Sun Java™ System

# Web Proxy Server 4.0.1 Configuration File Reference

2005Q4

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

Part No: 819-3651-10

# Contents

# About This Guide

This guide describes how to configure and administer the Sun Java™ System Web Proxy Server 4, formerly known as Sun™ ONE Web Proxy Server and iPlanet™ Web Proxy Server (and hereafter referred to as Sun Java System Web Proxy Server or just Proxy Server).

This preface contains information about the following topics:

- Who Should Use This Guide
- How This Guide Is Organized
- Documentation Conventions
- Using the Documentation
- Contacting Sun Technical Support
- Third-Party Web Site References
- Feedback

## Who Should Use This Guide

This guide is intended for information technology administrators in production environments. The guide assumes familiarity with the following:

- Basic system administration tasks
- Installing software
- Using Web browsers
- Issuing commands in a terminal window

# How This Guide Is Organized

The guide is divided into parts, each of which addresses specific areas and tasks. The following table lists the parts of the guide and their contents.

**Table 1 Guide Organization**

| Chapter | Description |
|---|---|
| Chapter 1, "Basics of Server Operation" | This chapter introduces the major configuration files that control the Sun Java System Web Proxy Server and describes how to activate and edit them. |
| Chapter 2, "Server Configuration Elements in server.xml" | This chapter discusses the server.xml file, which controls most aspects of server operation. |
| Chapter 3, "Syntax and Use of magnus.conf" | This chapter discusses the directives you can set in the magnus.conf file to configure the Sun Java System Web Proxy Server during initialization. |
| Chapter 4, "Syntax and Use of obj.conf" | This chapter discusses the SAFs you can set in the configuration file obj.conf to configure the Sun Java System Web Proxy Server during initialization. |
| Chapter 5, "Predefined SAFs in obj.conf" | This chapter describes the predefined SAFs used in the obj.conf file. |
| Chapter 6, "MIME Types" | This chapter discusses the MIME types file, which maps file extensions to file types. |
| Chapter 7, "Other Server Configuration Files" | This chapter lists other important configuration files and provides a quick reference of their contents. |
| Appendix A, "Configuration Changes Between iPlanet Web Proxy Server 3.6 and Sun Java System Web Proxy Server 4" | This appendix describes the changes in configuration files between the iPlanet Web Proxy Server 3.6 and Sun Java System Web Proxy Server 4. |
| Appendix B, "Time Formats" | This appendix describes the format strings used for dates and times in the server log. |
| Appendix C, "Alphabetical List of Server Configuration Elements" | This chapter provide an alphabetical list for easy lookup of elements in server.xml and directives in magnus.conf. |
| Appendix D, "Alphabetical List of Predefined SAFs" | This chapter provide an alphabetical list for easy lookup of directives in obj.conf. |

# Documentation Conventions

The following table lists the documentation conventions used in this guide.

**Table 2 Documentation Conventions**

| Element | Usage |
|---|---|
| File and directory paths | Given in UNIX® format, with forward slashes separating directory names |
| Installation root directories | Indicated as*`install_dir`*. |
| *Italic* text | Titles, emphasis, terms |
| `monospace` text | Code examples, file names, path names, command names, programming language keywords, properties |
| i*`italic monospace`* text | Variable path names, environment variables in paths |

# Using the Documentation

The Sun Java System Web Proxy Server 4.0.1 documentation is available in PDF and HTML formats at:
http://docs.sun.com/app/docs/coll/1311.1

The following table lists the tasks and concepts described in guide.

**Table 3 Proxy Server Documentation**

| For Information About | See |
|---|---|
| Late-breaking information about the software and documentation | *Release Notes* |
| Performing installation and migration tasks:<br>• Supported platforms and environments<br>• Installing Sun Java System Web Proxy Server<br>• Migrating from version 3.6 to version 4 | *Installation and Migration Guide* |

**Table 3 Proxy Server Documentation**

| For Information About | See |
|---|---|
| Performing administration and management tasks:<br>• Using the Administration and command-line interfaces<br>• Configuring server preferences<br>• Managing users and groups<br>• Monitoring and logging server activity<br>• Using certificates and public key cryptography to secure the server<br>• Controlling server access<br>• Proxying and routing URLs<br>• Caching<br>• Filtering content<br>• Using a reverse proxy<br>• Using SOCKS<br>• Tuning the Proxy Server to optimize performance | *Administration Guide*<br>(and the online Help included with the product) |
| Creating custom Netscape Server Application Programmer's Interface (NSAPI) plugins | *NSAPI Developer's Guide* |
| Editing configuration files | *Configuration File Reference* |

# Contacting Sun Technical Support

For technical questions about this product not answered in the product documentation, go to:
http://www.sun.com/service/contacting

# Third-Party Web Site References

Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

# Feedback

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to http://docs.sun.com and click "Send comments." Be sure to provide the document title and part number in the online form.

Chapter 1

# Basics of Server Operation

The configuration and behavior of Sun Java System Web Proxy 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
- Directory Structure
- Dynamic Reconfiguration

## Configuration Files

The configuration and operation of the Sun Java System Web Proxy Server is controlled by configuration files. The configuration files reside in the directory <*Instance_Directory*>`/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` -- contains most of the server configuration.
- `magnus.conf` -- contains global server initialization information.
- `obj.conf` -- contains instructions for handling HTTP requests from clients.
- `mime.types` -- contains information for determining the content type of requested resources.

# server.xml

This file contains most of the server configuration. A schema file, sun-web-proxy-server_4_0.dtd, defines its format and content.

For more information about how the server uses sun-web-proxy-server_4_0.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 looks at this file and executes the settings on startup. The server does not look at this file again until it is restarted.

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

# obj.conf

This file contains instructions for the Sun Java System Web Proxy Server about how to handle HTTP requests from clients and proxy requests to the origin server that services the content. The server looks at 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 Proxy 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 <*Instance_Directory*>/config directory.

The obj.conf file is essential to the operation of the Sun Java System Web Proxy 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, "Syntax and Use of 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, while 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 7, “Other Server Configuration Files.”

## Directory Structure

The following section describes the directory structure created when you first install Sun Java System Web Proxy Server 4.

### Default Directory Structure

The default directory structure of the proxy server environment consists of the following items:

- <*Install_Root*>`/alias` Contains the key-pair files for all server instances installed in this installation directory.
- <*Install_Root*>`/bin` Contains the binary executables for the Proxy Server, itself.
- <*Install_Root*>`/extras` Contains the command line utilities for the Proxy Server.
- <*Install_Root*>`/httpacl` Contains the access control list (ACL) files for all server instances installed in this installation directory.
- <*Install_Root*>`/manual` Contains the HTML documentation for the Proxy Server.
- <*Install_Root*>`/ns-iconsns` Contains graphical images for proxied FTP browsing.

- *<Install_Root>*/`plugins` Contains plug-ins installed for this installation of the Proxy Server.
- *<Install_Root>*/`proxy-admserv` Contains an HTTP server instance used to manage the Proxy and SOCKS servers for this installation.

## Proxy Server Directory Structure

The default directory structure of the Proxy Server instance immediately after installation consists of the following items:

- *<Instance_Directory>*/`cache` Contains the initial cache file system for this instance of the Proxy Server.
- *<Instance_Directory>*/`conf_bk` Contains back up versions of the Proxy Server configuration files.
- *<Instance_Directory>/config* Contains the current versions of the Proxy Server configuration files.
- *<Instance_Directory>*/`logs` Contains the errors and access log files for the Proxy Server instance.
- *<Instance_Directory>*/`pac` Contains the proxy auto configuration files.
- *<Instance_Directory>*/`reconfig` - command line script to perform dynamic reconfiguration of the Proxy Server configuration files.
- *<Instance_Directory>/start-sockd* - command line script to start the SOCKS daemon.
- *<Instance_Directory>*/`start-` command line script to start the Proxy Server.

# Dynamic Reconfiguration

Dynamic reconfiguration is a feature in Proxy Server 4 that allows you to make configuration changes to a live Proxy Server without having to stop and restart the Proxy Server for the changes to take effect. You can dynamically change all configuration settings and attributes in the server.xml file as well as many other configuration files without having to restart the server.

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_Directory*>`/config` directory. A schema file, `sun-web-proxy-server_4_0.dtd`, determines the format and content of the `server.xml` file.

This chapter describes `server.xml` and `sun-web-proxy-server_4_0.dtd` in the following sections:

- The sun-web-proxy-server_4_0.dtd File
- Elements in the server.xml File
- Core Server Elements
- Listener Elements
- Cache Elements
- The Sun Java System LDAP Schema
- Variables
- Sample server.xml File

## The sun-web-proxy-server_4_0.dtd File

The `sun-web-proxy-server_4_0.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-proxy-server_4_0.dtd` file is located in the <*Install_Directory*>`/bin/proxy/dtds` directory.

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

- Subelements
- Data
- Attributes

## Subelements

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

```
<!ELEMENT LS (DESCRIPTION?, SSLPARAMS?)>
```

The ELEMENT tag specifies that a LSCLASS element can contain DESCRIPTION, and SSLPARAMS 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     ACLFILE
id            ID            #REQUIRED
file          CDATA         #REQUIRED
```

An `ACLFILE` element can contain `id`, and `file` 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 Proxy 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:

- Core Server Elements
- Listener Elements
- Cache Elements

**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 C, "Alphabetical List of Server Configuration Elements."

# Core Server Elements

General elements are as follows:

- SERVER
- PROPERTY
- DESCRIPTION
- LOG
- EVENT
- EVENTTIME
- EVENTACTION

## 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 |
|---|---|---|
| PROPERTY | zero or more | Specifies a property of the server. |
| LS | one or more | Defines one or more HTTP listen sockets. |
| MIME | zero or one | Defines mime type. |
| ACLFILE | zero or one | References one or more ACL files. |
| USERDB | zero or more | Defines the user database used. |
| FILECACHE | only one | Configures NSFC parameters. |
| CACHE | zero or one | Configures the disk cache parameters. |
| LOG | zero or one | Configures the system logging service. |

**Table 2-2** SERVER subelements

| Element | Required | Description |
|---|---|---|
| EVENT | zero or more | Configures events. |

## Attributes

The following table describes attributes for the SERVER element.

**Table 2-3** SERVER attributes

| Attribute | Default | Description |
|---|---|---|
| objectfile | obj.conf | Specifies the obj.conf file for the server. |
| 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 server; any server behavior that is configured in the default object affects the entire server.<br>If you specify an object that does not exist, the server does not report an error until a client tries to retrieve a document. |

# 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 45.

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 Proxy Server
- Needed by a system or object that Sun Java System Web Proxy Server doesn't have knowledge of, such as an LDAP server or a Java class

For example,

