information for determining the content type of requested resources.

server.xml

`server.xml`file contains most of the server configuration. A schema file, `sun-web-server_6_1.dtd`, defines its format and content.

For more information about how the server uses `sun-web-server_6_1.dtd` and `server.xml`, see Chapter 2, Server Configuration Elements in server.xml.

magnus.conf

This file sets values of variables that configure the server during initialization. The server executes the settings in this file on startup. The server does not check at this file again until it is restarted.

See Chapter 3, Syntax and Use of magnus.conf for a list of all the variables and Init directives that can be set in `magnus.conf`.

obj.conf

This file contains instructions for the Sun Java System Web Server about how to handle HTTP requests from clients and service web server content such as native server plug-ins and CGI programs. The server checks the configuration defined by this file every time it processes a request from a client.

This file contains a series of instructions (directives) that tell the Sun Java System Web Server what to do at each stage in the request-response process.You can modify and extend the request handling process by adding or changing the instructions in `obj.conf`.

All `obj.conf` files are located in the *server_root*`/config` directory. There is one `obj.conf` file for each virtual server class. Whenever this guide refers to "the `obj.conf` file," it refers to all `obj.conf` files or to the `obj.conf` file for the virtual server class described.

By default, each active `obj.conf` file is named *vs_class*`-obj.conf`. Editing one of these files directly or through the Administration interface changes the configuration of a virtual server class.

The `obj.conf` file is essential to the operation of the Sun Java System Web Server. When you make changes to the server through the Administration interface, the system automatically updates `obj.conf`.

For information about how the server uses `obj.conf`, see Chapter 4, Predefined SAFs in obj.conf.

mime.types

This file maps file extensions to MIME types to enable the server to determine the content type of a requested resource. For example, requests for resources with `.html` extensions indicate that the client is requesting an HTML file, requests for resources with `.gif` extensions indicate that the client is requesting an image file in GIF format.

For more information about how the server uses `mime.types`, see "MIME Types."

Other Configuration Files

For information about other important configuration files, see Chapter 6, Other Server Configuration Files.

Directory Structure

The following section describes the directory structure created when you first install Sun Java System Web Server 6.1. The information is organized in two parts:

- "All Platforms" on page 23
- "UNIX and Linux Platforms" on page 25

All Platforms

For all platforms, the following directories are created under the server root directory:

- **alias** contains the key and certificate files for all Sun Java System servers (for example, `https-admserv-server_id-cert8.db` and `secmod.db`).
- **bin** contains the binary files for the server, such as the actual server, the Administration Server forms, and so on. In addition, this directory includes the `https/install` folder that contains files needed for migrating server settings and default configuration files needed for backward compatibility.
- **docs** is the server's default primary document directory, where your server's content files are usually kept. If you are migrating settings from an existing server, this directory doesn't appear until you finish the migration process.
- **extras** contains the log analyzer and log analysis tools.
  - The `flexanlg` directory contains a command-line log analyzer. This log analyzer analyzes files in flexlog format.
  - The `log_anly` directory contains the log analysis tool that runs through the Server Manager. This log analyzer analyzes files in common log format only.

httpacl contains the files that store access control configuration information in the `generated.`*server-id*`.acl` and `genwork.`*server-id*`.acl` files. The file `generated.`*server-id*`.acl` contains changes you make using the Server Manager access control forms after saving your changes; `genwork.`*server-id*`.acl` contains your changes *before* you save your changes.

- **https-admserv** contains the directories for the Administration Server. This directory has the following subdirectories and files:
  - For UNIX/Linux platforms, this directory contains shell scripts to start, stop, and restart the server and a script to rotate log files.
  - `ClassCache` contains classes and Java files, generated as a result of the compilation of JavaServer pages.
  - `conf_bk` contains backup copies of the administration server's configuration files.
  - `config` contains the server's configuration files.
  - `logs` contains any error or access log files.
  - `SessionData` contains session database data from `MMapSessionManager`.
  - `startsvr.bat` is the script that starts the Server Manager on Windows machines. The Server Manager lets you configure all servers installed in the server root directory.
  - `stopsvr.bat` is the script that stops the Server Manager on Windows machines.

  https-server_id are the directories for each server you have installed on the machine. Each server directory has the following subdirectories and files:
  - `ClassCache` contains classes and Java files, generated as result of the compilation of JavaServer pages.
  - `conf_bk` contains backup copies of the server's configuration files.
  - `config` contains the server instance configuration files.
  - `logs` contains the server instance log files.
  - `reconfig` is the script used to reconfigure the server dynamically. If you make non-global changes to the server, you can use this script to reconfigure the server without stopping and starting it. Note that changes to ACL files and `magnus.conf` require you to stop and restart the server.
  - `restart` is the script that restarts the server.
  - `rotate` rotates server log files without affecting users who may be connected to the server.
  - `search` contains the following directories: `admin` and `collections`
  - `SessionData` contains session database data from `MMapSessionManager`.
  - `startsvr.bat` is the script that starts the Server Manager. The Server Manager lets you configure all servers installed in the server root directory.
  - `stopsvr.bat` is the script that stops the Server Manager.

manual contains the online manuals for the product.

- **plug-ins** contains directories for Java, search, and other plug-ins. This directory has the following subdirectories:
  - `htaccess` contains server plug-in for `.htaccess` access control and `htconvert`, an `.nsconfig` to `.htaccess` converter.
  - `digest` contains the Digest Authentication Plug-in for Sun Java System Directory Server 5.0, as well as information about the plug-in.
  - `samples` contains samples and example components, plug-ins and technologies supported by the Sun Java System Web Server servlet engine. This includes binaries, all code, and a build enviroment.
  - `servlets` contains information about and examples of web-apps applications.
  - `include` contains various include files.
  - `lib` contains shared libraries.
  - `nsacl` contains information for your server's access control lists.
  - `loadbal` contains the required files for the Resonate load-balancer integration plug-in.
  - `nsapi` contains header files and example code for creating your own functions using NSAPI. For more information, see the Sun Java System documentation web site at:

    `http://docs.sun.com/app/docs/coll/1308.5`
  - `search` contains information for your server's search plug-ins.
  - `snmp` contains information for your server's SNMP plug-ins.

  setup contains the various Sun Java System Web Server setup files, including `setup.log` and `uninstall.inf`.
- *userdb* contains user databases and related information.
- *LICENSE.txt* is the license file.
- *README.txt* is the readme file that contains a link to the Sun Java System Web Server *Release Notes*.

UNIX and Linux Platforms

In addition to the files and directories described in "All Platforms" on page 23 the following files are created at the `server-root` directory for UNIX and Linux platforms:

- **startconsole** launches a browser to the Administration Server page.

  The following files are created under the *server-root*`/https-admserv` directory for UNIX and Linux platforms:
  - `ClassCache` contains classes and Java files, generated as result of the compilation of JavaServer pages.

- `conf_bk` contains backup copies of the server's configuration files.
- `config` contains the Administration Server configuration files.
- `logs` contains the Administration Server log files.
- `SessionData` contains session database data from `MMapSessionManager`.
- `restart` is the script that restarts the Server Manager.
- `start` is the script that starts the Server Manager. The Server Manager lets you configure all servers installed in the server root directory.
- `stop` is the script that stops the Server Manager.

Dynamic Reconfiguration

Dynamic reconfiguration enables you to make configuration changes to a live web server without having to stop and restart the web server for the changes to take effect. You can dynamically change all configuration settings and attributes in the `server.xml` file and its associated files without restarting the server.

To access the dynamic reconfiguration screen and install a new configuration dynamically, click the Apply link in the upper right corner of the Server Manager, Class Manager, and Virtual Server Manager pages, click the Load Configuration Files button on the Apply Changes page. If there are errors in installing the new configuration, the previous configuration is restored.

◆ ◆ ◆ CHAPTER 2

Server Configuration Elements in server.xml

The `server.xml` file contains most of the server configuration. The encoding is UTF-8 to maintain compatibility with regular UNIX text editors. The `server.xml` file is located in the *instance_dir*`/config` directory. A schema file, `sun-web-server_6_1.dtd`, determines the format and content of the `server.xml` file.

This chapter describes `server.xml` and `sun-server_1_0.dtd` in the following sections:

sun-web-server_6_1.dtd File

The `sun-web-server_6_1.dtd` file defines the structure of the `server.xml` file, including the elements it can contain and the subelements and attributes these elements can have. The `sun-web-server_6_1.dtd` file is located in the *install_dir*`/bin/https/dtds` directory.

Each element defined in a DTD file (which might be present in the corresponding XML file) can contain the following:

Subelements

Elements can contain subelements. For example, the following file fragment defines the `VSCLASS` element.

```
<!ELEMENT VSCLASS (VARS?, VS*, QOSPARAMS?)>
```

The ELEMENT tag specifies that a `VSCLASS` element can contain `VARS`, `VS`, and `QOSPARAMS` elements in that order.

The following table shows how optional suffix characters of subelements determine the requirement rules or number of allowed occurrences for the subelements.

TABLE 2–1 Requirement Rules and Subelement Suffixes

Subelement Suffix	Requirement Rule
*element**	Can contain *zero or more* of this subelement
element?	Can contain *zero or one* of this subelement
element+	Must contain *one or more* of this subelement
element (no suffix)	Must contain *only one* of this subelement

If an element cannot contain other elements, you see EMPTY or (#PCDATA) instead of a list of element names in parentheses.

Data

Some elements contain character data instead of subelements. These elements have definitions of the following format:

```
<!ELEMENT element-name (#PCDATA)>
```

For example:

```
<!ELEMENT DESCRIPTION (#PCDATA)>
```

In the `server.xml` file, white space is treated as part of the data in a data element. Therefore, there should be no extra white space before or after the data delimited by a data element. For example:

```
<DESCRIPTION>myserver</DESCRIPTION>
```

Attributes

Elements that have ATTLIST tags contain attributes (name-value pairs). For example:

```
<!ATTLIST JDBCRESOURCE

jndiname CDATA #REQUIRED

poolname CDATA #REQUIRED

enabled %boolean; "true">
```

A `JDBCRESOURCE` element can contain `jndiname`, `poolname`, and `enabled` attributes.

The #REQUIRED label means that a value must be supplied. The `#IMPLIED` label means that the attribute is optional, and that Sun Java System Web Server generates a default value. Wherever possible, explicit defaults for optional attributes (such as "`true`") are listed.

Attribute declarations specify the type of the attribute. For example, `CDATA` means character data, and `%boolean` is a predefined enumeration.

Elements in the server.xml File

This section describes the XML elements in the `server.xml` file. Elements are grouped as follows:

Note – Subelements must be defined in the order in which they are listed under each Subelements heading unless otherwise noted.

For an alphabetical listing of elements in `server.xml`, see "Appendix D, Alphabetical List of Server Configuration Elements.

Core Server Elements

General elements are as follows:

SERVER

Defines a server. This is the root element. There can only be one server element in a `server.xml` file.

Subelements

The following table describes subelements for the SERVER element.

TABLE 2–2 SERVER Subelements

Element	Required	Description
"VARS" on page 32	zero or one	Defines variables that can be given values in `server.xml` and referenced in `obj.conf`
"PROPERTY" on page 31	zero or more	Specifies a property of the serve.
"LS" on page 33	one or more	Defines one or more HTTP listen sockets
"MIME" on page 36	zero or more	Defines the mime type
"ACLFILE" on page 37	zero or more	References one or more ACL files
"VSCLASS" on page 38	one or more	Defines a virtual server class
"QOSPARAMS" on page 41	zero or one	Defines quality of service parameters
"JAVA" on page 50	zero or one	Configures Java™ Virtual Machine (JVM) parameters
"LOG" on page 65	zero or one	Configures the system logging service

Attributes

The following table describes attributes for the SERVER element.

TABLE 2–3 SERVER Attributes

Attribute	Default	Description
`qosactive`	`no`	Enables quality of service features, which let you set limits on server entities or view server statistics for bandwidth and connections. Allowed values are `yes`, `no`, `on`, `off`, `true`, `false`, `1` or `0`.
`qosmetricsinterval`	`30`	(optional) The interval in seconds during which the traffic is measured.
`qosrecomputeinterval`	`100`	(optional) The period in milliseconds in which the bandwidth gets recomputed for all server entities.

PROPERTY

Specifies a property, or a variable that is defined in `server.xml` and referenced in `obj.conf`. For information about variables, see "Variables" on page 69.

For a list of variables commonly defined in server.xml, see "Variables Used in the Interface."

A property adds configuration information to its parent element that is one or both of the following:

- Optional with respect to Sun Java System Web Server
- Needed by a system or object that Sun Java System Web Server does not have knowledge of, such as an LDAP server or a Java class

  For example, an `AUTHREALM` element can include `PROPERTY` subelements:

  ```
  <AUTHREALM name="file"
  classname="com.iplanet.ias.security.auth.realm.file.FileRealm">
  <PROPERTY name="file" value="instance_dir/config/keyfile>
  <PROPERTY name="jaas-context" value="fileRealm">
  </AUTHREALM>
  ```

  The properties that an `AUTHREALM` element uses depends on the value of the `AUTHREALM` element's name attribute. The `file` realm uses `file` and `jaas-context` properties. Other realms use different properties.

Subelements

The following table describes subelements for the `PROPERTY` element.

TABLE 2–4 PROPERTY Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of the property.

Attributes

The following table describes attributes for the `PROPERTY` element.

TABLE 2–5 PROPERTY Attributes

Attribute	Default	Description
`name`	none	Specifies the name of the property or variable.
`value`	none	Specifies the value of the property or variable.

DESCRIPTION

Contains a text description of the parent element.

Subelements

None

Attributes

None

VARS

Defines variables that can be given values in `server.xml` and referenced in `obj.conf`. For more information, see "Variables" on page 69.

Subelements

none

Attributes

none

Listener Elements

The listener elements are as follows:

LS

Defines an HTTP listen socket.

Note – When you create a secure listen socket through the Server Manager, security is automatically turned on globally in `magnus.conf`. When you create a secure listen socket manually in `server.xml`, security must be turned on by editing `magnus.conf`.

The `CONNECTIONGROUP` element from the schema file for `server.xml` in version 6.0 of Web Server is no longer supported. Its attributes and the subelement `SSLPARAMS` are added to the `LS` element in Sun Java System Web Server 6.1.

Subelements

The following table describes subelements for the `LS` element.

TABLE 2–6 LS Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of the listen socket.
"SSLPARAMS" on page 35	zero or one	Defines Secure Socket Layer (SSL) parameters.

Attributes

The following table describes attributes for the `LS` element.

TABLE 2–7 LS Attributes

Attribute	Default	Description
`id`	none	(Optional) The socket family type. A socket family type cannot begin with a number. When you create a secure listen socket in the `server.xml` file, security must be turned on in `magnus.conf`. When you create a secure listen socket in the Server Manager, security is automatically turned on globally in `magnus.conf`.
`ip`	any	Specifies the IP address of the listen socket. Can be in dotted-pair or IPv6 notation. Can also be `any` for `INADDR_ANY`.
`port`	none	Port number to create the listen socket on. Legal values are `1` - `65535`. On UNIX, creating sockets that listen on ports `1` - `1024` requires superuser privileges. Configuring an SSL listen socket to listen on port 443 is recommended. Two different IP addresses can't use the same port.
`security`	`false`	(Optional) Determines whether the listen socket runs SSL. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`. You can turn SSL2 or SSL3 on or off and set ciphers using an `SSLPARAMS` subelement for this listen socket. The `Security` setting in the `magnus.conf` file globally enables or disables SSL by making certificates available to the server instance. Therefore, `Security` in `magnus.conf` must be `on` or security in `server.xml` does not work. For more information, see Chapter 3, Syntax and Use of magnus.conf
`acceptorthreads`	`1`	(Optional) Number of acceptor threads for the listener. The recommended value is the number of processors in the machine. Legal values are `1` - `1024`.
`family`	none	(Optional) The socket family type. Legal values are `inet, inet6, and nca`. Use the value `inet6` for IPv6 listen sockets. When using the value of `inet6`, IPv4 addresses are prefixed with `::ffff:` in the log file. Specify `nca` to use the Solaris Network Cache and Accelerator.

TABLE 2–7 LS Attributes *(Continued)*

Attribute	Default	Description
`blocking`	false	(Optional) Determines whether the listen socket and the accepted socket are put in to blocking mode. Use of blocking mode may improve benchmark scores. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`defaultvs`	none	The `id` attribute of the default virtual server for this particular listen socket.
`servername`	none	Tells the server what to put in the host name section of any URLs it sends to the client. This affects URLs the server automatically generates. it does not affect the URLs for directories and files stored in the server. This name should be the alias name if your server uses an alias. If you append a colon and port number, that port will be used in URLs the server sends to the client.

SSLPARAMS

Defines SSL (Secure Socket Layer) parameters.

Subelements

None

Attributes

The following table describes attributes for the `SSLPARAMS` element.

TABLE 2–8 SSLPARAMS Attributes

Attribute	Default	Description
`servercertnickname`	`Server-Cert`	The nickname of the server certificate in the certificate database or the PKCS#11 token. In the certificate, the name format is *tokenname*:*nickname*. Including the *tokenname*: part of the name in this attribute is optional.

TABLE 2–8 SSLPARAMS Attributes *(Continued)*

Attribute	Default	Description
ssl2	false	(Optional) Determines whether SSL2 is enabled. Legal values are on, off, yes, no, 1, 0, true, and false. If both SSL2 and SSL3 are enabled for a virtual server, the server tries SSL3 encryption first. If that fails, the server tries SSL2 encryption.
ssl2ciphers	none	(Optional) A space-separated list of the SSL2 ciphers used, with the prefix + to enable or - to disable. For example +rc4. Allowed values are rc4, rc4export, rc2, rc2export, idea, des, desede3.
ssl3	true	(optional) Determines whether SSL3 is enabled. Legal values are on, off, yes, no, 1, 0, true and false. If both SSL2 and SSL3 are enabled for a virtual server, the server tries SSL3 encryption first. If that fails, the server tries SSL2 encryption.
ssl3tlsciphers	none	(optional) A space-separated list of the SSL3 ciphers used, with the prefix + to enable or - to disable, for example +rsa_des_sha. Allowed SSL3 values are rsa_rc4_128_md5, rsa_3des_sha, rsa_des_sha, rsa_rc4_40_md5, rsa_rc2_40_md5, rsa_null_md5. Allowed TLS values are rsa_des_56_sha, rsa_rc4_56_sha.
tls	true	(optional) Determines whether TLS is enabled. Legal values are on, off, yes, no, 1, 0, true, and false.
tlsrollback	true	(optional) Determines whether TLS rollback is enabled. Legal values are on, off, yes, no, 1, 0, true, and false. TLS rollback should be enabled for Microsoft Internet Explorer 5.0 and 5.5.
clientauth	false	(optional) Determines whether SSL3 client authentication is performed on every request, independent of ACL-based access control. Legal values are on, off, yes, no, 1, 0, true, and false.

MIME

Defines MIME types.

The most common way that the server determines the MIME type of a requested resource is by invoking the `type-by-extension` directive in the `ObjectType` section of the `obj.conf` file. The `type-by-extension` function does not work if no `mime` element has been defined in the "SERVER" on page 30 element.

Attributes

The following table describes attributes for the `MIME` element.

TABLE 2–9 MIME Attributes

Attribute	Default	Description
`id`	none	Internal name for the MIME types listing. Used in a "VS" on page 39 element to define the MIME types used by the virtual server. The MIME types name cannot begin with a number.
`file`	none	The name of a MIME types file. For more information, see Chapter 5, MIME Types.

ACLFILE

References one or more ACL files

Subelements

The following table describes subelements for the `ACLFILE` element.

TABLE 2–10 ACLFILE Subelements

Element	Required	Description
"DESCRIPTION" on page 32	Zero or one	Contains a text description of the `ACLFILE` element

Attributes

The following table describes attributes for the `ACLFILE` element.

TABLE 2–11 ACLFILE Attributes

Attribute	Default	Description
`id`	none	Internal name for the ACL file listing. Used in a "VS" on page 39 element to define the ACL file used by the virtual server. An ACL file listing name cannot begin with a number.
`file`	none	A space-separated list of ACL files. Each ACL file must have a unique name. For information about the format of an ACL file, see the *Sun Java System Web Server 6.1 SP9 Administrator's Guide*. The name of the default ACL file is `generated.https-`*server_id*`.acl`, and the file resides in the *server_root*`/`*server_id*`/httpacl` directory. To use this file, you must reference it in `server.xml`.

VSCLASS

Defines a virtual server class.

Subelements

The following table describes subelements for the `VSCLASS` element.

TABLE 2–12 VSCLASS Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of the `VSCLASS`.
"VARS" on page 32	zero or one	Specifies a property of the `VSCLASS`.
"PROPERTY" on page 31	zero or more	Specifies a property of the `VSCLASS`.
"VS" on page 39	zero or more	Defines a virtual server.
"QOSPARAMS" on page 41	zero or one	Defines quality of service parameters.

Attributes

The following table describes attributes for the `VSCLASS` element.

TABLE 2–13 VSCLASS Attributes

Attribute	Default	Description
`id`	none	Virtual server class ID. This is a unique ID that allows lookup of a specific virtual server class. A virtual server class ID cannot begin with a number.
`objectfile`	`obj.conf`	The `obj.conf` file for this class of virtual servers. Cannot be overridden in a VS element.
`rootobject`	default	(Optional) Tells the server which object loaded from an `obj.conf` file is the default. The default object is expected to have all the name translation (`NameTrans`) directives for the virtual server, any server behavior that is configured in the default object affects the entire server. If you specify an object that doesn't exist, the server does not report an error until a client tries to retrieve a document. The Server Manager assumes the default to be the object named `default`. Do not deviate from this convention if you use (or plan to use) the Server Manager.
`acceptlanguage`	`false`	(Optional) If `true`, the server parses the `Accept-Language` header and sends an appropriate language version based on which language the client can accept. You should set this value to `on` only if the server supports multiple languages. Can be overridden in a VS element. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, and `false`.

VS

Defines a virtual server. A virtual server, also called a virtual host, is a virtual web server that serves content targeted for a specific URL. Multiple virtual servers may serve content using the same or different host names, port numbers, or IP addresses. The HTTP service can direct incoming web requests to different virtual servers based on the URL.

Subelements

The following table describes subelements for the `VS` element.

TABLE 2–14 VS Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.
"VARS" on page 32	zero or one	Specifies a property or a variable of the VS.
"PROPERTY" on page 31	zero or more	Specifies a property or a variable of the VS.
"QOSPARAMS" on page 41	zero or one	Defines quality of service parameters.
"USERDB" on page 42	zero or more	Defines the user database for the virtual server.
"DAV" on page 43	zero or one	Defines the WebDAV configuration for the virtual server.
"SEARCH" on page 47	zero or one	Defines the search configuration for the virtual server.
"WEBAPP" on page 49	zero or more	Specifies a web application.

Attributes

The following table describes attributes for the VS element.

TABLE 2–15 VS Attributes

Attribute	Default	Description
id	none	Virtual server ID. This is a unique ID that allows lookup of a specific virtual server. Can also be referred to as the variable $id in an obj.conf file. A virtual server ID cannot begin with a number.
connections	none	(optional) A space-separated list of LS ids that specify the connection(s) the virtual server uses. Required only for a VS that is not the defaultvs of a listen socket.
urlhosts	none	A space-separated list of values allowed in the Host request header to select the current virtual server. Each VS that is configured to the same listen socket must have a unique urlhosts value for that group.
objectfile	objectfile of the enclosing VSCLASS	(optional) The file name of the obj.conf file for this virtual server.

TABLE 2–15 VS Attributes *(Continued)*

Attribute	Default	Description
`rootobject`	`default`	(optional) Tells the server which object loaded from an `obj.conf` file is the default. Tells the server which object loaded from an `obj.conf` file is the default. The default object is expected to have all the name translation (NameTrans) directives for the virtual server; any server behavior that is configured in the default object affects the entire server. If you specify an object that doesn't exist, the server doesn't report an error until a client tries to retrieve a document.
`mime`	none	The `id` of the "MIME" on page 36 element used by the virtual server.
`aclids`	none	(optional) One or more `id` attributes of "ACLFILE" on page 37 elements, separated by commas. Specifies the ACL file(s) used by the virtual server.
`errorlog`	none	(optional) Specifies a log file for virtual-server-specific error messages. See the "LOG" on page 65 description for details about logs.
`acceptlanguage`	`off`	(optional) If true, the server parses the Accept-Language header and sends an appropriate language version based on which language the client can accept. You should set this value to `on` only if the server supports multiple languages. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`state`	`on`	(optional) Determines whether a virtual-server is active (`on`) or inactive (`off`, `disabled`). The default is `on` (active). When inactive, a virtual server does not service requests. If a virtual server is disabled, only the global server administrator can turn it on.

QOSPARAMS

Defines quality of service parameters of an "SERVER" on page 30, "VSCLASS" on page 38, or "VS" on page 39 element.

Subelements

none

Attributes

The following table describes attributes for the QOSPARAMS element.

TABLE 2–16 QOSPARAMS Attributes

Attribute	Default	Description
maxbps	none	(required if enforcebandwidth is yes) The maximum bandwidth limit for the server, vsclass, or vs in bytes per second.
enforcebandwidth	false	(optional) Specifies whether the bandwidth limit should be enforced or not. Allowed values are yes, no, true, false, on, off, 1, 0.
maxconn	none	(required if enforceconnections is yes) The maximum number of concurrent connections for the "SERVER" on page 30, "VSCLASS" on page 38, or "VS" on page 39.
enforceconnections	false	(optional) Specifies whether the connection limit should be enforced or not. Allowed values are yes, no, true, false, on, off, 1, 0.

USERDB

Defines the user database used by the "VS" on page 39 element.

Subelements

The following table describes subelements for the USERDB element.

TABLE 2–17 USERDB Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.

Attributes

The following table describes attributes for the USERDB element.

TABLE 2–18 USERDB Attributes

Attribute	Default	Description
`id`	none	The user database name in the virtual server's ACL file. A user database name cannot begin with a number.
`database`	none	The user database name in the `dbswitch.conf` file.
`basedn`	none	(optional) Overrides the base DN lookup in the `dbswitch.conf` file. However, the `basedn` value is still relative to the base DN value from the `dbswitch.conf` entry.
`certmaps`	none	(optional) Specifies which certificate mapped to LDAP entry mappings (defined in `certmap.conf`) to use. If not present, all mappings are used. All lookups based on mappings in `certmap.conf` are relative to the final base DN of the "VS" on page 39.

WebDAV Elements

The WebDAV elements are as follows:

- "DAV" on page 43
- "DAVCOLLECTION" on page 45

DAV

Defines the WebDAV (Web-based Distributed Authoring and Versioning) configuration for the "VS" on page 39 element.

Subelements

The following table describes subelements for the `DAV` element.

TABLE 2–19 DAV Subelements

Element	Required	Description
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.

TABLE 2–19 DAV Subelements *(Continued)*

Element	Required	Description
"DAVCOLLECTION" on page 45	zero or more	Collections for which DAV is enabled.

Attributes

The following table describes attributes for the `DAV` element.

TABLE 2–20 DAV Attributes

Attribute	Default	Description
`lockdb`	*server-instance*`/lock-db/vs`	(optional) Specifies the directory where the locking database will be maintained.
`lockdbupdateinterval`	0	(optional) specifies the frequency with which the memory representation of the lock database should be synced up to the disk copy of the lock database. The interval is specified in seconds. A value of zero disables the memory representation of the lock database.
`minlocktimeout`	none	(optional) Minimum lifetime of a lock in seconds, `-1` implies never expires. A value of `0` sets `minlocktimeout` to infinity.
`propdbupdateinterval`	0	(optional) specifies the frequency with which the memory representation of the property database should be synced up to the disk copy of the database. The interval is specified in seconds. A value of zero disables the memory representation of the property database.
`maxpropdbsize`	8192	(optional) specifies an upper limit on the total size of the memory representation of the property databases in the collection. When this size is reached, any additional databases accessed in this collection will not have a memory representation.
`maxxmlrequestbodysize`	8192	(optional) Maximum size of the XML request body. Needed to prevent potential Denial of Service (DOS) attacks.

TABLE 2–20 DAV Attributes *(Continued)*

Attribute	Default	Description
`maxpropdepth`	`1`	(optional) The depth of the PROPFIND request. If the request is to a collection, then the depth of the subdirectories included in the response is specified by this attribute. Legal values are `0`, `1`, and `infinity`.
`enabled`	`true`	(optional) Specifies if DAV functionality is enabled for a virtual server. Legal values are `yes`, `no`, `true`, `false`, `on`, `off`, `1`, `0`.

DAVCOLLECTION

Defines a DAV-enabled collection of documents rooted at a URI; the source of the documents are accessed via a separate URI space.

The `DAVCOLLECTION` element defines WebDAV functionality for a URI space. The attributes specified on a collection override any virtual server attribute values.

Subelements

The following table describes subelements for the `DAVCOLLECTION` element.

TABLE 2–21 DAVCOLLECTION Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.

Attributes

The following table describes attributes for the `DAVCOLLECTION` element.

TABLE 2–22 DAVCOLLECTION Attributes

Attribute	Default	Description
`uri`	none	(required) Specifies the URI by which the output content is accessed.
`sourceuri`	none	(optional) Specifies the URI by which the source content of the documents are accessed.
`lockdb`	`lockdb` value specified in the "DAV" on page 43 element	(optional) Specifies the directory where the locking database will be maintained.

TABLE 2–22 DAVCOLLECTION Attributes *(Continued)*

Attribute	Default	Description
`lockdbupdateinterval`	The value specified in the "DAV" on page 43 element.	(optional) specifies the frequency with which the memory representation of the lock database should be synced up to the disk copy of the lock database. The interval is specified in seconds. A value of zero disables the memory representation of the lock database.
`minlocktimeout`	`minlocktimeout` attribute value specified in the "DAV" on page 43 element	(optional) Minimum lifetime of a lock in seconds, `-1` implies never expires, `0` turns locking off.
`propdbupdateinterval`	The value specified in the "DAV" on page 43 element.	(optional) specifies the frequency with which the memory representation of the property database should be synced up to the disk copy of the database. The interval is specified in seconds. A value of zero disables the memory representation of the property database.
`maxpropdbsize`	The value specified in the "DAV" on page 43 element.	(optional) specifies an upper limit on the total size of the memory representation of the property databases in the collection. When this size is reached, any additional databases accessed in this collection will not have a memory representation.
`maxxmlrequestbodysize`	The value specified in the "DAV" on page 43 element.	(optional) Maximum size of the XML request body. Needed to prevent potential Denial of Service (DOS) attacks.
`maxpropdepth`	The value specified in the "DAV" on page 43 element.	(optional) The maximum depth permitted for a DAV PROPFIND request. Allowed values are `0`, `1`, and `infinity`.
`enabled`	`true`	(optional) Specifies if DAV functionality is enabled for this collection.