```
<PROPERTY name="accesslog"
value="<Install_Root>/<Instance_Directory>/logs/access"/>
```

## Subelements

The following table describes subelements for the PROPERTY element.

**Table 2-4** PROPERTY subelements

| Element | Required | Description |
|---|---|---|
| DESCRIPTION | 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

## LOG

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

- The errors log file stores messages from the server. The default name is errors.
- The access log file stores HTTP access messages from the server. The default name is access.log. To configure the access log, you use server application functions in the obj.conf files.

## Subelements

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

**Table 2-6** `LOG` subelements

| Element | Required | Description |
|---|---|---|
| `PROPERTY` | zero or more | Specifies a property or a variable. |

## Attributes

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

**Table 2-7** `LOG` attributes

| Attribute | Default | Description |
|---|---|---|
| `file` | `errors` | Specifies the file that stores messages from the server. |
| `loglevel` | `info` | Controls the default type of messages logged by other elements to the error log.<br>Allowed values are as follows, from highest to lowest.<br>`finest, finer, fine, info, warning, failure, config, security, and catastrophe`. |
| `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`. |

# EVENT

An event can be scheduled to run at (a) specific time(s) either on (a) day(s) of the week or on (a) day(s) of the month or when the server starts up or shuts down.

## Subelements

The following table describes subelements for the EVENT element.

**Table 2-8** EVENT subelements

| Element | Required | Description |
|---|---|---|
| DESCRIPTION | zero or one | Descriptive text about the event. Used for informational purposes. This is an optional element. |
| EVENTTIME | only one | Container element that specifies the time at which the event is to be executed. This is a required element. |
| EVENTACTION | only one | Container element that specifies the event action to be executed. This is a required element. |
| PROPERTY | zero or more | Specifies a property or a variable. |

## Attributes

The following table describes attributes for the EVENT element.

**Table 2-9** EVENT attributes

| Attribute | Default | Description |
|---|---|---|
| enabled | true | Indicates whether the specified event is to be scheduled or not. |
| name | none | Specifies the name of the event. |

# EVENTTIME

Container element that specifies the time at which the event is to be executed. This is a required element.

## Subelements

The following table describes subelements for the EVENTTIME element.

**Table 2-10** EVENTTIME subelements

| Element | Required | Description |
|---|---|---|
| TIMEOFDAY | only one | A space separated list of times (in 24 hr hh:mm notation) at which the event should be run. This is a required element. If neither DAYOFWEEK or DAYOFMONTH is specified then the event will be scheduled at these times every day of the week.<br>For example,<br>`<TIMEOFDAY>00:30 6:30 12:30 18:30</TIMEOFDAY>` |
| DAYOFWEEK | zero or one | A space separated list of weekday names on which the event should be run at the time specified by the TIMEOFDAY value. A value for either this element or the DAYOFMONTH element must be specified. The valid names for weekdays are - Mon, Tue, Wed, Thu, Fri, Sat, Sun.<br>For example,<br>`<DAYOFWEEK>Mon Wed Fri</DAYOFWEEK>` |
| DAYOFMONTH | zero or one | A space separated list of integers from 1-31 that denotes the day of the month on which the event is to be run. The TIMEOFDAY value specifies the time at which the event will be run. A value for either this element or the TIMEOFDAY element must be specified.<br>For example,<br>`<DAYOFMONTH>1 15</DAYOFMONTH>` |
| ONSTARTUP | only one | The event is scheduled to occur when the server starts up. |
| ONSHUTDOWN | only one | The event is scheduled to occur when the server shuts down. |

# EVENTACTION

Container element that specifies the event action to be executed.

### Subelements

The following table describes subelements for the EVENTACTION element.

**Table 2-11** EVENTACTION subelements

| Element | Required | Description |
|---|---|---|
| RESTART | zero or one | If specified, this event will restart the server at the specified times |
| RECONFIG | zero or one | If specified, this event will dynamically reconfigure the server at the specified times. |
| ROTATELOGS | zero or one | If specified, this event will rotate the server access and error log files at the specified times. |
| COMMAND | zero or one | The command line of the executable to run at the scheduled time(s). This is an optional sub element of EVENTACTION. |

# Listener Elements

The Listener elements are as follows:

- LS
- SSLPARAMS
- MIME
- ACLFILE
- USERDB

## 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.

## Subelements

The following table describes subelements for the LS element.

**Table 2-12** LS subelements

| Element | Required | Description |
|---|---|---|
| DESCRIPTION | zero or one | Contains a text description of the listen socket. |
| SSLPARAMS | zero or one | Defines Secure Socket Layer (SSL) parameters. |

## Attributes

The following table describes attributes for the LS element.

**Table 2-13** LS attributes

| Attribute | Default | Description |
|---|---|---|
| id | none | (optional) The socket family type. A socket family type cannot begin with a number.<br>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. |

**Table 2-13** LS attributes

| Attribute | Default | Description |
|---|---|---|
| `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.<br>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 will be prefixed with `::ffff:` in the log file. Specify `nca` to make use of the Solaris Network Cache and Accelerator. |
| `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`. |
| `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 doesn’t affect the URLs for directories and files stored in the server. This name should be the alias name if your server uses an alias.<br>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 2-14** 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. |
| `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`. |

**Table 2-14** SSLPARAMS attributes

| Attribute | Default | Description |
|---|---|---|
| `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 element.

## Attributes

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

**Table 2-15** MIME attributes

| Attribute | Default | Description |
|---|---|---|
| `id` | none | Internal name for the MIME types listing. The MIME types name cannot begin with a number. |
| `file` | none | The name of a MIME types file. For more information, see Chapter 6, “MIME Types.”. |

# TYPE

Defines the type of the requested resource.

## Subelements

none

## Attributes

The following table describes attributes for the TYPE element.

**Table 2-16** TYPE attributes

| Attribute | Default | Description |
|---|---|---|
| `type` | none | Defines the type of the requested resource. |
| `language` | none | Defines the content language. |
| `encoding` | none | Defines the content-encoding. |
| `extensions` | none | Defines the file extensions associated with the specified resource. |

# ACLFILE

References one ACL file.

## Subelements

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

**Table 2-17** ACLFILE subelements

| Element | Required | Description |
|---|---|---|
| `DESCRIPTION` | zero or one | Contains a text description of the `ACLFILE` element. |

## Attributes

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

**Table 2-18** ACLFILE attributes

| Attribute | Default | Description |
|---|---|---|
| `id` | none | Internal name for the ACL file listing. An ACL file listing name cannot begin with a number. |

**Table 2-18** ACLFILE attributes

| Attribute | Default | Description |
|---|---|---|
| `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 Proxy Server 4.0.1 *Administration Guide.*<br>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`. |

# USERDB

Defines the user database used by the server.

## Subelements

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

**Table 2-19** USERDB subelements

| Element | Required | Description |
|---|---|---|
| `DESCRIPTION` | zero or one | Contains a text description of this element. |

## Attributes

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

**Table 2-20** USERDB attributes

| Attribute | Default | Description |
|---|---|---|
| `id` | none | The user database name in the server's ACL file. A user database name cannot begin with a number. |
| `database` | none | The user database name in the `dbswitch.conf` file. |

**Table 2-20** USERDB attributes

| Attribute | Default | Description |
|---|---|---|
| `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 server. |

# Cache Elements

Cache elements are as follows:

- FILECACHE
- CACHE
- PARTITION
- GC

## FILECACHE

Configures the in-memory cache.

### Subelements

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

**Table 2-21** FILECACHE subelements

| Element | Required | Description |
|---|---|---|
| `DESCRIPTION` | zero or one | Contains a text description of this element. |

### Attributes

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

**Table 2-22** FILECACHE attributes

| Attribute | Default | Description |
|---|---|---|
| `enabled` | true | Select this option, if not already selected. |
| `transmitfile` | false | When you enable Transmit File, the server caches open file descriptors for files in the file cache, rather than the file contents, and PR_TransmitFile is used to send the file contents to a client. When Transmit File is enabled, the distinction normally made by the file cache between small, medium, and large files no longer applies, because only the open file descriptor is being cached. |
| `contentcache` | true | Enables caching file content. |
| `tempdir` | | Specifies the directory to store temporary files. |
| `maxage` | 30 | The maximum age in seconds of a valid cache entry. This setting controls how long cached information will continue to be used once the file is cached. An entry older than MaxAge is replaced by a new entry for the same file, if the same file is referenced through the cache. |
| `mediumfilesizelimit` | 537600 | Size in bytes of the largest (non-small) file that is considered to be medium size. The contents of medium files are cached by mapping the file into virtual memory (currently only on Unix platforms). The contents of "large" files (larger than "medium") are not cached, although information about large files is cached. |
| `mediumfilespace` | 10485760 | Specifies how much virtual memory will be used to map all medium sized files. |
| `smallfilesizelimit` | 2048 | Size in bytes of the largest file that is considered to be "small". The contents of small files are cached by allocating heap space and reading the file into it. |
| `smallfilespace` | 1048576 | Specifies how much heap space will be used for the cache, including heap space used to cache small files. |
| `maxfiles` | 1024 | The maximum number of files that may be in the cache at once. |
| `hashinitsize` | 0 | |

# CACHE

Configures the disk cache.

## Subelements

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

**Table 2-23** CACHE subelements

| Element | Required | Description |
|---|---|---|
| `DESCRIPTION` | zero or one | Contains a text description of this element. |
| `PARTITION` | one or more | Cache partition is a reserved part of disk or memory that is set aside for caching purposes. |
| `GC` | zero or one | Cache garbage collector is used to delete files from the cache. Garbage collection can be done in either the automatic mode or the explicit mode. |

## Attributes

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

**Table 2-24** CACHE attributes

| Attribute | Default | Description |
|---|---|---|
| `enabled` | true | Select this option, if not already selected. |
| `cachedir` | *<Install_Root>*`/`*<Instance_Directory>*`/cache` | Specifies the directory for caching. |
| `cachecapacity` | 2000MB | The cache capacity should be set equal to or greater than the cache size. Setting the capacity larger than the cache size can be helpful if you know that you plan to increase the cache size later, such as by adding an external disk. |

# PARTITION

Configures the storage area on a disk that you set aside for caching. If you wish to have your cache span several disks, you need to configure at least one cache partition for each disk. Each partition can be independently administered. In other words, you can enable, disable, and configure a partition independently of all other partitions.

## Subelements

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

**Table 2-25** CACHE subelements

| Element | Required | Description |
|---|---|---|
| `DESCRIPTION` | zero or one | Contains a text description of this element. |

## Attributes

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

**Table 2-26** CACHE attributes

| Attribute | Default | Description |
|---|---|---|
| `enabled` | true | Select this option, if not already selected. |
| `partitiondir` | *<Install_Root>/<Instance_Directory>*`/cache` | Specify the directory where the partition is to be created. |
| `partitionname` | part1 | Specify a name for the partition. |
| `maxsize` | 1600MB | The optional number for the maximum size, in megabytes, to allow for the cache partition to grow. |
| `minspace` | 5MB | The minimum amount of available space, in megabytes, on the physical partition. This is the actual disk on which the cache partition resides. If less space is available, the proxy stops caching to that cache partition, even if it has not reached the maximum size (max-size). It continues to write to other partitions that are not full. |

# GC

Configures the cache garbage collector that deletes files from the cache. Garbage collection can be done in either the automatic mode or the explicit mode.

## Subelements

The following table describes subelements for the GC element.

**Table 2-27** CACHE subelements

| Element | Required | Description |
|---|---|---|
| DESCRIPTION | zero or one | Contains a text description of this element. |

## Attributes

The following table describes attributes for the GC element.

**Table 2-28** CACHE attributes

| Attribute | Default | Description |
|---|---|---|
| enabled | true | Select this option, if not already selected. |
| gchimargin | 80 | Controls the percentage of the maximum cache size that, when reached, triggers garbage collection. |
| gclomargin | 70 | Controls the percentage of the maximum cache size that the garbage collector targets. |
| gcleavefsfull | 60 | Determines the percentage of the cache partition size below which garbage collection will not go. |
| gcextramargin | 30 | Sets the percentage of the cache to be removed by the garbage collector. |

# The 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 267.

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.

## The 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 e-mail name (specifically, it must be a valid local-part as defined in RFC822). It is recommended that the `cn` contain *name initial sn*. It is recommended that the RDN of the user entry be 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`.

## The 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` would contain the attribute `inetDomainBaseDN` with value `o=Cust2,o=isp,o=root`.

If an `inetDomainStatus` attribute is present in an entry and 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="accesslog"
value="<Install_Root>/<Instance_Directory>/logs/access"/>
```

The variable is then used in the `obj.conf` file. For example:

```
Init fn="flex-init" access="$accesslog" format.access="%Ses->client.ip% -
%Req->vars.auth-user% [%SYSDATE%] '%Req->reqpb.clf-request%'
%Req->srvhdrs.clf-status% %Req->srvhdrs.content-length%"
```

Using this `accesslog` 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.

## Other Important Variables

In a default installation, the following variables are used to configure various aspects of the server's operation.

### 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 2-29** General Variables

| Property | Description |
|---|---|
| `accesslog` | The access log file for the server. |

## Variable Evaluation

Variables are evaluated when generating specific objectsets. Evaluation is recursive: variable values can contain other variables.

# 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
Proxy Server 4.0//EN"
"file:///space/proxy40/bin/proxy/dtds/sun-web-proxy-server_4_0.dtd">

<SERVER>

    <PROPERTY name="accesslog"
value="/space/proxy40/proxy-server1/logs/access"/>

<LS id="ls1" port="8080" servername="agneyam"/>

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

<ACLFILE id="acl1"
file="/space/proxy40/httpacl/generated.proxy-server1.acl"/>

<USERDB id="default"/>

<FILECACHE enabled="true"  maxage="30" mediumfilesizelimit="537600"
mediumfilespace="10485760" smallfilesizelimit="2048"
smallfilespace="1048576" transmitfile="false" maxfiles="1024"
hashinitsize="0"/>

<CACHE enabled="true" cachecapacity="2000"
cachedir="/space/proxy40/proxy-server1/cache">

    <PARTITION  partitionname="part1"
partitiondir="/space/proxy40/proxy-server1/cache" maxsize="1600"
minspace="5" enabled="true"/>

    <GC enabled="true" gchimargin="80" gclomargin="70" gcleavefsfull="60"
gcextramargin="30"/>

</CACHE>

<LOG file="/space/proxy40/proxy-server1/logs/errors" loglevel="finest"/>

</SERVER>
```

Chapter 3

# Syntax and Use of magnus.conf

When the Sun Java System Web Proxy 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 Proxy 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 Proxy Server 4.

The categories are:

- Server Information
- DNS Lookup
- Processes
- Error Logging and Statistic Collection
- Security
- Summary of Directives in magnus.conf

For an alphabetical list of directives, see Appendix C, "Alphabetical List of Server Configuration Elements."

# Server Information

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

- Server Name
- Server ID
- User
- NetsiteRoot

## Server Name

Specifies the server name.

## Server ID

Specifies the server ID.

## 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
```

### NetsiteRoot

Specifies the server root. This directive is set during installation and is commented out. Unlike other directives, the server expects this directive to start with #. Do not change this directive.

**Syntax**

`#NetsiteRoot` *path*

**Example**

`#ServerRoot` *<Install_Root>* / *<Instance_Directory>*

## DNS Lookup

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

- AsyncDNS
- DNS

### 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, 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.

### Syntax

```
DNS [on|off]
```

### Default

DNS host name resolution is `off` as a default

### Example

```
DNS off
```

# Processes

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

- MaxProcs (UNIX Only)

## 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 Proxy Server 4, there is always a primordial process in addition to the number of active processes specified by this setting.

### Default

1

# 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
- PidLog

## ErrorLogDateFormat

### Syntax

`ErrorLogDateFormat` *format*

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

### Default

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

## 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*

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

### Example

```
PidLog /home/xx12345/builds/install1/proxy-server1/logs/pid
```

# Security

This section describes the directive in `magnus.conf` that affects server access and security issues for Sun Java System Web Proxy Server 4.

- Security

## Security

The Security directive globally enables or disables SSL by making certificates available to the server instance. It must be on for the server 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
```

# Summary of Directives in magnus.conf

**Purpose**

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

Directives have the following syntax:

*directive value*

## Directives

The following table lists

**Table 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| `AcceptLanguage` | `on`, `off` | `off` | Determines whether (`on`) or not (`off`) the server parses the Accept-Language header sent by the client to indicate which languages the client accepts. |
| `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 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| `Address` | IP address | `not enabled` | When Address is enabled, proxy will bind all connect sockets (sockets used to connect to the web server) to the IP address specified in the directive.<br>If Address is "0.0.0.0", then proxy does not perform any bind operation and lets the operating system handle the binding of socket when connect() is called. |
| `AsyncDNS` | `on`, `off` | `off` | Specifies whether asynchronous DNS is allowed. |
| `CanonicalizeURI` | `0` (off), `1` (on) | `1` (on) | Enable / disable URI canonicalization. |
| `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. |
| `Chroot` | A path | (none) | (Unix only) lets the Unix system administrator place the server under a constraint such that it has access only to files in a given directory, termed the “Chroot directory”. |

**Table 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| 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 proxy server can have. |
| DefaultLanguage | en (English),<br>fr (French),<br>de (German),<br>ja (Japanese) | en | Specifies the default language for the server. The default language is used for both the client responses and administration. |
| 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 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). |
| flushTimer | Any number of milliseconds | 3000<br>(3 seconds). | If the interval in milliseconds between subsequent write operations for an application is greater than this value, further buffering is disabled. |

**Table 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| `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. |
| `KeepAliveQueryMaxSleepTime` | | `100`<br>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.<br>Note that CPU usage will increase with lower KeepAliveQueryMeanTime values. | This directive specifies the desired keep-alive latency in milliseconds. |

**Table 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| `KeepAliveIdleTime` | Any number of milliseconds | `200` | Specifies the idle time between polls within each thread in the keep-alive subsystem. |
| `KeepAlivePollTimeout` | Any number of milliseconds | `1000` | Specifies the timeout to the `poll()` call within each thread in the keep-alive subsystem. |
| `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. |
| `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 - 32768` | | Specifies the maximum number of Keep-Alive and persistent connections that the server can have open simultaneously. |
| `MaxProcs` | | `1` | (UNIX only) Specifies the maximum number of processes that the server can have running simultaneously. |

**Table 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| 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 | | 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. |
| 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) | | Specifies the maximum number of simultaneous request processing threads that the server can handle simultaneously per socket.<br>This setting can have performance implications. For more information, see the Sun Java System Web Proxy Server 4 *Performance Tuning, Sizing, and Scaling Guide.* |

**Table 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| RqThrottleMin | Any number less than RqThrottle | | 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. |
| 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. |

**Table 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| TempDir | A path | /tmp (UNIX)<br>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. |
| 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. |

**Table 3-1** magnus.conf directives

| Directive | Allowed Values | Default Value | Description |
|---|---|---|---|
| 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.<br>(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

# Syntax and Use of obj.conf

The obj.conf configuration file contains directives that instruct the Sun™ Java SystemWeb Proxy Server how to handle HTTP and HTTPS requests from clients. 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.

By default, the obj.conf file for the server is named obj.conf.

This chapter discusses server instructions in obj.conf, the use of OBJECT tags, the use of variables, the flow of control in obj.conf, the syntax rules for editing obj.conf, and a note about example directives.

**NOTE** For detailed information about the standard directives and predefined Server Application Functions (SAFs) that are used in the obj.conf file, see Chapter 5, "Predefined SAFs in obj.conf."

This chapter has the following sections:

- How the Proxy Server Functions
- Dynamic Reconfiguration
- Server Instructions in obj.conf
- Configuring HTTP Compression
- The Object and Client Tags
- Variables Defined in server.xml
- Flow of Control in obj.conf

- Changes in Function Flow
- Syntax Rules for Editing obj.conf
- About obj.conf Directive Examples

## How the Proxy Server Functions

'Proxy' is a general term that means 'to act on behalf of a user in an authorized capacity'. A web proxy server intercepts client connections and obtains the requested content from an origin server (the owner of the content) on behalf of the client.

Typical web proxies accept connections from clients, make decisions as to whether or not the clients are permitted to use the proxy (or access the requested resources), and then completes connections on behalf of the clients to the various origin servers. In this manner, the web proxy acts as both a server as well as a client of the requested resource.

There are two basic types of web proxy servers: a forward proxy and a reverse proxy. While they share much of the same functionality, there are some definite differences between the two.

## Forward Proxy Scenario

A forward proxy provides internal clients access through a firewall to resources on the Internet. This service is often provided as part of a larger intranet security strategy. Forward proxying allows clients to access resources outside of the firewall without compromising the integrity of the private network.

A forward proxy can be configured to keep copies of content within their local cache. Subsequent requests for that content can then be serviced from the local cache rather than obtaining it from the origin server. Caching increases performance by decreasing the time it takes to traverse the network.

Most proxy servers have the capability to filter requests from users. Administrators can choose to limit access to certain resources that may not be appropriate for the workplace and therefore deny such access.

In a forward proxy scenario, the client is aware of the proxy server and is configured to use it for various requests. The firewall can then be configured to only allow certain traffic from the proxy server rather than permitting such access to all internal clients.

## Reverse Proxy Scenario

A proxy server can also provide external clients with access to internal resources the reside behind the corporate firewall. When a proxy server is used to handle connections into a private network, the process is called reverse proxying. The term, reverse, refers to the fact that traffic flows in the opposite direction than normal proxy traffic flow.

While forward proxies are best used to filter content, increase performance, and log user accesses, reverse proxies provide these benefits and more. A reverse proxy may be used to load balance across multiple servers, provide fail-over capabilities, and provide access to corporate resources in a safe and secure manner.

In a reverse proxy scenario, the client is not even aware that it is using a proxy server. This is one of the key differences between a forward and reverse proxy server scenario.

## NSAPI Filters

The NSAPI API allows multiple Server Application Functions (SAFs) to interact in request processing. For example, one SAF could be used to authenticate the client after which a second SAF would generate the content.

## Steps in the Request-handling Process

When the server first starts up it performs some initialization and then waits for a request from a client (such as a browser).

The `obj.conf` file for the server specifies how the request is handled in the following steps:

1. **Init**

   The Init functions load and initialize server modules and plugins, and initialize log files.

2. **AuthTrans** (authorization translation)

   Verify any authorization information (such as name and password) sent in the request.

3. **NameTrans** (name translation)

   Translate the logical URI into a local file system path.

4. **PathCheck** (path checking)

   Check the local file system path for validity and check that the requestor has access privileges to the requested resource on the file system.

5. **ObjectType** (object typing)

   Determine the MIME-type (Multi-purpose Internet Mail Encoding) of the requested resource (for example, `text/html`, `image/gif`, and so on).

6. **Input** (prepare to read input)

   Select filters that will process incoming request data read by the `Service` step.

7. **Output** (prepare to send output)

   Select filters that will process outgoing response data generated by the `Service` step.

8. **Service** (generate the response)

   Generate and return the response to the client.

9. **AddLog** (adding log entries)

   Add entries to log file(s).

10. **Error** (service)

    This step is executed only if an error occurs in the previous steps. If an error occurs, the server logs an error message and aborts the process.

11. **Connect**

    Call the connect function you specify.

12. **DNS**

    Call either the `dns-config` built-in function or a DNS function that you specify.

13. **Filter**

    Run an external command and then pipe the data through the external command before processing that data in the proxy.

14. **Route**

    Specify information about where the proxy server should route requests.

## Directives for Handling Requests

The file `obj.conf` contains a series of instructions, known as directives, that tell the Sun Java System Web Proxy Server what to do at each stage in the request-handling process. Each directive invokes a Server Application Function (SAF) with one or more arguments. Each directive applies to a specific stage in the request-handling process. The stages are `Init`, `AuthTrans`, `NameTrans`, `PathCheck`, `ObjectType`, `Input`, `Output`, `Service`, `AddLog`, `Connect`, `DNS`, `Filter`, and `Route`..

# Dynamic Reconfiguration

You do not need to restart the server for changes to certain configuration files to take effect (for example, `obj.conf`, `mime.types`, and `server.xml`,). All you need to do is apply the changes by clicking the Apply link and then clicking the Load Configuration Files button on the Apply Changes screen. If there are errors in installing the new configuration, the previous configuration is restored.

When you edit `obj.conf` and apply the changes, a new configuration is loaded into memory that contains all of the information from the dynamically configurable files.

Every new connection references the newest configuration. Once the last session referencing a configuration ends, the now unused old configuration is deleted.

# Server Instructions in obj.conf

The `obj.conf` file contains directives that instruct the server how to handle requests received from clients such as browsers. These directives appear inside `OBJECT` tags.

Each directive calls a function, indicating when to call it and specifying arguments for it.

The syntax of each directive is:

`Directive fn=`*func-name name1*`="`*value1*`"...`*nameN*`="`*valueN*`"`

For example:

```
Init fn="flex-init" access="$accesslog" format.access="%Ses->client.ip% -
%Req->vars.auth-user% [%SYSDATE%] '%Req->reqpb.clf-request%'
%Req->srvhdrs.clf-status% %Req->srvhdrs.content-length%”
```

Directive indicates when this instruction is executed during the request-handling process. The value is one of Init, AuthTrans, NameTrans, PathCheck, ObjectType, Service, AddLog, Error, Connect, DNS, Filter, and Route.

The value of the fn argument is the name of the SAF to execute. All directives must supply a value for the fn parameter; if there's no function, the instruction won't do anything.

The remaining parameters are the arguments needed by the function, and they vary from function to function.

Sun Java System Web Proxy Server is shipped with a set of built-in Server Application Functions (SAFs) that you can use to create and modify directives in obj.conf.

## Summary of the Directives

Following are the categories of server directives and a description of what each does. Each category corresponds to a stage in the request-handling process. The section "Flow of Control in obj.conf" on page 80explains exactly how the server decides which directive or directives to execute in each stage.

- Init

  The Init functions load and initialize server modules and plugins, and initialize log files.

- AuthTrans

  Verifies any authorization information (normally sent in the Authorization header) provided in the HTTP request and translates it into a user and / or a group. Server access control occurs in two stages. AuthTrans verifies the authenticity of the user. Later, PathCheck tests the user's access privileges for the requested resource.

  ```
  AuthTrans fn=basic-auth userfn=ntauth auth-type=basic userdb=none
  ```

  This example calls the basic-auth function, which calls a custom function (in this case ntauth, to verify authorization information sent by the client. The Authorization header is sent as part of the basic server authorization scheme.

- NameTrans

  Translates the URL specified in the request from a logical URL to a physical file system path for the requested resource. This may also result in redirection to another site. For example:

```
NameTrans fn="document-root" root="D:/Sun/WebServer61/Server1/docs"
```

This example calls the `document-root` function with a `root` argument of `D:/Sun/WebServer61/Server1/docs`. The function `document-root` function translates the `http://`*server_name*`/` part of the requested URL to the document root, which in this case is `D:/Sun/WebServer61/Server1/docs`. Thus a request for `http://`*server-name*`/doc1.html` is translated to `D:/Sun/WebServer61/Server1/docs/doc1.html`.

- `PathCheck`

  Performs tests on the physical path determined by the `NameTrans` step. In general, these tests determine whether the path is valid and whether the client is allowed to access the requested resource. For example:

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

  This example calls the `find-index` function with an `index-names` argument of `index.html,home.html`. If the requested URL is a directory, this function instructs the server to look for a file called either `index.html` or `home.html` in the requested directory.

- `ObjectType`

  Determines the MIME (Multi-purpose Internet Mail Encoding) type of the requested resource. The MIME type has attributes `type` (which indicates content type), `encoding`, and `language`. The MIME type is sent in the headers of the response to the client. The MIME type also helps determine which `Service` directive the server should execute.

  The resulting type may be:

  - A common document type such as `text/html` or `image/gif` (for example, the file name extension `.gif` translates to the MIME type `image/gif`).
  - An internal server type. Internal types always begin with `magnus-internal`.

  For example:

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

  This example calls the `type-by-extension` function, which causes the server to determine the MIME type according to the requested resource’s file extension.

- `Input`

Selects filters that will process incoming request data read by the Service step. The Input directive allows you to invoke the insert-filter SAF in order to install filters that process incoming data. All Input directives are executed when the server or a plugin first attempts to read entity body data from the client. The Input directives are executed at most once per request. For example:

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

This directive instructs the insert-filter function to add a filter named http-decompression to the filter stack, which would decompress incoming HTTP request data before passing it to the Service step.

- Output

  Selects filters that will process outgoing response data generated by the Service step. The Output directive allows you to invoke the insert-filter SAF to install filters that process outgoing data. All Output directives are executed when the server or a plugin first attempts to write entity body data from the client. The Output directives are executed at most once per request. For example:

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

  This directive instructs the insert-filter function to add a filter named http-compression to the filter stack, which would compress outgoing HTTP response data generated by the Service step.

- Service

  Generates and sends the response to the client. This involves setting the HTTP result status, setting up response headers (such as Content-Type and Content-Length), and generating and sending the response data. The default response is to invoke the send-file function to send the contents of the requested file along with the appropriate header files to the client.

  The default Service directive is:

  ```
  Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
  fn="send-file"
  ```

  This directive instructs the server to call the send-file function in response to any request whose method is GET, HEAD, or POST, and whose type does not begin with magnus-internal/. (Note the use of the special characters *~ to mean “does not match.”)

  Another example is:

  ```
  Service method="(GET|HEAD)" type="magnus-internal/imagemap"
  fn="imagemap"
  ```

In this case, if the method of the request is either GET or HEAD, and the type of the requested resource is "magnus-internal/imagemap," the function imagemap is called.

- AddLog

  Adds an entry to a log file to record information about the transaction. For example:

  ```
  AddLog fn="flex-log" name="access"
  ```

  This example calls the flex-log function to log information about the current request in the log file named access.

- Error

  Handles an HTTP error. This directive is invoked if a previous directive results in an error. Typically the server handles an error by sending a custom HTML document to the user describing the problem and possible solutions.

  For example:

  ```
  Error fn="send-error" reason="Unauthorized"
  path="D:/Sun/WebServer61/Server1/errors/unauthorized.html"
  ```

  In this example, the server sends the file in D:/Sun/WebServer61/Server1/errors/unauthorized.html whenever a client requests a resource that it is not authorized to access.

- Connect

  The Connect directive calls the connect function you specify.

  Only the first applicable Connect function is called, starting from the most restrictive object. Occasionally it is desirable to call multiple functions (until a connection is established). The function returns REQ_NOACTION if the next function should be called. If it fails to connect, the return value is REQ_ABORT. If it connects successfully, the connected socket descriptor will be returned.

- DNS

  The DNS directive calls either the dns-config built-in function or a DNS function that you specify.

- Filter

  The Filter directive runs an external command and then pipes the data through the external command before processing that data in the proxy. This is accomplished using the pre-filter function.

- Route

  The Route directive specifies information about where the proxy server should route requests.

# Configuring HTTP Compression

When compression is enabled in the server, an entry gets added to the obj.conf file. A sample entry is shown below:

```
Output fn="insert-filter" filter="http-compression" type="text/*"
```

Depending on the options specified, this line might also contain these options:

```
vary="on" compression-level="9"
```

To restrict compression to documents of only a particular type, or to exclude browsers that don’t work well with compressed content, you would need to edit the obj.conf file, as discussed below.

The option that appears as:

```
type="text/*"
```

restricts compression to documents that have a MIME type of text/* (for example, text/ascii, text/css, text/html, and so on). This can be modified to compress only certain types of documents. If you want to compress only HTML documents, for example, you would change the option to:

```
type="text/html"
```

Alternatively, you can specifically exclude browsers that are known to misbehave when they receive compressed content (but still request it anyway) by using the <Client> tag as follows:

```
<Client match="none"\
 browser="*MSIE [1-3]*"\
 browser="*MSIE [1-5]*Mac*"\
 browser="Mozilla/[1-4]*Nav*">
Output fn="insert-filter" filter="http-compression" type="text/*"
</Client>
```

This restricts compression to browsers that are *not* any of the following:

- Internet Explorer for Windows earlier than version 4
- Internet Explorer for Macintosh earlier than version 6
- Netscape Navigator/Communicator earlier than version 6

Internet Explorer on Windows earlier than version 4 may request compressed data at times, but does not correctly support it. Internet Explorer on Macintosh earlier than version 6 does the same. Netscape Communicator version 4.x requests compression, but only correctly handles compressed HTML. It will not correctly handle linked CSS or JavaScript from the compressed HTML, so administrators often simply prevent their servers from sending any compressed content to that browser (or earlier).

For more information about the `<Client>` tag, see "The Client Tag" on page 77.

# The Object and Client Tags

This section discusses the use of `<Object>` and `<Client>` tags in the file `obj.conf`.

`<Object>` tags group directives that apply to requests for particular resources, while `<Client>` tags group directives that apply to requests received from specific clients.

These tags are described in the following topics:

- The Object Tag
- The Client Tag

## The Object Tag

Directives in the `obj.conf` file are grouped into objects that begin with an `<Object>` tag and end with an `</Object>` tag. The default object provides instructions to the server about how to process requests by default. Each new object modifies the default object's behavior.

An `Object` tag may have a `name` attribute or a `ppath` attribute. Either parameter may be a wildcard pattern. For example:

```
<Object name="cgi">
```

- or -

```
<Object ppath="/usr/sun/webserver61/server1/docs/private/*">
```

The server always starts handling a request by processing the directives in the default object. However, the server switches to processing directives in another object after the `NameTrans` stage of the default object if either of the following conditions is true:

- The successful `NameTrans` directive specifies a `name` argument.
- The physical path name that results from the `NameTrans` stage matches the `ppath` attribute of another object.

When the server has been alerted to use an object other than the default object, it processes the directives in the other object before processing the directives in the default object. For some steps in the process, the server stops processing directives in that particular stage (such as the `Service` stage) as soon as one is successfully executed, whereas for other stages the server processes all directives in that stage, including the ones in the default object as well as those in the additional object. For more details, see "Flow of Control in obj.conf" on page 80

## Objects that Use the name Attribute

If a `NameTrans` directive in the default object specifies a `name` argument, the server switches to processing the directives in the object of that name before processing the remaining directives in the default object.

For example, the following `NameTrans` directive in the default object assigns the name `cgi` to any request whose URL starts with `http://`*server_name*`/cgi/`:

```
<Object name="default">
NameTrans fn="pfx2dir" from="/cgi" dir="<Install_Root>/<Instance_Directory>/mycgi"
name="cgi"
...
</Object>
```

When that `NameTrans` directive is executed, the server starts processing directives in the object named `cgi`:

```
<Object name="cgi">
more directives...
</Object>
```

### Objects that Use the ppath Attribute

When the server finishes processing the NameTrans directives in the default object, the logical URL of the request will have been converted to a physical path name. If this physical path name matches the ppath attribute of another object in obj.conf, the server switches to processing the directives in that object before processing the remaining ones in the default object.

For example, the following NameTrans directive translates the http://*server_name/* part of the requested URL to <*Install_Root*>/<*Instance_Directory*>/docs (which is the document root directory):

```
<Object name="default">
NameTrans fn="document-root" root="<Install_Root>/<Instance_Directory>/docs"
...
</Object>
```

The URL http://*server_name*/internalplan1.html would be translated to <*Install_Root*>/<*Instance_Directory*>/docs/internalplan1.html. However, suppose that obj.conf contains the following additional object:

```
<Object ppath="*internal*">
more directives...
</Object>
```

In this case, the partial path *internal* matches the path <*Install_Root*>/<*Instance_Directory*>/docs/internalplan1.html. So now the server starts processing the directives in this object before processing the remaining directives in the default object.

## The Client Tag

The <Client> tag is used to limit execution of a set of directives to requests received from specific clients. Directives listed between the <Client> and </Client> tags are executed only when information in the client request matches the parameter values specified.

## Client Tag Parameters

The following table lists the `<Client>` tag parameters.

**Table 4-1** Client Tag Parameters

| Parameter | Description |
|---|---|
| `browser` | User-agent string sent by a browser to the Web Server |
| `chunked` | Boolean value set by a client requesting chunked encoding |
| `code` | HTTP response code |
| `dns` | DNS name of the client |
| `internal` | Boolean value indicating internally generated request |
| `ip` | IP address of the client |
| `keep-alive` | Boolean value indicating the client has requested a keep-alive connection |
| `keysize` | Key size used in an SSL transaction |
| `match` | Match mode for the `<Client>` tag; valid values are `all`, `any`, and `none` |
| `method` | HTTP method used by the browser |
| `name` | Name of an object as specified in a previous `NameTrans` statement |
| `odds` | Sets a random value for evaluating the enclosed directive; specified as either a percentage or a ratio (for example, 20% or 1/5) |
| `path` | Physical path to the requested resource |
| `ppath` | Physical path of the requested resource |
| `query` | Query string sent in the request |
| `reason` | Text version of the HTTP response code |
| `restarted` | Boolean value indicating a request has been restarted |
| `secret-keysize` | Secret key size used in an SSL transaction |
| `security` | Indicates an encrypted request |
| `type` | Type of document requested (such as `text/html` or `image/gif`) |
| `uri` | URI section of the request from the browser |
| `urlhost` | DNS name of the virtual server requested by the client (the value is provided in the `Host` header of the client request) |

The `<Client>` tag parameters provide greater control over when and if directives are executed. In the following example, use of the `odds` parameter gives a request a 25% chance of being redirected:

```
<Client odds="25%">
NameTrans fn="redirect" from="/Pogues"
url-prefix="http://pogues.example.com"
</Client>
```

One or more wildcard patterns can be used to specify Client tag parameter values.

Wildcards can also be used to exclude clients that match the parameter value specified in the <`Client` tag>. In the following example, the `<Client>` tag and the `AddLog` directive are combined to direct the Web Server to log access requests from all clients *except* those from the specified subnet:

```
<Client ip="~192.85.250.*">
AddLog fn="flex-log" name="access"
</Client>
```

Using the ~ wildcard negates the expression, so the Web Server excludes clients from the specified subnet.

You can also create a negative match by setting the `match` parameter of the `Client` tag to `none`. In the following example, access requests from the specified subnet are excluded, as are all requests to the server `www.mycompany.com`:

```
<Client match="none" ip="192.85.250.*" urlhost="www.mycompany.com">
AddLog fn="flex-log" name="access"
</Client>
```

# Variables Defined in server.xml

You can define variables in the server.xml file and reference them in an obj.conf file. For example, the following server.xml code defines and uses a variable called docroot:

```
<!DOCTYPE SERVER SYSTEM "server.dtd" [
<!ELEMENT LS (DESCRIPTION?,SSLPARAMS?)>

<!ATTLIST LS

          id ID #REQUIRED

          ip CDATA "any"

          port CDATA #REQUIRED

          security %boolean; "false"

          acceptorthreads CDATA "1"

          family CDATA #IMPLIED

          blocking %boolean; "false"

          servername CDATA #REQUIRED

>
```

You can reference the variable in obj.conf as follows:

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

Using this docroot variable saves you from having to define document roots for virtual server classes in the obj.conf files. It also allows you to define different document roots for different virtual servers within the same virtual server class.

| | |
|---|---|
| **NOTE** | Variable substitution is allowed only in an obj.conf file. It is not allowed in any other Sun Java System Web Proxy Server configuration files. Any variable referenced in an obj.conf file must be defined in the server.xml file. |

# Flow of Control in obj.conf

Before the server can process a request, it must direct the request to the correct server.

After the server is determined, the server executes the `obj.conf` file for the server. This section discusses how the server decides which directives to execute in `obj.conf`.

## Init

The Init functions load and initialize server modules and plugins, and initialize log files.

## AuthTrans

When the server receives a request, it executes the `AuthTrans` directives in the default object to check that the client is authorized to access the server.

If there is more than one `AuthTrans` directive, the server executes them all (unless one of them results in an error). If an error occurs, the server skips all other directives except for `Error` directives.

## NameTrans

Next, the server executes a `NameTrans` directive in the default object to map the logical URL of the requested resource to a physical path name on the server's file system. The server looks at each `NameTrans` directive in the default object in turn, until it finds one that can be applied.

If there is more than one `NameTrans` directive in the default object, the server considers each directive until one succeeds.

The `NameTrans` section in the default object must contain exactly one directive that invokes the `document-root` function. This functions translates the `http://`*server_name*`/`part of the requested URL to a physical directory that has been designated as the server's document root. For example:

```
NameTrans fn="document-root" root="<Install_Root>/<Instance_Directory>/docs"
```

The directive that invokes `document-root` must be the last directive in the `NameTrans` section so that it is executed if no other `NameTrans` directive is applicable.

The `pfx2dir` (prefix to directory) function is used to set up additional mappings between URLs and directories. For example, the following directive translates the URL `http://`*server_name*`/cgi/` into the directory path name <*Install_Root*>`/`<*Instance_Directory*>`/docs/mycgi/`:

```
NameTrans fn="pfx2dir" from="/cgi"
dir="<Install_Root>/<Instance_Directory>/docs/mycgi"
```

Notice that if this directive appeared *after* the one that calls `document-root`, it would never be executed, with the result that the resultant directory path name would be <*Install_Root*>/<*Instance_Directory*>`/docs/cgi/` (not `mycgi`). This illustrates why the directive that invokes `document-root` must be the last one in the `NameTrans` section.

## How and When the Server Processes Other Objects

As a result of executing a `NameTrans` directive, the server might start processing directives in another object. This happens if the `NameTrans` directive that was successfully executed specifies a name or generates a partial path that matches the `name` or `ppath` attribute of another object.

If the successful `NameTrans` directive assigns a name by specifying a `name` argument, the server starts processing directives in the named object (defined with the `OBJECT` tag) before processing directives in the default object for the rest of the request-handling process.

For example, the following `NameTrans` directive in the default object assigns the name `cgi` to any request whose URL starts with `http://`*server_name*`/cgi/`.

```
<Object name="default">
...
NameTrans fn="pfx2dir" from="/cgi"
dir="<Install_Root>/<Instance_Directory>/docs/mycgi" name="cgi"
...
</Object>
```

When that `NameTrans` directive is executed, the server starts processing directives in the object named `cgi`:

```
<Object name="cgi">
more directives...
</Object>
```

When a NameTrans directive has been successfully executed, there will be a physical path name associated with the requested resource. If the resultant path name matches the ppath (partial path) attribute of another object, the server starts processing directives in the other object before processing directives in the default object for the rest of the request-handling process.

For example, suppose obj.conf contains an object as follows:

```
<Object ppath="*internal*">
more directives...
</Object>
```

Now suppose the successful NameTrans directive translates the requested URL to the path name *<Install_Root>*/*<Instance_Directory>*/docs/internalplan1.html. In this case, the partial path *internal* matches the path *<Install_Root>*/*<Instance_Directory>*/docs/internalplan1.html. So now the server would start processing the directives in this object before processing the remaining directives in the default object.

## PathCheck

After converting the logical URL of the requested resource to a physical path name in the NameTrans step, the server executes PathCheck directives to verify that the client is allowed to access the requested resource.

If there is more than one PathCheck directive, the server executes all of the directives in the order in which they appear, unless one of the directives denies access. If access is denied, the server switches to executing directives in the Error section.

If the NameTrans directive assigned a name or generated a physical path name that matches the name or ppath attribute of another object, the server first applies the PathCheck directives in the matching object before applying the directives in the default object.

# ObjectType

Assuming that the PathCheck directives all approve access, the server next executes the ObjectType directives to determine the MIME type of the request. The MIME type has three attributes: type, encoding, and language. When the server sends the response to the client, the type, language, and encoding values are transmitted in the headers of the response. The type also frequently helps the server to determine which Service directive to execute to generate the response to the client.

If there is more than one ObjectType directive, the server applies all of the directives in the order in which they appear. However, once a directive sets an attribute of the MIME type, further attempts to set the same attribute are ignored. The reason that all ObjectType directives are applied is that one directive may set one attribute, for example type, while another directive sets a different attribute, such as language.

As with the PathCheck directives, if another object has been matched to the request as a result of the NameTrans step, the server executes the ObjectType directives in the matching object before executing the ObjectType directives in the default object.

## Setting the Type By File Extension

Usually the default way the server figures out the MIME type is by calling the type-by-extension function. This function instructs the server to look up the MIME type according to the requested resource's file extension in the MIME types table. This table was created during virtual server initialization by the MIME types file (which is usually called mime.types).

For example, the entry in the MIME types table for the extensions .html and .htm is usually:

```
type=text/html  exts=htm,html
```

which says that all files with the extension .htm or .html are text files formatted as HTML, and the type is text/html.

Note that if you make changes to the MIME types file, you must reconfigure the server before those changes can take effect.

## Forcing the Type

If no previous ObjectType directive has set the type, and the server does not find a matching file extension in the MIME types table, the type still has no value even after type-by-expression has been executed. Usually if the server does not recognize the file extension, it is a good idea to force the type to be text/plain, so that the content of the resource is treated as plain text. There are also other situations where you might want to set the type regardless of the file extension, such as forcing all resources in the designated CGI directory to have the MIME type `magnus-internal/cgi`.

The function that forces the type is `force-type`.

For example, the following directives first instruct the server to look in the MIME types table for the MIME type, then if the `type` attribute has not been set (that is, the file extension was not found in the MIME types table), set the `type` attribute to `text/plain`.

```
ObjectType fn="type-by-extension"
ObjectType fn="force-type" type="text/plain"
```

If the server receives a request for a file `abc.dogs`, it looks in the MIME types table, does not find a mapping for the extension `.dogs`, and consequently does not set the `type` attribute. Since the `type` attribute has not already been set, the second directive is successful, forcing the `type` attribute to `text/plain`.

The following example illustrates another use of `force-type`. In this example, the `type` is forced to `magnus-internal/cgi` before the server gets a chance to look in the MIME types table. In this case, all requests for resources in http://*server_name*/cgi/ are translated into requests for resources in the directory <*Install_Root*>/<*Instance_Directory*>/docs/mycgi/. Since a `name` is assigned to the request, the server processes `ObjectType` directives in the object named `cgi` before processing the ones in the default object. This object has one `ObjectType` directive, which forces the `type` to be `magnus-internal/cgi`.

```
NameTrans fn="pfx2dir" from="/cgi"
dir="<Install_Root>/<Instance_Directory>/docs/mycgi" name="cgi"
<Object name="cgi">
ObjectType fn="force-type" type="magnus-internal/cgi"
Service fn="send-cgi"
</Object>
```

The server continues processing all `ObjectType` directives including those in the default object, but since the `type` attribute has already been set, no other directive can set it to another value.

## Input

The `Input` directive selects filters that will process incoming request data read by the `Service` step. It allows you to invoke the `insert-filter` SAF in order to install filters that process incoming data.

The `Input` directives are executed at most once per request.

You can 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 `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.

## Output

The `Output` directive selects filters that will process outgoing response data generated by the `Service` step. The `Output` directive allows you to invoke the `insert-filter` SAF to install filters that process outgoing data. All `Output` directives are executed when the server or a plugin first attempts to write entity body data from the client.

The `Output` directives are executed at most once per request.

You can 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, `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.

# Service

Next, the server needs to execute a `Service` directive to generate the response to send to the client. The server looks at each `Service` directive in turn, to find the first one that matches the type, method and query string. If a `Service` directive does not specify type, method, or query string, then the unspecified attribute matches anything.

If there is more than one `Service` directive, the server applies the first one that matches the conditions of the request, and ignores all remaining `Service` directives.

As with the `PathCheck` and `ObjectType` directives, if another object has been matched to the request as a result of the `NameTrans` step, the server considers the `Service` directives in the matching object before considering the ones in the default object. If the server successfully executes a `Service` directive in the matching object, it will not get around to executing the `Service` directives in the default object, since it only executes one `Service` directive.

## Service Examples

For an example of how `Service` directives work, consider what happens when the server receives a request for the URL `D:/`*server_name*`/jos.html`. In this case, all directives executed by the server are in the default object.

- The following `NameTrans` directive translates the requested URL to <*Install_Root*>`/`<*Instance_Directory*>`/docs/jos.html`:

  ```
  NameTrans fn="document-root" root="<Install_Root>/<Instance_Directory>/docs"
  ```

- Assume that the `PathCheck` directives all succeed.

- The following `ObjectType` directive tells the server to look up the resource's MIME type in the MIME types table:

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

- The server finds the following entry in the MIME types table, which sets the `type` attribute to `text/html`:

  ```
  type=text/html exts=htm,html
  ```

- The server invokes the following `Service` directive. The value of the `type` parameter matches anything that does *not* begin with `magnus-internal/`. This directive sends the requested file, `jos.html`, to the client.

```
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*" fn="send-file""
```

Here is an example that involves using another object:

- The following NameTrans directive assigns the name personnel to the request.

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

- As a result of the name assignment, the server switches to processing the directives in the object named personnel. This object is defined as:

```
<Object name="personnel">
Service fn="index-simple"
</Object>
```

- The personnel object has no PathCheck or ObjectType directives, so the server processes the PathCheck and ObjectType directives in the default object. Let's assume that all PathCheck and ObjectType directives succeed.
- When processing Service directives, the server starts by considering the Service directive in the personnel object, which is:

  `Service fn="index-simple"`
- The server executes this Service directive, which calls the index-simple function.

  Since a Service directive has now been executed, the server does not process any other Service directives. (However, if the matching object had not had a Service directive that was executed, the server would continue looking at Service directives in the default object.)

## Default Service Directive

There is usually a Service directive that does the default task (sends a file) if no other Service directive matches a request sent by a browser. This default directive should come last in the list of Service directives in the default object, to ensure it only gets called if no other Service directives have succeeded. The default Service directive is usually:

```
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*" fn="send-file"
```

This directive matches requests whose method is GET, HEAD, or POST, which covers nearly virtually all requests sent by browsers. The value of the type argument uses special pattern-matching characters.

The characters "*~" mean "anything that doesn't match the following characters," so the expression *~magnus-internal/ means "anything that doesn't match magnus-internal/." An asterisk by itself matches anything, so the whole expression *~magnus-internal/* matches anything that does not begin with magnus-internal/.

So if the server has not already executed a Service directive when it reaches this directive, it executes the directive so long as the request method is GET, HEAD or POST, and the value of the type attribute does not begin with magnus-internal/. The invoked function is send-file, which simply sends the contents of the requested file to the client.

## AddLog

After the server generate the response and sends it to the client, it executes AddLog directives to add entries to the log files.

All AddLog directives are executed. The server can add entries to multiple log files.

Depending on which log files are used and which format they use, the Init section in magnus.conf may need to have directives that initialize the logs. For example, if one of the AddLog directives calls flex-log, which uses the extended log format, the Init section must contain a directive that invokes flex-init to initialize the flexible logging system.

For more information about initializing logs, see the discussion of the functions flex-init and init-clf in Chapter 5, "Predefined SAFs in obj.conf."

## Error

If an error occurs during the request-handling process, such as if a PathCheck or AuthTrans directive denies access to the requested resource, or the requested resource does not exist, the server immediately stops executing all other directives and immediately starts executing the Error directives.

## Connect

The Connect directive calls the connect function you specify.

Only the first applicable `Connect` function is called, starting from the most restrictive object. Occasionally it is desirable to call multiple functions (until a connection is established). The function returns REQ_NOACTION if the next function should be called. If it fails to connect, the return value is REQ_ABORT. If it connects successfully, the connected socket descriptor will be returned.

## DNS

The `DNS` directive calls either the `dns-config` built-in function or a DNS function that you specify.

## Filter

The `Filter` directive runs an external command and then pipes the data through the external command before processing that data in the proxy. This is accomplished using the `pre-filter` function.

## Route

The Route directive specifies information about where the proxy server should route requests.

# Changes in Function Flow

There are times when the function flow changes from the normal request-handling process. This happens during internal redirects, restarts, and URI translation functions.

## Internal Redirects

An example of an internal redirect is a servlet include or forward. In this case, because there is no exposed NSAPI function to handle an internal redirect, when an internal redirect occurs, the `request` structure is copied into `rq->orig_rq`.

## Restarts

A restart occurs when a REQ_RESTART is returned from a PathCheck or Service function. For example, when a CGI is redirected using a relative path.

On a restart, much of the request is cleared. Some elements of the HTTP request (rq->reqpb), the server's "working" variables (rq->vars), and response headers (rq->srvhdrs) are cleared. The method, protocol, and clf-request variables from rq->reqpb are saved. The saved variables are put back into the data structure. The new URI is inserted (and if there is a query string in the new URI, that too is inserted) into rq->reqpb. The parameter rq->rq_attr.req_restarted is set to 1.

## URI Translation

At times it is necessary to find the physical path for a URI without actually running a request. The function request_translate_uri does this. A new request structure is created and run through the AuthTrans and NameTrans stages to get the physical path. Thereafter, the new request is freed.

# Syntax Rules for Editing obj.conf

Several rules are important in the obj.conf file. Be very careful when editing this file. Simple mistakes can make the server fail to start or operate correctly.

**CAUTION** Do not remove any directives from any obj.conf file that are present in the obj.conf file that exists when you first install Sun Java System Web Proxy Server. The server may not function properly.

## Order of Directives

The order of directives is important, since the server executes them in the order they appear in obj.conf. The outcome of some directives affect the execution of other directives.

For PathCheck directives, the order within the PathCheck section is not so important, since the server executes all PathCheck directives. However, the order within the ObjectType section is very important, because if an ObjectType directive sets an attribute value, no other ObjectType directive can change that value. For example, if the default ObjectType directives were listed in the following order (which is the wrong way around), every request would have its type value set to text/plain, and the server would never have a chance to set the type according to the extension of the requested resource.