Search Elements

Search elements are as follows:

- "SEARCH" on page 47
- "SEARCHCOLLECTION" on page 47
- "DISPLAYNAME" on page 48

SEARCH

Defines search related configuration parameters for a given "VS" on page 39.

Subelements

The following table describes subelements for the `SEARCH` element.

TABLE 2–23 SEARCH Subelements

Element	Required	Description
"WEBAPP" on page 49	zero or one	The default search web application for this virtual server
"SEARCHCOLLECTION" on page 47	zero or more	Specifies a searchable index of documents called a collection.
"PROPERTY" on page 31	zero or more	Specifies name-value pairs to pass extra configuration information to the search engine.

Attributes

The following table describes attributes for the `SEARCH` element.

TABLE 2–24 SEARCH Attributes

Attribute	Default	Description
`maxhits`	none	The maximum number of results that will be retrieved by the search engine in a single search.

SEARCHCOLLECTION

Specifies a searchable index of documents called a search collection.

Subelements

The following table describes subelements for the `SEARCHCOLLECTION` element.

TABLE 2–25 SEARCHCOLLECTION Subelements

Element	Required	Description
"DISPLAYNAME" on page 48	zero or one	Optional display name that can be used while displaying searchable collections to the end user.

TABLE 2–25 SEARCHCOLLECTION Subelements *(Continued)*

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of the collection.
"PROPERTY" on page 31	zero or more	Contains name-value pairs to pass extra configuration information to the search engine.

Attributes

The following table describes attributes for the `SEARCHCOLLECTION` element.

TABLE 2–26 SEARCHCOLLECTION Attributes

Attribute	Default	Description
`name`	none	Specifies unique identifier for this collection. Should be a legal XML ID type.
`path`	none	Specifies a file system location for storing search collection meta data.
`uri`	none	Specifies a URI for the indexable collection of documents.
`docroot`	none	Specifies a file system path for the collection of documents.
`enabled`	`true`	Specifies whether a collection can be searched. Legal values are `yes`, `no`, `true`, `false`, `on`, `off`, `1`, and `0`.

DISPLAYNAME

Specifies a human-readable name for the collection to be used while displaying the collection to the end user. Example:

```
<DISPLAYNAME>  Omega Manual </DISPLAYNAME>
```

Subelements

none

Attributes

none

Web Application Elements

The Web application elements are as follows:

WEBAPP

Defines a Java web application rooted at a given URI within a "VS" on page 39.

Subelements

The following table describes subelements for the `WEBAPP` element.

TABLE 2–27 webapp Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.

Attributes

The following table describes attributes for the `WEBAPP` element.

TABLE 2–28 `WEBAPP` attributes

Attribute	Default	Description
`uri`	empty string	This is the context path at which the web application is installed (Section 5.4 of the Servlet 2.3 specification). If this attribute is "/" then this web application is designated to be the default web application for the virtual server. The default web application for a virtual server responds to all requests that cannot be resolved to other web applications deployed to the virtual server. Every virtual server has a default web application.
`path`	none	A fully qualified or relative path to the directory in which the contents of the `.war` file have been extracted.

TABLE 2–28 WEBAPP attributes *(Continued)*

Attribute	Default	Description
`enabled`	`true`	This attribute can be used to temporarily disable the web application from servicing requests without removing the contents of the web application (on disk).Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.

Java Configuration Elements

The Java configuration elements are as follows:

JAVA

Defines configurable properties for the integrated Java Virtual Machine (JVM), and for Java-based security and resources.

Subelements

The following table describes subelements for the `JAVA` element.

TABLE 2–29 JAVA Subelements

Element	Required	Description
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.
"JVMOPTIONS" on page 52	zero or more	Contains JVM command line options.
"PROFILER" on page 53	zero or one	Configures a profiler for use with the server.
"SECURITY" on page 54	zero or one	Defines parameters and configuration information needed by the security service.
"RESOURCES" on page 56	zero or one	Specifies configured resources.

Attributes

The following table describes attributes for the `JAVA` element.

TABLE 2–30 JAVA Attributes

Attribute	Default	Description
`javahome`	*<install-root>*`/bin/https/jdk` For SVR 4 package-based installation for Solaris: `/usr/java`	The path to the directory where the JDK is installed.
`debug`	`false`	(optional) If `true`, the server starts up in debug mode ready for attachment with a JPDA-based (Java Platform Debugger Architecture-based) debugger. Legal values are `on`, `off`, `yes`, `no`, `true`, `false`, `1`, `0`.
`debugoptions`	`-Xdebug -Xrunjdwp:transport=dt_socket, server=y,suspend=n`	(optional) Specifies JPDA options. A list of debugging options that you can include is available at: `http://java.sun.com/products/jpda/doc/conninv.html#Invocation`
`classpathprefix`	none	(optional) Specifies a prefix for the system classpath. You should only prefix the system classpath if you wish to override system classes, such as the XML parser classes. Use this attribute with caution.
`serverclasspath`	none	(optional) Specifies the classpath for the environment from which the server was started. This classpath can be accessed using `System.getProperty("java.class.path")`.
`classpathsuffix`	none	(optional) Specifies a suffix for the system classpath.
`nativelibrarypathprefix`	none	(optional) Specifies a prefix for the native library path. The native library path is the automatically constructed concatenation of the path to the server's native shared libraries, the standard JRE (Java Runtime Environment) native library path, the shell environment setting (LD_LIBRARY_PATH on UNIX), and any path specified in the "PROFILER" on page 53 element. Since this is synthesized, it does not appear explicitly in the server configuration.
`nativelibrarypathsuffix`	none	(optional) Specifies a suffix for the native library path.

TABLE 2–30 JAVA Attributes *(Continued)*

Attribute	Default	Description
`envclasspathignored`	`true`	(optional) If `false`, the CLASSPATH environment variable is read and appended to the server classpath. The CLASSPATH environment variable is added after the `classpath-suffix`, at the very end. For a development environment, this value should be set to false. For a production environment, this value should be set to `true` to prevent environment variable side effects. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`bytecodepreprocessors`	none	(optional) A comma-separated list of class names, each of which must implement the `com.sun.appserv.BytecodePreprocessor` interface. Each of the specified preprocessor classes is called in the order specified.
`dynamicreloadinterval`	`2`	Specifies the interval, in seconds, after which a deployed application is reloaded.
`loglevel`	Value of `level` attribute of "LOG" on page 65 element	(optional) Controls the type of messages logged by this element to the errors log. For details, see the description of the `level` attribute of the "LOG" on page 65 element.

JVMOPTIONS

Defines configurable system-wide Java VM properties., for example:

```
<JVMOPTIONS>-Xdebug -Xmx128m</JVMOPTIONS>
```

In addition, web server looks for a system property, `-Dcom.sun.webserv.startupclasses`, whose value is a comma-separated list of fully qualified Java classes that server loads into the Virtual Machine upon startup. Example:

```
<JVMOPTIONS>
-Dcom.sun.webserv.startupclasses=com.sample.MyInitializer,com.jdo
.PersistenceManagerFactory
</JVMOPTIONS>
```

For information about the available options, see
`http://java.sun.com/docs/hotspot/VMOptions.html`.

Note – (On Windows only). Logging out of the machine where the web server is installed might abort the web server process. To avoid this problem, perform these steps:

1. Add the following line to the `server.xml` file of the web server instance.

   ```
   <JVMOPTIONS>-Xrs</JVMOPTIONS>
   ```

2. Restart the instance.

Subelements

none

Attributes

none

PROFILER

Configures a profiler for use with the server.

Subelements

The following table describes subelements for the `PROFILER` element.

TABLE 2–31 PROFILER Subelements

Element	Required	Description
"PROPERTY" on page 31	zero or more	Specifies a property.
"JVMOPTIONS" on page 52	zero or more	Contains profiler-specific JVM command line options.

Attributes

The following table describes attributes for the `PROFILER` element.

TABLE 2–32 PROFILER Attributes

Attribute	Default	Description
`classpath`	none	(optional) Specifies the classpath for the profiler.
`nativelibrarypath`	none	(optional) Specifies the native library path for the profiler.

TABLE 2–32 PROFILER Attributes *(Continued)*

Attribute	Default	Description
`enabled`	`true`	(optional) Determines whether the profiler is enabled. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.

SECURITY

Defines parameters and configuration information needed by the security service.

Subelements

The following table describes subelements for the `SECURITY` element.

TABLE 2–33 `SECURITY` Subelements

Element	Required	Description
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.
"AUTHREALM" on page 55	one or more	Defines a realm for authentication.

Attributes

The following table describes attributes for the `SECURITY` element.

TABLE 2–34 `SECURITY` Attributes

Attribute	Default	Description
`defaultrealm`	`file`	(optional) Specifies the default authentication realm (an "AUTHREALM" on page 55 name attribute) for this server instance. The default realm will be used to process authentication events for any web applications which do not otherwise specify which realm to use.
`anonymousrole`	`ANYONE`	(optional) Used as the name for default, or anonymous, role. The anonymous role is always assigned to all principals.

TABLE 2–34 SECURITY Attributes *(Continued)*

Attribute	Default	Description
audit	false	(optional) If true, additional access logging is performed to provide audit information. Legal values are on, off, yes, no, 1, 0, true, false. Audit information consists of: ■ Authentication success and failure events ■ Servlet access grants and denials
loglevel	Value of level attribute of "LOG" on page 65 element	(optional) Controls the type of messages logged by this element to the errors log. For details, see the description of the level attribute of the "LOG" on page 65 element.

AUTHREALM

Defines a realm for authentication.

Authentication realms require provider-specific properties, which vary depending on the needs of a particular implementation.

Here is an example of the default file realm:

```
<authrealm name="file"

classname="com.iplanet.ias.security.auth.realm.file.FileRealm">

<property name="file" value="instance_dir/config/keyfile">

<property name="jaas-context" value="fileRealm">

</authrealm>
```

Which properties an AUTHREALM element uses depends on the value of the AUTHREALM element's name attribute. The file realm uses file and jaas-context properties. Other realms use different properties.

Subelements

The following table describes subelements for the AUTHREALM element.

TABLE 2–35 AUTHREALM Subelements

Element	Required	Description
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the AUTHREALM element.

TABLE 2–36 AUTHREALM Attributes

Attribute	Default	Description
name	none	Specifies the name of this realm.
classname	none	Specifies the Java class that implements this realm.

Properties

The standard realms provided have both required and optional properties. A custom realm may have different properties. For details about the properties and configuration characteristics of the AUTHREALM realms, refer to the chapter "Securing Web Applications" in the Sun Java System Web Server 6.1 *Programmer's Guide to Web Applications*.

Resource Elements

Resource elements are as follows:

RESOURCES

Contains configured resources, such as database connections.

Subelements

The following table describes subelements for the RESOURCES element.

TABLE 2–37 RESOURCES Subelements

Element	Required	Description
"CUSTOMRESOURCE" on page 57	zero or more	Defines a custom resource.
"EXTERNALJNDIRESOURCE" on page 58	zero or more	Defines a resource that resides in an external JNDI (Java Naming and Directory Interface) repository.
"JDBCRESOURCE" on page 59	zero or more	Defines a JDBC (Java Database Connectivity) resource.
"JDBCCONNECTIONPOOL" on page 59	zero or more	Defines the properties that are required for creating a JDBC connection pool.
"MAILRESOURCE" on page 64	zero or more	Defines the properties that are required for creating a mail resource.

Attributes

none

CUSTOMRESOURCE

Defines a custom resource, which specifies a custom server-wide resource object factory. Such object factories implement the `javax.naming.spi.ObjectFactory` interface.

Subelements

The following table describes subelements for the `CUSTOMRESOURCE` element.

TABLE 2–38 `CUSTOMRESOURCE` Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the `CUSTOMRESOURCE` element.

TABLE 2–39 CUSTOMRESOURCE Attributes

Attribute	Default	Description
jndiname	none	Specifies the JNDI name for the resource.
restype	none	Specifies the fully qualified type of the resource.
factoryclass	none	Specifies the fully qualified name of the user-written factory class, which implements javax.naming.spi.ObjectFactory.
enabled	true	(optional) Determines whether this resource is enabled at runtime. Legal values are on, off, yes, no, 1, 0, true, false.

EXTERNALJNDIRESOURCE

Defines a resource that resides in an external JNDI repository. For example, a generic Java object could be stored in an LDAP server. An external JNDI factory must implement the javax.naming.spi.InitialContextFactory interface.

Subelements

The following table describes subelements for the EXTERNALJNDIRESOURCE element.

TABLE 2–40 EXTERNALJNDIRESOURCE Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the EXTERNALJNDIRESOURCE element.

TABLE 2–41 EXTERNALJNDIRESOURCE Attributes

Attribute	Default	Description
jndiname	none	Specifies the JNDI name for the resource.
jndilookupname	none	Specifies the JNDI lookup name for the resource.
restype	none	Specifies the fully qualified type of the resource.

TABLE 2–41 EXTERNALJNDIRESOURCE Attributes *(Continued)*

Attribute	Default	Description
factoryclass	none	Specifies the fully qualified name of the factory class, which implements javax.naming.spi.InitialContextFactory.
enabled	true	(optional) Determines whether this resource is enabled at runtime. Legal values are on, off, yes, no, 1, 0, true, false.

JDBCRESOURCE

Defines a JDBC (javax.sql.DataSource) resource.

Subelements

The following table describes subelements for the JDBCRESOURCE element.

TABLE 2–42 JDBCRESOURCE Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the JDBCRESOURCE element.

TABLE 2–43 JDBCRESOURCE Attributes

Attribute	Default	Description
jndiname	none	Specifies the JNDI name for the resource.
poolname	none	Specifies the name of the associated JDBC connection pool, defined in a "JDBCCONNECTIONPOOL" on page 59 element.
enabled	true	(optional) Determines whether this resource is enabled at runtime. Legal values are on, off, yes, no, 1, 0, true, false.

JDBCCONNECTIONPOOL

Defines the properties that are required for creating a JDBC connection pool.

Note – The `restype` attribute of the `JDBCCONNECTIONPOOL` element is reserved and ignored in Sun Java System Web Server 6.1. Any value set for this attribute is ignored by the server.

Subelements

The following table describes subelements for the `JDBCCONNECTIONPOOL` element.

TABLE 2–44 `JDBCCONNECTIONPOOL` Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.
"CONNECTIONPROPERTY" on page 63	zero or more	Specifies the connection properties for the connection pool.

Attributes

The following table describes attributes for the `JDBCCONNECTIONPOOL` element.

TABLE 2–45 `JDBCCONNECTIONPOOL` Attributes

Attribute	Default	Description
`name`	none	Specifies the name of the connection pool. A "JDBCRESOURCE" on page 59 element's `poolname` attribute refers to this `name`.
`datasourceclassname`	none	Specifies the class name of the associated vendor-supplied data source. This class must implement `java.sql.DataSource` or `java.sql.XADataSource` or both.
`steadypoolsize`	`8`	(optional) Specifies the initial and minimum number of connections maintained in the pool.
`maxpoolsize`	`32`	(optional) Specifies the maximum number of connections that can be created to satisfy client requests.
`maxwaittime`	`60000`	(optional) Specifies the amount of time, in milliseconds, that the caller is willing to wait for a connection. If `0`, the caller is blocked indefinitely until a resource is available or an error occurs.

TABLE 2–45 JDBCCONNECTIONPOOL Attributes *(Continued)*

Attribute	Default	Description
poolresizequantity	2	(optional) Specifies the number of connections to be destroyed if the existing number of connections is above the steady-pool-size (subject to the max-pool-size limit). This is enforced periodically at the idle-time-out-in-seconds interval. An idle connection is one that has not been used for a period of idle-time-out-in-seconds.
idletimeout	300	(optional) Specifies the maximum time that a connection can remain idle in the pool. After this amount of time, the pool can close this connection.
transactionisolationlevel	default JDBC driver isolation level	(optional) Specifies the transaction isolation level on the pooled database connections. Allowed values are read-uncommitted, read-committed, repeatable-read, or serializable. Applications that change the isolation level on a pooled connection programmatically risk polluting the pool, which can lead to errors. See isolationlevelguaranteed for more details.
isolationlevelguaranteed	true	(optional) Applicable only when transactionisolationlevel is explicitly set. If true, every connection obtained from the pool is guaranteed to have the desired isolation level. This may impact performance on some JDBC drivers. You can set this attribute to false if you are certain that the hosted applications do not return connections with altered isolation levels.
connectionvalidationrequired	false	(optional) Specifies whether connections must be validated before being given to the application. If a resource's validation fails, it is destroyed, and a new resource is created and returned. Legal values are on, off, yes, no, 1, 0, true, false.

TABLE 2–45 JDBCCONNECTIONPOOL Attributes *(Continued)*

Attribute	Default	Description
connectionvalidationmethod	auto-commit	(optional) Legal values are as follows: ■ auto-commit (default), which uses Connection.setAutoCommit(Connection.getAutoCommit()) ■ meta-data, which uses Connection.getMetaData() ■ table, which performs a query on a table specified in the validation-table-name attribute
validationtablename	none	(optional) Specifies the table name to be used to perform a query to validate a connection. This parameter is mandatory if and only if connectionvalidationtype is set to table.
failallconnections	false	(optional) If true, closes all connections in the pool if a single validation check fails. This parameter is mandatory if and only if isconnectionvalidationrequired is set to true. Legal values are on, off, yes, no, 1, 0, true, false.

Properties

Most JDBC 2.0 drivers allow use of standard property lists to specify the user, password, and other resource configuration information. Although properties are optional with respect to Sun Java System Web Server, some properties may be necessary for most databases. For details, see Section 5.3 of the JDBC 2.0 Standard Extension API.

When properties are specified, they are passed to the vendor's data source class (specified by the datasourceclassname attribute) using set*Name*(*value*) methods.

The following table describes some common properties for the JDBCCONNECTIONPOOL element. The left column lists the property name, and the right column describes what the property does.

TABLE 2–46 JDBCCONNECTIONPOOL Properties

Property	Description
user	Specifies the user name for this connection pool.
password	Specifies the password for this connection pool.
databaseName	Specifies the database for this connection pool.

TABLE 2–46 JDBCCONNECTIONPOOL Properties *(Continued)*

Property	Description
`serverName`	Specifies the database server for this connection pool.
`port`	Specifies the port on which the database server listens for requests.
`networkProtocol`	Specifies the communication protocol.
`roleName`	Specifies the initial SQL role name.
`datasourceName`	Specifies an underlying `XADataSource`, or a `ConnectionPoolDataSource` if connection pooling is done.
`description`	Specifies a text description.
`url`	Specifies the URL for this connection pool. Although this is not a standard property, it is commonly used.

CONNECTIONPROPERTY

Specifies the connection properties for a JDBC connection pool.

Subelements

The following table describes subelements for the `CONNECTIONPROPERTY` element.

TABLE 2–47 `CONNECTIONPROPERTY` Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.

Attributes

The following table describes attributes for the `CONNECTIONPROPERTY` element.

TABLE 2–48 `CONNECTIONPROPERTY` Attributes

Attribute	Default	Description
`name`	none	Specifies a name for the connection property.
`value`	`none`	Specifies a value for the connection property.
`invocationfrequency`	`at-creation`	(optional) Specifies the frequency with which the connection property is invoked. Legal values are `at-creation` and `every-lease`.

MAILRESOURCE

Defines a JavaMail (`javax.mail.Session`) resource.

Subelements

The following table describes subelements for the `MAILRESOURCE` element.

TABLE 2–49 `MAILRESOURCE` Subelements

Element	Required	Description
"DESCRIPTION" on page 32	zero or one	Contains a text description of this element.

Attributes

The following table describes attributes for the `MAILRESOURCE` element.

TABLE 2–50 `MAILRESOURCE` Attributes

Attribute	Default	Description
`jndiname`	none	Specifies the JNDI name for the resource.
`storeprotocol`	`imap`	(optional) Specifies the storage protocol service, which connects to a mail server, retrieves messages, and saves messages in folder(s). Example values are `imap` and `pop3`.
`storeprotocolclass`	`com.sun.mail.imap.IMAPStore`	(optional) Specifies the service provider implementation class for storage. You can find this class at: ■ `http://java.sun.com/products/javamail/` ■ `http://java.sun.com/products/javabeans/glasgow/`
`transportprotocol`	`smtp`	(optional) Specifies the transport protocol service, which sends messages.
`transportprotocolclass`	`com.sun.mail.smtp.SMTPTransport`	(optional) Specifies the service provider implementation class for transport. You can find this class at: ■ `http://java.sun.com/products/javamail/` ■ `http://java.sun.com/products/javabeans/glasgow/`
`host`	none	The mail server host name

TABLE 2–50 MAILRESOURCE Attributes *(Continued)*

Attribute	Default	Description
user	none	The mail server user name
from	none	The e-mail address the mail server uses to indicate the message sender.
enabled	true	(optional) Determines whether this resource is enabled at runtime. Legal values are on, off, yes, no, 1, 0, true, false.

LOG

Configures the system logging service, which includes the following log files:

- The errors log file stores messages from the default virtual server. Messages from other configured virtual servers also go here, unless the logfile attribute is explicitly specified in the "VSCLASS" on page 38 or "VS" on page 39 element. The default name is errors.
- The access log file stores HTTP access messages from the default virtual server. The default name is access.log. To configure the access log, you use server application functions in the magnus.conf and obj.conf files.
- A virtual server log file stores messages from a "VSCLASS" on page 38 or "VS" on page 39 element that has an explicitly specified log-file attribute.

Subelements

The following table describes subelements for the LOG element.

TABLE 2–51 LOG Subelements

Element	Required	Description
"PROPERTY" on page 31	zero or more	Specifies a property or a variable.

Attributes

The following table describes attributes for the LOG element.

TABLE 2–52 LOG Attributes

Attribute	Default	Description
`file`	`errors`	Specifies the file that stores messages from the default virtual server. Messages from other configured virtual servers also go here, unless the `errorlog` attribute is explicitly specified in the "VS" on page 39 element.
`loglevel`	`info`	Controls the default type of messages logged by other elements to the error log. Allowed values are as follows, from highest to lowest: `finest`, `finer`, `fine`, `info`, `warning`, `failure`, `config`, `security`, and `catastrophe`.
`logvsid`	`false`	(optional) If `true`, virtual server IDs are displayed in the virtual server logs. This is useful if multiple "VS" on page 39 elements share the same log file. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`logstdout`	`true`	(optional) If `true`, redirects `stdout` output to the errors log. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`logstderr`	`true`	(optional) If `true`, redirects `stderr` output to the errors log. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`logtoconsole`	`true`	(optional, UNIX only) If `true`, redirects log messages to the console.
`createconsole`	`false`	(optional, Windows only) If `true`, creates a Windows console. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.
`usesyslog`	`false`	(optional) If `true`, uses the UNIX `syslog` service or Windows Event Logging to produce and manage logs. Legal values are `on`, `off`, `yes`, `no`, `1`, `0`, `true`, `false`.

User Database Selection

A "USERDB" on page 42 object selects a user database for the parent virtual server. This selection occurs in the following manner:

- The "USERDB" on page 42 element's id attribute maps to an ACL file's database attribute.
- The "USERDB" on page 42 element's database attribute maps to a `dbswitch.conf` entry.

This layer between the ACL file and the `dbswitch.conf` file gives the server administrator full control over the databases to which virtual server administrators and users have access.

The `dbswitch.conf` file establishes the root of the search tree for LDAP databases as follows:

- The base DN in the LDAP URL in `dbswitch.conf` defines a root object for all further DN specifications. So, for most new installations, it can be empty, because the final base DN is determined in other ways -- either through a dc tree lookup or an explicit `basedn` value in the "USERDB" on page 42 tag.
- A new `dbswitch.conf` attribute for LDAP databases, `dcsuffix`, defines the root of the dc tree. This root is relative to the base DN in the LDAP URL. If the database is *schema compliant* you can use `dcsuffix` . Requirements for schema compliance are listed in "Sun Java System LDAP Schema" on page 67.

A user database is selected for a virtual server as follows:

- If a "VS" on page 39 has no "USERDB" on page 42 subelement, user- or group-based ACLs fail.
- When no `database` attribute is present in a virtual server's ACL definition, the "VS" on page 39 must have a "USERDB" on page 42 subelement with an `id` attribute of `default`. The `database` attribute of the "USERDB" on page 42 then points to a database in `dbswitch.conf`. If no `database` attribute is present, `default` is used.
- If an LDAP database is schema compliant, the base DN of the access is computed using a dc tree lookup of the VS element's hosts attribute that matches the client-supplied host header. If no hosts attribute matches, the `servername` attribute of the parent "SERVER" on page 30 is used. The dc tree lookup is based at the `dcsuffix` DN. The result must contain an `inetDomainBaseDN` attribute that contains the base DN. This base DN is taken as is and is not relative to any of the base DN values.
- If the `basedn` attribute of the "USERDB" on page 42 element is not present and the database is not schema compliant, the access requests are relative to the base DN in the `dbswitch.conf` entry, as in previous Sun Java System Web Server versions.

Sun Java System LDAP Schema

This section describes the Sun Java System LDAP Schema that defines a set of rules for directory data.

You can use the `dcsuffix` attribute in the `dbswitch.conf` file if your LDAP database meets the requirements outlined in this section. For more information about the `dbswitch.conf` file, see "dbswitch.conf" on page 211.

The subtree rooted at an ISP entry (for example, `o=isp`) is called the *convergence tree*. It contains all directory data related to organizations (customers) served by an ISP.

The subtree rooted at `o=internet` is called the *domain component tree,* or *dc tree*. It contains a sparse DNS tree with entries for the customer domains served. These entries are links to the appropriate location in the convergence tree where the data for that domain is located.

The directory tree may be single rooted, which is recommended (for example, `o=root` may have `o=isp` and `o=internet` under it), or have two separate roots, one for the convergence tree and one for the `dc` tree.

Convergence Tree

The top level of the convergence tree must have one organization entry for each customer (or organization), and one for the ISP itself.

Underneath each organization, there must be two `organizationalUnit` entries: `ou=People` and `ou=Groups`. A third, `ou=Devices`, can be present if device data is to be stored for the organization.

Each user entry must have a unique `uid` value within a given organization. The namespace under this subtree can be partitioned into various `ou` entries that aggregate user entries in convenient groups (for example, `ou=eng`, `ou=corp`). User `uid` values must still be unique within the entire `People` subtree.

User entries in the convergence tree are of type `inetOrgPerson`. The `cn`, `sn`, and `uid` attributes must be present. The `uid` attribute must be a valid email name (specifically, it must be a valid local-part as defined in RFC822). The `cn` contains *name initial sn*. The RDN of the user entry the `uid` value. User entries must contain the auxiliary class `inetUser` if they are to be considered enabled for service or valid.

User entries can also contain the auxiliary class `inetSubscriber`, which is used for account management purposes. If an `inetUserStatus` attribute is present in an entry and has a value of `inactive` or `deleted`, the entry is ignored.

Groups are located under the `Groups` subtree and consist of LDAP entries of type `groupOfUniqueNames`.

Domain Component (dc) Tree

The `dc` tree contains hierarchical `domain` entries, each of which is a DNS name component.

Entries that represent the domain name of a customer are overlaid with the LDAP auxiliary class `inetDomain`. For example, the two LDAP entries `dc=customer1,dc=com,o=Internet,o=root` and `dc=customer2,dc=com,o=Internet,o=root` contain the `inetDomain` class, but `dc=com,o=Internet,o=root` does not. The latter is present only to provide structure to the tree.

Entries with an `inetDomain` attribute are called virtual domains. These must have the attribute `inetDomainBaseDN` filled with the DN of the top level organization entry where the data of this domain is stored in the convergence tree. For example, the virtual domain entry in `dc=cust2,dc=com,o=Internet,o=root` contains the attribute `inetDomainBaseDN` with value `o=Cust2,o=isp,o=root`.

If an `inetDomainStatus` attribute has a value of `inactive` or `deleted`, the entry is ignored.

Variables

Some variables are defined in `server.xml` for use in the `obj.conf` file. The following file fragment defines a `docroot` variable:

```
<PROPERTY name="docroot" value="/server/docs/class2/acme" >
```

A `docroot` variable allows different document root directories to be assigned for different virtual servers. The variable is then used in the `obj.conf` file. For example:

```
NameTrans fn=document-root root="$docroot"
```

Using this `docroot` variable allows you to define different document roots for different virtual servers within the same virtual server class.

Format of a Variable

A variable is found in `obj.conf` when the following regular expression matches:

```
\$[A-Za-z][A-Za-z0-9_]*
```

This expression represents a `$` followed by one or more alphanumeric characters. A delimited version ("`${property}`") is not supported. To get a regular `$` character, use `$$` to have variable substitution.

The id Variable

A special variable, `id`, is always available within a "VS" on page 39 element and refers to the value of the `id` attribute. It is predefined and cannot be overridden. The `id` attribute uniquely identifies a virtual server. For example:

```
<PROPERTY name=docroot value="/export/$id" >
```

If the `id` attribute of the parent "VS" on page 39 element is `myserver`, the `docroot` variable is set to the value `/export/myserver`.

Other Important Variables

In a default installation, the following variables are used to configure various aspects of the server's operation. Unlike the `$id` variable, they are not predefined in the server, and they can be overridden.

General Variables

The following table lists general `server.xml` variables. The left column lists variables, and the right column lists descriptions of those variables.

TABLE 2–53 General Variables

Property	Description
`docroot`	The document root of the virtual server. Typically evaluated as the parameter to the `document-root` function in the `obj.conf` file.
accesslog	The access log file for a virtual server.

send-cgi Variables

The following table lists `server.xml` variables used by the `send-cgi` function in the `obj.conf` file. The left column lists variables, and the right column lists descriptions of those variables.

TABLE 2–54 send-cgi Variables

Property	Description
`user`	The value of the `user` CGI parameter.
group	The value of the `group` CGI parameter.
chroot	The value of the `chroot` CGI parameter.
dir	The value of the `dir` CGI parameter.
nice	The value of the `nice` CGI parameter.

For more information about the `send-cgi` function, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

Variable Evaluation

Variables are evaluated when generating specific objectsets for individual virtual servers. Evaluation is recursive: variable values can contain other variables. For example:

```
...
<VSCLASS>
   ...
   <VS ...>
      ...
      <PROPERTY name=docroot value="$docrootbase/nonjava/$id" >
   </VS>
   <VS...>
      ...
      <PROPERTY name=docroot value="$docrootbase/java/$id" >
   </VS>
   ...
   <PROPERTY name=docrootbase value="/export" >
</VSCLASS>
...
```

Variables in subelements override variables in the parent elements. For example, it is possible to set a variable for a class of virtual servers and override it with a definition of the same variable in an individual virtual server.