```
ObjectType fn="force-type" type="text/plain"
ObjectType fn="type-by-extension"
```

Similarly, the order of directives in the Service section is very important. The server executes the first Service directive that matches the current request and does not execute any others.

## Parameters

The number and names of parameters depends on the function. The order of parameters on the line is not important.

## Case Sensitivity

Items in the obj.conf file are case-sensitive including function names, parameter names, many parameter values, and path names.

## Separators

The C language allows function names to be composed only of letters, digits, and underscores. You may use the hyphen (-) character in the configuration file in place of underscore (_) for your C code function names. This is only true for function names.

## Quotes

Quotes (") are only required around value strings when there is a space in the string. Otherwise they are optional. Each open-quote must be matched by a close-quote.

## Spaces

- Spaces are not allowed at the beginning of a line except when continuing the previous line.
- Spaces are not allowed before or after the equal (=) sign that separates the name and value.
- Spaces are not allowed at the end of a line or on a blank line.

## Line Continuation

A long line may be continued on the next line by beginning the next line with a space or tab.

## Path Names

Always use forward slashes (/) rather than backslashes (\) in path names under Windows. Backslash escapes the next character.

## Comments

Comments begin with a pound (#) sign. If you manually add comments to `obj.conf`, then use the Server Manager interface to make changes to your server, the Server Manager will wipe out your comments when it updates `obj.conf`.

# About obj.conf Directive Examples

Every line in the `obj.conf` file begins with one of the following keywords:

```
Init
AuthTrans
NameTrans
PathCheck
ObjectType
Input
Output
Service
AddLog
```

```
Error
Connect
DNS
Filter
Route
<Object>
</Object>
```

If any line of any example begins with a different word in the manual, the line is wrapping in a way that it does not in the actual file. In some cases this is due to line length limitations imposed by the PDF and HTML formats of the manuals.

For example, the following directive is all on one line in the actual `obj.conf` file:

```
Init fn="flex-init" access="$accesslog" format.access="%Ses->client.ip% -
%Req->vars.auth-user% [%SYSDATE%] '%Req->reqpb.clf-request%'
%Req->srvhdrs.clf-status% %Req->srvhdrs.content-length%"
```

Chapter 5

# Predefined SAFs in obj.conf

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.

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 Proxy 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:

- Init
- AuthTrans
- NameTrans
- PathCheck
- ObjectType
- Input
- Output
- Service
- AddLog
- Error

- Connect
- DNS
- Filter
- Route

For an alphabetical list of predefined SAFs, see Appendix D, “Alphabetical List of Predefined SAFs.”

The following table lists the SAFs that can be used with each directive.

**Table 5-1** Available Server Application Functions (SAFs) Per Directive

| Directive | Server Application Functions |
|---|---|
| Init | define-perf-bucket<br>host-dns-cache-init<br>flex-init<br>flex-rotate-init<br>icp-init<br>init-clf<br>init-filter-order<br>init-j2ee<br>init-proxy<br>init-uhome<br>init-url-filter<br>ip-dns-cache-init<br>load-modules<br>load-types<br>nt-console-init<br>pa-init-parent-array<br>pa-init-proxy-array<br>perf-init<br>pool-init<br>register-http-method<br>stats-init<br>suppress-request-headers<br>thread-pool-init<br>tune-cache<br>tune-proxy |
| AuthTrans | basic-auth<br>basic-ncsa<br>get-sslid<br>match-browser<br>proxy-auth<br>set-variable |

**Table 5-1** Available Server Application Functions (SAFs) Per Directive

| Directive | Server Application Functions |
|---|---|
| NameTrans | assign-name<br>document-root<br>home-page<br>map<br>match-browser<br>ntrans-j2ee<br>pac-map<br>pat-map<br>pfx2dir<br>redirect<br>set-variable<br>strip-params<br>unix-home |
| PathCheck | block-multipart-posts<br>check-acl<br>deny-existence<br>deny-service<br>find-compressed<br>find-index<br>find-links<br>find-pathinfo<br>get-client-cert<br>load-config<br>match-browser<br>nt-uri-clean<br>ntcgicheck<br>require-auth<br>require-proxy-auth<br>set-variable<br>set-virtual-index<br>ssl-check<br>ssl-logout<br>unix-uri-clean<br>url-check<br>url-filter<br>user-agent-check |

**Table 5-1** Available Server Application Functions (SAFs) Per Directive

| Directive | Server Application Functions |
|---|---|
| `ObjectType` | `block-auth-cert`<br>`block-cache-info`<br>`block-cipher`<br>`block-ip`<br>`block-issuer-dn`<br>`block-keysize`<br>`block-proxy-auth`<br>`block-secret-keysize`<br>`block-ssl-id`<br>`block-user-dn`<br>`cache-enable`<br>`cache-setting`<br>`force-type`<br>`forward-auth-cert`<br>`forward-cache-info`<br>`forward-cipher`<br>`forward-ip`<br>`forward-issuer-dn`<br>`forward-keysize`<br>`suppress-request-headers`<br>`forward-proxy-auth`<br>`forward-secret-keysize`<br>`forward-ssl-id`<br>`forward-user-dn`<br>`http-client-config`<br>`java-ip-check`<br>`match-browser`<br>`reverse-map`<br>`set-basic-auth`<br>`set-default-type`<br>`set-variable`<br>`shtml-hacktype`<br>`ssl-client-config`<br>`type-by-exp`<br>`type-by-extension` |
| `Input` | `insert-filter`<br>`match-browser`<br>`remove-filter`<br>`set-variable` |
| `Output` | `content-rewrite`<br>`insert-filter`<br>`match-browser`<br>`remove-filter`<br>`set-variable` |

**Table 5-1** Available Server Application Functions (SAFs) Per Directive

| Directive | Server Application Functions |
|---|---|
| Service | add-footer<br>add-header<br>append-trailer<br>imagemap<br>index-common<br>index-simple<br>key-toosmall<br>list-dir<br>make-dir<br>match-browser<br>proxy-retrieve<br>query-handler<br>remove-dir<br>remove-file<br>remove-filter<br>rename-file<br>send-error<br>send-file<br>send-range<br>send-shellcgi<br>send-wincgi<br>service-dump<br>service-j2ee<br>service-trace<br>set-variable<br>shtml_send<br>stats-xml<br>upload-file |
| AddLog | common-log<br>flex-log<br>match-browser<br>record-useragent<br>set-variable |
| Error | error-j2ee<br>match-browser<br>query-handler<br>remove-filter<br>send-error<br>set-variable |
| Connect | |
| DNS | dns-config<br>your-dns-function |

**Table 5-1** Available Server Application Functions (SAFs) Per Directive

| Directive | Server Application Functions |
|---|---|
| `Filter` | `filter-ct`<br>`filter-html`<br>`pre-filter` |
| `Route` | `icp-route`<br>`pa-enforce-internal-routing`<br>`pa-set-parent-route`<br>`set-proxy-server`<br>`set-origin-server`<br>`set-socks-server`<br>`unset-proxy-server`<br>`unset-socks-server` |

# The bucket Parameter

The following performance buckets are predefined in Sun Java System Web Proxy 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` `Service` function.

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

For more information about performance buckets, see the Sun Java System Web Proxy Server 4.0.1 *Administration Guide*.

# Init

The Init functions load and initialize server modules and plugins, and initialize log files.

### Syntax

```
Init fn=function-name [parm1=value1]...[parmN=valueN]
```

`function-name` identifies the server initialization function to call. These functions shouldn't be called more than once.

`parm=value` pairs are values for function-specific parameters. The number of parameters depends on the function you use. The order of the parameters doesn't matter. The functions of the **Init** directive listed here are described in detail in the following sections.

- `define-perf-bucket` creates a performance bucket.
- `flex-init` initializes the flex-log flexible access logging feature
- `flex-rotate-init` enables rotation for flexible logs.
- `host-dns-cache-init` caches host names of the origin servers.
- `icp-init` initializes the ICP feature.
- `init-clf` initializes the Common Log File subsystem.
- `init-filter-order` controls the position of specific filters within filter stacks.
- `init-j2ee` initializes the Java subsystem. This is applicable only to the Administration Server.
- `init-proxy` initializes the networking code used by the proxy.
- `init-uhome` loads user home directory information.
- `init-url-filter` specifies one or more filter files of URLs. A filter file is a file that contains a list of URLs.
- `ip-dns-cache-init` configures DNS caching.
- `load-modules` tells the server to load functions from a shared object file.
- `load-types` maps file extensions to MIME types.
- `nt-console-init` enables the Windows console, which is the command-line shell that displays standard output and error streams.
- `pa-init-parent-array` initializes a parent array member and specifies information about the PAT file for the parent array of which it is a member.
- `pa-init-proxy-array` initializes a proxy array member and specifies information about the PAT file for the array of which it is a member.
- `perf-init` enables system performance measurement via performance buckets.

- pool-init configures pooled memory allocation.
- register-http-method lets you extend the HTTP protocol by registering new HTTP methods.
- stats-init enables reporting of performance statistics in XML format.
- suppress-request-headers configures the proxy server to remove outgoing headers from the request.
- thread-pool-init configures an additional thread pool.
- tune-cache allows you to tune the performance of your proxy server's cache.
- tune-proxy allows you to tune the performance of your proxy server.You should not change the default settings.

# define-perf-bucket

Applicable in Init-class directives.

The define-perf-bucket function creates a performance bucket, which you can use to measure the performance of SAFs in obj.conf.

### Parameters

The following table describes parameters for the define-perf-bucket function.

**Table 5-2** define-perf-bucket parameters

| Parameter | Description |
|---|---|
| name | Name for the bucket (for example, cgi-bucket). |
| description | Description of what the bucket measures (for example, CGI Stats). |

### Example

```
Init fn="define-perf-bucket" name="cgi-bucket" description="CGI Stats"
```

### See Also

perf-init

# flex-init

Applicable in Init-class directives.

The flex-init function opens the named log file to be used for flexible logging and establishes a record format for it. The log format is recorded in the first line of the log file. You cannot change the log format while the log file is in use by the server.

The flex-log function (applicable in AddLog-class directives) writes entries into the log file during the AddLog stage of the request-handling process.

The log file stays open until the server is shut down or restarted (at which time all logs are closed and reopened).

**NOTE** If the server has AddLog-stage directives that call flex-log, the flexible log file must be initialized by flex-init during server initialization.

You may specify multiple log file names in the same flex-init function call. Then use multiple AddLog directives with the flex-log function to log transactions to each log file.

The flex-init function may be called more than once. Each new log file name and format will be added to the list of log files.

If you move, remove, or change the currently active log file without shutting down or restarting the server, client accesses might not be recorded. To save or backup the currently active log file, you need to rename the file and then restart the server. The server first looks for the log file by name, and if it doesn't find it, creates a new one (the renamed original log file is left for you to use).

For information on rotating log files, see flex-rotate-init.

The flex-init function has three parameters: one that names the log file, one that specifies the format of each record in that file, and one that specifies the logging mode.

### Parameters

The following table describes parameters for the flex-init function.

**Table 5-3** flex-init parameters

| Parameter | Description |
|---|---|
| *logFileName* | Name of the parameter is the name of the log file. The value of the parameter specifies either the full path to the log file or a file name relative to the server's `logs` directory. For example:<br>`access="/usr/netscape/server4/https-servername/logs/access"`<br>`mylogfile = "log1"`<br>You will use the log file name later, as a parameter to the `flex-log` function (applicable in `AddLog`-class directives). |
| `buffer-size` | Specifies the size of the global log buffer. The default is `8192`. See the third `flex-init` example below. |
| `buffers-per-file` | Specifies the number of buffers for a given log file. The default value is determined by the server.<br>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-log-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. |
| `format.`*logFileName* | Specifies the format of each log entry in the log file.<br>For information about the format, see the "More on Log Format" section below. |

### More on Log Format

The `flex-init` function recognizes anything contained between percent signs (%) as the name portion of a name-value pair stored in a parameter block in the server. (The one exception to this rule is the `%SYSDATE%` component, which delivers the current system date.) `%SYSDATE%` is formatted using the time format `%d/%b/%Y:%H:%M:%S` plus the offset from GMT.

Any additional text is treated as literal text, so you can add to the line to make it more readable. Typical components of the formatting parameter are listed in the following table Table 5-4. Certain components might contain spaces, so they should be bounded by escaped quotes (\").

If no format parameter is specified for a log file, the common log format is used:

```
"%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%]
\"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length%"
```

You can now log cookies by logging the `Req->headers.cookie.name` component.

In the following table, the components that are enclosed in escaped double quotes (\") are the ones that could potentially resolve to values that have white spaces.

**Table 5-4** Typical Components of flex-init Formatting

| Flex-log Option | Component |
|---|---|
| Client host name (unless `iponly` is specified in flex-log or DNS name is not available) or IP address | `%Ses->client.ip%` |
| Client DNS name | `%Ses->client.dns%` |
| System date | `%SYSDATE%` |
| Full HTTP request line | `\"%Req->reqpb.clf-request%\"` |
| Status | `%Req->srvhdrs.clf-status%` |
| Response content length | `%Req->srvhdrs.content-length%` |
| Response content type | `%Req->srvhdrs.content-type%` |
| Referer header | `\"%Req->headers.referer%\"` |
| User-agent header | `\"%Req->headers.user-agent%\"` |
| HTTP method | `%Req->reqpb.method%` |
| HTTP URI | `%Req->reqpb.uri%` |
| HTTP query string | `%Req->reqpb.query%` |
| HTTP protocol version | `%Req->reqpb.protocol%` |
| Accept header | `%Req->headers.accept%` |
| Date header | `%Req->headers.date%` |
| If-Modified-Since header | `%Req->headers.if-modified-since%` |
| Authorization header | `%Req->headers.authorization%` |
| Any header value | `%Req->headers.headername%` |
| Name of authorized user | `%Req->vars.auth-user%` |
| Value of a cookie | `%Req->headers.cookie.name%` |

**Table 5-4** Typical Components of flex-init Formatting

| Flex-log Option | Component |
|---|---|
| Value of any variable in `Req->vars` | `%Req->vars.varname%` |
| Duration | `%duration%`<br>Records the time in microseconds the server spent handling the request. Statistics must be enabled for the server instance before `%duration%` can be used. For information about enabling statistics, see the Sun Java System Web Proxy Server 4.0.1 *Administration Guide*. |

### Examples

The first example below initializes flexible logging into the file *<Install_Root><Instance_Directory>*`/logs/access`.

```
Init fn="flex-init" access="$accesslog" format.access="%Ses->client.ip% -
%Req->vars.auth-user% [%SYSDATE%] '%Req->reqpb.clf-request%'
%Req->srvhdrs.clf-status% %Req->srvhdrs.content-length%"
```

This will record the following items:

- IP or host name, followed by the three characters “ - ”
- User name, followed by the two characters “ [”
- System date, followed by the two characters “] ”
- Full HTTP request in quotes, followed by a single space
- HTTP result status in quotes, followed by a single space
- Content length

This is the default format, which corresponds to the Common Log Format (CLF).

It is advisable that the first six elements of any log always be in exactly this format, because a number of log analyzers expect that as output.

The second example initializes flexible logging into the file *<Install_Root><Instance_Directory>*`/logs/extended`.

```
Init fn=flex-init extended="<Install_Root><Instance_Directory>/logs/extendedd"
format.extended="%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%]
\"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length% %Req->headers.referer%
\"%Req->headers.user-agent%\" %Req->reqpb.method% %Req->reqpb.uri%
%Req->reqpb.query% %Req->reqpb.protocol%"
```

The third example shows how logging can be tuned to prevent request handling threads from making blocking calls when writing to log files, instead delegating these calls to the log flush thread.

Doubling the size of the `buffer-size` and `num-buffers` parameters from their defaults and lowering the value of the `LogFlushInterval` `magnus.conf` directive to 4 seconds frees the request-handling threads to quickly write the log data.

```
Init fn=flex-init buffer-size=16384 num-buffers=2000
access="/<Install_Root><Instance_Directory>/logs/access"
format.access="%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%]
\"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length%"
```

**See Also**

`flex-rotate-init`

# flex-rotate-init

Applicable in `Init`-class directives.

The `flex-rotate-init` function configures log rotation for all log files on the server, including error logs and the `common-log`, `flex-log`, and `record-useragent` `AddLog` SAFs. Call this function in the `Init` section of `magnus.conf` before calling `flex-init`. The `flex-rotate-init` function allows you to specify a time interval for rotating log files. At the specified time interval, the server moves the log file to a file whose name indicates the time of moving. The log functions in the AddLog stage in obj.conf then start logging entries in a new log file. The server does not need to be shut down while the log files are being rotated.

**NOTE** The server keeps all rotated log files forever, so you will need to clean them up as necessary to free disk space.

By default, log rotation is disabled.

### Parameters

The following table describes parameters for the flex-rotate-init function.

**Table 5-5** flex-rotate-init parameters

| Parameter | Description |
|---|---|
| rotate-start | Indicates the time to start rotation. This value is a four-digit string indicating the time in 24-hour format. For example, 0900 indicates 9 a.m., while 1800 indicates 9 p.m. |
| rotate-interval | Indicates the number of minutes to elapse between each log rotation. |
| rotate-access | (Optional) Determines whether common-log, flex-log, and record-useragent logs are rotated (AddLog SAFs). Values are yes (the default), and no. |
| rotate-error | (Optional) Determines whether error logs are rotated. Values are yes (the default), and no. |
| rotate-callback | (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. |

### Example

This example enables log rotation, starting at midnight and occurring every hour.

```
Init fn=flex-rotate-init rotate-start=2400 rotate-interval=60
```

### See Also

flex-init

## host-dns-cache-init

Applicable in Init-class directives.

The host-dns-cache-init function is used to cache host names of the origin servers. If DNS looks up are caches, then when the server gets a request from the client servers, it caches the server's host name information.

### Parameters

The following table describes parameters for the dns-cache-init function.

**Table 5-6** host-dns-cache-init parameters

| Parameter | Description |
|---|---|
| cache-size | (Optional) Specifies how many entries are contained in the cache. Acceptable values are 32 to 32768; the default value is 1024. |
| expire | (Optional) Specifies how long (in seconds) it takes for a cache entry to expire. Acceptable values are 1 to 31536000 (1 year); the default is 1200 seconds (20 minutes). |
| negative-dns-cache | To enable/disable caching invalid hostnames; the default is yes. |

### Example

```
Init fn="host-dns-cache-init" cache-size="2140" expire="600"
```

## icp-init

Applicable in Init-class directives.

The icp-init function enables and initializes ICP. ICP (Internet Cache Protocol) is an object location protocol that enables caches to communicate with one another. Caches can use ICP to send queries and replies about the existence of cached URLs and about the best locations from which to retrieve those URLs.

### Syntax

```
Init fn=icp-init
   config_file=file name
   status=on|off
```

### Parameters

The following table describes parameters for the `icp-init` function.

**Table 5-7** icp-init parameters

| Parameter | Description |
|---|---|
| `config_file` | The name of the ICP configuration file. |
| `status` | Specifies whether ICP is enabled or disabled.<br>Possible values are:<br>• `on` means that ICP is enabled.<br>• `off` means that ICP is disabled. |

### Example

```
Init fn=icp-init
   config_file=icp.conf
   status=on
```

## init-clf

Applicable in `Init`-class directives.

The `init-clf` function opens the named log files to be used for common logging. The `common-log` function writes entries into the log files during the `AddLog` stage of the request-handling process. The log files stay open until the server is shut down (at which time the log files are closed) or restarted (at which time the log files are closed and reopened).

**NOTE** If the server has an `AddLog-stage` directive that calls `common-log`, common log files must be initialized by `init-clf` during initialization.

**NOTE** This function should only be called once. If it is called again, the new call will replace log file names from all previous calls.

If you move, remove, or change the log file without shutting down or restarting the server, client accesses might not be recorded. To save or backup a log file, you need to rename the file (and for UNIX, send the `-HUP` signal), and then restart the server. The server first looks for the log file by name, and if it doesn't find it, creates a new one (the renamed original log file is left for you to use).

For information on rotating log files, see `flex-rotate-init`.

### Parameters

The following table describes parameters for the `init-clf` function.

**Table 5-8** init-clf parameters

| Parameter | Description |
|---|---|
| *logFileName* | Name of the parameter is the name of the log file. The value of the parameter specifies either the full path to the log file or a file name relative to the server's `logs` directory. For example:<br>`access="/usr/netscape/server4/https-servername/logs/access"`<br>`mylogfile = "log1"`<br>You will use the log file name later, as a parameter to the `common-log` function (applicable in `AddLog-class` directives). |

### Examples

.

```
Init fn=init-clf access=/usr/netscape/server4/https-boots/logs/access

Init fn=init-clf templog=/tmp/mytemplog templog2=/tmp/mytemplog2
```

### See Also

`flex-rotate-init`

# init-filter-order

Applicable in `Init`-class directives.

The `init-filter-order` `Init` SAF can be used to control the position of specific filters within filter stacks. For example, `init-filter-order` can be used to ensure that a filter that converts outgoing XML to XHTML is inserted above a filter that converts outgoing XHTML to HTML.

Filters that appear higher in the filter stack are given an earlier opportunity to process outgoing data, and filters that appear lower in the filter stack are given an earlier opportunity to process incoming data.

The appropriate position of a specific filter within the filter stack is defined by the filter developer. For example, filters that translate content from XML to HTML are placed higher in the filter stack than filters that compress data for transmission. Filter developers use the `filter_create` function to define the filter's position in the filter stack. `init-filter-order` can be used to override the position defined by the filter developer.

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. For example, consider two filters, `xhtml-to-html` and `xml-to-xhtml`, which convert XHTML to HTML and XML to XHTML, respectively. Since both filters transform data from one format to another, they may be defined to occupy the same position in the filter stack. To transform XML documents to XHTML and then to HTML before sending the data to the client, `Output fn="insert-filter"` directives in `obj.conf` would appear in the following order:

```
Output fn="insert-filter" filter="xhtml-to-html"

Output fn="insert-filter" filter="xml-to-xhtml"
```

In general, administrators should use the order of `Input fn="insert-filter"` and `Output fn="insert-filter"` directives in `obj.conf` to control the position of filters in the filter stack. `init-filter-order` should only be used to address specific filter interoperability problems.

**NOTE** The `load-module` SAFs that create the filters should be called before `init-filter-order` attempts to order them.

### Parameters

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

**Table 5-9** init-filter-order parameters

| Parameter | Description |
|---|---|
| `filters` | Comma-separated list of filters in the order they should appear within a filter stack, listed from highest to lowest. |

### Example

```
Init fn="init-filter-order"
filters="xml-to-xhtml,xhtml-to-html,http-compression"
```

## init-j2ee

This is applicable to only the Administration Server.

Applicable in `Init`-class directives.

The `init-j2ee` function initializes the Java subsystem.

### Parameters

This function requires a `LateInit=yes` parameter.

### Example

```
Init fn="load-modules" shlib="install_dir/lib/libj2eeplugin.so"
funcs="init-j2ee,ntrans-j2ee,service-j2ee,error-j2ee"
shlib_flags="(global|now)"
Init fn="init-j2ee" LateInit=yes
```

## init-proxy

Applicable in `Init`-class directives.

The `init-proxy` function initializes the Proxy Server’ s internal settings. This function is called during the initialization of the Proxy Server, but it should also be specified in the `obj.conf` to ensure that the values are initialized properly.

### Syntax

```
Init fn=init-proxy
   timeout=<seconds>
   timeout-2=seconds
```

### Parameters

The following table describes parameters for the `init-proxy` function.

Table 5-10 init-proxy parameters

| Parameter | Description |
|---|---|
| `timeout` | The number of seconds of delay allowed between consecutive network packets received from the remote server. If the delay exceeds the timeout, the connection is dropped. The default is 120 seconds (2 minutes). This is not the maximum time allowed for an entire transaction, but the delay between the packets. For example, the entire transaction can last 15 minutes, as long as at least one packet of data is received before each timeout period. |
| `timeout-2`<br>(timeout after interrupt) | The timeout after interrupt value tells the Proxy Server how much time it has to continue writing a cache file after a client has aborted the transaction. In other words, if the Proxy Server has almost finished caching a document and the client aborts the connection, the server can continue caching the document until it reaches the timeout after interrupt value.<br>The highest recommended timeout after interrupt value is 5 minutes. The default value is 15 seconds. |

### Example

```
Init fn=init-proxy
    timeout=120
```

## init-uhome

Applicable in `Init`-class directives.

**UNIX Only.** The `init-uhome` function loads information about the system’s user home directories into internal hash tables. This increases memory usage slightly, but improves performance for servers that have a lot of traffic to home directories.

### Parameters

The following table describes parameters for the `init-uhome` function.

**Table 5-11** init-uhome parameters

| Parameter | 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. |

### Examples

```
Init fn=init-uhome

Init fn=init-uhome pwfile=/etc/passwd-http
```

## init-url-filter

Applicable in `Init`-class directives.

The `init-url-filter` function specifies one or more filter files of URLs. A filter file is a file that contains a list of URLs.

### Parameters

You can pass one or more parameters to this SAF and associate each parameter to a filter file of URLs. These parameter names may be used later in `url-filter` SAF to either allow or deny these filter files of URLs.

### Example

```
PathCheck fn="init-url-filter" filt1="/path/to/filter/file1"
filt2="/path/to/filter/file2" filt3="/path/to/filter/file3" etc...
```

## ip-dns-cache-init

Applicable in `Init`-class directives.

The ip-dns-cache-init function specifies that DNS lookups should be cached when DNS lookups are enabled. If DNS lookups are cached, then when the server gets a client's host name information, it stores that information in the DNS cache. If the server needs information about the client in the future, the information is available in the DNS cache.

You may specify the size of the DNS cache and the time it takes before a cache entry becomes invalid. The DNS cache can contain 32 to 32768 entries; the default value is 1024 entries. Values for the time it takes for a cache entry to expire (specified in seconds) can range from 1 second to 1 year; the default value is 1200 seconds (20 minutes).

### Parameters

The following table describes parameters for the ip-dns-cache-init function.

**Table 5-12** ip-dns-cache-init parameters

| Parameter | Description |
|---|---|
| cache-size | (Optional) Specifies how many entries are contained in the cache. Acceptable values are 32 to 32768; the default value is 1024. |
| expire | (Optional) Specifies how long (in seconds) it takes for a cache entry to expire. Acceptable values are 1 to 31536000 (1 year); the default is 1200 seconds (20 minutes). |

### Example

```
Init fn="ip-dns-cache-init" cache-size="2140" expire="600"
```

## load-modules

Applicable in Init-class directives.

The load-modules function loads a shared library or dynamic-link library (DLL) into the server code. Specified functions from the library can then be executed from any subsequent directives. Use this function to load new plugins or SAFs.

If you define your own SAFs, you get the server to load them by using the load-modules function and specifying the shared library or DLL to load.

### Parameters

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

Table 5-13 load-modules parameters

| Parameter | Description |
|---|---|
| shlib | Specifies either the full path to the shared library or DLL, or a file name relative to the server configuration directory. |
| funcs | Comma-separated list of the names of the functions in the shared library or DLL to be made available for use by other Init directives or by Service directives in obj.conf. The list should not contain any spaces. The dash (-) character may be used in place of the underscore (_) character in function names. |
| NativeThread | (Optional) Specifies which threading model to use.<br>no causes the routines in the library to use user-level threading.<br>yes enables kernel-level threading. The default is yes. |
| pool | Name of a custom thread pool, as specified in thread-pool-init. |

### Examples

```
Init fn=load-modules shlib="C:/mysrvfns/corpfns.dll" funcs="moveit"

Init fn=load-modules shlib="/mysrvfns/corpfns.so" funcs="myinit,myservice"
Init fn=myinit
```

## load-types

Applicable in Init-class directives.

The load-types function scans a file that tells it how to map filename extensions to MIME types. MIME types are essential for network navigation software like Netscape Navigator to tell the difference between file types. For example, they are used to tell an HTML file from a GIF file. See “MIME Types.”Chapter 6, “MIME Types,”for more information.

Calling this function is crucial if you use Web Proxy Server Manager online forms or the FTP proxying capability.

### Syntax

```
Init fn=load-types
   mime-types="mime.types"
```

This function loads the MIME type file `mime.types` from the configuration directory (the same directory as `magnus.conf` and `obj.conf`). This function call is mandatory and in practice is always as shown in the syntax.

### Parameters

The following table describes parameters for the `load-types` function.

**Table 5-14** load-types parameters

| Parameter | Description |
|---|---|
| `mime-types` | specifies either the full path to the global MIME types file or a filename relative to the server configuration directory. The proxy server comes with a default file called `mime.types`. |
| `local-types` | Optional parameter to a file with the same format as the global MIME types file, but it is used to maintain types that are applicable only to your server. |

### Example

```
Init fn=load-types mime-types=mime.types
```

```
Init fn=load-types mime-types=/tp/mime.types \
   local-types=local.types
```

## nt-console-init

Applicable in `Init`-class directives.

The `nt-console-init` function enables the Windows console, which is the command-line shell that displays standard output and error streams.

### Parameters

The following table describes parameters for the `nt-console-init` function.

**Table 5-15** nt-console-init parameters

| Parameter | Description |
|---|---|
| `stderr` | Directs error messages to the Windows console. The required and only value is `console`. |
| `stdout` | Directs output to the Windows console. The required and only value is `console`. |

### Example

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

## pa-init-parent-array

Applicable in `Init`-class directives.

The `pa-init-parent-array` function initializes a parent array member and specifies information about the PAT file for the parent array of which it is a member.

**NOTE** The `load modules` directive should come before the `pa-init-proxy-array` function in the `obj.conf` file.

### Syntax

```
Init fn=pa-init-parent-array
   set-status-fn=pa-set-member-status
   poll="yes|no"
   file="absolute filename"
   pollhost="host name"
   pollport="port number"
   pollhdrs="absolute filename"
   pollurl="url"
   status="on|off"
```

### Parameters

The following table describes parameters for the `pa-init-parent-array` function.

**Table 5-16** pa-init-parent-array parameters

| Parameter | Description |
|---|---|
| `set-status-fn` | specifies the function that sets the status for the member. |
| `poll` | Tells the array member whether or not it needs to poll for a PAT file.<br>• `yes` means that the member should poll for the PAT file. A member should only poll for a PAT file if it is not the master proxy. The master proxy has a local copy of the PAT file, and therefore, does not need to poll for it.<br>• `no` means that the member should not poll for the PAT file. A member should not poll for the PAT file if it is the master proxy. |
| `file` | The full pathname of the PAT file. |
| `pollhost` | The host name of the proxy to be polled for the PAT file. This parameter only needs to be specified if the `poll` parameter is set to yes, meaning that the member is not the master proxy. |
| `pollport` | is the port number on the pollhost that should be contacted when polling for the PAT file. This parameter only needs to be specified if the `poll` parameter is set to yes, meaning that the member is not the master proxy. |
| `pollhdrs` | is the full pathname of the file that contains any special headers that must be sent with the HTTP request for the PAT file. This parameter is optional and should only be specified if the `poll` parameter is set to yes, meaning that the member is not the master proxy. |
| `pollurl` | is the URL of the PAT file to be polled for. This parameter only needs to be specified if the `poll` parameter is set to yes, meaning that the member is not the master proxy. |
| `status` | specifies whether the parent array member is on or off.<br>• `on` means that the member is on.<br>• `off` means that the member is off. |

### Example

The following example tells the member not to poll for the PAT file. This example would apply to a master proxy.