Sample server.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
   Copyright (c) 2003 Sun Microsystems, Inc.  All rights reserved.
   Use is subject to license terms.
-->
<!DOCTYPE SERVER PUBLIC "-//Sun Microsystems Inc.//DTD Sun Java
System Web Server 6.1//EN"
"file:///home/nb136819/space/servers/s1ws61/bin/https/dtds/sun-web-
server_6_1.dtd">

<SERVER>
    <PROPERTY name="docroot"
value="/home/nb136819/space/servers/s1ws61/docs">
    <PROPERTY name="accesslog"
value="/home/nb136819/space/servers/s1ws61/https-admserv/logs/
access">
    <PROPERTY name="user" value="">
    <PROPERTY name="group" value="">
    <PROPERTY name="chroot" value="">
    <PROPERTY name="dir" value="">
    <PROPERTY name="nice" value="">
<LS id="ls1" port="5555" servername="plaza.india.sun.com"
defaultvs="vs-admin">

<LS id="ls2" port="9999" servername="plaza.india.sun.com"
```

```
defaultvs="useradmin">

<MIME id="mime1" file="mime.types">

<ACLFILE id="acl1"
file="/home/nb136819/space/servers/s1ws61/httpacl/generated.https
-admserv.acl">
<VSCLASS id="vsclass-admin" objectfile="obj.conf">
        <VS id="vs-admin" connections="ls1" mime="mime1"
aclids="acl1" urlhosts="plaza.india.sun.com">

        <PROPERTY name="docroot"
value="/home/nb136819/space/servers/s1ws61/docs">
            <USERDB id="default">
            <WEBAPP uri="/admin-app"
path="/home/nb136819/space/servers/s1ws61/bin/https/webapps/admin-app">
        </VS>
    </VSCLASS>
<VSCLASS id="userclass" objectfile="userclass.obj.conf">
        <VS id="useradmin" connections="ls2" mime="mime1"
aclids="acl1" urlhosts="plaza.india.sun.com">
            <PROPERTY name="docroot"
value="/home/nb136819/space/servers/s1ws61/docs">
            <USERDB id="default">
            <WEBAPP uri="/user-app"
path="/home/nb136819/space/servers/s1ws61/bin/https/webapps/user-app">
        </VS>
    </VSCLASS>
<JAVA javahome="/home/nb136819/space/servers/s1ws61/bin/https/jdk"
serverclasspath="/home/nb136819/space/servers/s1ws61/bin/https/jar/
webserv-rt.jar:${java.home}/lib/tools.jar:/home/nb136819/space/serv
ers/s1ws61/bin/https/jar/webserv-ext.jar:/home/nb136819/space/serve
rs/s1ws61/bin/https/jar/webserv-jstl.jar:/home/nb136819/space/serve
rs/s1ws61/bin/https/jar/webserv-admin.jar:/home/nb136819/space/serv
ers/s1ws61/bin/https/jar/ktsearch.jar" classpathsuffix=""
envclasspathignored="true" nativelibrarypathprefix="" debug="false"
debugoptions="-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=n"
dynamicreloadinterval="-1">

<JVMOPTIONS>-Dorg.xml.sax.parser=org.xml.sax.helpers.XMLReaderAdapter
</JVMOPTIONS>

<JVMOPTIONS>-Dorg.xml.sax.driver=org.apache.crimson.parser.XMLReaderImpl
</JVMOPTIONS>

<JVMOPTIONS>-Djava.security.manager
</JVMOPTIONS>
```

```
<JVMOPTIONS>-Djava.security.policy=/home/nb136819/space/servers/s1ws61
/https-admserv/config/server.policy</JVMOPTIONS>

<JVMOPTIONS>-Djava.security.auth.login.config=/home/nb136819/space
/servers/s1ws61/https-admserv/config/login.conf
</JVMOPTIONS>

<JVMOPTIONS>-Djava.util.logging.manager=com.iplanet.ias.server.logging
.ServerLogManager
</JVMOPTIONS>

<JVMOPTIONS>-Xms128m -Xmx256m
</JVMOPTIONS>

  <SECURITY defaultrealm="file" anonymousrole="ANYONE" audit="false">
            <AUTHREALM name="file"
classname="com.iplanet.ias.security.auth.realm.file.FileRealm">

             <PROPERTY name="file"
value="/home/nb136819/space/servers/s1ws61/https-admserv/config/keyfile">

             <PROPERTY name="jaas-context" value="fileRealm">
            </AUTHREALM>
    </SECURITY>
<RESOURCES>
            </RESOURCES>
    </JAVA>
<LOG
file="/home/nb136819/space/servers/s1ws61/https-admserv/
logs/errors" loglevel="info">

</SERVER>
```

◆ ◆ ◆ CHAPTER 3

Syntax and Use of magnus.conf

When the Sun Java™ System Web Server starts up, it looks in a file called `magnus.conf` in the *server-id*`/config` directory to establish a set of global variable settings that affect the server's behavior and configuration. Sun Java System Web Server executes all the directives defined in `magnus.conf`. The order of the directives is not important.

Note – When you edit the `magnus.conf` file, you must restart the server for the changes to take effect.

This chapter lists the global settings that can be specified in `magnus.conf` in Sun Java System Web Server 6.1.

The categories are:

For an alphabetical list of directives, see Appendix D, Alphabetical List of Server Configuration Elements

For a list of magnus.conf directives deprecated in Sun Java System Web Server 6.1, see "Deprecated Directives" on page 96.

Note – Much of the functionality of the file cache is controlled by a configuration file called `nsfc.conf`. For information about `nsfc.conf`, see "nsfc.conf" on page 215.

Init Functions

The `Init` functions load and initialize server modules and plug-ins, and initialize log files. For more information about these functions, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

Server Information

This sub-section lists the directives in `magnus.conf` that specify information about the server. They are:

ExtraPath

Appends the specified directory name to the `PATH` environment variable. This is used for configuring Java on Windows. There is no default value; you must specify a value.

Syntax

`ExtraPath` *path*

TempDir

Specifies the directory on the local volume that the server uses for its temporary files. On UNIX, this directory must be owned by, and writable by, the user the server runs as. See also the directives "User" on page 77 and "TempDirSecurity" on page 77.

Syntax

`TempDir` *path*

Default

`/tmp` (UNIX)

`TEMP` (environment variable for Windows)

TempDirSecurity

Determines whether the server checks if the TempDir directory is secure. On UNIX, specifying `TempDirSecurity off` allows the server to use `/tmp` as a temporary directory.

Caution – Specifying `TempDirSecurity off` or using `/tmp` as a temporary directory on UNIX is highly discouraged. Using `/tmp` as a temporary directory opens a number of potential security risks.

Syntax

```
TempDirSecurity [on|off]
```

Default

`on`

User

Windows: The `User` directive specifies the user account the server runs with. By using a specific user account (other than LocalSystem), you can restrict or enable system features for the server. For example, you can use a user account that can mount files from another machine.

UNIX: The `User` directive specifies the UNIX user account for the server. If the server is started by the superuser or root user, the server binds to the port you specify and then switches its user ID to the user account specified with the `User` directive. This directive is ignored if the server isn't started as `root`. The user account you specify should have *read* permission to the server's root and subdirectories. The user account should have *write* access to the `logs` directory and *execute* permissions to any CGI programs. The user account should not have write access to the configuration files. This ensures that in the unlikely event that someone compromises the server, they won't be able to change configuration files and gain broader access to your machine. Although you can use the `nobody` user, it isn't recommended.

Syntax

`User` *name*

`name` is the 8-character (or less) login name for the Unix user account.

Default

If there is no `User` directive, the server runs with the user account it was started with.

Examples

```
User http
```

```
User server
```

```
User nobody
```

Language Issues

This section lists the directives in `magnus.conf` related to language issues. The following directive is supported:

- "DefaultLanguage" on page 78

DefaultLanguage

For an international version of the server, this directive specifies the default language for the server. The default language is used for both the client responses and administration. Values are `en` (English), `fr` (French), `de` (German) or `ja` (Japanese).

Default

The default is `en`.

DNS Lookup

This section lists the directives in `magnus.conf` that affect DNS (Domain Name System) lookup. The directives are:

- "AsyncDNS" on page 78
- "DNS" on page 79

AsyncDNS

Specifies whether asynchronous DNS is allowed. This directive is ignored. Even if the value is set to on, the server does not perform asynchronous DNS lookups.

DNS

The `DNS` directive specifies whether the server performs DNS lookups on clients that access the server. When a client connects to your server, the server knows the client's IP address but not its host name (for example, it knows the client as 198.95.251.30, rather than its host name `www.a.com`). The server will resolve the client's IP address into a host name for operations like access control, CGI, JSP, Servlet, error reporting, and access logging.

If your server responds to many requests per day, you might want (or need) to stop host name resolution; doing so can reduce the load on the DNS or NIS (Network Information System) server.

Threads, Processes, and Connections

In Sun Java System Web Server 6.1, acceptor threads on a listen socket accept connections and put them onto a connection queue. Session threads then pick up connections from the queue and service the requests. The session threads post more session threads if required at the end of the request. The policy for adding new threads is based on the connection queue state:

- Each time a new connection is returned, the number of connections waiting in the queue (the backlog of connections) is compared to the number of session threads already created. If it is greater than the number of threads, more threads are scheduled to be added the next time a request completes.
- The previous backlog is tracked, so that if it is seen to be increasing over time, and if the increase is greater than the `ThreadIncrement` value, and the number of session threads minus the backlog is less than the `ThreadIncrement` value, then another `ThreadIncrement` number of threads are scheduled to be added.
- The process of adding new session threads is strictly limited by the `RqThrottle` value.
- To avoid creating too many threads when the backlog increases suddenly (such as the startup of benchmark loads), the decision whether more threads are needed is made only once every 16 or 32 times a connection is made based on how many session threads already exist.

  This subsection lists the directives in `magnus.conf` that affect the number and timeout of threads, processes, and connections. They are:

Also see the section "Native Thread Pools" on page 86 for directives for controlling the pool of native kernel threads.

For more information about performance tuning, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*.

AcceptTimeout

Specifies the number of seconds the server waits for data to arrive from the client. If data does not arrive before the timeout expires then the connection is closed. By setting it to less than the default 30 seconds, you can free up threads sooner. However, you may also disconnect users with slower connections.

Syntax

AcceptTimeout *seconds*

Default

30 seconds for servers that don't use hardware encryption devices and 300 seconds for those that do.

ConnQueueSize

Specifies the number of outstanding (yet to be serviced) connections that the web server can have. It is recommended that this value always be greater than the operating system limit for the maximum number of open file descriptors per process.

This setting can have performance implications. For more information, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*.

Default

The default value is `4096`.

HeaderBufferSize

The size (in bytes) of the buffer used by each of the request processing threads for reading the request data from the client. The maximum number of request processing threads is controlled by the "RqThrottle" on page 84 setting.

Default

The default value is `8192` (8 KB).

KeepAliveQueryMaxSleepTime

This directive specifies an upper limit to the time slept (in milliseconds) after polling keep-alive connections for further requests.

Default

The default is 100.

On lightly loaded systems that primarily service keep-alive connections, you can lower this number to enhance performance. However doing so can increase CPU usage.

KeepAliveQueryMeanTime

This directive specifies the desired keep-alive latency in milliseconds.

Default

The default value of 100 is appropriate for almost all installations.

Note that CPU usage will increase with lower KeepAliveQueryMeanTime values.

KeepAliveThreads

This directive determines the number of threads in the keep-alive subsystem. It is recommended that this number be a small multiple of the number of processors on the system (for example, a 2 CPU system should have 2 or 4 keep alive threads). The maximum number of keep-alive connections allowed ("MaxKeepAliveConnections" on page 83) should also be taken into consideration when choosing a value for this setting.

Default

1

KeepAliveTimeout

This directive determines the maximum time that the server holds open an HTTP Keep-Alive connection or a persistent connection between the client and the server. The Keep-Alive feature for earlier versions of the server allows the client/server connection to stay open while the server processes the client request. The default connection is a persistent connection that remains open until the server closes it or the connection has been open for longer than the time allowed by `KeepAliveTimeout`.

The timeout countdown starts when the connection is handed over to the keep-alive subsystem. If there is no activity on the connection when the timeout expires, the connection is closed.

Default

The default value is `30` seconds. The maximum value is `300` seconds (5 minutes).

KernelThreads

Sun Java System Web Server can support both kernel-level and user-level threads whenever the operating system supports kernel-level threads. Local threads are scheduled by NSPR (Netscape Portable Runtime) within the process, whereas kernel threads are scheduled by the host operating system. Usually, the standard debugger and compiler are intended for use with kernel-level threads. By setting `KernelThreads` to `1` (on), you ensure that the server uses only kernel-level threads, not user-level threads. By setting `KernelThreads` to `0` (off), you ensure that the server uses only user-level threads, which may improve performance.

Default

The default is 0 (off).

ListenQ

Specifies the maximum number of pending connections on a listen socket. Connections that time out on a listen socket whose backlog queue is full will fail.

Default

The default value is platform-specific: 4096 (AIX), 200 (Windows), 128 (all others).

MaxKeepAliveConnections

Specifies the maximum number of Keep-Alive and persistent connections that the server can have open simultaneously. Values range from 0 to 1048576 for the 64–bit server.

MaxProcs (UNIX Only)

Specifies the maximum number of processes that the server can have running simultaneously. If you don't include `MaxProcs` in your `magnus.conf` file, the server defaults to running a single process.

One process per processor is recommended if you are running in multi-process mode. In Sun Java System Web Server 6.1, there is always a primordial process in addition to the number of active processes specified by this setting.

Additional discussion of this and other server configuration and performance tuning issues can be found in the Sun Java System Web Server 6.1 *Performance Tuning, Sizing, and Scaling Guide*.

Default

1

PostThreadsEarly

If this directive is set to 1 (on), the server checks the whether the minimum number of threads are available at a listen socket after accepting a connection but before sending the response to the request. Use this directive when the server will be handling requests that take a long time to handle, such as those that do long database connections.

Default

0 (off)

RcvBufSize

Specifies the size (in bytes) of the receive buffer used by sockets. Allowed values are determined by the operating system.

Default

The default value is determined by the operating system. Typical defaults are 4096 (4K), 8192 (8K).

RqThrottle

Specifies the maximum number of request processing threads that the server can handle simultaneously. Each request runs in its own thread.

This setting can have performance implications. For more information, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*.

RqThrottleMin

Specifies the number of request processing threads that are created when the server is started. As the load on the server increases, more request processing threads are created (up to a maximum of `RqThrottle` threads).

SndBufSize

Specifies the size (in bytes) of the send buffer used by sockets.

Default

The default value is determined by the operating system. Typical defaults are `4096` (4K), `8192` (8K).

StackSize

Determines the maximum stack size for each request handling thread.

Default

The most favorable machine-specific stack size.

StrictHttpHeaders

Controls strict HTTP header checking. If strict HTTP header checking is on, the server rejects connections that include inappropriately duplicated headers.

Syntax

StrictHttpHeaders `[on|off]`

Default

`off`

TerminateTimeout

Specifies the time that the server waits for all existing connections to terminate before it shuts down.

Default

30 seconds

ThreadIncrement

The number of additional or new request processing threads created to handle an increase in the load on the server, for example when the number of pending connections (in the request processing queue) exceeds the number of idle request processing threads.

When a server starts up, it creates `RqThrottleMin` number of request processing threads. As the load increases, it creates `ThreadIncrement` additional request processing threads until `RqThrottle` request processing threads have been created.

Default

The default value is 10.

UseNativePoll (UNIX only)

Uses a platform-specific poll interface when set to 1(on). Uses the NSPR poll interface in the KeepAlive subsystem when set to 0 (off).

Default

1 (on)

Native Thread Pools

This section lists the directives for controlling the size of the native kernel thread pool. You can also control the native thread pool by setting the system variables `NSCP_POOL_STACKSIZE`, `NSCP_POOL_THREADMAX`, and `NSCP_POOL_WORKQUEUEMAX`. If you have set these values as environment variables and also in `magnus.conf`, the environment variable values will take precedence.

The native pool on UNIX is normally not engaged, as all threads are OS-level threads. Using native pools on UNIX may introduce a small performance overhead as they'll require an additional context switch; however, they can be used to localize the `jvm.stickyAttach` effect or for other purposes, such as resource control and management or to emulate single-threaded behavior for plug-ins.

On Windows, the default native pool is always being used and Sun Java System Web Server uses fibers (user-scheduled threads) for initial request processing. Using custom additional pools on Windows introduces no additional overhead.

The directives are:

NativePoolStackSize

Determines the stack size of each thread in the native (kernel) thread pool.

Default

0

NativePoolMaxThreads

Determines the maximum number of threads in the native (kernel) thread pool.

NativePoolMinThreads

Determines the minimum number of threads in the native (kernel) thread pool.

Default

1

NativePoolQueueSize

Determines the number of threads that can wait in the queue for the thread pool. If all threads in the pool are busy, then the next request-handling thread that needs to use a thread in the native pool must wait in the queue. If the queue is full, the next request-handling thread that tries to get in the queue is rejected, with the result that it returns a busy response to the client. It is then free to handle another incoming request instead of being tied up waiting in the queue.

Default

0

CGI

This section lists the directives in `magnus.conf` that affect requests for CGI programs. The directives are:

CGIExpirationTimeout

This directive specifies the maximum time in seconds that CGI processes are allowed to run before being killed.

The value of `CGIExpirationTimeout` should not be set too low— 300 seconds (5 minutes) would be a good value for most interactive CGIs; but if you have CGIs that are expected to take longer without misbehaving, then you should set it to the maximum duration you expect a CGI program to run normally. A value of `0` disables CGI expiration, which means that there is no time limit for CGI processes.

Note that on Windows platforms `init-cgi` time-out does not work, so you must use `CGIExpirationTimeout`.

Default

`0`

CGIStubIdleTimeout

This directive causes the server to kill any CGIStub processes that have been idle for the number of seconds set by this directive. Once the number of processes is at `MinCGIStubs`, the server does not kill any more processes.

Default

`30`

CGIWaitPid (UNIX Only)

For UNIX platforms, when `CGIWaitPid` is set to on, the action for the SIGCHLD signal is the system default action for the signal. If a NSAPI plug-in fork/execs a child process, it should call `waitpid` with its child process `pid` when `CGIWaitPid` is enabled to avoid leaving "defunct" processes when its child process terminates. When `CGIWaitPid` is enabled, the SHTML engine waits explicitly on its exec cmd child processes. Note that this directive has no effect on CGI.

Default

`on`

MaxCGIStubs

Controls the maximum number of CGIStub processes the server can spawn. This is the maximum concurrent CGIStub processes in execution, not the maximum number of pending requests. The default value should be adequate for most systems. Setting this too high may actually reduce throughput.

Default

`10`

MinCGIStubs

Controls the number of processes that are started by default. The first CGIStub process is not started until a CGI program has been accessed. Note that if you have an `init-cgi` directive in the `magnus.conf` file, the minimum number of CGIStub processes are spawned at startup. The value must be less than the `MaxCGIStubs` value.

Default

`2`

WincgiTimeout

WinCGI processes that take longer than this value are terminated when this timeout (in seconds) expires.

Default

`60`

Error Logging and Statistic Collection

This section lists the directives in `magnus.conf` that affect error logging and the collection of server statistics. They are:

ErrorLogDateFormat

The `ErrorLogDateFormat` directive specifies the date format that the server logs use.

Syntax

`ErrorLogDateFormat` *format*

The *format* can be any format valid for the C library function `strftime`. See Appendix C, "Time Formats"

Default

`%d/%b/%Y:%H:%M:%S`

LogFlushInterval

This directive determines the log flush interval, in seconds, of the log flush thread for the access log.

Default

`2`

PidLog

`PidLog` specifies a file in which to record the process ID (pid) of the base server process. Some of the server support programs assume that this log is in the server root, in `logs/pid`.

To shut down your server, kill the base server process listed in the pid log file by using a `-TERM` signal. To tell your server to reread its configuration files and reopen its log files, use `kill` with the `-HUP` signal.

If the `PidLog` file isn't writable by the user account that the server uses, the server does not log its process ID anywhere. The server won't start if it can't log the process ID.

Syntax

`PidLog` *file*

The *file* is the full path name and file name where the process ID is stored.

Default

There is no default.

Examples

```
PidLog /var/ns-server/logs/pid
```

```
PidLog /tmp/ns-server.pid
```

ACL

This section lists the directives in `magnus.conf` relevant to access control lists (ACLs). They are:

ACLCacheLifetime

`ACLCacheLifetime` determines the number of seconds before cache entries expire. Each time an entry in the cache is referenced, its age is calculated and checked against `ACLCacheLifetime`. The entry is not used if its age is greater than or equal to the `ACLCacheLifetime`. If this value is set to 0, the cache is turned off.

If you use a large number for this value, you may need to restart the Sun Java System Web Server when you make changes to the LDAP entries. For example, if this value is set to 120 seconds, the Sun Java System Web Server might be out of sync with the LDAP server for as long as two minutes. If your LDAP entries are not likely to change often, use a large number.

Default

120

ACLUserCacheSize

`ACLUserCacheSize` determines the number of users in the User Cache.

Default

200

ACLGroupCacheSize

`ACLGroupCacheSize` determines how many group IDs can be cached for a single UID/cache entry.

Default

4

Security

This section lists the directives in `magnus.conf` that affect server access and security issues for Sun Java System Web Server. They are:

Security

The Security directive globally enables or disables SSL by making certificates available to the server instance. It must be on for virtual servers to use SSL. If enabled, the user is prompted for the administrator password (in order to access certificates, and so on).

Note – When you create a secure listen socket through the Server Manager, security is automatically turned on globally in `magnus.conf`. When you create a secure listen socket manually in `server.xml`, security must be turned on by editing `magnus.conf`.

Syntax

```
Security [on|off]
```

Default

```
off
```

Example

```
Security off
```

ServerString

Allows the administrator to change the string sent with the `Server` HTTP header.

Syntax

`ServerString` *string*

string is the new string to send as the header. All characters, including quotes, will be sent. The string `none`, will cause the header to not be sent at all.

Example

```
ServerString My Own Server/1.0
ServerString none
```

SSLCacheEntries

Specifies the number of SSL sessions that can be cached. There is no upper limit.

Syntax

`SSLCacheEntries` *number*

If the *number* is 0, the default value, which is 10000, is used.

SSLClientAuthDataLimit

Specifies the maximum amount of application data, in bytes, that is buffered during the client certificate handshake phase.

Default

The default value is `1048576` (1 MB).

SSLClientAuthTimeout

Specifies the number of seconds after which the client certificate handshake phase times out.

Default

`60`

SSLSessionTimeout

The `SSLSessionTimeout` directive controls SSL2 session caching.

Syntax

`SSLSessionTimeout` *seconds*

The *seconds* value is the number of seconds until a cached SSL2 session becomes invalid. If the `SSLSessionTimeout` directive is specified, the value of seconds is silently constrained to be between 5 and 100 seconds.

Default

The default value is 100.

SSL3SessionTimeout

The `SSL3SessionTimeout` directive controls SSL3 session caching.

Syntax

`SSL3SessionTimeout` *seconds*

The *seconds* value is the number of seconds until a cached SSL3 session becomes invalid. The default value is 86400 (24 hours). If the `SSL3SessionTimeout` directive is specified, the value of seconds is silently constrained to be between 5 and 86400 seconds.

Chunked Encoding

This section lists directives that control chunked encoding. For more information, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

These directives have equivalent Service SAF parameters in `obj.conf`. The `obj.conf` parameters override these directives. For more information, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

UseOutputStreamSize

The `UseOutputStreamSize` directive determines the default output stream buffer size for the net_read and netbuf_grab NSAPI functions.

Note – The `UseOutputStreamSize` parameter can be set to `0` in the `obj.conf` file to disable output stream buffering. For the `magnus.conf` file, setting `UseOutputStreamSize` to `0` has no effect.

Syntax

`UseOutputStreamSize` *size*

The *size* value is the number of bytes.

Default

The default value is `8192` (8 KB).

ChunkedRequestBufferSize

The `ChunkedRequestBufferSize` directive determines the default buffer size for "un-chunking" request data.

Syntax

`ChunkedRequestBufferSize` *size*

The *size* value is the number of bytes.

Default

The default value is 8192.

ChunkedRequestTimeout

The `ChunkedRequestTimeout` directive determines the default timeout for "un-chunking" request data.

Syntax

`ChunkedRequestTimeout` *seconds*

The *seconds* value is the number of seconds.

Default

The default value is 60 (1 minute).

Miscellaneous Directives

This section lists the following miscellaneous directives in `magnus.conf`:

Note – Directives noted with boolean values have the following equivalent values: `on/yes/true` and `off/no/false`.

ChildRestartCallback

This directive forces the callback of NSAPI functions that were registered using the daemon_atrestart function when the server is restarting or shutting down. Values are `on`, `off`, `yes`, `no`, `true`, or `false`.

Default

`no directive`

Favicon

To turn off the internal `favicon.ico` support, add the following line to `magnus.conf`:

```
Favicon off
```

HTTPVersion

The current HTTP version used by the server in the form *m.n*, where *m* is the major version number and *n* the minor version number.

Default

The default value is `1.1`.

MaxRqHeaders

Specifies the maximum number of header lines in a request. Values range from 1 to 512.

Default

64

Umask (UNIX only)

This directive specifies the umask value used by the NSAPI functions `System_fopenWA()` and `System_fopenRW()` to open files in different modes. Valid values for this directive are standard UNIX umask values.

For more information on these functions, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

Deprecated Directives

The following directives have been deprecated in Sun Java System Web Server 6.1:

- `AdminLanguage`
- `ClientLanguage`
- `NetsiteRoot`
- `ServerID`
- `ServerName`
- `ServerRoot`

Summary of Init Functions and Directives in magnus.conf

Purpose

Contains global variable settings that affect server functioning. This file is read only at server start-up.

Location

server_root/https-admserv/config
server_root/https-admserv/conf_bk
server_root/https-*server_id*/config
server_root/https-*server_id*/conf_bk

Syntax

`Init` functions have the following syntax:

`Init fn=`*function* *param1*`="`*value1*`" ...`*paramN*`="`*valueN*`"`

In the following table"Init Functions" on page 97, functions are in bold to distinguish them from parameters.

Directives have the following syntax:

directive *value*

See Also

Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide

Init Functions

The following table lists the Init functions available in the magnus.conf file:

TABLE 3–1 magnus.conf Init Functions

Function/Parameter	Allowed Values	Default Value	Description
cindex-init			Changes the default characteristics for fancy indexing.
opts	s	(None)	(optional) is a string of letters specifying the options to activate. Currently there is only one possible option: ■ s tells the server to scan each HTML file in the directory being indexed for the contents of the HTML <TITLE> tag to display in the description field. The <TITLE> tag must be within the first 255 characters of the file.
widths	Comma separated numbers of characters	Minimums required to display column titles	(optional) Specifies the width for each of the four columns in the indexing display: name, last-modified date, size, and description respectively. The final three values can each be set to 0 to turn the display for that column off. The name column cannot be turned off.
timezone	GMT or local	local	(optional, iPlanet Web Server 4.*x* only) Indicates whether the last-modified time is shown in local time or in Greenwich Mean Time.
format	Format for the UNIX function strftime()	%d-%b-%Y %H:%M	(optional, iPlanet Web Server 4.*x* only) Determines the format of the last modified date display.
ignore	Wildcard pattern	.*	(optional) Specifies a wildcard pattern for file names the server should ignore while indexing. File names starting with a period (.) are always ignored.

TABLE 3–1 magnus.conf Init Functions *(Continued)*

Function/Parameter	Allowed Values	Default Value	Description
`icon-uri`		`/mc-icons/`	(optional) Specifies the URI prefix the `index-common` function uses when generating URLs for file icons (`.gif` files). If `icon-uri` is different from the default, the `pfx2dir` function in the `NameTrans` directive must be changed so that the server can find these icons.
`define-perf-bucket`			Creates a performance bucket, which you can use to measure the performance of SAFs in `obj.conf` (see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*). This function works only if the `perf-init` function is enabled.
`name`			A name for the bucket, for example `cgi-bucket`.
`description`			A description of what the bucket measures, for example `CGI Stats`.
`dns-cache-init`			Configures DNS caching.
`cache-size`	`32` to `32768` (32K)	`1024`	(optional) Specifies how many entries are contained in the cache.
`expire`	`1` to `31536000` seconds (1 year)	`1200` seconds (20 minutes)	(optional) specifies how long (in seconds) it takes for a cache entry to expire.
`flex-init`			Initializes the flexible logging system.
logFileName	A path or file name		The full path to the log file or a file name relative to the server's `logs` directory. In this example, the log file name is `access` and the path is `/logdir/access`: `access="/logdir/access"`

TABLE 3–1 magnus.conf Init Functions *(Continued)*

Function/Parameter	Allowed Values	Default Value	Description
`format.`*logFileName*			Specifies the format of each log entry in the log file. See the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide* for more information.
`buffer-size`	Number of bytes	`8192`	Specifies the size of the global log buffer.
`buffers-per-file`	The lower bound is 1. There always needs to be at least one buffer per file. The upper bound is dictated by the number of buffers that exist. The upper bound on the number of buffers that exist can be defined by the num-buffers parameter.	Determined by the server	Specifies the number of buffers for a given log file
`num-buffers`		`1000`	Specifies the maximum number of logging buffers to use.
`flex-rotate-init`			Enables rotation for logs.
`rotate-start`	A 4-digit string indicating the time in 24-hour format		Indicates the time to start rotation. For example, 0900 indicates 9 am while 1800 indicates 9 pm.
`rotate-interval`	Number of minutes		Indicates the number of minutes to elapse between each log rotation.
`rotate-access`	`yes`, `no`	`yes`	(optional) determines whether `common-log`, `flex-log`, and `record-useragent` logs are rotated. For more information, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.
`rotate-error`	`yes`, `no`	`yes`	(optional) determines whether error logs are rotated.

TABLE 3–1 magnus.conf Init Functions *(Continued)*

Function/Parameter	Allowed Values	Default Value	Description
`rotate-callback`	A path		(optional) specifies the file name of a user-supplied program to execute following log file rotation. The program is passed the post-rotation name of the rotated log file as its parameter.
`init-cgi`			Changes the default settings for CGI programs.
`timeout`	Number of seconds	`300`	(optional) specifies how many seconds the server waits for CGI output before terminating the script.
`cgistub-path`			(optional) specifies the path to the CGI stub binary. If not specified, iPlanet Web Server looks in the following directories, in the following order, relative to the server instance's `config` directory: `../private/Cgistub`, then `../../bin/https/bin/Cgistub`. For information about installing an suid Cgistub, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.
env-variable			(optional) specifies the name and value for an environment variable that the server places into the environment for the CGI.
`init-clf`			Initializes the Common Log subsystem.
logFileName	A path or file name		Specifies either the full path to the log file or a file name relative to the server's `logs` directory.
`init-uhome`			Loads user home directory information.

TABLE 3–1 magnus.conf Init Functions *(Continued)*

Function/Parameter	Allowed Values	Default Value	Description
`pwfile`			(optional) specifies the full file system path to a file other than `/etc/passwd`. If not provided, the default UNIX path (`/etc/passwd`) is used.
`load-modules`			Loads shared libraries into the server.
`shlib`			Specifies either the full path to the shared library or dynamic link library or a file name relative to the server configuration directory.
`funcs`	A comma separated list with no spaces		A list of the names of the functions in the shared library or dynamic link library to be made available for use by other `Init` or `Service` directives. The dash (-) character may be used in place of the underscore (_) character in function names.
`NativeThread`	`yes`, `no`	`yes`	(optional) specifies which threading model to use. `no` causes the routines in the library to use user-level threading. `yes` enables kernel-level threading.
`pool`			The name of a custom thread pool, as specified in `thread-pool-init`.
`nt-console-init`			Enables the NT console, which is the command-line shell that displays standard output and error streams.
`stderr`	`console`		Directs error messages to the NT console.
`stdout`	`console`		Directs output to the NT console.
`perf-init`			Enables system performance measurement via performance buckets.

TABLE 3–1 magnus.conf Init Functions *(Continued)*

Function/Parameter	Allowed Values	Default Value	Description
`disable`	`true`, `false`	`true`	Disables the function when `true`.
`pool-init`			Configures pooled memory allocation.
`block-size`			(optional) size (in bytes) of the memory blocks allocated by the internal pooled memory allocator. Default value is `32768`.
`free-size`	`1048576` bytes or less		(optional) maximum size in bytes of free block list.
`disable`	`true`, `false`	`false`	(optional) flag to disable the use of pooled memory if `true`.
`register-http-method`			Lets you extend the HTTP protocol by registering new HTTP methods.
`methods`	A comma separated list		Names of the methods you are registering.
`stats-init`			Enables reporting of performance statistics in XML format.
`profiling`	`yes`, `no`	`no`	Enables NSAPI performance profiling using buckets. This can also be enabled through `perf-init`.
`update-interval`	`1` or greater	`5`	The period in seconds between statistics updates within the server.
`virtual-servers`	`1` or greater	`1000`	The maximum number of virtual servers for which statistics are tracked. This number should be set higher than the number of virtual servers configured.
`thread-pool-init`			Configures an additional thread pool.
`name`			Name of the thread pool.

TABLE 3–1 magnus.conf Init Functions *(Continued)*

Function/Parameter	Allowed Values	Default Value	Description
`maxthreads`			Maximum number of threads in the pool.You can use only five threads at a time in the FastTrack edition of iPlanet Web Server.
`minthreads`			Minimum number of threads in the pool.
`queueSize`	Number of bytes		Size of the queue for the pool.
`stackSize`	Number of bytes		Stack size of each thread in the native (kernel) thread pool.

Directives

The following table lists

TABLE 3–2 magnus.conf Directives

Directive	Allowed Values	Default Value	Description
`AcceptTimeout`	Any number of seconds	`30` for servers that don't use hardware encryption devices and 300 for those that do	Specifies the number of seconds the server waits for data to arrive from the client. If data does not arrive before the timeout expires then the connection is closed.
`ACLCacheLifetime`	Any number of seconds	`120`	Determines the number of seconds before cache entries expire. Each time an entry in the cache is referenced, its age is calculated and checked against `ACLCacheLifetime`. The entry is not used if its age is greater than or equal to the `ACLCacheLifetime`. If this value is set to `0`, the cache is turned off.
`ACLUserCacheSize`		`200`	Determines the number of users in the User Cache.
`ACLGroupCacheSize`		`4`	Determines how many group IDs can be cached for a single UID/cache entry.

TABLE 3–2 magnus.conf Directives *(Continued)*

Directive	Allowed Values	Default Value	Description
`AsyncDNS`	`on`, `off`	`off`	Specifies whether asynchronous DNS is allowed.
`CGIExpirationTimeout`	Any number of seconds	`300` (5 minutes) recommended	Specifies the maximum time in seconds that CGI processes are allowed to run before being killed.
`CGIStubIdleTimeout`	Any number of seconds	`30`	Causes the server to kill any CGIStub processes that have been idle for the number of seconds set by this directive. Once the number of processes is at `MinCGIStubs`, the server does not kill any more processes.
`CGIWaitPid`	`on`, `off`	`on`	(UNIX only) makes the action for the SIGCHLD signal the system default action for the signal. Makes the SHTML engine wait explicitly on its exec cmd child processes.
`ChildRestartCallback`	`on`, `off`, `yes`, `no`, `true`, `false`	`no`	Forces the callback of NSAPI functions that were registered using the `daemon_atrestart` function when the server is restarting or shutting down.
`ChunkedRequestBufferSize`	Any number of bytes	`8192`	Determines the default buffer size for "un-chunking" request data.
`ChunkedRequestTimeout`	Any number of seconds	`60` (1 minute).	Determines the default timeout for "un-chunking" request data.
`ConnQueueSize`	Any number of connections (including 0)	`4096`	Specifies the number of outstanding (yet to be serviced) connections that the web server can have. This setting can have performance implications. For more information, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*.
`DefaultLanguage`	`en` (English), `fr` (French), `de` (German), `ja` (Japanese)	`en`	Specifies the default language for the server. The default language is used for both the client responses and administration.

TABLE 3–2 magnus.conf Directives *(Continued)*

Directive	Allowed Values	Default Value	Description
`DNS`	`on`, `off`	`on`	Specifies whether the server performs DNS lookups on clients that access the server.
`ErrorLogDateFormat`	See the manual page for the C library function `strftime`	`%d/%b/%Y:%H:%M:%S`	The date format for the error log.
`ExtraPath`	A path	(none)	Appends the specified directory name to the `PATH` environment variable. This is used for configuring Java on Windows NT. There is no default value; you must specify a value.
`Favicon`	`On / Off`	`on`	Provides the web server administrator the ability to disable or change the icon which appears in the web address book or favorites list on Internet Explorer browsers (so, `favicon` translates as `favorite icon`).
`HeaderBufferSize`	Any number of bytes	`8192` (8 KB)	The size (in bytes) of the buffer used by each of the request processing threads for reading the request data from the client. The maximum number of request processing threads is controlled by the `RqThrottle` setting.
`HTTPVersion`	*m.n*; *m* is the major version number and *n* the minor version number	`1.1`	The current HTTP version used by the server.

TABLE 3–2 magnus.conf Directives *(Continued)*

Directive	Allowed Values	Default Value	Description
`KeepAliveQueryMaxSleepTime`		`100` On lightly loaded systems that primarily service keep-alive connections, you can lower this number to enhance performance. However doing so can increase CPU usage.	This directive specifies an upper limit to the time slept (in milliseconds) after polling keep-alive connections for further requests.
`KeepAliveQueryMeanTime`		`100` is appropriate for almost all installations. Note that CPU usage will increase with lower KeepAliveQueryMeanTime values.	This directive specifies the desired keep-alive latency in milliseconds.
`KeepAliveThreads`	Any number of threads	`1`	Specifies the number of threads in the keep-alive subsystem. It is recommended that this number be a small multiple of the number of processors on the system.
`KeepAliveTimeout`	`300` seconds maximum	`30`	Determines the maximum time that the server holds open an HTTP Keep-Alive connection or a persistent connection between the client and the server.
`KernelThreads`	`0` (off), `1` (on)	`0` (off)	If on, ensures that the server uses only kernel-level threads, not user-level threads. If off, uses only user-level threads.
`ListenQ`	Ranges are platform-specific	`4096` (AIX), `200` (NT), `128` (all others)	Defines the number of incoming connections for a server socket.

TABLE 3–2 magnus.conf Directives *(Continued)*

Directive	Allowed Values	Default Value	Description
`LogFlushInterval`	Any number of seconds	`30`	Determines the log flush interval, in seconds, of the log flush thread.
`MaxCGIStubs`	Any number of CGI stubs	`10`	Controls the maximum number of CGIStub processes the server can spawn. This is the maximum concurrent CGIStub processes in execution, not the maximum number of pending requests.
`MaxKeepAliveConnections`	`0` - `1048576` (for 64–bit server)	`25616`	Specifies the maximum number of Keep-Alive and persistent connections that the server can have open simultaneously.
`MaxProcs`	Any number of processes1	`1`	(UNIX only) Specifies the maximum number of processes that the server can have running simultaneously.
`MaxRqHeaders`	`1` - `512`	`64`	Specifies the maximum number of header lines in a request.
`MinCGIStubs`	Any number less than MaxCGIStubs	`2`	Controls the number of processes that are started by default.
`NativePoolMaxThreads`	Any number of threads	`1285`	Determines the maximum number of threads in the native (kernel) thread pool.
`NativePoolMinThreads`	Any number of threads	`1`	Determines the minimum number of threads in the native (kernel) thread pool.
`NativePoolQueueSize`	Any nonnegative number	`0`	Determines the number of threads that can wait in the queue for the thread pool.
`NativePoolStackSize`	Any nonnegative number	`0`	Determines the stack size of each thread in the native (kernel) thread pool.
`PidLog`	A valid path to a file	(none)	Specifies a file in which to record the process ID (pid) of the base server process.

TABLE 3–2 magnus.conf Directives *(Continued)*

Directive	Allowed Values	Default Value	Description
`PostThreadsEarly`	`1` (on), 0 (off)	0 (off)	If on, checks whether the minimum number of threads are available at a socket after accepting a connection but before sending the response to the request.
`RcvBufSize`	Range is platform-specific	`0` (uses platform-specific default)	Controls the size of the receive buffer at the server's sockets.
`RqThrottle`	Any number of requests (including 0)	`51216`	Specifies the maximum number of simultaneous request processing threads that the server can handle simultaneously per socket. This setting can have performance implications. For more information, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*.
`RqThrottleMin`	Any number less than `RqThrottle`	`484`	Specifies the number of request processing threads that are created when the server is started. As the load on the server increases, more request processing threads are created (up to a maximum of `RqThrottle` threads).
`Security`	`on`, `off`	`off`	Globally enables or disables SSL by making certificates available to the server instance. Must be `on` for virtual servers to use SSL.
`SndBufSize`	Range is platform-specific	`0` (uses platform-specific default)	Controls the size of the send buffer at the server's sockets.
`SSL3SessionTimeout`	`5` - `86400`	`86400` (24 hours).	The number of seconds until a cached SSL3 session becomes invalid.
`SSLCacheEntries`	A non-negative integer	`10000` (used if `0` is specified)	Specifies the number of SSL sessions that can be cached. There is no upper limit.
`SSLClientAuthDataLimit`	Number of Bytes	`1048576` (1MB)	Specifies the maximum amount of application data that is buffered during the client certificate handshake phase.

TABLE 3–2 magnus.conf Directives *(Continued)*

Directive	Allowed Values	Default Value	Description
`SSLClientAuthTimeout`	Any number of seconds	`60`	Specifies the number of seconds after which the client certificate handshake phase times out.
`SSLSessionTimeout`	`5` - `100`	`100`	Specifies the number of seconds until a cached SSL2 session becomes invalid.
`StackSize`	Number of Bytes	The most favorable machine-specific stack size.	Determines the maximum stack size for each request handling thread.
`StrictHttpHeaders`	`on`, `off`	`off`	If `on`, rejects connections that include inappropriately duplicated headers.
`TempDir`	A path	`/tmp` (UNIX) `TEMP` (environment variable for Windows NT)	Specifies the directory the server uses for its temporary files. On UNIX, this directory should be owned by, and writable by, the user the server runs as.
`TempDirSecurity`	`on`, `off`	`on`	Determines whether the server checks if the `TempDir` directory is secure. On UNIX, specifying `TempDirSecurity off` allows the server to use `/tmp` as a temporary directory.
`TerminateTimeout`	Any number of seconds	`30`	Specifies the time in seconds that the server waits for all existing connections to terminate before it shuts down.
`ThreadIncrement`	Any number of threads	`10`	The number of additional or new request processing threads created to handle an increase in the load on the server.
`Umask`	A standard UNIX umask value	(none)	UNIX only: Specifies the umask value used by the NSAPI functions `System_fopenWA()` and `System_fopenRW()` to open files in different modes.

TABLE 3–2 magnus.conf Directives *(Continued)*

Directive	Allowed Values	Default Value	Description
`UseNativePoll`	`1` (on), 0 (off)	`1` (on)	Uses a platform-specific poll interface when set to `1` (on). Uses the NSPR poll interface in the KeepAlive subsystem when set to `0` (off).
`UseOutputStreamSize`	Any number of bytes	`8192` (8 KB)	Determines the default output stream buffer size for the `net_read` and `netbuf_grab` NSAPI functions.
`User`	A login name, 8 characters or less	(none)	(Windows NT) specifies the user account the server runs with, allowing you to restrict or enable system features for the server. (UNIX) if the server is started by the `superuser` or `root` user, the server binds to the Port you specify and then switches its user ID to the user account specified with the User directive. This directive is ignored if the server isn't started as `root`.
`WincgiTimeout`	Any number of seconds	`60`	WinCGI processes that take longer than this value are terminated when this timeout expires.

◆ ◆ ◆ CHAPTER 4

Predefined SAFs in obj.conf

The `obj.conf` configuration file contains directives that instruct the Sun Java System Web Server how to handle HTTP and HTTPS requests from clients and service web server content such as native server plug-ins and CGI programs. You can modify and extend the request-handling process by adding or changing the instructions in `obj.conf`.

All obj.conf files are located in the *instance_dir*`/config` directory, where *instance_dir* is the path to the installation directory of the server instance. There is one `obj.conf` file for each virtual server class, unless several virtual server classes are configured to share an `obj.conf` file. Whenever this guide refers to "the `obj.conf` file," it refers to all `obj.conf` files or to the `obj.conf` file for the virtual server class being described.

By default, the `obj.conf` file for the initial virtual server class is named `obj.conf`, and the `obj.conf` files for the administrator-defined virtual server classes are named *virtual_server_class_id*`.obj.conf`. Editing one of these files directly or through the Administration interface changes the configuration of a virtual server class.

This chapter describes the standard directives and predefined Server Application Functions (SAFs) that are used in the `obj.conf` file to give instructions to the server. For details about the syntax and use of the `obj.conf` file, refer to the Sun Java System Web Server 6.1 *NSAPI Programmer's Guide*.

Each SAF has its own arguments, which are passed to it by a directive in `obj.conf`. Every SAF is also passed additional arguments that contain information about the request (such as what resource was requested and what kind of client requested it), and any other server variables created or modified by SAFs called by previously invoked directives. Each SAF may examine, modify, or create server variables. Each SAF returns a result code that tells the server whether it succeeded, did nothing, or failed.

This chapter includes functions that are part of the core functionality of Sun Java System Web Server. It does not include functions that are available only if additional components, such as server-parsed HTML, are enabled.

This chapter covers the following stages:

- "AuthTrans" on page 116
- "NameTrans" on page 126
- "PathCheck" on page 136
- "ObjectType" on page 151
- "Input" on page 157
- "Output" on page 160
- "Service" on page 162
- "AddLog" on page 195
- "Error" on page 198

For an alphabetical list of predefined SAFs, see Appendix E, Alphabetical List of Predefined SAFs

List of SAFs

The following table lists the Server Application Functions (SAFs) that can be used with each directive.

TABLE 4–1 AvailableSAFs per Directive

Directive	Server Application Functions
"AuthTrans" on page 116	"basic-auth" on page 117 "basic-ncsa" on page 118"get-sslid" on page 119 "match-browser" on page 120"qos-handler" on page 121"set-variable" on page 122
"NameTrans" on page 126	"assign-name" on page 126"document-root" on page 128 "home-page" on page 129 "match-browser" on page 120"ntrans-dav" on page 130 "ntrans-j2ee" on page 131 "pfx2dir" on page 131 "redirect" on page 133 "set-variable" on page 122"strip-params" on page 134"unix-home" on page 135
"PathCheck" on page 136	"check-acl" on page 136 "deny-existence" on page 139 "find-index" on page 139 "find-links" on page 140 "find-pathinfo" on page 141 "get-client-cert" on page 142 "load-config" on page 143 "match-browser" on page 120"nt-uri-clean" on page 145 "ntcgicheck" on page 146 "require-auth" on page 147 "set-variable" on page 122"set-virtual-index" on page 148"ssl-check" on page 149"ssl-logout" on page 150"unix-uri-clean" on page 150
"ObjectType" on page 151	"force-type" on page 152 "match-browser" on page 120"set-default-type" on page 153 "set-variable" on page 122"shtml-hacktype" on page 154 "type-by-exp" on page 155 "type-by-extension" on page 156

TABLE 4–1 AvailableSAFs per Directive *(Continued)*

Directive	Server Application Functions
"Input" on page 157	"insert-filter" on page 158"match-browser" on page 120"remove-filter" on page 159"set-variable" on page 122
"Output" on page 160	"insert-filter" on page 160"match-browser" on page 120"remove-filter" on page 161"set-variable" on page 122
"Service" on page 162	"add-footer" on page 165"add-header" on page 166"append-trailer" on page 167 "imagemap" on page 169 "index-common" on page 169 "index-simple" on page 171 "key-toosmall" on page 172 "list-dir" on page 173 "make-dir" on page 175"match-browser" on page 120"query-handler" on page 176 "remove-dir" on page 176 "delete-file" on page 177 "remove-filter" on page 178"rename-file" on page 179 "send-cgi" on page 180 "send-error" on page 182 "send-file" on page 183 "send-range" on page 185 "send-shellcgi" on page 185 "send-wincgi" on page 186 "service-dump" on page 188"service-j2ee" on page 189"service-trace" on page 190"set-variable" on page 122"shtml_send" on page 191"stats-xml" on page 192"upload-file" on page 194
"AddLog" on page 195	"common-log" on page 195 "flex-log" on page 196"match-browser" on page 120"record-useragent" on page 197 "set-variable" on page 122
"Error" on page 198	"error-j2ee" on page 199"match-browser" on page 120"qos-error" on page 200"query-handler" on page 200"remove-filter" on page 201"send-error" on page 202"set-variable" on page 122

The bucket Parameter

The following performance buckets are predefined in Sun Java System Web Server:

- The `default-bucket` records statistics for the functions not associated with any user-defined or built-in bucket.
- The `all-requests` bucket records `.perf` statistics for all NSAPI SAFs, including those in the `default-bucket`.

  You can define additional performance buckets in the `magnus.conf` file (see the `perf-init` and `define-perf-bucket` functions).

  You can measure the performance of any SAF in `obj.conf` by adding a `bucket=`*bucket-name* parameter to the function, for example `bucket=cache-bucket`.

  To list the performance statistics, use the "service-dump" on page 188 `Service` function.

  As an alternative, you can use the "stats-xml" on page 192 `Service` function to generate performance statistics; use of buckets is optional.

  For more information about performance buckets, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*.

AuthTrans

AuthTrans stands for Authorization Translation. AuthTrans directives give the server instructions for checking authorization before allowing a client to access resources. AuthTrans directives work in conjunction with PathCheck directives. Generally, an AuthTrans function checks if the user name and password associated with the request are acceptable, but it does not allow or deny access to the request; that is left to a PathCheck function.

The server handles the authorization of client users in two steps:

- "AuthTrans" on page 116 validates authorization information sent by the client in the Authorization header.
- "PathCheck" on page 136 checks that the authorized user is allowed access to the requested resource.

  The authorization process is split into two steps so that multiple authorization schemes can be easily incorporated, and to provide the flexibility to have resources that record authorization information, but do not require it.

  AuthTrans functions get the user name and password from the headers associated with the request. When a client initially makes a request, the user name and password are unknown so the AuthTrans functions and PathCheck functions work together to reject the request, since they can't validate the user name and password. When the client receives the rejection, its usual response is to present a dialog box asking for the user name and password to enter the appropriate realm, and then the client submits the request again, this time including the user name and password in the headers.

  If there is more than one AuthTrans directive in obj.conf, each function is executed in order until one succeeds in authorizing the user.

  The following AuthTrans-class functions are described in detail in this section:
- "basic-auth" on page 117 calls a custom function to verify user name and password. Optionally determines the user's group.
- "basic-ncsa" on page 118 verifies user name and password against an NCSA-style or system DBM database. Optionally determines the user's group.
- "get-sslid" on page 119 retrieves a string that is unique to the current SSL session and stores it as the ssl-id variable in the Session->client parameter block.
- "match-browser" on page 120 matches specific strings in the User-Agent string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "qos-handler" on page 121 handles the current quality of service statistics.
- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.

basic-auth

Applicable in AuthTrans-class directives.

The basic-auth function calls a custom function to verify authorization information sent by the client. The Authorization header is sent as part of the basic server authorization scheme.

This function is usually used in conjunction with the PathCheck-class function "require-auth" on page 147.

Parameters

The following table describes parameters for the basic-auth function.

TABLE 4–2 basic-auth Parameters

Parameter	Description
auth-type	Specifies the type of authorization to be used. This should always be basic.
userdb	(Optional) Specifies the full path and file name of the user database to be used for user verification. This parameter will be passed to the user function.
userfn	Name of the user custom function to verify authorization. This function must have been previously loaded with load-modules. It has the same interface as all of the SAFs, but it is called with the user name (user), password (pw), user database (userdb), and group database (groupdb) if supplied, in the pb parameter. The user function should check the name and password using the database and return REQ_NOACTION if they are not valid. It should return REQ_PROCEED if the name and password are valid. The basic-auth function will then add auth-type, auth-user (user), auth-db (userdb), and auth-password (pw, Windows only) to the rq->vars pblock.
groupdb	(Optional) Specifies the full path and file name of the user database. This parameter will be passed to the group function.
groupfn	(Optional) Name of the group custom function that must have been previously loaded with load-modules. It has the same interface as all of the SAFs, but it is called with the user name (user), password (pw), user database (userdb), and group database (groupdb) in the pb parameter. It also has access to the auth-type, auth-user (user), auth-db (userdb), and auth-password (pw, Windows only) parameters in the rq->vars pblock. The group function should determine the user's group using the group database, add it to rq->vars as auth-group, and return REQ_PROCEED if found. It should return REQ_NOACTION if the user's group is not found.
bucket	(Optional) Common to all obj.conf functions.

Examples

In `magnus.conf`:

```
Init fn=load-modules shlib=/path/to/mycustomauth.so
funcs=hardcoded_auth
```

In `obj.conf`:

```
AuthTrans fn=basic-auth auth-type=basic
userfn=hardcoded_authPathCheck fn=require-auth
auth-type=basic realm="Marketing Plans"
```

See Also

"require-auth" on page 147

basic-ncsa

Applicable in `AuthTrans`-class directives.

The `basic-ncsa` function verifies authorization information sent by the client against a database. The `Authorization` header is sent as part of the basic server authorization scheme.

This function is usually used in conjunction with the `PathCheck`-class function "require-auth" on page 147.

Parameters

The following table describes parameters for the `basic-ncsa` function.

TABLE 4–3 basic-auth Parameters

Parameter	Description
`auth-type`	Specifies the type of authorization to be used. This should always be `basic`.
`dbm`	(Optional) Specifies the full path and base file name of the user database in the server's native format. The native format is a system DBM file, which is a hashed file format allowing instantaneous access to billions of users. If you use this parameter, don't use the `userfile` parameter as well.
`userfile`	(Optional) Specifies the full path name of the user database in the NCSA-style HTTPD user file format. This format consists of lines using the format *name:password*, where *password* is encrypted. If you use this parameter, don't use `dbm`.

TABLE 4–3 basic-auth Parameters *(Continued)*

Parameter	Description
`grpfile`	(Optional) Specifies the NCSA-style HTTPD group file to be used. Each line of a group file consists of *group* : *user1 user2 ... userN* where each user is separated by spaces.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
AuthTrans fn=basic-ncsa auth-type=basic
dbm=/sun/server61/userdb/rs

PathCheck fn=require-auth auth-type=basic realm="Marketing Plans"
AuthTrans fn=basic-ncsa auth-type=basic
userfile=/sun/server61/.htpasswd grpfile=/sun/server61/.grpfile

PathCheck fn=require-auth auth-type=basic realm="Marketing Plans"
```

See Also

"require-auth" on page 147

get-sslid

Applicable in `AuthTrans`-class directives.

Note – This function is provided for backward compatibility only. The functionality of `get-sslid` has been incorporated into the standard processing of an SSL connection.

The `get-sslid` function retrieves a string that is unique to the current SSL session, and stores it as the `ssl-id` variable in the `Session->client` parameter block.

If the variable `ssl-id` is present when a CGI is invoked, it is passed to the CGI as the `HTTPS_SESSIONID` environment variable.

The `get-sslid` function has no parameters and always returns `REQ_NOACTION`. It has no effect if SSL is not enabled.

Parameters

The following table describes parameters for the `get-sslid` function.

TABLE 4–4 get-sslid Parameters

Parameter	Description
`bucket`	(Optional) Common to all `obj.conf` functions.

match-browser

Applicable in all stage directives.

The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.

Syntax

stage `fn="match-browser" browser=`*"string" name="value"* [*name="value"* `...`]

Parameters

The following table describes parameter values for the `match-browser` function.

TABLE 4–5 match-browser Parameter Values

Value	Description
stage	Stage directive used in `obj.conf` processing (`NameTrans`, `PathCheck`, and so on). The `match-browser` function is applicable in all stage directives.
string	Wildcard pattern to compare against the `User-Agent` header (for example, `"*Mozilla*"`).
name	Variable to be changed. The `match-browser` SAF indirectly invokes the "set-variable" on page 122 SAF. For a list of valid variables, see "set-variable" on page 122.
value	New value for the specified variable.

Example

The following `AuthTrans` directive instructs Sun Java System Web Server to do as follows when the browser's `User-Agent` header contains the string `Broken` or `broken`. The server will:

- Not send the SSL3 and TLS `close_notify` packet (see "set-variable" on page 122).
- Not honor requests for HTTP Keep-Alive (see "set-variable" on page 122
- Use the HTTP/1.0 protocol rather than HTTP/1.1 (see "set-variable" on page 122).

```
AuthTrans fn="match-browser"
                browser="*[Bb]roken*"
                ssl-unclean-shutdown="true"
                keep-alive="disabled"
                http-downgrade="1.0"
```

See Also

"set-variable" on page 122

qos-handler

Applicable in `AuthTrans`-class directives.

The `qos-handler` function examines the current quality of service statistics for the virtual server, virtual server class, and global server, logs the statistics, and enforces the QOS parameters by returning an error. This must be the first `AuthTrans` function configured in the `default` object in order to work properly.

The code for this SAF is one of the examples provided in the Sun Java System Web Server 6.1 *NSAPI Programmer's Guide*.

For more information, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*.

Parameters

The following table describes parameters for the `qos-handler` function.

TABLE 4–6 qos-handler Parameters

Parameter	Description
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
AuthTrans fn=qos-handler
```

See Also

"qos-error" on page 200

set-variable

Applicable in all stage directives.

The `set-variable` function enables you to change server settings based upon conditional information in a request. It can also be used to manipulate variables in parameter blocks with the following commands:

- `insert-`*pblock*`="`*name*`=`*value*`"`

  Adds a new value to the specified *pblock*.
- `set-`*pblock*`="`*name*`=`*value*`"`

  Sets a new value in the specified *pblock*, replacing any existing value(s) with the same name.
- `remove-`*pblock*`="`*name*`"`

  Removes all values with the given name from the specified *pblock*.

Note – For more information about parameter blocks, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

Syntax

stage `fn="set-variable" [{insert|set|remove}-`*pblock*`="`*name*`=`*value*`" ...][`*name*`="`*value*`" ...]`

Parameters

The following table describes parameter values for the `set-variable` function.

TABLE 4–7 set-variable Parameter Values

Value	Description
pblock	One of the following Session/Request parameter block names: ■ `client`: Contains the IP address of the client machine and the DNS name of the remote machine. For more information, see the description of the `Session->client` function in the "Data Structure Reference" chapter of the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*. ■ `vars`: Contains the server's working variables, which includes anything not specifically found in the `reqpb`, `headers`, or `srvhdrs` pblocks. The contents of this pblock differ, depending upon the specific request and the type of SAF. ■ `reqpb`: Contains elements of the HTTP request, which includes the HTTP method (GET, POST, and so on), the URI, the protocol (generally HTTP/1.0), and the query string. This pblock doesn't usually change during the request-response process.`headers`: Contains all the request headers (such as `User-Agent`, `If-Modified-Since`, and so on) received from the client in the HTTP request. This pblock doesn't usually change during the request-response process. For more information about request headers, see the "Hypertext Transfer Protocol" chapter of the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*. ■ `srvhdrs`: Contains the response headers (such as `Server`, `Date`, `Content-type`, `Content-length`, and so on) that are to be sent to the client in the HTTP response. For more information about response headers, see the "Hypertext Transfer Protocol" chapter of the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*. **Note –** For more information about parameter blocks, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.
name	The variable to set.
value	The string assigned to the variable specified by *name*.

Variables

The following tables lists variables supported by the `set-variable` SAF.

TABLE 4–8 Supported Variables

Parameter	Description
`abort`	A value of `true` indicates the result code should be set to `REQ_ABORTED`. Setting the result code to `REQ_ABORTED` will abort the current request and send an error to the browser. For information about result codes, see the "Creating Custom SAFs" chapter of the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.
`error`	Sets the error code to be returned in the event of an aborted browser request.
`escape`	A boolean value signifying whether a URL should be escaped using `util_uri_escape`. For information about `util_uri_escape`, see the "NSAPI Function Reference" chapter of the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.
`find-pathinfo-forward`	Path information after the file name in a URI. See "find-pathinfo" on page 141.
`http-downgrade`	HTTP version number (for example, 1.0).
`http-upgrade`	HTTP version number (for example, 1.0).
`keep-alive`	A boolean value that establishes whether a keep-alive request from a browser will be honored.
`name`	Specifies an additional named object in the `obj.conf` file whose directives will be applied to this request. See also "assign-name" on page 126.
`noaction`	A value of `true` indicates the result code should be set to `REQ_NOACTION`. For `AuthTrans`, `NameTrans`, `Service`, and `Error` stage SAFs, setting the result code to `REQ_NOACTION` indicates that subsequent SAFs in that stage should be allowed to execute. For information about result codes, see the "Creating Custom SAFs" chapter of the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.
`nostat`	Causes the server *not* to perform the `stat()` function for a URL when possible. See also "assign-name" on page 126.
`senthdrs`	A boolean value that indicates whether HTTP response headers have been sent to the client.
`ssl-unclean-shutdown`	A boolean value that can be used to alter the way SSL3 connections are closed. As this violates the SSL3 RFCs, you should only use this with great caution if you know that you are experiencing problems with SSL3 shutdowns.

TABLE 4–8 Supported Variables *(Continued)*

Parameter	Description
`stop`	A value of `true` indicates the result code should be set to `REQ_PROCEED`. For `AuthTrans`, `NameTrans`, `Service`, and `Error` stage SAFs, setting the result code to `REQ_PROCEED` indicates that no further SAFs in that stage should be allowed to execute. For information about result codes, see the "Creating Custom SAFs" chapter of the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.
`url`	Redirect requests to a specified URL.

Examples

- To deny HTTP keep-alive requests for a specific server class (while still honoring keep-alive requests for the other classes), add this `AuthTrans` directive to the `obj.conf` for the server class, and set the variable `keep-alive` to `disabled`:

  ```
  AuthTrans fn="set-variable" keep-alive="disabled"
  ```

  To cause that same server class to use HTTP/1.0 while the rest of the server classes use HTTP/1.1, the `AuthTrans` directive would be:

  ```
  AuthTrans fn="set-variable" keep-alive="disabled" http-downgrade="true"
  ```

- To insert an HTTP header into each response, add a `NameTrans` directive to `obj.conf`, using the `insert-`*`pblock`* command and specifying `srvhdrs` as your Session/Request parameter block.

  For example, to insert the HTTP header `P3P`, you would add the following line to each request:

  ```
  NameTrans fn="set-variable" insert-srvhdrs="P3P"
  ```

- To terminate processing a request based upon certain URIs, use a `<Client>` tag to specify the URIs and an `AuthTrans` directive that sets the variable `abort` to `true` when there is a match. Your `<Client>` tag would be comparable to the following:

  ```
  <Client uri="*(system32|root.exe)*">AuthTrans fn="set-variable"
  abort="true"</Client>
  ```

See Also

"match-browser" on page 120

NameTrans

`NameTrans` stands for Name Translation. `NameTrans` directives translate virtual URLs to physical directories on your server. For example, the URL

```
http://www.test.com/some/file.html
```

could be translated to the full file system path

```
/usr/Sun/WebServer61/server1/docs/some/file.html
```

`NameTrans` directives should appear in the default object. If there is more than one `NameTrans` directive in an object, the server executes each one in order until one succeeds.

The following `NameTrans`-class functions are described in detail in this section:

- "assign-name" on page 126 tells the server to process directives in a named object.
- "document-root" on page 128 translates a URL into a file system path by replacing the `http://`*server-name*`/` part of the requested resource with the document root directory.
- "home-page" on page 129 translates a request for the server's root home page (`/`) to a specific file.
- "match-browser" on page 120 matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "ntrans-dav" on page 130 determines whether a request should be handled by the WebDAV subsystem and if so, creates a `dav` objectset.
- "ntrans-j2ee" on page 131 determines whether a request maps to a Java™ technology-based web application context.
- "pfx2dir" on page 131 translates any URL beginning with a given prefix to a file system directory and optionally enables directives in an additional named object.
- "redirect" on page 133 redirects the client to a different URL.
- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.
- "strip-params" on page 134 removes embedded semicolon-delimited parameters from the path.
- "unix-home" on page 135 translates a URL to a specified directory within a user's home directory.

assign-name

Applicable in `NameTrans`-class directives.

The `assign-name` function specifies the name of an object in `obj.conf` that matches the current request. The server then processes the directives in the named object in preference to the ones in the default object.

For example, consider the following directive in the default object:

```
NameTrans fn=assign-name name=personnel from=/personnel
```

Let's suppose the server receives a request for `http://`*server-name*`/personnel`. After processing this `NameTrans` directive, the server looks for an object named `personnel` in `obj.conf`, and continues by processing the directives in the `personnel` object.

The `assign-name` function always returns `REQ_NOACTION`.

Parameters

The following table describes parameters for the `assign-name` function.

TABLE 4–9 assign-name Parameters

Parameter	Description
`from`	Wildcard pattern that specifies the path to be affected.
`name`	Specifies an additional named object in `obj.conf` whose directives will be applied to this request.
`find-pathinfo-forward`	(Optional) Makes the server look for the `PATHINFO` forward in the path right after the `ntrans-base` instead of backward from the end of path as the server function `assign-name` does by default. The value you assign to this parameter is ignored. If you do not wish to use this parameter, leave it out. The `find-pathinfo-forward` parameter is ignored if the `ntrans-base` parameter is not set in `rq->vars`. By default, `ntrans-base` is set. This feature can improve performance for certain URLs by reducing the number of stats performed.

TABLE 4–9 assign-name Parameters *(Continued)*

Parameter	Description
`nostat`	(Optional) Prevents the server from performing a stat on a specified URL whenever possible. The effect of `nostat="`*virtual-path*`"` in the `NameTrans` function `assign-name` is that the server assumes that a stat on the specified *virtual-path* will fail. Therefore, use `nostat` only when the path of the *virtual-path* does not exist on the system, for example, for NSAPI plug-in URLs, to improve performance by avoiding unnecessary stats on those URLs. When the default `PathCheck` server functions are used, the server does not stat for the paths */ntrans-base/virtual-path* and */ntrans-base/virtual-path/** if *ntrans-base* is set (the default condition); it does not stat for the URLs */virtual-path* and */virtual-path/** if *ntrans-base* is not set.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
# This NameTrans directive is in the default object.
NameTrans fn=assign-name name=personnel from=/a/b/c/pers
...
<Object name=personnel>
...additional directives..
</Object>

NameTrans fn="assign-name" from="/perf" find-pathinfo-forward=""
name="perf"

NameTrans fn="assign-name" from="/nsfc"  nostat="/nsfc"
name="nsfc"
```

document-root

Applicable in `NameTrans`-class directives.

The `document-root` function specifies the root document directory for the server. If the physical path has not been set by a previous `NameTrans` function, the `http://`*server-name*`/` part of the path is replaced by the physical path name for the document root.

When the server receives a request for `http://`*server-name*`/somepath/somefile`, the `document-root` function replaces `http://`*server-name*`/` with the value of its `root` parameter. For example, if the document root directory is `/usr/sun/webserver61/server1/docs`, then when the server receives a request for `http://`*server-name*`/a/b/file.html`, the `document-root` function translates the path name for the requested resource to `/usr/sun/webserver61/server1/docs/a/b/file.html`.

This function always returns REQ_PROCEED. NameTrans directives listed after this will never be called, so be sure that the directive that invokes document-root is the last NameTrans directive.

There can be only one root document directory. To specify additional document directories, use the "pfx2dir" on page 131 function to set up additional path name translations.

Parameters

The following table describes parameters for the document-root function.

TABLE 4–10 document-root Parameters

Parameter	Description
root	File system path to the server's root document directory.
bucket	(Optional) Common to all obj.conf functions.

Examples

```
NameTrans fn=document-root root=/usr/sun/webserver61/server1/docs

NameTrans fn=document-root root=$docroot
```

See Also

"pfx2dir" on page 131

home-page

Applicable in NameTrans-class directives.

The home-page function specifies the home page for your server. Whenever a client requests the server's home page (/), they'll get the document specified.

Parameters

The following table describes parameters for the home-page function.

TABLE 4–11 home-page Parameters

Parameter	Description
`path`	Path and name of the home page file. If `path` starts with a slash (`/`), it is assumed to be a full path to a file. This function sets the server's `path` variable and returns `REQ_PROCEED`. If `path` is a relative path, it is appended to the URI and the function returns `REQ_NOACTION` continuing on to the other `NameTrans` directives.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
NameTrans fn="home-page" path="/path/to/file.html"
NameTrans fn="home-page" path="/path/to/$id/file.html"
```

match-browser

Applicable in all stage directives. The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables. See "match-browser" on page 120.

ntrans-dav

Applicable in `NameTrans`-class directives.

The `ntrans-dav` function determines whether a request should be handled by the WebDAV subsystem and if so, adds a `dav` object to the pipeline.

Parameters

The following table describes parameters for the `ntrans-dav` function.

TABLE 4–12 ntrans-dav Parameters

Parameter	Description
`name`	Specifies an additional named object in `obj.conf` whose directives will be applied to this request.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
NameTrans fn="ntrans-dav" name="dav"
```

See Also

"service-dav" on page 187

ntrans-j2ee

Applicable in `NameTrans`-class directives.

The `ntrans-j2ee` function determines whether a request maps to a Java web application context.

Parameters

The following table describes parameters for the `ntrans-j2ee` function.

TABLE 4–13 ntrans-j2ee Parameters

Parameter	Description
`name`	Named object in `obj.conf` whose directives are applied to requests made to Java web applications.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
NameTrans fn="ntrans-j2ee" name="j2ee"
```

See Also

"service-j2ee" on page 189, "error-j2ee" on page 199

pfx2dir

Applicable in `NameTrans`-class directives.

The `pfx2dir` function replaces a directory prefix in the requested URL with a real directory name. It also optionally allows you to specify the name of an object that matches the current request. (See the discussion of "assign-name" on page 126 for details of using named objects.)

Parameters

The following table describes parameters for the `pfx2dir` function.

TABLE 4–14 pfx2dir Parameters

Parameter	Description
`from`	URI prefix to convert. It should not have a trailing slash (/).
`dir`	Local file system directory path that the prefix is converted to. It should not have a trailing slash (/).
`name`	(Optional) Specifies an additional named object in `obj.conf` whose directives will be applied to this request.
`find-pathinfo-forward`	(Optional) Makes the server look for the `PATHINFO` forward in the path right after the `ntrans-base` instead of backward from the end of path as the server function `find-pathinfo` does by default. The value you assign to this parameter is ignored. If you do not wish to use this parameter, leave it out. The `find-pathinfo-forward` parameter is ignored if the `ntrans-base` parameter is not set in `rq->vars` when the server function `find-pathinfo` is called. By default, `ntrans-base` is set. This feature can improve performance for certain URLs by reducing the number of stats performed in the server function `find-pathinfo`. On Windows, this feature can also be used to prevent the `PATHINFO` from the server URL normalization process (changing '\' to '/') when the `PathCheck` server function `find-pathinfo` is used. Some double-byte characters have hexadecimal values that may be parsed as URL separator characters such as \ or ~. Using the `find-pathinfo-forward` parameter can sometimes prevent incorrect parsing of URLs containing double-byte characters.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

In the first example, the URL `http://`*server-name*`/cgi-bin/`*resource* (such as `http://x.y.z/cgi-bin/test.cgi`) is translated to the physical path name `/httpd/cgi-local/`*resource* (such as `/httpd/cgi-local/test.cgi`), and the server also starts processing the directives in the object named `cgi`.

```
NameTrans fn=pfx2dir from=/cgi-bin dir=/httpd/cgi-local name=cgi
```

In the second example, the URL `http://`*server-name*`/icons/`*resource* (such as `http://x.y.z/icons/happy/smiley.gif`) is translated to the physical path name `/users/nikki/images/`*resource* (such as `/users/nikki/images/smiley.gif`).

```
NameTrans fn=pfx2dir from=/icons/happy dir=/users/nikki/images
```

The third example shows the use of the `find-pathinfo-forward` parameter. The URL `http://`*server-name*`/cgi-bin/`*resource* is translated to the physical path name `/export/home/cgi-bin/`*resource*.

```
NameTrans fn="pfx2dir" find-pathinfo-forward="" from="/cgi-bin"
dir="/export/home/cgi-bin" name="cgi"
```

redirect

Applicable in `NameTrans`-class directives.

The `redirect` function lets you change URLs and send the updated URL to the client. When a client accesses your server with an old path, the server treats the request as a request for the new URL.

Parameters

The following table describes parameters for the `redirect` function.

TABLE 4–15 redirect Parameters

Parameter	Description
`from`	Specifies the prefix of the requested URI to match.
`url`	(Maybe optional) Specifies a complete URL to return to the client. If you use this parameter, don't use `url-prefix` (and vice versa).
`url-prefix`	(Maybe optional) The new URL prefix to return to the client. The `from` prefix is simply replaced by this URL prefix. If you use this parameter, don't use `url` (and vice versa).
`escape`	(Optional) Flag that tells the server to `util_uri_escape` the URL before sending it. It should be `yes` or `no`. The default is `yes`. For more information about `util_uri_escape`, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

In the first example, any request for `http://`*server-name*`/`*whatever* is translated to a request for `http://tmpserver/`*whatever*.

```
NameTrans fn=redirect from=/ url-prefix=http://tmpserver
```

In the second example, any request for `http://`*server-name*`/toopopular/`*whatever* is translated to a request for `http://bigger/better/stronger/morepopular/`*whatever*.

```
NameTrans fn=redirect from=/toopopular url=http://bigger/better/stronger/morepopular
```

set-variable

Applicable in all stage directives. The `set-variable` SAF enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands. See "set-variable" on page 122.

strip-params

Applicable in `NameTrans`-class directives.

The `strip-params` function removes embedded semicolon-delimited parameters from the path. For example, a URI of `/dir1;param1/dir2` would become a path of `/dir1/dir2`. When used, the `strip-params` function should be the first `NameTrans` directive listed.

Parameters

The following table describes parameters for the `strip-params` function.

TABLE 4–16 strip-params Parameters

Parameter	Description
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
NameTrans fn=strip-params
```

unix-home

Applicable in `NameTrans`-class directives.

UNIX Only. The `unix-home` function translates user names (typically of the form ~username) into the user's home directory on the server's UNIX machine. You specify a URL prefix that signals user directories. Any request that begins with the prefix is translated to the user's home directory.

You specify the list of users with either the `/etc/passwd` file or a file with a similar structure. Each line in the file should have this structure (elements in the `passwd` file that are not needed are indicated with *):

```
username:*:*:groupid:*:homedir:*
```

If you want the server to scan the password file only once at startup, use the `Init`-class function `init-uhome` in `magnus.conf`.

Parameters

The following table describes parameters for the `unix-home` function.

TABLE 4–17 unix-home Parameters

Parameter	Description
`subdir`	Subdirectory within the user's home directory that contains their web documents.
`pwfile`	(Optional) Full path and file name of the password file if it is different from `/etc/passwd`.
`name`	(Optional) Specifies an additional named object whose directives will be applied to this request.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
NameTrans fn=unix-home from=/~ subdir=public_html
NameTrans fn=unix-home from /~ pwfile=/mydir/passwd subdir=public_html
```

See Also

"find-links" on page 140

PathCheck

`PathCheck` directives check the local file system path that is returned after the `NameTrans` step. The path is checked for things such as CGI path information and for dangerous elements such as `/./`and `/../` and `//`, and then any access restriction is applied.

If there is more than one `PathCheck` directive, each of the functions is executed in order.

The following `PathCheck`-class functions are described in detail in this section:

- "check-acl" on page 136 checks an access control list for authorization.
- "deny-existence" on page 139 indicates that a resource was not found.
- "find-index" on page 139 locates a default file when a directory is requested.
- "find-links" on page 140 denies access to directories with certain file system links.
- "find-pathinfo" on page 141 locates extra path info beyond the file name for the `PATH_INFO` CGI environment variable.
- "get-client-cert" on page 142 gets the authenticated client certificate from the SSL3 session.
- "load-config" on page 143 finds and loads extra configuration information from a file in the requested path.
- "match-browser" on page 120 matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "nt-uri-clean" on page 145 denies access to requests with unsafe path names by indicating not found.
- "ntcgicheck" on page 146 looks for a CGI file with a specified extension.
- "pcheck-dav" on page 147 inserts a DAV-specific service function.
- "require-auth" on page 147 denies access to unauthorized users or groups.
- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.
- "set-virtual-index" on page 148 specifies a virtual index for a directory.
- "ssl-check" on page 149 checks the secret keysize.
- "ssl-logout" on page 150 invalidates the current SSL session in the server's SSL session cache.
- "unix-uri-clean" on page 150 denies access to requests with unsafe path names by indicating not found.

check-acl

Applicable in `PathCheck`-class directives.

The `check-acl` function specifies an access control list (ACL) to use to check whether the client is allowed to access the requested resource. An access control list contains information about who is or is not allowed to access a resource, and under what conditions access is allowed.

Regardless of the order of `PathCheck` directives in the object, `check-acl` functions are executed first. They cause user authentication to be performed, if required by the specified ACL, and will also update the access control state.

Parameters

The following table describes parameters for the `check-acl` function.

TABLE 4–18 check-acl Parameters

Parameter	Description
`acl`	Name of an access control list.
`path`	(Optional) Wildcard pattern that specifies the path for which to apply the ACL.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
PathCheck fn=check-acl acl="*HRonly*"
```

find-compressed

Applicable in `PathCheck`-class directives.

The `find-compressed` function checks if a compressed version of the requested file is available. If the following conditions are met, `find-compressed` changes the `path` to point to the compressed file:

- A compressed version is available.
- The compressed version is at least as recent as the noncompressed version.
- The client supports compression.

  Not all clients support compression. The `find-compressed` function allows you to use a single URL for both the compressed and noncompressed versions of a file. The version of the file that is selected is based on the individual clients' capabilities.

  A compressed version of a file must have the same file name as the noncompressed version but with a `.gz` suffix. For example, the compressed version of a file named `/httpd/docs/index.html` would be named `/httpd/docs/index.html.gz`. To compress files, you can use the freely available gzip program.

Because compressed files are sent as is to the client, you should not compress files such as SHTML pages, CGI programs, or pages created with JavaServer Pages™ (JSP™) technology that need to be interpreted by the server. To compress the dynamic content generated by these types of files, use the `http-compression` filter.

The `find-compressed` function does nothing if the HTTP method is not `GET` or `HEAD`.

Parameters

The following table describes parameters for the `find-compressed` function.

TABLE 4–19 find-compressed Parameters

Parameter	Description
`check-age`	Specifies whether to check if the compressed version is older than the noncompressed version. Possible values are `yes` and `no`. ■ If set to `yes`, the compressed version will not be selected if it is older than the noncompressed version. ■ If set to `no`, the compressed version will always be selected, even if it is older than the noncompressed version. By default, the value is set to `yes`.
`vary`	Specifies whether to insert a `Vary: Accept-Encoding` header. Possible values are `yes` or `no`. ■ If set to `yes`, a `Vary: Accept-Encoding` header is always inserted when a compressed version of a file is selected. ■ If set to `no`, a `Vary: Accept-Encoding` header is never inserted. By default, the value is set to `yes`.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
<Object name="default">
NameTrans fn="assign-name" from="*.html" name="find-compressed"
...
</Object>
<Object name="find-compressed">
PathCheck fn="find-compressed"
</Object>
```

See Also

`http-compression`

deny-existence

Applicable in `PathCheck`-class directives.

The `deny-existence` function sends a "not found" message when a client tries to access a specified path. The server sends "not found" instead of "forbidden," so the user cannot tell if the path exists.

Parameters

The following table describes parameters for the `deny-existence` function.

TABLE 4–20 deny-existence Parameters

Parameter	Description
`path`	(Optional) Wildcard pattern of the file system path to hide. If the path does not match, the function does nothing and returns `REQ_NOACTION`. If the path is not provided, it is assumed to match.
`bong-file`	(Optional) Specifies a file to send rather than responding with the "not found" message. It is a full file system path.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
PathCheck fn=deny-existence path=/usr/sun/server61/docs/private
PathCheck fn=deny-existence bong-file=/svr/msg/go-away.html
```

find-index

Applicable in `PathCheck`-class directives.

The `find-index` function investigates whether the requested path is a directory. If it is, the function searches for an index file in the directory, and then changes the path to point to the index file. If no index file is found, the server generates a directory listing.

Note that if the file `obj.conf` has a `NameTrans` directive that calls "home-page" on page 129, and the requested directory is the root directory, then the home page rather than the index page is returned to the client.

The `find-index` function does nothing if there is a query string, if the HTTP method is not `GET`, or if the path is that of a valid file.

Parameters

The following table describes parameters for the `find-index` function.

TABLE 4–21 find-index Parameters

Parameter	Description
`index-names`	Comma-separated list of index file names to look for. Use spaces only if they are part of a file name. Do not include spaces before or after the commas. This list is case-sensitive if the file system is case-sensitive.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
PathCheck fn=find-index index-names=index.html,home.html
```

find-links

Applicable in `PathCheck`-class directives.

UNIX Only. The `find-links` function searches the current path for symbolic or hard links to other directories or file systems. If any are found, an error is returned. This function is normally used for directories that are not trusted (such as user home directories). It prevents someone from pointing to information that should not be made public.

Parameters

The following table describes parameters for the `find-links` function.

TABLE 4–22 find-links Parameters

Parameter	Description
`disable`	Character string of links to disable: ■ `h` is hard links ■ `s` is soft links ■ `o` allows symbolic links from user home directories only if the user owns the target of the link

TABLE 4–22 find-links Parameters *(Continued)*

Parameter	Description
`dir`	Directory to begin checking. If you specify an absolute path, any request to that path and its subdirectories is checked for symbolic links. If you specify a partial path, any request containing that partial path is checked for symbolic links. For example, if you use `/user/` and a request comes in for `some/user/directory`, then that directory is checked for symbolic links.
`checkFileExistence`	Checks linked file for existence and aborts request with `403 (forbidden)` if this check fails.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
PathCheck fn=find-links disable=sh dir=/foreign-dir
PathCheck fn=find-links disable=so dir=public_html
```

See Also

"unix-home" on page 135

find-pathinfo

Applicable in `PathCheck`-class directives.

The `find-pathinfo` function finds any extra path information after the file name in the URL and stores it for use in the CGI environment variable `PATH_INFO`.

Parameters

The following table describes parameters for the `find-pathinfo` function.

TABLE 4–23 find-pathinfo Parameters

Parameter	Description
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
PathCheck fn=find-pathinfo
PathCheck fn=find-pathinfo find-pathinfo-forward=""
```

get-client-cert

Applicable in `PathCheck`-class directives.

The `get-client-cert` function gets the authenticated client certificate from the SSL3 session. It can apply to all HTTP methods, or only to those that match a specified pattern. It only works when SSL is enabled on the server.

If the certificate is present or obtained from the SSL3 session, the function returns `REQ_NOACTION`, allowing the request to proceed; otherwise, it returns `REQ_ABORTED` and sets the protocol status to `403 FORBIDDEN`, causing the request to fail and the client to be given the `FORBIDDEN` status.

Parameters

The following table describes parameters for the `get-client-cert` function.

TABLE 4–24 get-client-cert Parameters

Parameter	Description
`dorequest`	Controls whether to actually try to get the certificate, or just test for its presence. If `dorequest` is absent, the default value is `0`. ■ `1` tells the function to redo the SSL3 handshake to get a client certificate, if the server does not already have the client certificate. This typically causes the client to present a dialog box to the user to select a client certificate. The server may already have the client certificate if it was requested on the initial handshake, or if a cached SSL session has been resumed. ■ `0` tells the function not to redo the SSL3 handshake if the server does not already have the client certificate. If a certificate is obtained from the client and verified successfully by the server, the ASCII base64 encoding of the DER-encoded X.509 certificate is placed in the parameter `auth-cert` in the `Request->vars` pblock, and the function returns `REQ_PROCEED`, allowing the request to proceed.
`require`	Controls whether failure to get a client certificate will abort the HTTP request. If `require` is absent, the default value is `1`. ■ `1` tells the function to abort the HTTP request if the client certificate is not present after `dorequest` is handled. In this case, the HTTP status is set to `PROTOCOL_FORBIDDEN`, and the function returns `REQ_ABORTED`. ■ `0` tells the function to return `REQ_NOACTION` if the client certificate is not present after `dorequest` is handled.

TABLE 4–24 get-client-cert Parameters *(Continued)*

Parameter	Description
`method`	(Optional) Specifies a wildcard pattern for the HTTP methods for which the function will be applied. If `method` is absent, the function is applied to all requests.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
# Get the client certificate from the session.
# If a certificate is not already associated with the
# session, request one.
# The request fails if the client does not present a
# valid certificate.PathCheck fn="get-client-cert" dorequest="1"
```

load-config

Applicable in `PathCheck`-class directives.

The `load-config` function searches for configuration files in document directories and adds the file's contents to the server's existing configuration. These configuration files (also known as dynamic configuration files) specify additional access control information for the requested resource. Depending on the rules in the dynamic configuration files, the server may or may not allow the client to access the requested resource.

Each directive that invokes `load-config` is associated with a base directory, which is either stated explicitly through the `basedir` parameter or derived from the root directory for the requested resource. The base directory determines two things:

- The topmost directory for which requests will invoke this call to the `load-config` function.

  For example, if the base directory is `D:/sun/server61/docs/nikki/`, then only requests for resources in this directory or its subdirectories (and their subdirectories) trigger the search for dynamic configuration files. A request for the resource `D:/sun/server61/docs/somefile.html` does not trigger the search in this case, since the requested resource is in a parent directory of the base directory.
- The topmost directory in which the server looks for dynamic configuration files to apply to the requested resource.

  If the base directory is `D:/sun/server61/docs/nikki/`, the server starts its search for dynamic configuration files in this directory. It may or may not also search subdirectories (but never parent directories), depending on other factors.

  When you enable dynamic configuration files through the Server Manager interface, the system writes additional objects with `ppath` parameters into the `obj.conf` file. If you manually add directives that invoke `load-config` to the default object (rather than putting them in separate objects), the Server Manager interface might not reflect your changes.

If you manually add PathCheck directives that invoke load-config to the file obj.conf, put them in additional objects (created with the <OBJECT> tag) rather than putting them in the default object. Use the ppath attribute of the OBJECT tag to specify the partial path name for the resources to be affected by the access rules in the dynamic configuration file. The partial path name can be any path name that matches a pattern, which can include wildcard characters.

For example, the following <OBJECT> tag specifies that requests for resources in the directory D:/sun/server61/docs are subject to the access rules in the file my.nsconfig.

```
<Object ppath="D:/sun/server61/docs/*">
PathCheck fn="load-config" file="my.nsconfig" descend=1
basedir="D:/sun/server61/docs"
</Object>
```

Note – If the ppath resolves to a resource or directory that is higher in the directory tree (or is in a different branch of the tree) than the base directory, the load-config function is not invoked. This is because the base directory specifies the highest-level directory for which requests will invoke the load-config function.

The load-config function returns REQ_PROCEED if configuration files were loaded, REQ_ABORTED on error, or REQ_NOACTION when no files are loaded.

Parameters

The following table describes parameters for the load-config function.

TABLE 4–25 load-config Parameters

Parameter	Description
file	(Optional) Name of the dynamic configuration file containing the access rules to be applied to the requested resource. If not provided, the file name is assumed to be .nsconfig.
disable-types	(Optional) Specifies a wildcard pattern of types to disable for the base directory, such as magnus-internal/cgi. Requests for resources matching these types are aborted.
descend	(Optional) If present, specifies that the server should search in subdirectories of this directory for dynamic configuration files. For example, descend=1 specifies that the server should search subdirectories. No descend parameter specifies that the function should search only the base directory.

TABLE 4–25 load-config Parameters *(Continued)*

Parameter	Description
`basedir`	(Optional) Specifies base directory. This is the highest-level directory for which requests will invoke the `load-config` function, and is also the directory where the server starts searching for configuration files. If `basedir` is not specified, the base directory is assumed to be the root directory that results from translating the requested resource's URL to a physical path name. For example, if the request is for `http://`*server-name*`/a/b/file.html`, the physical file name would be */document-root*`/a/b/file.html`.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

In this example, whenever the server receives a request for any resource containing the substring `secret` that resides in `D:/Sun/WebServer61/server1/docs/nikki/` or a subdirectory thereof, it searches for a configuration file called `checkaccess.nsconfig`.

The server starts the search in the directory `D:/Sun/WebServer61/server1/docs/nikki`, and searches subdirectories too. It loads each instance of `checkaccess.nsconfig` that it finds, applying the access control rules contained therein to determine whether the client is allowed to access the requested resource.

```
<Object ppath="*secret*">
PathCheck fn="load-config" file="checkaccess.nsconfig"
basedir="D:/Sun/WebServer61/server1/docs/nikki" descend="1"
</Object>
```

match-browser

Applicable in all stage directives. The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables. See "match-browser" on page 120.

nt-uri-clean

Applicable in `PathCheck`-class directives.

Windows Only. The `nt-uri-clean` function denies access to any resource whose physical path contains `\.\`, `\..\` or `\` (these are potential security problems).

Parameters

The following table describes parameters for the `nt-uri-clean` function.

TABLE 4–26 nt-uri-clean Parameters

Parameter	Description
`tildeok`	If present, allows tilde (~) characters in URIs. This is a potential security risk on the Windows platform, where `longfi~1.htm` might reference `longfilename.htm` but does not go through the proper ACL checking. If present, "`//`" sequences are allowed.
`dotdirok`	If present, "`//`" sequences are allowed.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
PathCheck fn=nt-uri-clean
```

See Also

"unix-uri-clean" on page 150

ntcgicheck

Applicable in `PathCheck`-class directives.

Windows Only. The `ntcgicheck` function specifies the file name extension to be added to any file name that does not have an extension, or to be substituted for any file name that has the extension `.cgi`.

Parameters

The following table describes parameters for the `ntcgicheck` function.

TABLE 4–27 ntcgicheck Parameters

Parameter	Description
`extension`	The replacement file extension.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
PathCheck fn=ntcgicheck extension=pl
```

See Also

"send-cgi" on page 180, "send-wincgi" on page 186, "send-shellcgi" on page 185

pcheck-dav

Applicable in `PathCheck`-class directives.

The `pcheck-dav` function inserts a DAV-specific service function as the first service function if the `Translate:f` header is present, DAV is enabled for the request uri, and a corresponding source uri for the request uri exists. During the `Service` stage, this inserted service function restarts the request if necessary; otherwise, `REQ_NOACTION` is returned.

Parameters

The following table describes parameters for the `pcheck-dav` function.

TABLE 4–28 pcheck-dav Parameters

Parameter	Description
`bucket`	(Optional) Common to all `obj.conf` functions.

See Also

"ntrans-dav" on page 130, "service-dav" on page 187

require-auth

Applicable in `PathCheck`-class directives.

The `require-auth` function allows access to resources only if the user or group is authorized. Before this function is called, an authorization function (such as `basic-auth`) must be called in an `AuthTrans` directive.