```
Init fn=pa-init-parent-array
   poll="no"
   file="c:/netscape/server/bin/proxy/pa1.pat"
```

The following example specifies that the member should poll for a PAT file. This member is not the master proxy.

```
Init fn=pa-init-parent-array
   poll="yes"
   file="c:/netscape/server/bin/proxy/pa2.pat"
   pollhost="proxy1"
   pollport="8080"
   pollhdrs="c:/netscape/server/proxy-name/parray/pa2.hdr"
   status="on"
   set-status-fn=set-member-status
   pollurl="/pat"
```

# pa-init-proxy-array

Applicable in `Init`-class directives.

The `pa-init-proxy-array` function initializes a proxy array member and specifies information about the PAT file for the array of which it is a member.

**NOTE** The `load modules` directive should come before the `pa-init-proxy-array` function in the `obj.conf` file.

### Syntax

```
Init fn=pa-init-proxy-array
   set-status-fn=pa-set-member-status
   poll="yes|no"
   file="absolute filename"
   pollhost="host name"
   pollport="port number"
   pollhdrs="absolute filename"
   pollurl="url"
   status="on|off"
```

### Parameters

The following table describes parameters for the `pa-init-proxy-array` function.

**Table 5-17** pa-init-proxy-array parameters

| Parameter | Description |
|---|---|
| `set-status-fn` | specifies the function that sets the status for the member. |

**Table 5-17** pa-init-proxy-array parameters

| Parameter | Description |
|---|---|
| `poll` | Tells the array member whether or not it needs to poll for a PAT file.<br>• `yes` means that the member should poll for the PAT file. A member should only poll for a PAT file if it is not the master proxy. The master proxy has a local copy of the PAT file, and therefore, does not need to poll for it.<br>• `no` means that the member should not poll for the PAT file. A member should not poll for the PAT file if it is the master proxy. |
| `file` | The full pathname of the PAT file. |
| `pollhost` | The host name of the proxy to be polled for the PAT file. This parameter only needs to be specified if the `poll` parameter is set to `yes`, meaning that the member is not the master proxy. |
| `pollport` | iThe port number on the pollhost that should be contacted when polling for the PAT file. This parameter only needs to be specified if the `poll` parameter is set to `yes`, meaning that the member is not the master proxy. |
| `pollhdrs` | iThe full pathname of the file that contains any special headers that must be sent with the HTTP request for the PAT file. This parameter is optional and should only be specified if the `poll` parameter is set to `yes`, meaning that the member is not the master proxy. |
| `pollurl` | iThe URL of the PAT file to be polled for. This parameter only needs to be specified if the `poll` parameter is set to `yes`, meaning that the member is not the master proxy. |
| `status` | Specifies whether the parent array member is on or off.<br>• `on` means that the member is on.<br>• `off` means that the member is off. |

### Example

The following example tells the member not to poll for the PAT file. This example would apply to a master proxy.

```
Init fn=pa-init-proxy-array
    poll="no"
    file="c:/netscape/server/bin/proxy/pa1.pat"
```

The following example specifies that the member should poll for a PAT file. This member is not the master proxy.

```
Init fn=pa-init-proxy-array
   poll="yes"
   file="c:/netscape/server/bin/proxy/pa2.pat"
   pollhost="proxy1"
   pollport="8080"
   pollhdrs="c:/netscape/server/proxy-name/parray/pa2.hdr"
   status="on"
   set-status-fn=set-member-status
   pollurl="/pat"
```

# perf-init

Applicable in `Init`-class directives.

The `perf-init` function enables system performance measurement via performance buckets.

### Parameters

The following table describes parameters for the `perf-init` function.

**Table 5-18** perf-init parameters

| Parameter | Description |
|---|---|
| `disable` | Flag to disable the use of system performance measurement via performance buckets. Should have a value of `true` or `false`. Default value is `true`. |

### Example

```
Init fn=perf-init disable=false
```

### See Also

`define-perf-bucket`

# pool-init

Applicable in `Init`-class directives.

The pool-init function changes the default values of pooled memory settings. The size of the free block list may be changed or pooled memory may be entirely disabled.

Memory allocation pools allow the server to run significantly faster. If you are programming with the NSAPI, note that MALLOC, REALLOC, CALLOC, STRDUP, and FREE work slightly differently if pooled memory is disabled. If pooling is enabled, the server automatically cleans up all memory allocated by these routines when each request completes. In most cases, this will improve performance and prevent memory leaks. If pooling is disabled, all memory is global and there is no clean-up.

If you want persistent memory allocation, add the prefix PERM_ to the name of each routine (PERM_MALLOC, PERM_REALLOC, PERM_CALLOC, PERM_STRDUP, and PERM_FREE).

**NOTE** Any memory you allocate from Init-class functions will be allocated as persistent memory, even if you use MALLOC. The server cleans up only the memory that is allocated while processing a request, and because Init-class functions are run before processing any requests, their memory is allocated globally.

### Parameters

The following table describes parameters for the pool-init function.

**Table 5-19** pool-init parameters

| Parameter | Description |
|---|---|
| free-size | (Optional) Maximum size in bytes of free block list. May not be greater than 1048576. |
| disable | (Optional) Flag to disable the use of pooled memory. Should have a value of true or false. Default value is false. |

### Example

```
Init fn=pool-init disable=true
```

# register-http-method

Applicable in Init-class directives.

This function lets you extend the HTTP protocol by registering new HTTP methods. (You do not need to register the default HTTP methods.)

Upon accepting a connection, the server checks if the method it received is known to it. If the server does not recognize the method, it returns a “501 Method Not Implemented” error message.

### Parameters

The following table describes parameters for the register-http-method function.

**Table 5-20** register-http-method parameters

| Parameter | Description |
|---|---|
| methods | Comma-separated list of the names of the methods you are registering. |

### Example

The following example shows the use of register-http-method and a Service function for one of the methods.

```
Init fn="register-http-method" methods="MY_METHOD1,MY_METHOD2"
Service fn="MyHandler" method="MY_METHOD1"
```

## stats-init

Applicable in Init-class directives.

The stats-init function enables reporting of performance statistics in XML format. The actual report is generated by the stats-xml function in obj.conf.

### Parameters

The following table describes parameters for the stats-init function.

**Table 5-21** stats-init parameters

| Parameter | Description |
|---|---|
| update-interval | Period in seconds between statistics updates within the server. Set higher for better performance, lower for more frequent updates. The minimum value is 1; the default is 5. |

**Table 5-21** stats-init parameters

| Parameter | Description |
|---|---|
| profiling | Enables NSAPI performance profiling using buckets if set to yes. This can also be enabled through the perf-init Init SAF. The default is no, which results in slightly better server performance. |

### Example

```
Init fn="stats-init" update-interval="5" virtual-servers="2000"
profiling="yes"
```

## suppress-request-headers

Applicable in Init-class and ObjectType-class directives.

If you specify this function at the Init stage it applies to the entire proxy (for all the requests ).

If you specify this function at ObjectType stage you can control suppressing outgoing headers functionality for different objects in the obj.conf file.

The suppress-request-headers function configures the proxy server to remove outgoing headers from the request. It accepts one or more hdr parameters through which you can specify multiple headers you want to suppress.

For example, you might want to prevent the from  and Cookie headers from going out because it reveals the user's credentials.

### Parameters

The following table describes parameters for the suppress-request-headers function.

**Table 5-22** suppress-request-headers parameters

| Parameter | Description |
|---|---|
| hdr | Name of the HTTP request header to be suppressed. |

### Example

```
Init fn="suppress-request-headers" hdr="from" hdr="Cookie"
```

## thread-pool-init

Applicable in Init-class directives.

The thread-pool-init function creates a new pool of user threads. A pool must be declared before it is used. To tell a plugin to use the new pool, specify the pool parameter when loading the plugin with the Init-class function load-modules.

One reason to create a custom thread pool would be if a plugin is not thread-aware, in which case you can set the maximum number of threads in the pool to 1.

The older parameter NativeThread=yes always engages one default native pool, called NativePool.

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 plugins.

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

In addition, native thread pool parameters can be added to the magnus.conf file for convenience.

### Parameters

The following table describes parameters for the thread-pool-init function.

**Table 5-23** thread-pool-init parameters

| Parameter | Description |
|---|---|
| name | Name of the thread pool. |
| maxthreads | Maximum number of threads in the pool. |
| minthreads | Minimum number of threads in the pool. |

**Table 5-23** thread-pool-init parameters

| Parameter | Description |
|---|---|
| `queueSize` | Size of the queue for the pool. If all threads in the pool are busy, further request-handling threads that want to get a thread from the pool will wait in the pool queue. The number of request-handling threads that can wait in the queue is limited by the queue size. If the queue is full, the next request-handling thread that comes to the queue is turned away, with the result that the request is turned down, but the request-handling thread remains free to handle another request instead of becoming locked up in the queue. |
| `stackSize` | Stack size of each thread in the native (kernel) thread pool. |

### Example

```
Init fn=thread-pool-init name="my-custom-pool" maxthreads=5 minthreads=1
queuesize=200
Init fn=load-modules shlib="C:/mydir/myplugin.dll" funcs="tracker"
pool="my-custom-pool"
```

### See Also

`load-modules`

## tune-cache

Applicable in `Init`-class directives.

The `tune-cache` function allows you to tune the performance of your proxy server's cache. You should not change the default settings unless directed to do so by Sun Technical Support.

### Syntax

```
Init fn=tune-cache
    byte-ranges
```

### Parameters

The following table describes the parameter for the `tune-cache` function.

**Table 5-24** tune-cache parameters

| Parameter | Description |
|---|---|
| `byte-ranges` | Determines whether or not the proxy is allowed to generate byte-range responses from the cache. By default, this feature is disabled. |

### Example

```
Init fn=tune-cache
    byte-ranges=off
```

## tune-proxy

Applicable in `Init`-class directives.

The `tune-proxy` function allows you to tune the performance of your proxy server.You should not change the default settings.

### Syntax

```
Init fn=tune-proxy
   ftp-listing-width=number
```

### Parameters

The following table describes the parameter for the `tune-proxy` function.

**Table 5-25** tune-cache parameters

| Parameter | Description |
|---|---|
| `ftp-listing-width` | You may want to modify the width of FTP listings to better suit your needs.Increasing listing width allows longer file names and thus reduces filename truncation. The default width is 80 characters. |

### Example

```
Init fn=tune-proxy
    ftp-listing-width=80
```

## Summary of Init Functions

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

**Table 5-26** Init functions

| Function/Parameter | Allowed Values | Default Value | Description |
|---|---|---|---|
| **`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 Proxy Server4.0.1 NSAPI Developer'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`:<br>`access="/logdir/access"` |

**Table 5-26** Init functions

| Function/Parameter | Allowed Values | Default Value | Description |
|---|---|---|---|
| `format.`*logFileName* | | | Specifies the format of each log entry in the log file. |
| `relaxed.`*logFileName* | `true`, `on`, `yes`, or `1`; `false`, `off`, `no`, or `0` | | Turns on relaxed logging, which skips logging components that would normally block static page acceleration if static page acceleration is enabled. |
| `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.<br>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. |
| `thread-buffer-size` | Number of bytes | `8192` (8 KB) | Specifies the size of the per thread log buffer. |
| **`flex-rotate-init`** | | | Enables rotation for logs. |

**Table 5-26** Init functions

| Function/Parameter | Allowed Values | Default Value | Description |
|---|---|---|---|
| `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 Proxy Server 4.0.1 NSAPI Developer's Guide*. |
| `rotate-error` | `yes`, `no` | `yes` | (optional) determines whether error logs are rotated. |
| `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. |

**Table 5-26** Init functions

| Function/Parameter | Allowed Values | Default Value | Description |
|---|---|---|---|
| `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`.<br>For information about installing an suid Cgistub, see the *Sun Java System Web Proxy Server 4.0.1 NSAPI Developer'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. |
| `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. |

**Table 5-26** Init functions

| Function/Parameter | Allowed Values | Default Value | Description |
|---|---|---|---|
| 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. |
| disable | true, false | true | Disables the function when true. |
| **pool-init** | | | Configures pooled memory allocation. |

**Table 5-26** Init functions

| Function/Parameter | Allowed Values | Default Value | Description |
|---|---|---|---|
| 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. |
| maxthreads | | | Maximum number of threads in the pool. |
| 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. |

# 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 validates authorization information sent by the client in the Authorization header.
- PathCheck 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 calls a custom function to verify user name and password. Optionally determines the user's group.
- basic-ncsa verifies user name and password against an NCSA-style or system DBM database. Optionally determines the user's group.
- get-sslid 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 matches specific strings in the User-Agent string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.

- proxy-auth translates authorization information provided through the basic proxy authorization scheme.
- set-variable 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.

### Parameters

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

**Table 5-27** 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. |

**Table 5-27** basic-auth parameters

| Parameter | Description |
|---|---|
| `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_auth

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

### See Also

`require-auth`

## 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.

### Parameters

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

**Table 5-28** 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. |
| 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`

## 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 5-29** 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 Proxy 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 5-30** 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 SAF. For a list of valid variables, see set-variable. |
| *value* | New value for the specified variable. |

### Example

The following AuthTrans directive instructs Sun Java System Web Proxy 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 "ssl-unclean-shutdown" on page 146).
- Not honor requests for HTTP Keep-Alive (see "keep-alive" on page 146).
- Use the HTTP / 1.0 protocol rather than HTTP / 1.1 (see "http-downgrade" on page 146).

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

### See Also

set-variable

# proxy-auth

Applicable in AuthTrans-class directives.

The proxy-auth function of the AuthTrans directive translates authorization information provided through the basic proxy authorization scheme. This scheme is similar to the HTTP authorization scheme but doesn't interfere with it, so using proxy authorization doesn't block the ability to authenticate to the remote server.

This function is usually used with the PathCheck fn=require-proxy-auth function.

### Syntax

```
AuthTrans fn=proxy-auth auth-type=basic
   dbm=full path name
```

```
AuthTrans fn=proxy-auth auth-type=basic
   userfile=full path name
   grpfile=full path name
```

### Parameters

The following table describes parameter values for the proxy-auth function.

**Table 5-31** proxy-auth parameter values

| Value | Description |
|---|---|
| auth-type | Specifies the type of authorization to be used. The type should be "basic" unless you are running a UNIX proxy and are going to use your own function to perform authentication. |
| dbm | Specifies the full path and base filename 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. |
| userfile | Specifies the full pathname of the user database in the NCSA-style httpd user file format. This format consists of name:password lines where password is encrypted. If you use this parameter, don't use dbm. |
| 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. |

### Example

A UNIX example:

```
AuthTrans fn=proxy-auth auth-type=basic
    dbm=/usr/ns-home/proxy-EXAMPLE/userdb/rs
A Windows NT example:
AuthTrans fn=proxy-auth auth-type=basic
    userfile=\netscape\server\proxy-EXAMPLE\.htpasswd
    grpfile=\netscape\server\proxy-EXAMPLE\.grpfile
```

It is possible to have authentication be performed by a user-provided function by passing the user-fn parameter to the `proxy-auth` function.

### Syntax

```
AuthTrans fn=proxy-auth auth-type=basic
   user-fn=your function
   userdb=full path name
```

### Parameters

The following table describes parameter values for the user provided `proxy-auth` function.

**Table 5-32** user provide `proxy-auth` parameter values

| Value | Description |
|---|---|
| `user-fn` | Specifies the name of the user-provided function that will be used to perform authentication in place of the built-in authentication. If authentication succeeds, the function should return REQ-PROCEED and if authentication fails, it should return REQ-NOACTION. |
| userdb | Specifies the full path and base filename 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. |

# 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 Proxy Server 4.0.1 *NSAPI Developer'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 5-33** set-variable parameter values

| Value | Description |
|---|---|
| *pblock* | One of the following Session/Request parameter block names:<br>• `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 Proxy Server 4.0.1 *NSAPI Developer’s Guide*.<br>• `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.<br>• `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.<br>`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.<br>• `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.<br>**Note:** For more information about parameter blocks, see the Sun Java System Web Proxy Server 4.0.1 *NSAPI Developer’s Guide*. |
| *name* | The variable to set. |
| *value* | The string assigned to the variable specified by *name*. |

### Variables

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

**Table 5-34** 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. |

**Table 5-34** Supported Variables

| Parameter | Description |
|---|---|
| `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 Proxy Server 4.0.1 *NSAPI Developer’s Guide*. |
| `find-pathinfo-forward` | Path information after the file name in a URI. See `find-pathinfo`. |
| `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`. |
| `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. |
| `nostat` | Causes the server *not* to perform the `stat()` function for a URL when possible. See also `assign-name`. |
| `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. |
| `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. |
| `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

# 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/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 tells the server to process directives in a named object.
- document-root 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 translates a request for the server's root home page (/) to a specific file.
- map looks for a certain URL prefix in the URL that the client is requesting.
- match-browser matches specific strings in the User-Agent string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.
- ntrans-j2ee  determines whether a request maps to a Java™ technology-based web application context. This is applicable only to the Administration Server.
- pac-map maps proxy-relative URLs to local files that are delivered to clients who request configuration.
- pat-map maps proxy-relative URLs to local files that are delivered to proxies that request configuration.
- pfx2dir translates any URL beginning with a given prefix to a file system directory and optionally enables directives in an additional named object.
- redirect redirects the client to a different URL.
- reverse-map rewrites HTTP response headers when the proxy server is functioning as a reverse proxy.
- set-variable 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 removes embedded semicolon-delimited parameters from the path.
- unix-home 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 5-35** 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.<br>The value you assign to this parameter is ignored. If you do not wish to use this parameter, leave it out.<br>The find-pathinfo-forward parameter is ignored if the ntrans-base parameter is not set in rq->vars. By default, ntrans-base is set.<br>This feature can improve performance for certain URLs by reducing the number of stats performed. |

**Table 5-35** assign-name parameters

| Parameter | Description |
|---|---|
| `nostat` | (Optional) Prevents the server from performing a stat on a specified URL whenever possible.<br>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 plugin URLs, to improve performance by avoiding unnecessary stats on those URLs.<br>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/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/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` function to set up additional path name translations.

### Parameters

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

**Table 5-36** 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/server1/docs

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

### See Also

`pfx2dir`

## 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 5-37** 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.<br>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"
```

## map

Applicable in NameTrans-class directives.