If a user was authorized in an `AuthTrans` directive, and the `auth-user` parameter is provided, then the user's name must match the `auth-user` wildcard value. Also, if the `auth-group` parameter is provided, the authorized user must belong to an authorized group, which must match the `auth-user` wildcard value.

Parameters

The following table describes parameters for the `require-auth` function.

TABLE 4–29 require-auth Parameters

Parameter	Description
`path`	(Optional) Wildcard local file system path on which this function should operate. If no path is provided, the function applies to all paths.
`auth-type`	Type of HTTP authorization used, and must match the `auth-type` from the previous authorization function in `AuthTrans`. Currently, `basic` is the only authorization type defined.
`realm`	String sent to the browser indicating the secure area (or realm) for which a user name and password are requested.
`auth-user`	(Optional) Specifies a wildcard list of users who are allowed access. If this parameter is not provided, any user authorized by the authorization function is allowed access.
`auth-group`	(Optional) Specifies a wildcard list of groups that are allowed access.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
PathCheck fn=require-auth auth-type=basic realm="Marketing Plans"
auth-group=mktg auth-user=(jdoe|johnd|janed)
```

See Also

"basic-auth" on page 117, "basic-ncsa" on page 118

set-variable

Applicable in all stage directives. The `set-variable` SAF enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands. See "set-variable" on page 122.

set-virtual-index

Applicable in `PathCheck`-class directives.

The `set-virtual-index` function specifies a virtual index for a directory, which determines the URL forwarding. The index can refer to a LiveWire application, a servlet in its own namespace, a Sun™ Java System Application Server applogic, and so on.

REQ_NOACTION is returned if none of the URIs listed in the from parameter match the current URI. REQ_ABORTED is returned if the file specified by the virtual-index parameter is missing, or if the current URI cannot be found. REQ_RESTART is returned if the current URI matches any one of the URIs mentioned in the from parameter, or if there is no from parameter.

Parameters

The following table describes parameters for the set-virtual-index function.

TABLE 4–30 set-virtual-index Parameters

Parameter	Description
virtual-index	URI of the content generator that acts as an index for the URI the user enters.
from	(Optional) Comma-separated list of URIs for which this virtual-index is applicable. If from is not specified, the virtual-index always applies.
bucket	(Optional) Common to all obj.conf functions.

Example

```
# MyLWApp is a LiveWire applicationPathCheck fn=set-virtual-index virtual-index=MyLWApp
```

ssl-check

Applicable in PathCheck-class directives.

If a restriction is selected that is not consistent with the current cipher settings under Security Preferences, this function opens a popup dialog warning that ciphers with larger secret keysizes need to be enabled. This function is designed to be used together with a Client tag to limit access of certain directories to nonexportable browsers.

The function returns REQ_NOACTION if SSL is not enabled, or if the secret-keysize parameter is not specified. If the secret keysize for the current session is less than the specified secret-keysize and the bong-file parameter is not specified, the function returns REQ_ABORTED with a status of PROTOCOL_FORBIDDEN. If the bong file is specified, the function returns REQ_PROCEED, and the path variable is set to the bong file name. Also, when a keysize restriction is not met, the SSL session cache entry for the current session is invalidated, so that a full SSL handshake will occur the next time the same client connects to the server.

Requests that use ssl-check are not cacheable in the accelerator file cache if ssl-check returns something other than REQ_NOACTION.

Parameters

The following table describes parameters for the `ssl-check` function.

TABLE 4–31 ssl-check Parameters

Parameter	Description
`secret-keysize`	(Optional) Minimum number of bits required in the secret key.
`bong-file`	(Optional) Name of a file (not a URI) to be served if the restriction is not met.
`bucket`	(Optional) Common to all `obj.conf` functions.

ssl-logout

Applicable in `PathCheck`-class directives.

The `ssl-logout` function invalidates the current SSL session in the server's SSL session cache. This does not affect the current request, but the next time the client connects, a new SSL session will be created. If SSL is enabled, this function returns `REQ_PROCEED` after invalidating the session cache entry. If SSL is not enabled, it returns `REQ_NOACTION`.

Parameters

The following table describes parameters for the `ssl-logout` function.

TABLE 4–32 ssl-logout Parameters

Parameter	Description
`bucket`	(Optional) Common to all `obj.conf` functions.

unix-uri-clean

Applicable in `PathCheck`-class directives.

UNIX Only. The `unix-uri-clean` function denies access to any resource whose physical path contains `/./`, `/../` or `//` (these are potential security problems).

Parameters

The following table describes parameters for the `unix-uri-clean` function.

TABLE 4–33 unix-uri-clean Parameters

Parameter	Description
`dotdirok`	If present, "`//`" sequences are allowed.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
PathCheck fn=unix-uri-clean
```

See Also

"nt-uri-clean" on page 145

ObjectType

`ObjectType` directives determine the MIME type of the file to send to the client in response to a request. MIME attributes currently sent are `type`, `encoding`, and `language`. The MIME type is sent to the client as the value of the `Content-Type` header.

`ObjectType` directives also set the `type` parameter, which is used by `Service` directives to determine how to process the request according to what kind of content is being requested.

If there is more than one `ObjectType` directive in an object, all of the directives are applied in the order they appear. If a directive sets an attribute and later directives try to set that attribute to something else, the first setting is used and the subsequent ones are ignored.

The `obj.conf` file almost always has an `ObjectType` directive that calls the "type-by-extension" on page 156 function. This function instructs the server to look in a particular file (the MIME types file) to deduce the content type from the extension of the requested resource.

The following `ObjectType`-class functions are described in detail in this section:

- "force-type" on page 152 sets the `Content-Type` header for the response to a specific type.
- "match-browser" on page 120 matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "set-default-type" on page 153 allows you to define a default `charset`, `content-encoding`, and `content-language` for the response being sent back to the client.

- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.
- "shtml-hacktype" on page 154 requests that `.htm` and `.html` files are parsed for server-parsed HTML commands.
- "type-by-exp" on page 155 sets the `Content-Type` header for the response based on the requested path.
- "type-by-extension" on page 156 sets the `Content-Type` header for the response based on the file's extension and the MIME types database.

force-type

Applicable in `ObjectType`-class directives.

The `force-type` function assigns a type to requests that do not already have a MIME type. This is used to specify a default object type.

Make sure that the directive that calls this function comes last in the list of `ObjectType` directives, so that all other `ObjectType` directives have a chance to set the MIME type first. If there is more than one `ObjectType` directive in an object, all of the directives are applied in the order they appear. If a directive sets an attribute and later directives try to set that attribute to something else, the first setting is used and the subsequent ones are ignored.

Parameters

The following table describes parameters for the `force-type` function.

TABLE 4–34 force-type Parameters

Parameter	Description
`type`	(Optional) Type assigned to a matching request (the `Content-Type` header).
`enc`	(Optional) Encoding assigned to a matching request (the `Content-Encoding` header).
`lang`	(Optional) Language assigned to a matching request (the `Content-Language` header).

TABLE 4–34 force-type Parameters *(Continued)*

Parameter	Description
`charset`	(Optional) Character set for the `magnus-charset` parameter in `rq->srvhdrs`. If the browser sent the `Accept-Charset` header or its `User-Agent` is Mozilla™/1.1 or newer, then append "`; charset=`*charset*" to `content-type`, where *charset* is the value of the `magnus-charset` parameter in `rq->srvhdrs`.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
ObjectType fn=force-type type=text/plain
ObjectType fn=force-type lang=en_US
```

See Also

"type-by-extension" on page 156, "type-by-exp" on page 155

match-browser

Applicable in all stage directives. The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables. See "match-browser" on page 120.

set-default-type

Applicable in `ObjectType`-class directives.

The `set-default-type` function allows you to define a default `charset`, `content-encoding`, and `content-language` for the response being sent back to the client.

If the `charset`, `content-encoding`, and `content-language` have not been set for a response, then just before the headers are sent the defaults defined by `set-default-type` are used. Note that by placing this function in different objects in `obj.conf`, you can define different defaults for different parts of the document tree.

Parameters

The following table describes parameters for the `set-default-type` function.

TABLE 4–35 set-default-type Parameters

Parameter	Description
`enc`	(Optional) Encoding assigned to a matching request (the `Content-Encoding` header).
`lang`	(Optional) Language assigned to a matching request (the `Content-Language` header).
`charset`	(Optional) Character set for the `magnus-charset` parameter in `rq->srvhdrs`. If the browser sent the `Accept-Charset` header or its `User-agent` is Mozilla/1.1 or newer, then append "`; charset=`*charset*" to `content-type`, where *charset* is the value of the `magnus-charset` parameter in `rq->srvhdrs`.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
ObjectType fn="set-default-type" charset="iso_8859-1"
```

set-variable

Applicable in all stage directives. The `set-variable` SAF enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands. See "set-variable" on page 122.

shtml-hacktype

Applicable in `ObjectType`-class directives.

The `shtml-hacktype` function changes the `Content-Type` of any `.htm` or `.html` file to `magnus-internal/parsed-html` and returns `REQ_PROCEED`. This provides backward compatibility with server-side includes for files with `.htm` or `.html` extensions. The function may also check the execute bit for the file on UNIX systems. The use of this function is not recommended.

Parameters

The following table describes parameters for the `shtml-hacktype` function.

TABLE 4–36 shtml-hacktype Parameters

Parameter	Description
`exec-hack`	(UNIX only, optional) Tells the function to change the `content-type` only if the execute bit is enabled. The value of the parameter is not important; it need only be provided. You may use `exec-hack=true`.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
ObjectType fn=shtml-hacktype exec-hack=true
```

type-by-exp

Applicable in `ObjectType`-class directives.

The `type-by-exp` function matches the current path with a wildcard expression. If the two match, the `type` parameter information is applied to the file. This is the same as "type-by-extension" on page 156, except you use wildcard patterns for the files or directories specified in the URLs.

Parameters

The following table describes parameters for the `type-by-exp` function.

TABLE 4–37 type-by-exp Parameters

Parameter	Description
`exp`	Wildcard pattern of paths for which this function is applied.
`type`	(Optional) Type assigned to a matching request (the `Content-Type` header).
`enc`	(Optional) Encoding assigned to a matching request (the `Content-Encoding` header).
`lang`	(Optional) Language assigned to a matching request (the `Content-Language` header).

TABLE 4–37 type-by-exp Parameters *(Continued)*

Parameter	Description
`charset`	(Optional) is the character set for the `magnus-charset` parameter in `rq->srvhdrs`. If the browser sent the `Accept-Charset` header or its `User-Agent` is Mozilla/1.1 or newer, then append "`; charset=`*charset*" to `content-type`, where *charset* is the value of the `magnus-charset` parameter in `rq->srvhdrs`.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
ObjectType fn=type-by-exp exp=*.test type=application/html
```

See Also

"type-by-extension" on page 156, "force-type" on page 152

type-by-extension

Applicable in `ObjectType`-class directives.

The `type-by-extension` function instructs the server to look in a table of MIME type mappings to find the MIME type of the requested resource according to the extension of the requested resource. The MIME type is added to the `Content-Type` header sent back to the client.

The table of MIME type mappings is created by a `MIME` element in the `server.xml` file, which loads a MIME types file or list and creates the mappings. For more information about `server.xml` and MIME types files, see the *Sun Java System Web Server 6.1 SP9 Administrator's Configuration File Reference*.

For example, the following two lines are part of a MIME types file:

```
type=text/html    exts=htm,htmltype=text/plain    exts=txt
```

If the extension of the requested resource is `htm` or `html`, the `type-by-extension` file sets the type to `text/html`. If the extension is `.txt`, the function sets the type to `text/plain`.

Parameters

The following table describes parameters for the `type-by-extension` function.

TABLE 4–38 type-by-extension Parameters

Parameter	Description
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
ObjectType fn=type-by-extension
```

See Also

"type-by-exp" on page 155, "force-type" on page 152

Input

All `Input` directives are executed when the server or a plug-in first attempts to read entity body data from the client.

The `Input` stage allows you to select filters that will process incoming request data read by the `Service` step.

NSAPI filters in Sun Java System Web Server 6.1 enable a function to intercept (and potentially modify) the content presented to or generated by another function.

You can add NSAPI filters that process incoming data by invoking the `insert-filter` SAF in the `Input` stage of the request-handling process. The `Input` directives are executed at most once per request.

You can also define the appropriate position of a specific filter within the filter stack. For example, filters that translate content from XML to HTML are placed higher in the filter stack than filters that compress data for transmission. You can use the `filter_create` function to define the filter's position in the filter stack, and the `init-filter-order` to override the defined position.

When two or more filters are defined to occupy the same position in the filter stack, filters that were inserted later will appear higher than filters that were inserted earlier. That is, the order of `Input fn="insert-filter"` and `Output fn="insert-filter"` directives in `obj.conf` becomes important.

The following `Input`-class functions are described in detail in this section:

- "insert-filter" on page 158 adds a filter to the filter stack to process incoming data.
- "match-browser" on page 120 matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "remove-filter" on page 159 removes a filter from the filter stack.
- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.

insert-filter

Applicable in `Input`-class directives.

The `insert-filter` SAF is used to add a filter to the filter stack to process incoming (client-to-server) data.

The order of `Input fn="insert-filter"` and `Output fn="insert-filter"` directives are important.

Returns

Returns `REQ_PROCEED` if the specified filter was inserted successfully or `REQ_NOACTION` if the specified filter was not inserted because it was not required. Any other return value indicates an error.

Parameters

The following table describes parameters for the `insert-filter` function.

TABLE 4–39 insert-filter Parameters

Parameter	Description
`filter`	Specifies the name of the filter to insert.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Input fn="insert-filter" filter="http-decompression"
```

match-browser

Applicable in all stage directives. The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables. See "match-browser" on page 120.

remove-filter

Applicable in `Input-`, `Output-`, `Service-`, and `Error`-class directives.

The `remove-filter` SAF is used to remove a filter from the filter stack. If the filter has been inserted multiple times then only the topmost instance is removed. In general, it is not necessary to remove filters with `remove-filter`, as they will be removed automatically at the end of the request.

Returns

Returns `REQ_PROCEED` if the specified filter was removed successfully, or `REQ_NOACTION` if the specified filter was not part of the filter stack. Any other return value indicates an error.

Parameters

The following table describes parameters for the `remove-filter` function.

TABLE 4–40 remove-filter Parameters

Parameter	Description
`filter`	Specifies the name of the filter to remove.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Input fn="remove-filter" filter="http-compression"
```

set-variable

Applicable in all stage directives. The `set-variable` SAF enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands. See "set-variable" on page 122.

Output

All `Output` directives are executed when the server or a plug-in first attempts to write entity body data from the client.

The `Output` stage allows you to select filters that will process outgoing data.

You can add NSAPI filters that process outcoming data by invoking the `insert-filter` SAF in the `Output` stage of the request-handling process. The `Output` directives are executed at most once per request.

You can define the position of a filter within the filter stack. For example, filters that translate content from XML to HTML are placed higher in the filter stack than filters that compress data for transmission. You can use the `filter_create` function to define the filter's position in the filter stack and the `init-filter-order` to override the defined position.

When two or more filters are defined to occupy the same position in the filter stack, filters that were inserted later will appear higher than filters that were inserted earlier.

The following `Output`-class functions are described in detail in this section:

- "insert-filter" on page 160 adds a filter to the filter stack to process outgoing data.
- "match-browser" on page 120 matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "remove-filter" on page 161 removes a filter from the filter stack.
- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.

insert-filter

Applicable in `Output`-class directives.

The `insert-filter` SAF is used to add a filter to the filter stack to process outgoing (server-to-client) data.

The order of `Input fn="insert-filter"` and `Output fn="insert-filter"` directives can be important.

Returns

Returns `REQ_PROCEED` if the specified filter was inserted successfully, or `REQ_NOACTION` if the specified filter was not inserted because it was not required. Any other return value indicates an error.

Parameters

The following table describes parameters for the `insert-filter` function.

TABLE 4–41 insert-filter Parameters

Parameter	Description
`filter`	Specifies the name of the filter to insert.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Output fn="insert-filter" filter="http-compression"
```

match-browser

Applicable in all stage directives. The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables. See "match-browser" on page 120.

remove-filter

Applicable in `Input-`, `Output-`, `Service-`, and `Error`-class directives.

The `remove-filter` SAF is used to remove a filter from the filter stack. If the filter has been inserted multiple times, only the topmost instance is removed. In general, it is not necessary to remove filters with `remove-filter`, as they will be removed automatically at the end of the request.

Returns

Returns `REQ_PROCEED` if the specified filter was removed successfully or `REQ_NOACTION` if the specified filter was not part of the filter stack. Any other return value indicates an error.

Parameters

The following table describes parameters for the `remove-filter` function.

TABLE 4–42 remove-filter Parameters

Parameter	Description
`filter`	Specifies the name of the filter to remove.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Output fn="remove-filter" filter="http-compression"
```

set-variable

Applicable in all stage directives. The `set-variable` SAF enables you to change server settings based upon conditional information in a request and to manipulate variables in parameter blocks by using specific commands. See "set-variable" on page 122.

Service

The `Service`-class of functions sends the response data to the client.

Every `Service` directive has the following optional parameters to determine whether the function is executed. All optional parameters must match the current request for the function to be executed.

`type`

(Optional) Specifies a wildcard pattern of MIME types for which this function will be executed. The `magnus-internal/*` MIME types are used only to select a `Service` function to execute.

`method`

(Optional) Specifies a wildcard pattern of HTTP methods for which this function will be executed. Common HTTP methods are `GET`, `HEAD`, and `POST`.

`query`

(Optional) Specifies a wildcard pattern of query strings for which this function will be executed.

UseOutputStreamSize

(Optional) Determines the default output stream buffer size, in bytes, for data sent to the client. If this parameter is not specified, the default is `8192` bytes.

Note – The `UseOutputStreamSize` parameter can be set to zero (`0`) in the `obj.conf` file to disable output stream buffering. For the `magnus.conf` file, setting `UseOutputStreamSize` to zero (`0`) has no effect.

---

flushTimer

(Optional) Determines the maximum number of milliseconds between write operations in which buffering is enabled. If the interval between subsequent write operations is greater than the `flushTimer` value for an application, further buffering is disabled. This is necessary for status-monitoring CGI applications that run continuously and generate periodic status update reports. If this parameter is not specified, the default is `3000` milliseconds.

ChunkedRequestBufferSize

(Optional) Determines the default buffer size, in bytes, for "un-chunking" request data. If this parameter is not specified, the default is `8192` bytes.

ChunkedRequestTimeout

(Optional) Determines the default timeout, in seconds, for "un-chunking" request data. If this parameter is not specified, the default is `60` seconds.

If there is more than one `Service`-class function, the first one matching the optional wildcard parameters (`type`, `method`, or `query`) is executed.

For more information about the `UseOutputStreamSize`, `flushTimer`, `ChunkedRequestBufferSize`, and `ChunkedRequestTimeout` parameters, see *Buffered Streams* in the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*. The `UseOutputStreamSize`, `ChunkedRequestBufferSize`, and `ChunkedRequestTimeout` parameters also have equivalent `magnus.conf` directives. For more information, see "Chunked Encoding" in the chapter "Syntax and Use of magnus.conf" in the *Sun Java System Web Server 6.1 SP9 Administrator's Configuration File Reference*. The `obj.conf` parameters override the `magnus.conf` directives.

By default, the server sends the requested file to the client by calling the "send-file" on page 183 function. The directive that sets the default is:

```
Service method="(GET|HEAD)" type="*~magnus-internal/*" fn="send-file"
```

This directive usually comes last in the set of `Service`-class directives to give all other `Service` directives a chance to be invoked. This directive is invoked if the method of the request is `GET`, `HEAD`, or `POST`, and the type does *not* start with `magnus-internal/`. Note here that the pattern `*~` means "does not match." For a list of characters that can be used in patterns, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

The following `Service`-class functions are described in detail in this section:

- "add-footer" on page 165 appends a footer specified by a file name or URL to an HTML file.
- "add-header" on page 166 prepends a header specified by a file name or URL to an HTML file.
- "append-trailer" on page 167 appends text to the end of an HTML file.
- "imagemap" on page 169 handles server-side image maps.
- "index-common" on page 169 generates a fancy list of the files and directories in a requested directory.
- "index-simple" on page 171 generates a simple list of files and directories in a requested directory.
- "key-toosmall" on page 172 indicates to the client that the provided certificate key size is too small to accept.
- "list-dir" on page 173 lists the contents of a directory.
- "make-dir" on page 175 creates a directory.
- "match-browser" on page 120 matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "query-handler" on page 176 handles the HTML ISINDEX tag.
- "remove-dir" on page 176 deletes an empty directory.
- "delete-file" on page 177 deletes a file.
- "remove-filter" on page 178 removes a refilter from the filter stack.
- "rename-file" on page 179 renames a file.
- "send-cgi" on page 180 sets up environment variables, launches a CGI program, and sends the response to the client.
- "send-error" on page 182 sends an HTML file to the client in place of a specific HTTP response status.
- "send-file" on page 183 sends a local file to the client.
- "send-range" on page 185 sends a range of bytes of a file to the client.
- "send-shellcgi" on page 185 sets up environment variables, launches a shell CGI program, and sends the response to the client.

- "send-wincgi" on page 186 sets up environment variables, launches a WinCGI program, and sends the response to the client.
- "service-dav" on page 187 services static content and restarts the request with the `sourceuri` for dynamic content.
- "service-dump" on page 188 creates a performance report based on collected performance bucket data.
- "service-j2ee" on page 189 services requests made to Java web applications.
- "service-trace" on page 190 services TRACE requests.
- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.
- "shtml_send" on page 191 parses an HTML file for server-parsed HTML commands.
- "stats-xml" on page 192 creates a performance report in XML format.
- "upload-file" on page 194 uploads and saves a file.

add-footer

Applicable in `Service`-class directives.

This function appends a footer to an HTML file that is sent to the client. The footer is specified either as a file name or a URI, thus the footer can be dynamically generated. To specify static text as a footer, use the "append-trailer" on page 167 function.

Parameters

The following table describes parameters for the `add-footer` function.

TABLE 4–43 add-footer Parameters

Parameter	Description
`file`	(Optional) Path name to the file containing the footer. Specify either `file` or `uri`. By default, the path name is relative. If the path name is absolute, pass the `NSIntAbsFilePath` parameter as `yes`.
`uri`	(Optional) URI pointing to the resource containing the footer. Specify either `file` or `uri`.
`NSIntAbsFilePath`	(Optional) If the file parameter is specified, the `NSIntAbsFilePath` parameter determines whether the file name is absolute or relative. The default is relative. Set the value to `yes` to indicate an absolute file path.

TABLE 4–43 add-footer Parameters *(Continued)*

Parameter	Description
"type" on page 162	(Optional) Common to all Service-class functions.
"method" on page 162	(Optional) Common to all Service-class functions.
"query" on page 162	(Optional) Common to all Service-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all Service-class functions.
"flushTimer" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all Service-class functions.
bucket	(Optional) Common to all obj.conf functions.

Examples

```
Service type=text/html method=GET fn=add-footer
file="footers/footer1.html"

Service type=text/html method=GET fn=add-footer
file="D:/Sun/WebServer61/server1/footers/footer1.html"
NSIntAbsFilePath="yes"
```

See Also

"append-trailer" on page 167, "add-header" on page 166

add-header

Applicable in Service-class directives.

This function prepends a header to an HTML file that is sent to the client. The header is specified either as a file name or a URI, thus the header can be dynamically generated.

Parameters

The following table describes parameters for the add-header function.

TABLE 4–44 add-header Parameters

Parameter	Description
`file`	(Optional) Path name to the file containing the header. Specify either `file` or `uri`. By default, the path name is relative. If the path name is absolute, pass the `NSIntAbsFilePath` parameter as `yes`.
`uri`	(Optional) URI pointing to the resource containing the header. Specify either `file` or `uri`.
`NSIntAbsFilePath`	(Optional) If the file parameter is specified, the `NSIntAbsFilePath` parameter determines whether the file name is absolute or relative. The default is relative. Set the value to `yes` to indicate an absolute file path.
"`type`" on page 162	(Optional) Common to all `Service`-class functions.
"`method`" on page 162	(Optional) Common to all `Service`-class functions.
"`query`" on page 162	(Optional) Common to all `Service`-class functions.
"`UseOutputStreamSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`flushTimer`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestBufferSize`" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
Service type=text/html method=GET fn=add-header
file="headers/header1.html"

Service type=text/html method=GET fn=add-footer
file="D:/Sun/WebServer61/server1/headers/header1.html"
NSIntAbsFilePath="yes"
```

See Also

"add-footer" on page 165, "append-trailer" on page 167

append-trailer

Applicable in `Service`-class directives.

The `append-trailer` function sends an HTML file and appends text to the end. It only appends text to HTML files. This is typically used for author information and copyright text. The date the file was last modified can be inserted.

Returns `REQ_ABORTED` if a required parameter is missing, if there is extra path information after the file name in the URL, or if the file cannot be opened for read-only access.

Parameters

The following table describes parameters for the `append-trailer` function.

TABLE 4–45 append-trailer Parameters

Parameter	Description
`trailer`	Text to append to HTML documents. The string is unescaped with `util_uri_unescape` before being sent. The text can contain HTML tags, and can be up to 512 characters long after unescaping and inserting the date. If you use the string `:LASTMOD:`, which is replaced by the date the file was last modified, you must also specify a time format with `timefmt`.
`timefmt`	(Optional) Time format string for `:LASTMOD:`. If `timefmt` is not provided, `:LASTMOD:` will not be replaced with the time.
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
Service type=text/html method=GET fn=append-trailer
 trailer="<hr><img src=/logo.gif> Copyright 1999"

# Add a trailer with the date in the format: MM/DD/YY
Service type=text/html method=GET fn=append-trailer timefmt="%D"
trailer="<HR>File last updated on: :LASTMOD:"
```

See Also

"add-footer" on page 165, "add-header" on page 166

imagemap

Applicable in `Service`-class directives.

The `imagemap` function responds to requests for imagemaps. Imagemaps are images that are divided into multiple areas that each have an associated URL. The information about which URL is associated with which area is stored in a mapping file.

Parameters

The following table describes parameters for the `imagemap` function.

TABLE 4–46 imagemap Parameters

Parameter	Description
"`type`" on page 162	(Optional) Common to all `Service`-class functions.
"`method`" on page 162	(Optional) Common to all `Service`-class functions.
"`query`" on page 162	(Optional) Common to all `Service`-class functions.
"`UseOutputStreamSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`flushTimer`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestBufferSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestTimeout`" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service type=magnus-internal/imagemap method=(GET|HEAD) fn=imagemap
```

index-common

Applicable in `Service`-class directives.

The index-common function generates a fancy (or common) list of files in the requested directory. The list is sorted alphabetically. Files beginning with a period (.) are not displayed. Each item appears as an HTML link. This function displays more information than "index-simple" on page 171, including the size, date last modified, and an icon for each file. It may also include a header and/or readme file into the listing.

The Init-class function cindex-init in magnus.conf specifies the format for the index list, including where to look for the images.

If obj.conf contains a call to index-common in the Service stage, magnus.conf must initialize fancy (or common) indexing by invoking cindex-init during the Init stage.

Indexing occurs when the requested resource is a directory that does not contain an index file or a home page, or no index file or home page has been specified by the functions "find-index" on page 139 or "home-page" on page 129.

The icons displayed are .gif files dependent on the content-type of the file, as listed in the following table:

TABLE 4–47 content-type Icons

Content-type	Icon
"text/*"	text.gif
"image/*"	image.gif
"audio/*"	sound.gif
"video/*"	movie.gif
"application/octet-stream"	binary.gif
directory	menu.gif
all others	unknown.gif

Parameters

The following table describes parameters for the index-common function.

TABLE 4–48 index-common Parameters

Parameter	Description
`header`	(Optional) Path (relative to the directory being indexed) and name of a file (HTML or plain text) that is included at the beginning of the directory listing to introduce the contents of the directory. The file is first tried with `.html` added to the end. If found, it is incorporated near the top of the directory list as HTML. If the file is not found, it is tried without the `.html` and incorporated as preformatted plain text (bracketed by `<PRE>` and).
`readme`	(Optional) Path (relative to the directory being indexed) and name of a file (HTML or plain text) to append to the directory listing. This file might give more information about the contents of the directory, indicate copyrights, authors, or other information. The file is first tried with `.html` added to the end. If found, it is incorporated at the bottom of the directory list as HTML. If the file is not found, it is tried without the `.html` and incorporated as preformatted plain text (enclosed by `<PRE>` and `</PRE>`).
"`type`" on page 162	(Optional) Common to all `Service`-class functions.
"`method`" on page 162	(Optional) Common to all `Service`-class functions.
"`query`" on page 162	(Optional) Common to all `Service`-class functions.
"`UseOutputStreamSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`flushTimer`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestBufferSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestTimeout`" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service fn=index-common type=magnus-internal/directory
method=(GET|HEAD) header=hdr readme=rdme.txt
```

See Also

"index-simple" on page 171, "find-index" on page 139, "home-page" on page 129

index-simple

Applicable in `Service`-class directives.

The `index-simple` function generates a simple index of the files in the requested directory. This function scans a directory and returns an HTML page to the browser displaying a bulleted list of the files and directories in the directory. The list is sorted alphabetically. Files beginning with a period (.) are not displayed. Each item appears as an HTML link.

Indexing occurs when the requested resource is a directory that does not contain either an index file or a home page, or no index file or home page has been specified by the functions "find-index" on page 139 or "home-page" on page 129.

Parameters

The following table describes parameters for the `index-simple` function.

TABLE 4–49 index-simple Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service type=magnus-internal/directory fn=index-simple
```

See Also

"index-common" on page 169

key-toosmall

Applicable in `Service`-class directives.

Note – This function is provided for backward compatibility only and was deprecated in Sun Java System Web Server 4.*x*. It is replaced by the `PathCheck`-class SAF "ssl-check" on page 149.

The `key-toosmall` function returns a message to the client specifying that the secret key size for SSL communications is too small. This function is designed to be used together with a `Client` tag to limit access of certain directories to nonexportable browsers.

Parameters

The following table describes parameters for the `key-toosmall` function.

TABLE 4–50 key-toosmall Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
<Object ppath=/mydocs/secret/*>Service fn=key-toosmall</Object>
```

list-dir

Applicable in `Service`-class directives.

The `list-dir` function returns a sequence of text lines to the client in response to a request whose method is `INDEX`. The format of the returned lines is:

name type size mimetype

The *name* field is the name of the file or directory. It is relative to the directory being indexed. It is URL-encoded, that is, any character might be represented by `%xx`, where `xx` is the hexadecimal representation of the character's ASCII number.

The *type* field is a MIME type such as `text/html`. Directories will be of type `directory`. A file for which the server doesn't have a type will be of type `unknown`.

The *size* field is the size of the file, in bytes.

The *mtime* field is the numerical representation of the date of last modification of the file. The number is the number of seconds since the epoch (Jan 1, 1970 00:00 UTC) since the last modification of the file.

When remote file manipulation is enabled in the server, the `obj.conf` file contains a `Service`-class function that calls `list-dir` for requests whose method is `INDEX`.

Parameters

The following table describes parameters for the `list-dir` function.

TABLE 4–51 list-dir Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service fn=list-dir method="INDEX"
```

make-dir

Applicable in `Service`-class directives.

The `make-dir` function creates a directory when the client sends a request whose method is `MKDIR`. The function can fail if the server can't write to that directory.

When remote file manipulation is enabled in the server, the `obj.conf` file contains a `Service`-class function that invokes `make-dir` when the request method is `MKDIR`.

Parameters

The following table describes parameters for the `make-dir` function.

TABLE 4–52 make-dir Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service fn="make-dir" method="MKDIR"
```

match-browser

Applicable in all stage directives. The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables. See "match-browser" on page 120.

query-handler

Applicable in `Service`- and `Error`-class directives.

Note – This function is provided for backward compatibility only and is used mainly to support the obsolete `ISINDEX` tag. If possible, use an HTML form instead.

The `query-handler` function runs a CGI program instead of referencing the path requested.

Parameters

The following table describes parameters for the `query-handler` function.

TABLE 4–53 query-handler Parameters

Parameter	Description
`path`	Full path and file name of the CGI program to run.
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
Service query=* fn=query-handler path=/http/cgi/do-grep
Service query=* fn=query-handler path=/http/cgi/proc-info
```

remove-dir

Applicable in `Service`-class directives.

The remove-dir function removes a directory when the client sends a request whose method is RMDIR. The directory must be empty (have no files in it). The function will fail if the directory is not empty or if the server doesn't have the privileges to remove the directory.

When remote file manipulation is enabled in the server, the obj.conf file contains a Service-class function that invokes remove-dir when the request method is RMDIR.

Parameters

The following table describes parameters for the remove-dir function.

TABLE 4–54 remove-dir Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all Service-class functions.
"method" on page 162	(Optional) Common to all Service-class functions.
"query" on page 162	(Optional) Common to all Service-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all Service-class functions.
"flushTimer" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all Service-class functions.
bucket	(Optional) Common to all obj.conf functions.

Example

```
Service fn="remove-dir" method="RMDIR"
```

delete-file

Applicable in Service-class directives.

The delete-file function deletes a file when the client sends a request whose method is DELETE. It deletes the file indicated by the URL if the user is authorized and the server has the needed file system privileges.

When remote file manipulation is enabled in the server, the obj.conf file contains a Service-class function that invokes delete-file when the request method is DELETE.

Parameters

The following table describes parameters for the `delete-file` function.

TABLE 4–55 delete-file Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service fn="delete-file" method="DELETE"
```

remove-filter

Applicable in `Input`-, `Output`-, `Service`-, and `Error`-class directives.

The `remove-filter` SAF is used to remove a filter from the filter stack. If the filter has been inserted multiple times, only the topmost instance is removed. In general, it is not necessary to remove filters with `remove-filter`, as they will be removed automatically at the end of the request.

Returns

Returns `REQ_PROCEED` if the specified filter was removed successfully, or `REQ_NOACTION` if the specified filter was not part of the filter stack. Any other return value indicates an error.

Parameters

The following table describes parameters for the `remove-filter` function.

TABLE 4–56 remove-filter Parameters

Parameter	Description
`filter`	Specifies the name of the filter to remove.
"`type`" on page 162	(Optional) Common to all `Service`-class functions.
"`method`" on page 162	(Optional) Common to all `Service`-class functions.
"`query`" on page 162	(Optional) Common to all `Service`-class functions.
"`UseOutputStreamSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`flushTimer`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestBufferSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestTimeout`" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service fn="remove-filter" filter="http-compression"
```

rename-file

Applicable in `Service`-class directives.

The `rename-file` function renames a file when the client sends a request with a `New-URL` header whose method is `MOVE`. This file renames the file indicated by the URL to `New-URL` within the same directory if the user is authorized and the server has the needed file system privileges.

When remote file manipulation is enabled in the server, the `obj.conf` file contains a `Service`-class function that invokes `rename-file` when the request method is `MOVE`.

Parameters

The following table describes parameters for the `rename-file` function.

TABLE 4–57 rename-file Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all Service-class functions.
"method" on page 162	(Optional) Common to all Service-class functions.
"query" on page 162	(Optional) Common to all Service-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all Service-class functions.
"flushTimer" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all Service-class functions.
bucket	(Optional) Common to all obj.conf functions.

Example

```
Service fn="rename-file" method="MOVE"
```

send-cgi

Applicable in Service-class directives.

The send-cgi function sets up the CGI environment variables, runs a file as a CGI program in a new process, and sends the results to the client.

For more information about the CGI environment variables and their NSAPI equivalents, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

For more information about CGI, see the *Sun Java System Web Server 6.1 SP9 Administrator's Guide* and the *Sun Java System Web Server 6.1 SP9 Programmer's Guide*.

There are three ways to change the timing used to flush the CGI buffer:

- Adjust the interval between flushes using the "Service" on page 162 parameter.
- Adjust the buffer size using the "Service" on page 162 parameter.
- Force Sun Java System Web Server to flush its buffer by forcing spaces into the buffer in the CGI script.

Parameters

The following table describes parameters for the `send-cgi` function.

TABLE 4–58 send-cgi Parameters

Parameter	Description
`user`	(UNIX only) Specifies the name of the user to execute CGI programs as.
`group`	(UNIX only) Specifies the name of the group to execute CGI programs as.
`chroot`	(UNIX only) Specifies the directory to chroot to before execution begins.
`dir`	(UNIX only) Specifies the directory to chdir to after chroot, but before execution begins.
`rlimit_as`	(UNIX only) Specifies the maximum CGI program address space in bytes. You can supply both current (soft) and maximum (hard) limits, separated by a comma. The soft limit must be listed first. If only one limit is specified, both limits are set to this value.
`rlimit_core`	(UNIX only) Specifies the maximum CGI program core file size. A value of `0` disables writing cores. You can supply both current (soft) and maximum (hard) limits, separated by a comma. The soft limit must be listed first. If only one limit is specified, both limits are set to this value.
`rlimit_nofile`	(UNIX only) Specifies the maximum number of file descriptors for the CGI program. You can supply both current (soft) and maximum (hard) limits, separated by a comma. The soft limit must be listed first. If only one limit is specified, both limits are set to this value.
`nice`	(UNIX only) Accepts an increment that determines the CGI program's priority relative to the server. Typically, the server is running with a `nice` value of `0` and the `nice` increment would be between `0` (the CGI program runs at same priority as server) and `19` (the CGI program runs at much lower priority than server). While it is possible to increase the priority of the CGI program above that of the server by specifying a nice increment of `-1`, this is not recommended.
"`type`" on page 162	(Optional) Common to all `Service`-class functions.
"`method`" on page 162	(Optional) Common to all `Service`-class functions.
"`query`" on page 162	(Optional) Common to all `Service`-class functions.
"`UseOutputStreamSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`flushTimer`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestBufferSize`" on page 163	(Optional) Common to all `Service`-class functions.

TABLE 4–58 send-cgi Parameters *(Continued)*

Parameter	Description
"ChunkedRequestTimeout" on page 163	(Optional) Common to all Service-class functions.
bucket	(Optional) Common to all obj.conf functions.

Example

The following example uses variables defined in the server.xml file for the send-cgi parameters. For more information about defining variables, see the *Sun Java System Web Server 6.1 SP9 Administrator's Configuration File Reference*.

```
<Object name="default">
...
NameTrans fn="pfx2dir" from="/cgi-bin"
dir="/home/foo.com/public_html/cgi-bin" name="cgi"
...
</Object>

<Object name="cgi">
 ObjectType fn="force-type" type="magnus-internal/cgi"
 Service fn="send-cgi" user="$user" group="$group" dir="$dir"
chroot="$chroot" nice="$nice"
</Object>
```

send-error

Applicable in Service-class directives.

The send-error function sends an HTML file to the client in place of a specific HTTP response status. This allows the server to present a message describing the problem. The HTML page may contain images and links to the server's home page or other pages.

Parameters

The following table describes parameters for the send-error function.

TABLE 4–59 send-error Parameters

Parameter	Description
path	Specifies the full file system path of an HTML file to send to the client. The file is sent as text/html regardless of its name or actual type. If the file does not exist, the server sends a simple default error page.

TABLE 4–59 send-error Parameters *(Continued)*

Parameter	Description
"type" on page 162	(Optional) Common to all Service-class functions.
"method" on page 162	(Optional) Common to all Service-class functions.
"query" on page 162	(Optional) Common to all Service-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all Service-class functions.
"flushTimer" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all Service-class functions.
bucket	(Optional) Common to all obj.conf functions.

Example

```
Error fn=send-error code=401 path=/sun/server61/docs/errors/401.html
```

send-file

Applicable in Service-class directives.

The send-file function sends the contents of the requested file to the client. This function provides the Content-Type, Content-Length, and Last-Modified headers.

Most requests are handled by this function using the following directive (which usually comes last in the list of Service-class directives in the default object, so that it acts as a default):

```
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*" fn="send-file"
```

This directive is invoked if the method of the request is GET, HEAD, or POST, and the type does *not* start with magnus-internal/. Note that the pattern *~ means "does not match." For a list of characters that can be used in patterns, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

Parameters

The following table describes parameters for the send-file function.

TABLE 4–60 send-file Parameters

Parameter	Description
`nocache`	(Optional) Prevents the server from caching responses to static file requests. For example, you can specify that files in a particular directory are not to be cached, which is useful for directories where the files change frequently. The value you assign to this parameter is ignored. If you do not wish to use this parameter, leave it out.
"`type`" on page 162	(Optional) Common to all `Service`-class functions.
"`method`" on page 162	(Optional) Common to all `Service`-class functions.
"`query`" on page 162	(Optional) Common to all `Service`-class functions.
"`UseOutputStreamSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`flushTimer`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestBufferSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestTimeout`" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service type="*~magnus-internal/*" method="(GET|HEAD)" fn="send-file"
```

In the following example, the server does not cache static files from `/export/somedir/` when requested by the URL prefix `/myurl`.

```
<Object name=default>
...
NameTrans fn="pfx2dir" from="/myurl" dir="/export/mydir",
name="myname"
...
Service method=(GET|HEAD|POST) type=*~magnus-internal/*
fn=send-file
...
</Object>
<Object name="myname">
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
nocache=""
</Object>
```

send-range

Applicable in `Service`-class directives.

When the client requests a portion of a document by specifying HTTP byte ranges, the `send-range` function returns the specified portion.

Parameters

The following table describes parameters for the `send-range` function.

TABLE 4–61 send-range Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service fn=send-range
```

send-shellcgi

Applicable in `Service`-class directives.

Windows Only. The `send-shellcgi` function runs a file as a shell CGI program and sends the results to the client. Shell CGI is a server configuration that lets you run CGI applications using the file associations set in Windows. For information about shell CGI programs, see the *Sun Java System Web Server 6.1 SP9 Administrator's Guide*.

Parameters

The following table describes parameters for the `send-shellcgi` function.

TABLE 4–62 send-shellcgi Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions

Examples

```
Service fn=send-shellcgi
Service type=magnus-internal/cgi fn=send-shellcgi
```

send-wincgi

Applicable in `Service`-class directives.

Windows Only. The `send-wincgi` function runs a file as a Windows CGI program and sends the results to the client. For information about Windows CGI programs, see *Sun Java System Web Server 6.1 SP9 Administrator's Guide*.

Parameters

The following table describes parameters for the `send-wincgi` function.

TABLE 4–63 send-wincgi Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all Service-class functions.
"method" on page 162	(Optional) Common to all Service-class functions.
"query" on page 162	(Optional) Common to all Service-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all Service-class functions.
"flushTimer" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all Service-class functions.
bucket	(Optional) Common to all obj.conf functions.

Examples

```
Service fn=send-wincgi
Service type=magnus-internal/cgi fn=send-wincgi
```

service-dav

Applicable in Service-class directives.

The service-dav function services a request to a WebDAV-enabled URI. In response to a request for a WebDAV resource, the service-dav function services the static content and restarts the request with the sourceuri for dynamic content. The sourceuri is identified by the magnus-internal setting. If no sourceuri is defined for dynamic content, an HTTP error message is returned.

Requests to WebDAV resources are authenticated and authorized by the AuthTrans and PathCheck NSAPI stages, respectively. By default, all access to sourceuri are restricted by the PathCheck entry in the dav object.

OPTIONS on a WebDAV-enabled URI are always handled by the default object's service-dav directive. Therefore, the OPTIONS method is not included in the service-dav directive of the dav object.

In response to an OPTIONS request to a WebDAV-enabled URI (or `sourceuri`), the `service-dav` function in the default object adds the necessary DAV headers and returns control to the core server, which then services the request.

For more information on access control for WebDAV resources, see the *Sun Java System Web Server 6.1 SP9 Administrator's Guide*.

Parameters

The following table describes parameters for the `service-dav` function.

TABLE 4–64 service-dav Parameters

Parameter	Description
"`method`" on page 162	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
<Object name="default">
....
Service method="(OPTIONS|PUT|DELETE|COPY|MOVE|PROPFIND|PROPPATCH|LOCK|UN
LOCK|MKCOL)" fn="service-dav"
</Object>

<Object name="dav">
PathCheck fn="check-acl" acl="dav-src"
Service fn="service-dav"
method="(PUT|DELETE|COPY|MOVE|PROPFIND|PROPPATCH|LOCK|UNLOCK|MKC
OL)"
</Object>
```

See Also

"stats-xml" on page 192

service-dump

Applicable in `Service`-class directives.

The `service-dump` function creates a performance report based on collected performance bucket data (see "The bucket Parameter" on page 115

To read the report, point the browser here:

`http://`*server_id*`:`*port*`/.perf`

Parameters

The following table describes parameters for the `service-dump` function.

TABLE 4–65 service-dump Parameters

Parameter	Description
"type" on page 162	Must be `perf` for this function.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
<Object name=default>
NameTrans fn="assign-name" from="/.perf" name="perf"
...
</Object>

<Object name=perf>
Service fn="service-dump"
</Object>
```

See Also

"stats-xml" on page 192

service-j2ee

Applicable in `Service-class` directives.

The `service-j2ee` function services requests made to Java web applications.

Parameters

The following table describes parameters for the `service-j2ee` function.

TABLE 4–66 service-j2ee Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
<Object name=default>
NameTrans fn="ntrans-j2ee" name="j2ee"
...
</Object>

<Object name=j2ee>
Service fn="service-j2ee"
</Object>
```

See Also

"ntrans-j2ee" on page 131, "error-j2ee" on page 199

service-trace

Applicable in `Service`-class directives.

The `service-trace` function services TRACE requests. TRACE requests are used to diagnose problems with web proxy servers located between a web client and web server.

Parameters

The following table describes parameters for the `service_trace` function.

TABLE 4–67 service-trace Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all Service-class functions.
"method" on page 162	(Optional) Common to all Service-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all Service-class functions.
"flushTimer" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all Service-class functions.
"query" on page 162	(Optional) Common to all Service-class functions.
bucket	(Optional) Common to all obj.conf functions.

Example

```
<Object name="default">
...
Service method="TRACE" fn="service-trace"
...
</Object>
```

set-variable

Applicable in all stage directives. The set-variable SAF enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands. See "set-variable" on page 122.

shtml_send

Applicable in Service-class directives.

The shtml_send function parses an HTML document and scans for embedded commands. These commands may provide information from the server, include the contents of other files, or execute a CGI program. The shtml_send function is only available when the Shtml plug-in (libShtml.so on UNIX libShtml.dll on Windows) is loaded. See the *Sun Java System Web Server 6.1 SP9 Programmer's Guide* for server-parsed HTML commands.

Parameters

The following table describes parameters for the `shtml_send` function.

TABLE 4–68 shtml-send Parameters

Parameter	Description
`ShtmlMaxDepth`	Maximum depth of include nesting allowed. The default value is `10`.
`addCgiInitVars`	(UNIX only) If present and equal to `yes` (the default is `no`), adds the environment variables defined in the `init-cgi` SAF to the environment of any command executed through the SHTML `exec` tag.
"`type`" on page 162	(Optional) Common to all `Service`-class functions.
"`method`" on page 162	(Optional) Common to all `Service`-class functions.
	(Optional) Common to all `Service`-class functions.
"`UseOutputStreamSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`flushTimer`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestBufferSize`" on page 163	(Optional) Common to all `Service`-class functions.
"`ChunkedRequestTimeout`" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service type=magnus-internal/shtml_send method=(GET|HEAD) fn=shtml_send
```

stats-xml

Applicable in `Service`-class directives.

The `stats-xml` function creates a performance report in XML format. If performance buckets have been defined, this performance report includes them.

However, you do need to initialize this function using the `stats-init` function in `magnus.conf`, then use the `NameTrans` function to direct requests to the `stats-xml` function. See the examples below.

The report is generated here:

`http://server_id:port/stats-xml/iwsstats.xml`

The associated DTD file is here:

`http://server_id:port/stats-xml/iwsstats.dtd`

For more information about the format of the `iwsstats.xml` file, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*

Parameters

The following table describes parameters for the `stats-xml` function.

TABLE 4–69 stats-xml Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all Service-class functions.
"method" on page 162	(Optional) Common to all Service-class functions.
"query" on page 162	(Optional) Common to all Service-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all Service-class functions.
"flushTimer" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all Service-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all Service-class functions.
bucket	(Optional) Common to all obj.conf functions.

Examples

In `magnus.conf`:

```
Init fn="stats-init" update-interval="5" virtual-servers="2000" profiling="yes"
```

In `obj.conf`:

```
<Object name="default">
...
NameTrans fn="assign-name" from="/stats-xml/*" name="stats-xml"
...
</Object>
```

```
...
<Object name="stats-xml">
Service fn="stats-xml"
</Object>
```

See Also

"service-dump" on page 188

upload-file

Applicable in `Service`-class directives.

The `upload-file` function uploads and saves a new file when the client sends a request whose method is `PUT`, if the user is authorized and the server has the needed file system privileges.

When remote file manipulation is enabled in the server, the `obj.conf` file contains a `Service`-class function that invokes `upload-file` when the request method is `PUT`.

Parameters

The following table describes parameters for the `upload-file` function.

TABLE 4–70 upload-file Parameters

Parameter	Description
"type" on page 162	(Optional) Common to all `Service`-class functions.
"method" on page 162	(Optional) Common to all `Service`-class functions.
"query" on page 162	(Optional) Common to all `Service`-class functions.
"UseOutputStreamSize" on page 163	(Optional) Common to all `Service`-class functions.
"flushTimer" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestBufferSize" on page 163	(Optional) Common to all `Service`-class functions.
"ChunkedRequestTimeout" on page 163	(Optional) Common to all `Service`-class functions.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Service fn=upload-file
```

AddLog

After the server has responded to the request, the `AddLog` directives are executed to record information about the transaction.

If there is more than one `AddLog` directive, all are executed.

The following `AddLog`-class functions are described in detail in this section:

- "common-log" on page 195 records information about the request in the common log format.
- "flex-log" on page 196 records information about the request in a flexible, configurable format.
- "match-browser" on page 120 matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "record-useragent" on page 197 records the client's IP address and `User-Agent` header.
- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.

common-log

Applicable in `AddLog`-class directives.

The `common-log` function records request-specific data in the common log format (used by most HTTP servers). There is a log analyzer in the `/extras/log_anly` directory for Sun Java System Web Server.

The common log must have been initialized previously by using the `init-clf` function. For information about rotating logs, see `flex-rotate-init` in the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

There are also a number of free statistics generators for the common log format.

Parameters

The following table describes parameters for the `common-log` function.

TABLE 4–71 common-log Parameters

Parameter	Description
`name`	(Optional) Gives the name of a log file, which must have been given as a parameter to the `init-clf` function in `magnus.conf`. If no name is given, the entry is recorded in the global log file.
`iponly`	(Optional) Instructs the server to log the IP address of the remote client rather than looking up and logging the DNS name. This will improve performance if DNS is off in the `magnus.conf` file. The value of `iponly` has no significance, as long as it exists; you may use `iponly=1`.
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
# Log all accesses to the global log file
AddLog fn=common-log
# Log accesses from outside our subnet (198.93.5.*) to
# nonlocallog
<Client ip="*~198.93.5.*">
AddLog fn=common-log name=nonlocallog
</Client>
```

See Also

"record-useragent" on page 197, "flex-log" on page 196

flex-log

Applicable in `AddLog`-class directives.

The `flex-log` function records request-specific data in a flexible log format. It may also record requests in the common log format. There is a log analyzer in the `/extras/flexanlg` directory for Sun Java System Web Server.

There are also a number of free statistics generators for the common log format. The log format is specified by using the `flex-init` function call. For information about rotating logs, see `flex-rotate-init` in the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

Parameters

The following table describes parameters for the `flex-log` function.

TABLE 4–72 flex-log Parameters

Parameter	Description
`name`	(Optional) Gives the name of a log file, which must have been given as a parameter to the `flex-init` function in `magnus.conf`. If no name is given, the entry is recorded in the global log file.
`iponly`	(Optional) Instructs the server to log the IP address of the remote client rather than looking up and logging the DNS name. This will improve performance if DNS is off in the `magnus.conf` file. The value of `iponly` has no significance, as long as it exists; you may use `iponly=1`.
`bucket`	(Optional) Common to all `obj.conf` functions.
`buffers-per-file`	Specifies the number of buffers for a given log file. The default value is determined by the server. Access log entries can be logged in strict chronological order by using a single buffer per log file. To accomplish this, add buffers-per-file="1" to the Init fn="flex-init" line in magnus.conf. This ensures that requests are logged in chronological order. Note that this approach will result in decreased performance when the server is under heavy load.

Examples

```
# Log all accesses to the global log file
AddLog fn=flex-log
# Log accesses from outside our subnet (198.93.5.*) to
# nonlocallog
<Client ip="*~198.93.5.*">
AddLog fn=flex-log name=nonlocallog
</Client>
```

See Also

"common-log" on page 195, "record-useragent" on page 197

match-browser

Applicable in all stage directives. The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables. See "match-browser" on page 120.

record-useragent

Applicable in `AddLog`-class directives.

The record-useragent function records the IP address of the client, followed by its User-Agent HTTP header. This indicates what version of the client was used for this transaction.

Parameters

The following table describes parameters for the record-useragent function.

TABLE 4–73 record-useragent Parameters

Parameter	Description
name	(Optional) Gives the name of a log file, which must have been given as a parameter to the init-clf function in magnus.conf. If no name is given, the entry is recorded in the global log file.
bucket	(Optional) Common to all obj.conf functions.

Example

```
# Record the client ip address and user-agent to browserlog
AddLog fn=record-useragent name=browserlog
```

See Also

"common-log" on page 195, "flex-log" on page 196

set-variable

Applicable in all stage directives. The set-variable SAF enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands. See "set-variable" on page 122.

Error

If a Server Application Function results in an error, it sets the HTTP response status code and returns the value REQ_ABORTED. When this happens, the server stops processing the request. Instead, it searches for an Error directive matching the HTTP response status code or its associated reason phrase, and executes the directive's function. If the server does not find a matching Error directive, it returns the response status code to the client.

The following Error-class functions are described in detail in this section:

- "error-j2ee" on page 199 handles errors that occur during execution of Java™ 2 Platform, Standard Edition (J2SE platform) applications and modules deployed to the Sun Java System Web Server.

- "match-browser" on page 120 matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables.
- "qos-error" on page 200 returns an error page stating which quality of service limits caused the error and what the value of the QOS statistic was.
- "query-handler" on page 200 runs a CGI program instead of referencing the path requested.
- "remove-filter" on page 201 removes a filter from the filter stack.
- "send-error" on page 202 sends an HTML file to the client in place of a specific HTTP response status.
- "set-variable" on page 122 enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.

error-j2ee

Applicable in `Error`-class directives.

The `error-j2ee` function handles errors that occur during execution of web applications deployed to the Sun Java System Web Server individually or as part of full J2SE applications.file name.

Parameters

The following table describes parameters for the `error-j2ee` function.

TABLE 4–74 error-j2ee Parameters

Parameter	Description
`bucket`	(Optional) Common to all `obj.conf` functions.

See Also

"ntrans-j2ee" on page 131, "service-j2ee" on page 189

match-browser

Applicable in all stage directives. The `match-browser` SAF matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Server based upon the results by setting values for specified variables. See "match-browser" on page 120.

qos-error

Applicable in `Error`-class directives.

The `qos-error` function returns an error page stating which quality of service limits caused the error, and what the value of the QOS statistic was.

The code for this SAF is one of the examples in the Sun Java System Web Server 6.1 *NSAPI Programmer's Guide*.

For more information, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide*.

Parameters

The following table describes parameters for the `qos-error` function.

TABLE 4–75 qos-error Parameters

Parameter	Description
`code`	(Optional) Three-digit number representing the HTTP response status code, such as `401` or `407`. The recommended value is `503`. This can be any HTTP response status code or reason phrase according to the HTTP specification. The following is a list of common HTTP response status codes and reason strings: ■ `401 Unauthorized` ■ `403 Forbidden` ■ `404 Not Found` ■ `500 Server Error`
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Error fn=qos-error code=503
```

See Also

"qos-handler" on page 121

query-handler

Applicable in `Service`- and `Error`-class directives.

Note – This function is provided for backward compatibility only and is used mainly to support the obsolete `ISINDEX` tag. If possible, use an HTML form instead.

The `query-handler` function runs a CGI program instead of referencing the path requested.

Parameters

The following table describes parameters for the `query-handler` function.

TABLE 4–76 query-handler Parameters

Parameter	Description
`path`	Full path and file name of the CGI program to run.
`reason`	(Optional) Text of one of the reason strings (such as "Unauthorized" or "Forbidden"). The string is not case-sensitive.
`code`	(Optional) Three-digit number representing the HTTP response status code, such as `401` or `407`. This can be any HTTP response status code or reason phrase according to the HTTP specification. The following is a list of common HTTP response status codes and reason strings: ■ `401 Unauthorized` ■ `403 Forbidden` ■ `404 Not Found` ■ `500 Server Error`
`bucket`	(Optional) Common to all `obj.conf` functions.

Examples

```
Error query=* fn=query-handler path=/http/cgi/do-grep
Error query=* fn=query-handler path=/http/cgi/proc-info
```

remove-filter

Applicable in `Input-`, `Output-`, `Service-`, and `Error`-class directives.

The `remove-filter` SAF is used to remove a filter from the filter stack. If the filter has been inserted multiple times, only the topmost instance is removed. In general, it is not necessary to remove filters with `remove-filter`, as they will be removed automatically at the end of the request.

Returns

Returns `REQ_PROCEED` if the specified filter was removed successfully, or `REQ_NOACTION` if the specified filter was not part of the filter stack. Any other return value indicates an error.

Parameters

The following table describes parameters for the `remove-filter` function.

TABLE 4–77 remove-filter Parameters

Parameter	Description
`filter`	Specifies the name of the filter to remove.
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Error fn="remove-filter" filter="http-compression"
```

send-error

Applicable in `Error`-class directives.

The `send-error` function sends an HTML file to the client in place of a specific HTTP response status. This allows the server to present a friendly message describing the problem. The HTML page may contain images and links to the server's home page or other pages.

Parameters

The following table describes parameters for the `send-error` function.

TABLE 4–78 send-error Parameters

Parameter	Description
`path`	Specifies the full file system path of an HTML file to send to the client. The file is sent as `text/html` regardless of its name or actual type. If the file does not exist, the server sends a simple default error page.

TABLE 4–78 send-error Parameters *(Continued)*

Parameter	Description
`reason`	(Optional) Text of one of the reason strings (such as "Unauthorized" or "Forbidden"). The string is not case-sensitive.
`code`	(Optional) Three-digit number representing the HTTP response status code, such as `401` or `407`. This can be any HTTP response status code or reason phrase according to the HTTP specification. The following is a list of common HTTP response status codes and reason strings: ■ `401 Unauthorized` ■ `403 Forbidden` ■ `404 Not Found` ■ `500 Server Error`
`bucket`	(Optional) Common to all `obj.conf` functions.

Example

```
Error fn=send-error code=401 path=/sun/server61/docs/errors/401.html
```

set-variable

Applicable in all stage directives. The `set-variable` SAF sets the HTTP status code and aborts the request by returning REQ_ABORTED. To set the HTTP status code without aborting the request, use the error parameter in conjunction with the *noaction* parameter.

To rewrite a HTTP status code, use a <Client> tag to match the original status code and an Output directive to set the new status code. The following example would rewrite all *302 Moved Temporarily* responses to *301 Moved Permanently responses*

Example

```
<Client code="302">
Output fn="set-variable" error="301 Moved Permanently" noaction="true"
</Client>
```

◆ ◆ ◆ CHAPTER 5

MIME Types

This chapter discusses the MIME types file.

The sections are:

Introduction

The MIME types file in the `config` directory contains mappings between MIME (Multipurpose Internet Mail Extensions) types and file extensions. For example, the MIME types file maps the extensions `.html` and `.htm` to the type `text/html`:

```
type=text/html exts=htm,html
```

When the Sun Java System Web Server receives a request for a resource from a client, it uses the MIME type mappings to determine what kind of resource is being requested.

MIME types are defined by three attributes: language (`lang`), encoding (`enc`), and content-type (`type`). At least one of these attributes must be present for each type. The most commonly used attribute is `type`. The server frequently considers the `type` when deciding how to generate the response to the client. The `enc` and `lang` attributes are rarely used.

The default MIME types file is called `mime.types`.

Determining the MIME Type

During the `ObjectType` step in the request handling process, the server determines the MIME type attributes of the resource requested by the client. Several different server application functions (SAFs) can be used to determine the MIME type, but the most commonly used one is `type-by-extension`. This function tells the server to look up the MIME type according to the requested resource's file extension in the MIME types table.

The directive in `obj.conf` that tells the server to look up the MIME type according to the extension is:

```
ObjectType fn=type-by-extension
```

If the server uses a different SAF, such as `force-type`, to determine the `type`, then the MIME types table is not used for that particular request.

For more details of the ObjectType step, see the *Sun Java System Web Server 6.1 SP9 NSAPI Programmer's Guide*.

How the Type Affects the Response

The server considers the value of the `type` attribute when deciding which `Service` directive in `obj.conf` to use to generate the response to the client.

By default, if the `type` does not start with `magnus-internal/`, the server just sends the requested file to the client. The directive in `obj.conf` that contains this instruction is:

```
Service method=(GET|HEAD|POST) type=*~magnus-internal/* fn=send-file
```

By convention, all values of `type` that require the server to do something other than just send the requested resource to the client start with `magnus-internal/`.

For example, if the requested resource's file extension is `.map`, the type is mapped to `magnus-internal/imagemap`. If the extension is `.cgi, .exe`, or `.bat`, the type is set to `magnus-internal/cgi`:

```
type=magnus-internal/imagemap       exts=map
type=magnus-internal/cgi            exts=cgi,exe,bat
```

If the `type` starts with `magnus-internal/`, the server executes the `Service` directive in `obj.conf` matches the specified type. For example, if the type is `magnus-internal/imagemap`, the server uses the `imagemap` function to generate the response to the client, as indicated by the following directive:

```
Service method=(GET|HEAD) type=magnus-internal/imagemap fn=imagemap
```

What Does the Client Do with the MIME Type?

The `Service` function generates the data and sends it to the client that made the request. When the server sends the data to the client, it also sends headers. These headers include the MIME type attributes are known (which is usually `type`).

When the client receives the data, it uses the MIME type to decide what to do with the data. For browser clients, display the data in the browser window.

If the requested resource cannot be displayed in a browser but needs to be handled by another application, its `type` starts with `application/`, for example, `application/octet-stream` (for `.bin` file extensions) or `application/x-maker` (for `.fm` file extensions). The client has its own set of user-editable mappings that tells it which application to use to handle which types of data.

For example, if the type is `application/x-maker`, the client usually handles it by opening Adobe® FrameMaker® to display the file.

Syntax of the MIME Types File

The first line in the MIME types file identifies the file format and must read:

```
#--Sun Microsystems MIME Information
```

Other non-comment lines have the following format:

```
type=type/subtype exts=[file extensions]
```

- `type/subtype` is the type and subtype.
- `exts` are the file extensions associated with this type.

Sample MIME Types File

Here is an example of a MIME types file:

```
#--Sun Microsystems MIME Information
      # Do not delete the above line. It is used to identify the file type.
      type=application/octet-stream      exts=bin,exe
      type=application/oda               exts=oda
      type=application/pdf               exts=pdf
      type=application/postscript        exts=ai,eps,ps
      type=application/rtf               exts=rtf
      type=application/x-mif             exts=mif,fm
      type=application/x-gtar            exts=gtar
      type=application/x-shar            exts=shar
```

```
type=application/x-tar              exts=tar
type=application/mac-binhex40       exts=hqx

type=audio/basic                    exts=au,snd
type=audio/x-aiff                   exts=aif,aiff,aifc
type=audio/x-wav                    exts=wav

type=image/gif                      exts=gif
type=image/ief                      exts=ief
type=image/jpeg                     exts=jpeg,jpg,jpe
type=image/tiff                     exts=tiff,tif
type=image/x-rgb                    exts=rgb
type=image/x-xbitmap                exts=xbm
type=image/x-xpixmap                exts=xpm
type=image/x-xwindowdump            exts=xwd

type=text/html                      exts=htm,html
type=text/plain                     exts=txt
type=text/richtext                  exts=rtx
type=text/tab-separated-values      exts=tsv
type=text/x-setext                  exts=etx

type=video/mpeg                     exts=mpeg,mpg,mpe
type=video/quicktime                exts=qt,mov
type=video/x-msvideo                exts=avi

enc=x-gzip                          exts=gz
enc=x-compress                      exts=z
enc=x-uuencode                      exts=uu,uue

type=magnus-internal/imagemap       exts=map
type=magnus-internal/parsed-html    exts=shtml
type=magnus-internal/cgi            exts=cgi,exe,bat
type=magnus-internal/jsp            exts=jsp
```

◆ ◆ ◆ CHAPTER 6

Other Server Configuration Files

This chapter summarizes the important configuration files not discussed in other chapters. Configuration files that should never be modified are not listed in this module.

The following configuration files are described in alphabetical order:

certmap.conf

Purpose

Configures how a certificate, designated by *name*, is mapped to an LDAP entry, designated by *issuerDN*.

Location

server_root/bin/https/install/misc
server_root/userdb

Syntax

```
certmap name issuerDNname:property1 [value1]
name:property2 [value2]
...
```

The default certificate is named `default`, and the default *issuerDN* is also named `default`. Therefore, the first certmap defined in the file must be as follows:

```
certmap default default
```

You can use # at the beginning of a line to indicate a comment.

See Also

Sun Java System Web Server 6.1 SP9 Administrator's Guide

The following table describes properties in the `certmap.conf` file. The left column lists the property names. The second column from the left lists allowed values. The third column from the left lists default values. The right column lists property descriptions.

TABLE 6–1 certmap.conf Properties

Attribute	Allowed Values	Default Value	Description
`DNComps`	See Description	Commented out	Used to form the base DN for performing an LDAP search while mapping the certificate to a user entry. Values are as follows: ■ Commented out: takes the user's DN from the certificate as is. ■ Empty: searches the entire LDAP tree (DN == suffix). ■ Comma-separated attributes: forms the DN.
`FilterComps`	See Description	Commented out	Used to form the filter for performing an LDAP search while mapping the certificate to a user entry. Values are as follows: ■ Commented out or empty: sets the filter to "`objectclass=*`". ■ Comma-separated attributes: forms the filter.
`verifycert`	`on` or `off`	`off` (commented out)	Specifies whether certificates are verified.

TABLE 6–1 certmap.conf Properties *(Continued)*

Attribute	Allowed Values	Default Value	Description
`CmapLdapAttr`	LDAP attribute name	certSubjectDN (commented out)	Specifies the name of the attribute in the LDAP database that contains the DN of the certificate.
`library`	Path to shared lib or dll	None	Specifies the library path for custom certificate mapping code.
`InitFn`	Name of initialization function	None	Specifies the initialization function in the certificate mapping code referenced by `library`.

dbswitch.conf

Purpose

Specifies the LDAP directory that Sun Java System Web Server uses.

Location

server_root`/userdb`

Syntax

`directory` *name LDAP_URLname*`:`*property1* `[`*value1*`]`
name`:`*property2* `[`*value2*`]`
`...`

The default contents of this file are as follows:

```
directory default null:///none
```

Edit the file as follows for anonymous binding over SSL:

```
directory default ldaps://directory.sun.com:636:/dc%3Dcom
```

Edit the file as follows for anonymous binding *not* over SSL:

```
directory default ldap://directory.sun.com:389:/dc%3Dcom
```

See Also

"User Database Selection" on page 66

The following table describes properties in the `dbswitch.conf` file. The Property column lists the property names. The Allowed Values column from the left lists allowed values. The Default Value column from the left lists default values. The Description column lists property descriptions.

TABLE 6–2 dbswitch.conf Properties

Property	Allowed Values	Default Value	Description
`nsessions`	A positive integer	`8`	The number of LDAP connections for the database.
`dyngroups`	`off`, `on`, `recursive`	`on`	Determines how dynamic groups are handled. If `off`, dynamic groups are not supported. If `on`, dynamic groups are supported. If `recursive`, dynamic groups can contain other groups.
`binddn`	A valid DN		The DN used for connecting to the database. If both `binddn` and `bindpw` are not present, binding is anonymous.
`bindpw`			The password used for connecting to the database. If both `binddn` and `bindpw` are not present, binding is anonymous.
`dcsuffix`	A valid DN (relative to the LDAP URL)	none	If present, the default value of the base DN for the request's virtual server is determined by a dc tree search of the connection group's `servername` attribute, starting at the `dcsuffix` DN. If not present, the default value of the base DN is the `base DN` value in the LDAP URL. The `basedn` attribute of a `USERDB` element in the `server.xml` file overrides this value.
`digestauth`	`off`, `on`	`off`	Specifies whether the database can perform digest authentication. If `on`, a special Directory Server plug-in is required. For information about how to install this plug-in, see the *Sun Java System Web Server 6.1 SP9 Administrator's Guide*.

TABLE 6–2 dbswitch.conf Properties *(Continued)*

Property	Allowed Values	Default Value	Description
`syntax`	`keyfile`, `digest`, `htaccess`	`keyfile`	Specifies what type of file auth-db will be used
`keyfile`			Specifies the path to the keyfile. Required, if `syntax` is set to `keyfile`.
`digestfile`			Specifies the path to the digestfile. Required, if `syntax` is set to `digestfile`.
`groupfile`			Path to the `AuthGroupFile`. If the groupfile is the same as the userfile, this file contains both user and group data, otherwise it contains only group data. Required if `syntax` is set to `htaccess`. For more information about the syntax of the `AuthGroupFile`, see the *Sun Java System Web Server 6.1 SP9 Administrator's Guide*.
`userfile`			Path to the `AuthUserFile`. If the userfile is the same as the groupfile, this file contains both user and group data, otherwise it contains only user data. Required if `syntax` is set to `htaccess`. For more information about the syntax of the `AuthUserFile`, see the *Sun Java System Web Server 6.1 SP9 Administrator's Guide*.

Deployment Descriptors

Purpose

Configures features specific to the Sun Java System Web Server for deployed web applications.

Location

The `META-INF` or `WEB-INF` directory of a module or application.

See Also

The following table shows where to find more information about Sun Java System Web Server deployment descriptors. The left column lists the deployment descriptors, and the right column lists where to find more information about those descriptors.

TABLE 6–3 Sun Java System Web Server Deployment Descriptors

Deployment Descriptor	Where to Find More Information
`sun-web.xml`	*Sun Java System Web Server 6.1 SP9 Programmer's Guide to Web Applications*.

generated.instance.acl

Purpose

Sets permissions for access to the server instance. This is the default ACL file; you can create and use others.

Location

server_root`/config`

See Also

Sun Java System Web Server 6.1 SP9 Administrator's Guide

login.conf

Purpose

The login module definition configuration file used by the Java Authentication and Authorization Service (JAAS) for client authentication.

Location

server_root`/config`

nsfc.conf

Purpose

Sets file cache parameters. This file is present only if file cache parameters have been changed from their defaults.

Location

server_root/https-admserv/config

Syntax

parameter=*value*

See Also

Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide

The following table describes properties in the nsfc.conf file. The left column lists the property names. The second column from the left lists allowed values. The third column from the left lists default values. The right column lists property descriptions.

TABLE 6–4 nsfc.conf Properties

Attribute	Allowed Values	Default Value	Description
FileCacheEnable	on, off	on	Enables the file cache.
CacheFileContent	on, off	on	Enables caching of file contents, as well as file information for files smaller than MediumFileSizeLimit (smaller than SmallFileSizeLimit if TransmitFile is on).
MaxAge	Number of seconds	30	The maximum age of a valid cache entry. This setting controls how long cached information is used once a file has been cached. An entry older than MaxAge is replaced by a new entry for the same file.

TABLE 6–4 nsfc.conf Properties *(Continued)*

Attribute	Allowed Values	Default Value	Description
`MediumFileSizeLimit`	Limited by available memory	`537600` (525K)	(UNIX only) Maximum size of a file that can be cached as a memory-mapped file (if `TransmitFile` is `off`).
`MediumFileSpace`	Limited by available memory	`10485760`(10 M)	Total size of all files that are cached as memory-mapped files (if `TransmitFile` is `off`).
`SmallFileSizeLimit`	Limited by available memory	`2048` (2K)	(UNIX only) Maximum size of a file that can be read into memory.
`SmallFileSpace`	Limited by available memory	`1048576` (UNIX, 1 M), `0` (Windows)	Total size of all files that are read into memory.
`TransmitFile`	`on`, `off`	`on` (Windows), `off` (UNIX)	Enables use of the `TransmitFile` system call. Not supported on IRIX, Compaq, Solaris, or Linux.
`MaxFiles`		`1024`	Maximum number of files in the file cache.
`HashInitSize`	Limited by available memory	`0`	Initial number of hash buckets. If `0`, the number of hash buckets is dynamically determined as 2 * `MaxFiles` + 1.
`CopyFiles`	`on`, `off`	`on`	(Windows only) Prevents sharing violations by copying files to a temporary directory.
`TempDir`	path	*<TempDir>*`/`*<server_id>*`-file-cache`	Specifies a temporary directory for the file cache if CopyFiles is on. *<TempDir>* is the value of TempDir in the magnus.conf file. See "TempDir" on page 76 *<server_id>* is the server instance id.

password.conf

Purpose

By default, the Sun Java System Web Server prompts the administrator for the SSL key database password before starting up. If you want the Web Server to be able to restart unattended, you need to save the password in a `password.conf` file. Be sure that your system is adequately protected so that this file and the key databases are not compromised.

Location

server_root/config

This file is not present by default. You must create it if you need it.

Syntax

PKCS#11_module_name:*password*

If you are using the internal PKCS#11 software encryption module that comes with the server, type the following:

internal:*password*

If you are using a different PKCS#11 module, for example for hardware encryption or hardware accelerators, you will need to specify the name of the PKCS#11 module, followed by the password.

See Also

Sun Java System Web Server 6.1 SP9 Administrator's Guide

server.policy

Purpose

Controls what access applications have to resources. This is the standard J2SE policy file. The J2SE SecurityManager is not active by default in Sun Java System Web Server 6.1. The policies granted in this policy file do not have any effect unless the SecurityManager is turned on in server.xml.

If you wish to use the J2SE SecurityManager you can turn it on by adding the following JVM options:

```
<JVMOPTIONS>-Djava.security.manager</JVMOPTIONS>
<JVMOPTIONS>-Djava.security.policy=server_root/config/server.policy</JVMOPTIONS>
```

Location

server_root/config

Syntax

`grant [codeBase "`*path*`"] {`

`permission` *permission_class* "*package*", "*permission_type*";

...

`};`

See Also

- *Sun Java System Web Server 6.1 SP9 Programmer's Guide*
- `http://java.sun.com/docs/books/tutorial/security/tour2/index.html`

*.clfilter

Purpose

The files `obj.conf.clfilter`, `magnus.conf.clfilter`, and `server.xml.clfilter` contain filter specifications for cluster management operations.

Location

server_root`/config`

◆ ◆ ◆ APPENDIX A

Configuration Changes Between iPlanet Web Server 4.1 and Sun Java System Web Server 6.1

This chapter summarizes major configuration file changes between the 4.1 and the 6.1 version of Sun Java System Web Server. The following 4.1 files are described:

magnus.conf

The following table summarizes the changes in `magnus.conf`:

TABLE A–1 magnus.conf Changes

4.x Directive	6.1 Directive	Comments
`AccelFileCache`	(none)	Obsolete because an NSAPI accelerator cache is no longer necessary
`AcceptLanguage`	(none)	See the `acceptlanguage` attribute of the "VSCLASS" on page 38 and "VS" on page 39 elements in `server.xml`
`ACLFile`	(none)	Maps to the "ACLFILE" on page 37 element in `server.xml`
`Address`	(none)	Maps to the "LS" on page 33 element in `server.xml`.
`AdminLanguage`	(none)	Deprecated.

TABLE A–1 magnus.conf Changes *(Continued)*

4.x Directive	6.1 Directive	Comments
AsyncDNS	AsyncDNS	Ignored. Even if the value is set to on, the server does not perform asynchronous DNS lookup.
BlockingListenSockets	(none)	See the blocking attribute of the "LS" on page 33 element in server.xml.
CGIWaitPid	(none)	Deprecated.
Ciphers	(none)	See the ssl2ciphers attribute of the "SSLPARAMS" on page 35 element in server.xml
ClientLanguage	(none)	Deprecated.
DaemonStats	(none)	Obsolete due to new performance statistics system. For more information, see the *Sun Java System Web Server 6.1 SP9 Performance Tuning, Sizing, and Scaling Guide* for further information.
DefaultCharSet	(none)	Deprecated
ErrorLog	(none)	See the file attribute of the "LOG" on page 65 element in server.xml.
IOTimeout	AcceptTimeout	Use the AcceptTimeout directive to specify the number of seconds the server must wait for data from a client before closing the connection.
LoadObjects	(none)	See the objectfile attribute in the "VSCLASS" on page 38 element in server.xml.
LogVerbose	(none)	See the "Attributes" on page 65 attribute in server.xml.
MaxThreads	(none)	Obsolete due to new thread handling system.
MinProcs	(none)	Obsolete due to new thread handling system.
MinThreads	(none)	Obsolete due to new thread handling system.
MtaHost	(none)	Ignored.
NetsiteRoot	(none)	Deprecated.

TABLE A–1 magnus.conf Changes *(Continued)*

4.x Directive	6.1 Directive	Comments
`Port`	(none)	See the "LS" on page 33 element in `server.xml`.
`RootObject`	(none)	See the `rootobject` attribute of the "VSCLASS" on page 38 element in `server.xml`.
`RqThrottleMinPerSocket`	(none)	See the `the acceptorthreads` attribute of the "LS" on page 33 element in `server.xml`.
(none)	`RqThrottleMin`	New. Specifies the number of request processing threads that are created when the server is started.
ServerID	(none)	Deprecated.
`ServerName`	(none)	Deprecated. See the `servername` attribute of the "LS" on page 33 element in the `server.xml` file.
`#ServerRoot`	(none)	Deprecated.
`SSL2`	(none)	See the `ssl2` attribute of the "SSLPARAMS" on page 35 element in `server.xml`
`SSL3`	(none)	See the `ssl3` attribute of the "SSLPARAMS" on page 35 element in `server.xml`
`SSL3Ciphers`	(none)	See the `the ssl3tlsciphers` attribute of "SSLPARAMS" on page 35 element in `server.xml`
`SSLClientAuth`	clientauth	See the `clientauth` attribute of the "SSLPARAMS" on page 35 element in `server.xml`
`VirtualServerFile`	(none)	Obsolete due to virtual server implementation

obj.conf

The `obj.conf` file has lost its Init directives to the `magnus.conf` file and acquired new directives and parameters. The following table summarizes the changes in the `obj.conf` file. Only the changed directives are listed.

TABLE A–2 obj.conf Changes

4.x Directive	6.1 Directive	Comments
Init functions	(none)	All functions have moved to `magnus.conf` except for `cache-init` and `load-types`, which are obsolete (for `load-types`, see the `MIME` element in the `server.xml` file).
`Service fn=parse-html`	`Service fn=shtml_send`	

contexts.properties

The `contexts.properties` file is no longer supported. Servlet contexts or web applications are now defined in the `server.xml` file and configured using the `sun-web.xml` file.

A few `contexts.properties` functions are now in the `server.xml` file.

The following table lists the equivalent functions in the `contexts.properties` and `sun-web.xml` files.

TABLE A–3 contexts.properties to sun-web.xml Correspondences

contexts.properties Property	sun-web.xml Element or Attribute	Comments
`sessionmgr`	`persistence-type` attribute of the `session-manager` element	
`sessionmgr.initArgs`	`manager-properties` and `store-properties` attributes of the `session-manager` element	
`initArgs`	(none)	Specified using the `context-param` element in `web.xml`. For more information, see Servlet 2.3 specification. To add context attributes, implement the `javax.servlet.ServletContextListener` interface. For more information, see to the Servlet 2.3 specification.
`respondCookieVersion`	(none)	Might be supported in a future release.
`tempDir`	`tempdir` property	

TABLE A–3 contexts.properties to sun-web.xml Correspondences *(Continued)*

contexts.properties Property	sun-web.xml Element or Attribute	Comments
`reloadInterval`	`dynamic-reload-interval` attribute of `class-loader` element	
`bufferSize`	(none)	Specified using the `UseOutputStreamSize` in `obj.conf`. for more information, see "service-j2ee" on page 189 for more information.
`docRoot`	(none)	Specified in the `server.xml` file for each virtual server.
`inputStreamLengthCheck`	(none)	Obsolete.
`outputStreamFlushTimer`	(none)	Obsolete.
`uri`	`uri` attribute of `WEBAPP` element in `server.xml`.	
`authdb`	`authdb` attribute of `auth-native` element	Obsolete.
`classpath`	`extra-class-path` attribute of `class-loader` element	
`singleClassLoader`	(none)	Obsolete because each web application has a single class loader as mandated by the Servlet 2.3 specification.
`serverName`	(none)	Specified in the `server.xml` file for each virtual server.
`contentTypeIgnoreFromSSI`	(none)	Obsolete due to web application support.
`parameterEncoding`	`parameter-encoding` element	
`isModifiedCheckAggressive`	(none)	Obsolete.
`includeTransparency`	(none)	Obsolete.

rules.properties

The `rules.properties` file is no longer supported in Sun Java System Web Server 6.1. The function of the `rules.properties` file is now handled by the `servlet-mapping` element in the `web.xml` file. For more information, see the Servlet 2.3 API specification at:

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

servlets.properties

The `servlets.properties` file is no longer supported for the default virtual server and other virtual servers. Most of the same functions are in the `sun-web.xml` file.

A few `servlets.properties` functions are in the `server.xml` file.

A few `servlets.properties` functions are in the `web.xml` file. For more information, see the Servlet 2.3 API specification at:

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

The following table lists the equivalent functions in the `servlets.properties` and `sun-web.xml` files.

TABLE A–4 servlets.properties to sun-web.xml Correspondences for Individual Servlet Properties

servlets.properties Property	sun-web.xml Element or Attribute	Comments
`code`	(none)	Specified in a `servlet-class` element in the `web.xml` file.
`context`	(none)	Obsolete because servlets are hosted within a web application which is deployed at the URI specified as the value of the `uri` attribute of the "WEBAPP" on page 49 element in `server.xml`.
`classpath`	(none)	The Servlet 2.3 specification specifies that servlet classes be packaged in the WEB-INF/classes directory or in .jar archives in the WEB-INF/lib directory.
`initArgs`	(none)	Use the `init-param` element of the `<servlet>` tag in `web.xml` to specify servlet-specific initialization parameters.
`startup`	(none)	Specified in a `load-on-startup` element in the `web.xml` file.

◆ ◆ ◆ APPENDIX B

Configuration Changes Between iPlanet Web Server 6.0 and Sun Java System Web Server 6.1

This chapter summarizes major configuration file changes between the 6.0 and the 6.1 version of Sun Java System Web Server. The following files are described:

magnus.conf

This section lists the magnus.conf-related changes in the following areas:

Init Functions

The magnus.conf file in SUN Java System Web Server 6.1 has acquired new Init SAFs as listed in the following table:

TABLE B–1 magnus.conf Init Functions

6.0 Function/Parameter	6.1 Function/Parameter	Comments
NSServletEarlyInit	(none)	Removed.
NSServletLateInit	(none)	Removed.

TABLE B–1 magnus.conf Init Functions *(Continued)*

6.0 Function/Parameter	6.1 Function/Parameter	Comments
`nt-console-init`	`createconsole`	Removed. On Windows, you can configure the `createconsole` attribute of the "LOG" on page 65 element to redirect stderr output to the console.

Directives

The `magnus.conf` file has lost directives to other configuration files and some directives supported by the `magnus.conf` file in previous releases are now deprecated. The following table summarizes the changes:

TABLE B–2 Changes in magnus.conf Directives

6.0 Directive	6.1 Value	Comments
`AdminLanguage`	(none)	Deprecated.
`AsyncDNS`	`AsyncDNS`	Ignored. Even if the value is set to `on`, the server does not perform asynchronous DNS lookup.
CGIWaitPid	(none)	Deprecated.
`ClientLanguage`	(none)	Deprecated.
`DefaultCharSet`	(none)	Ignored.
`ErrorLog`	(none)	See the "Attributes" on page 65 attribute of the "LOG" on page 65 element in `server.xml`.
`IOTimeout`	`AcceptTimeout`	Use the `AcceptTimeout` directive to specify the number of seconds the server must wait for data from a client before closing the connection.
`LogVerbose`	(none)	See the "Attributes" on page 65 attribute of the "LOG" on page 65 element in `server.xml`.
LogVsId	"Attributes" on page 65	See the "Attributes" on page 65 attribute of the "LOG" on page 65 element in `server.xml`.
`NetsiteRoot`	(none)	Deprecated.

TABLE B–2 Changes in magnus.conf Directives *(Continued)*

6.0 Directive	6.1 Value	Comments
ServerConfigurationFile	(none)	Ignored.
ServerID	(none)	Deprecated.
ServerName	(none)	Deprecated. See the `servername` attribute of the "LS" on page 33 element in the `server.xml` file.
#ServerRoot	(none)	Deprecated.

obj.conf

The `obj.conf` file has acquired new SAFs and parameters as listed in Table 6-7. Only the new and changed directives are listed.

TABLE B–3 obj.conf Changes

Supported in 6.0	Supported in 6.1	Comments
JSP092 object	(none)	Removed. Sun Java System Web Server 6.1 supports the JSP 2.3 specification and so, the JSP092 object is not required.

server.xml

This section describes the following changes:

- Table B–4
- Table B–5
- Table B–6

The following table lists the correspondences between the `server.xml` file in iPlanet Web Server 6.0 and the `server.xml` file in Sun Java System Web Server 6.1:

TABLE B–4 server.xml to server.xml Correspondences

`legacyls`	Not supported.

TABLE B–4 server.xml to server.xml Correspondences *(Continued)*

CONNECTIONGROUP	The CONNECTIONGROUP element is not supported. The defaultvs and servername attributes from the CONNECTIONGROUP element are added to the "LS" on page 33 element in Sun Java System Web Server 6.1 during migration.
SSLPARAMS	The SSLPARAMS element, in 6.0 parsed from the CONNECTIONGROUP element, is a subelement of the "LS" on page 33 element in Sun Java System Web Server 6.1.
VARS	The functionality of the VARS element is handled by the "PROPERTY" on page 31 element in Sun Java System Web Server 6.1. However, the VARS element is still retained for backward compatibility.
webapps_file webapps_enable	Removed. The "WEBAPP" on page 49 element of the VS element in server.xml handles web applications. Web container-specific configuration is handled by the sun-web.xml file.

The following table lists the correspondences between the start-jvm file in iPlanet Web Server 6.0 to the server.xml file in Sun Java System Web Server 6.1:

TABLE B–5 start-jvm and server.xml Correspondences

NSES_JDK	"Attributes" on page 50
NSES_CLASSPATH	"Attributes" on page 50
NSES_JRE_RUNTIME_LIBPATH	"Attributes" on page 50
NSES_JRE_RUNTIME_CLASSPATH	Use the -Xbootclasspath JVM option.

The following table lists the correspondences between the jvm12.conf file in iPlanet Web Server 6.0 and the server.xml file in Sun Java System Web Server 6.1:

TABLE B–6 jvm12.conf and server.xml Correspondences

jvm.minHeapSize	Use the -Xms<*value*> JVM option. Example: <JVMOPTIONS>-Xms128m -Xmx256m</JVMOPTIONS>
jvm.maxHeapSize	Use the -Xmx<value> JVM option. Example: <JVMOPTIONS>-Xms128m -Xmx256m</JVMOPTIONS>

TABLE B–6 jvm12.conf and server.xml Correspondences *(Continued)*

`jvm.enableClassGC`	Use the `-Xnoclassgc` JVM option to disable garbage collection.
jvm.option	`Use the` "JVMOPTIONS" on page 52 element.
jvm.profiler	`Use the` "PROFILER" on page 53 element.
`jvm.verboseMode`	Use the `-verbose` JVM option.
`jvm.printErrors`	Not supported.
`jvm.disableThreadRecycling`	Not supported.
`jvm.serializeAttach`	Not supported.
`jvm.stickyAttach`	Not supported.
`jvm.trace`	Configured in the `LOGLEVEL` element of the web container.
`jvm.allowExit`	Refer to information at the following URL for more information about configuring this in the server.policy file: `http://java.sun.com/products/archive/j2se/1.4.1_07/index.html`
jvm.include.CLASSPATH	`Use the` "Attributes" on page 50 attribute of the "JAVA" on page 50 element.
`jvm.enableDebug`	`Use the debug and` debugoptions `attributes of the` "JAVA" on page 50 element.
jvm.classpath	Use the "Attributes" on page 50 and "Attributes" on page 50 attributes of the JAVA element.

The following table lists the correspondences between the `web-apps.xml` file in iPlanet Web Server 6.0 and the `sun-web.xml` file in Sun Java System Web Server 6.1:

TABLE B–7 web-apps.xml and sun-web.xml Correspondences

`<config-param<param-name>`
`redirect-to-absolute-url</param-name>`
`<param-value>false`
`</param-value></config-param>`
	Use the <property name="relativeRedirectAllowed" value="true">

◆ ◆ ◆ APPENDIX C

Time Formats

This module describes the format strings used for dates and times in the server log. These formats are used by the NSAPI function `util_strftime`, by some built-in SAFs such as `append-trailer`, and by server-parsed HTML (`parse-html`).

The formats are similar to those used by the `strftime` C library routine, but not identical.

The following table describes the format strings for dates and times.

TABLE C–1 Format Strings

Attribute	Allowed Values
%a	Abbreviated weekday name (3 chars)
%d	Day of month as decimal number (01-31)
%S	Second as decimal number (00-59)
%M	Minute as decimal number (00-59)
%H	Hour in 24-hour format (00-23)
%Y	Year with century, as decimal number, up to 2099
%b	Abbreviated month name (3 chars)
%h	Abbreviated month name (3 chars)
%T	Time "HH:MM:SS"
%X	Time "HH:MM:SS"
%A	Full weekday name
%B	Full month name
%C	"%a %b %e %H:%M:%S %Y"

TABLE C–1 **Format Strings** *(Continued)*

Attribute	Allowed Values
%c	Date & time "%m/%d/%y %H:%M:%S"
%D	Date "%m/%d/%y"
%e	Day of month as decimal number (1-31) without leading zeros
%I	Hour in 12-hour format (01-12)
%j	Day of year as decimal number (001-366)
%k	Hour in 24-hour format (0-23) without leading zeros
%l	Hour in 12-hour format (1-12) without leading zeros
%m	Month as decimal number (01-12)
%n	line feed
%p	A.M./P.M. indicator for 12-hour clock
%R	Time "%H:%M"
%r	Time "%I:%M:%S %p"
%t	tab
%U	Week of year as decimal number, with Sunday as first day of week (00-51)
%w	Weekday as decimal number (0-6; Sunday is 0)
%W	Week of year as decimal number, with Monday as first day of week (00-51)
%x	Date "%m/%d/%y"
%y	Year without century, as decimal number (00-99)
%%	Percent sign

◆ ◆ ◆ APPENDIX D

Alphabetical List of Server Configuration Elements

◆ ◆ ◆ APPENDIX E

Alphabetical List of Predefined SAFs

This appendix provides an alphabetical list for the easy lookup of predefined SAFs.

Index

D

E

F

G

H

N

O

P

Q

R

S

T

U

V

W