The map function looks for a certain URL prefix in the URL that the client is requesting. If **map** finds the prefix, it replaces the prefix with the mirror site prefix. When you specify the URL, don't use trailing slashes—they cause "Not Found" errors.

### Syntax

```
NameTrans fn=map
       from="source site prefix"
       to="destination site prefix"
       name="named object"
```

### Parameters

The following table describes parameters for the map function.

**Table 5-38** map parameters

| Parameter | Description |
|---|---|
| from | The prefix to be mapped to the mirror site. |
| to | The mirror site prefix. |
| name | (optional) gives a named object from which to derive the configuration for this mirror site. |
| rewrite-host | (optional) indicates whether the Host HTTP request header is rewritten to match the host specified by the to parameter. In a reverse proxy configuration where the proxy server and origin server service the same set of virtual servers, you may wish to specify rewrite-host="false". The default is "true", meaning that the Host HTTP request header is rewritten. |

### Example

```
# Map site http://home.netscape.com/ to mirror site
http://mirror.com
NameTrans fn=map from="http://home.netscape.com"
    to="http://mirror.com"
```

# match-browser

See "match-browser" on page 140.

# ntrans-j2ee

This is applicable only to the Administration Server.

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 5-39** 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, error-j2ee

## pac-map

Applicable in NameTrans-class directives.

The pac-map function maps proxy-relative URLs to local files that are delivered to clients who request configuration.

### Syntax

```
NameTrans fn=pac-map
    from=URL
    to=prefix
    name=named object
```

### Parameters

The following table describes parameters for the pac-map function.

**Table 5-40** pac-map parameters

| Parameter | Description |
|---|---|
| from | The proxy URL to be mapped. |
| to | The local file to be mapped to. |
| name | (optional) gives a named object (template) from which to derive configuration. |

### Example

```
NameTrans fn=pac-map
    from=http://proxy.mysite.com/pac
    to=<Install_Root><Instance_Directory>pac/proxy.pac
    name=file
```

## pat-map

Applicable in `NameTrans`-class directives.

The `pat-map` function maps proxy-relative URLs to local files that are delivered to proxies who request configuration.

### Syntax

```
NameTrans fn=pat-map
    from=URL
    to=prefix
    name=named object
```

### Parameters

The following table describes parameters for the `pat-map` function.

**Table 5-41** pat-map parameters

| Parameter | Description |
|---|---|
| `from` | The proxy URL to be mapped. |
| `to` | The local file to be mapped to. |
| `name` | (optional) gives a named object (template) from which to derive configuration. |

### Example

```
NameTrans fn=pat-map
    from=http://proxy.mysite.com/pac
    to=<Install_Root><Instance_Directory>pac/proxy.pac
    name=file
```

# 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 for details of using named objects.)

### Parameters

The following table describes parameters for the pfx2dir function.

**Table 5-42** 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.<br>The value you assign to this parameter is ignored. If you do not wish to use this parameter, leave it out.<br>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.<br>This feature can improve performance for certain URLs by reducing the number of stats performed in the server function find-pathinfo.<br>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 5-43** 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`.<br>For more information about `util_uri_escape`, see the Sun Java System Web Proxy Server 4.0.1 *NSAPI Developer’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
```

## reverse-map

Applicable in `NameTrans`-class directives.

The `reverse-map` function is used to rewrite HTTP response headers when the proxy server is functioning as a reverse proxy. `reverse-map` looks for the URL prefix specified by the `from` parameter in certain response headers. If the `from` prefix matches the beginning of the response header value, `reverse-map` replaces the matching portion with the `to` prefix.

### Parameters

The following table describes parameters for the `reverse-map` function.

**Table 5-44** reverse-map parameters

| Parameter | Description |
|---|---|
| from | URL prefix to be rewritten. |
| to | URL prefix that will be substituted in place of the from prefix. |
| rewrite-location | (Optional) Boolean that indicates whether the Location HTTP response header should be rewritten. The default is "true", meaning the Location header is rewritten. |
| rewrite-content-location | (Optional) Boolean that indicates whether the Content-location HTTP response header should be rewritten. The default is "true", meaning the Content-location header is rewritten. |
| rewrite-headername | (Optional) Boolean that indicates whether the headername HTTP response header should be rewritten, where headername is a user-defined header name. With the exception of the Location and Content-location headers, the default is "false", meaning the headername header is not rewritten. |

## set-variable

See "set-variable" on page 143.

## 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 5-45** 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 5-46** 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. |

**Table 5-46** unix-home parameters

| Parameter | Description |
|---|---|
| `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`

# 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:

- `block-multipart-posts` blocks all multipart form file uploads when configured without any parameters.
- `check-acl` checks an access control list for authorization.
- `deny-existence` indicates that a resource was not found.
- `deny-service` sends a “Proxy Denies Access” error when a client tries to access a specific path.
- `find-index` locates a default file when a directory is requested.
- `find-links` denies access to directories with certain file system links.
- `find-pathinfo` locates extra path info beyond the file name for the `PATH_INFO` CGI environment variable.

- get-client-cert gets the authenticated client certificate from the SSL3 session.
- load-config finds and loads extra configuration information from a file in the requested path.
- match-browser matches specific strings in the User-Agent string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.
- nt-uri-clean denies access to requests with unsafe path names by indicating not found.
- ntcgicheck looks for a CGI file with a specified extension.
- require-auth denies access to unauthorized users or groups.
- require-proxy-auth makes sure that users are authenticated and triggers a password pop-up window.
- set-variable 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 specifies a virtual index for a directory.
- ssl-check checks the secret keysize.
- ssl-logout invalidates the current SSL session in the server's SSL session cache.
- unix-uri-clean denies access to requests with unsafe path names by indicating not found.
- url-check checks the validity of URL syntax.
- url-filter allows or denies URL patterns.
- user-agent-check restricts access to the proxy server based on the type and version of the client’s web browser.

## block-multipart-posts

Applicable in PathCheck-class directives.

The block-multipart-posts function blocks all multipart form file uploads when configured without any parameters. This can also be used to block requests based on specific content type, user-agent or HTTP method using content-type, user-agent and method parameters.

### Parameters

The following table describes parameters for the `block-multipart-posts` function.

**Table 5-47** block-multipart-posts parameters

| Parameter | Description |
|---|---|
| content-type | (Optional) Regular expression of the content type to be blocked. |
| user-agent | (Optional) Regular expression of the user agent to be blocked. |
| method | (Optional) Regular expression matching the HTTP request method to be blocked. |

### Example

```
PathCheck fn="block-multipart-posts" user-agent="Mozilla/.*"
method="(POST|PUT)"
```

## 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 5-48** 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. |

**Table 5-48** check-acl parameters

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

### Example

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

# 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 5-49** 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
```

# deny-service

Applicable in PathCheck-class directives and Service-class directives.

The deny-service function sends a “Proxy Denies Access” error when a client tries to access a specific path. If this directive appears in a client region, it performs access control on the specified clients.

The proxy specifically denies clients instead of specifically allowing them access to documents (for example, you don’t configure the proxy to allow a list of clients). The “default” object is used when a client doesn’t match any client region in objects, and because the “default” object uses the deny-service function, no one is allowed access by default.

### Syntax

```
PathCheck fn=deny-service path=.*someexpression.*
```

### Parameters

The following table describes the parameter for the deny-service function.

**Table 5-50** deny-service parameters

| Parameter | Description |
|---|---|
| path | A regular expression representing the path to check. Not specifying this parameter is equivalent to specifying *. URLs matching the expression are denied access to the proxy server. |

### Example

```
<Object ppath="http://sun/.*">
# Deny servicing proxy requests for fun GIFs
PathCheck fn=deny-service path=.*fun.*.gif
# Make sure nobody except Sun employees can use the object
# inside which this is placed.
<Client dns=*~.*.sun.com>
PathCheck fn=deny-service
</Client>
</Object>
```

# 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 5-51** 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.<br>• If set to yes, the compressed version will not be selected if it is older than the noncompressed version.<br>• If set to no, the compressed version will always be selected, even if it is older than the noncompressed version.<br>By default, the value is set to yes. |

**Table 5-51** find-compressed parameters

| Parameter | Description |
|---|---|
| vary | Specifies whether to insert a Vary: Accept-Encoding header. Possible values are yes or no.<br>• If set to yes, a Vary: Accept-Encoding header is always inserted when a compressed version of a file is selected.<br>• If set to no, a Vary: Accept-Encoding header is never inserted.<br>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

## 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, 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 5-52** 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 5-53** find-links parameters

| Parameter | Description |
|---|---|
| disable | Character string of links to disable:<br>• h is hard links<br>• s is soft links<br>• o allows symbolic links from user home directories only if the user owns the target of the link |
| 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

# 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 5-54** 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 5-55** 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`.<br>• `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.<br>• `0` tells the function not to redo the SSL3 handshake if the server does not already have the client certificate.<br>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`.<br>• `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`.<br>• `0` tells the function to return `REQ_NOACTION` if the client certificate is not present after `dorequest` is handled. |
| `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/server1/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/server1/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/server1/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/proxy4/docs are subject to the access rules in the file my.nsconfig.

```
<Object ppath="D:/sun/server1/docs/*">
PathCheck fn="load-config" file="my.nsconfig" descend=1
basedir="D:/sun/server1/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 5-56** 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 5-56** load-config parameters

| 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.<br>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

See "match-browser" on page 140.

## 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 5-57** 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

## 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 5-58** ntcgicheck parameters

| Parameter | Description |
|---|---|
| extension | The replacement file extension. |

**Table 5-58** ntcgicheck parameters

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

### Example

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

### See Also

send-wincgi, send-shellcgi

## 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 5-59** 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. |

**Table 5-59** require-auth parameters

| Parameter | Description |
|---|---|
| `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`, `basic-ncsa`

## require-proxy-auth

Applicable in `PathCheck`-class directives.

The `require-proxy-auth` function is a `PathCheck` function that makes sure that users are authenticated and triggers a password pop-up window.

### Syntax

```
PathCheck fn=require-proxy-auth
   auth-type=basic
   realm=name
   auth-group=group
   auth-users=name
```

### Parameters

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

**Table 5-60** require-proxy-auth parameters

| Parameter | Description |
|---|---|
| auth-type | Specifies the type of authorization to be used. The type should be “basic” unless you are running a UNIX proxy and are going to use your own function to perform authentication. |
| realm | A string (enclosed in double-quotation marks) sent to the client application so users can see what object they need authorization for. |
| auth-user | (optional) specifies a list of users who get access. The list should be enclosed in parentheses with each user name separated by the pipe \| symbol. |
| auth-group | (optional) specifies a list of groups that get access. Groups are listed in the password-type file. |

### Example

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

## set-variable

See “set-variable” on page 143.

## 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™ ONE 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 5-61** 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 application
PathCheck 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 5-62** 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 5-63** 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 5-64** 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

## url-check

Applicable in PathCheck-class directives.

The url-check function checks the validity of URL syntax.

## url-filter

Applicable in PathCheck-class directives.

The url-filter can be used to allow or deny URL patterns. You can use either regular expressions of URL patterns or names of filter files of URLs ("names" here refer to parameter names which were associated with filter files of URLs through init-url-filter SAF) as values for allow and deny parameters.

### Parameters

The following table describes parameters for the url-filter function.

**Table 5-65** url-filter parameters

| Parameter | Description |
|---|---|
| allow | Regular expression matching a URL pattern or name of a filter of URLs. |
| deny | Regular expression matching a URL pattern or name of a filter of URLs. |
| bong-file | Absolute path the custom error file (text or HTML) to be returned to the client. |

### Example

```
PathCheck fn="url-filter" allow="filt1"
deny=".*://.*.iplanet.com/.*"
```

## user-agent-check

Applicable in PathCheck-class directives.

The user-agent-check can be used to restrict access to the proxy server based on the type and version of the client's web browser. A regular expression to match with user-agent header sent from the client is passed as a parameter to this function.

### Parameters

The following table describes parameters for the user-agent-check function.

**Table 5-66** user-agent-check parameters

| Parameter | Description |
|---|---|
| ua | Regular expression matching user-agent header sent from the client to the proxy server. |

### Example

```
PathCheck fn = "user-agent-check" ua="Mozilla/.*"
```

# 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 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:

- block-auth-cert instructs the proxy server not to forward the client's SSL/TLS certificate to remote servers.
- block-cache-info instructs the proxy server not to forward information about local cache hits to remote servers.
- block-cipher instructs the proxy server to forward the name of the client's SSL/TLS cipher suite to remote servers.
- block-ip instructs the proxy server not to forward the client's IP address to remote servers.
- block-issuer-dn instructs the proxy server not to forward the distinguished name of the issuer of the client's SSL/TLS certificate to remote servers.
- block-keysize instructs the proxy server not to forward the size of the client's SSL/TLS key to remote servers.
- block-proxy-auth instructs the proxy server not to forward the client's proxy authentication credentials.
- block-secret-keysize instructs the proxy server not to forward the size of the client's SSL/TLS secret key to remote servers.

- `block-ssl-id` instructs the proxy server not to forward the client's SSL/TLS session ID to remote servers.
- `block-user-dn` instructs the proxy server not to forward the distinguished name of the subject of the client's SSL/TLS certificate to remote servers.
- `cache-enable` tells the proxy that an object is cacheable, based on specific criteria.
- `cache-setting` sets parameters used for cache control.
- `force-type` sets the `Content-Type` header for the response to a specific type.
- `forward-auth-cert` instructs the proxy server to forward the client's SSL/TLS certificate to remote servers.
- `forward-cache-info` instructs the proxy server to forward information about local cache hits to remote servers.
- `forward-cipher` instructs the proxy server to forward the name of the client's SSL/TLS cipher suite to remote servers.
- `forward-ip` instructs the proxy server to forward the client's IP address to remote servers.
- `forward-issuer-dn` instructs the proxy server to forward the distinguished name of the issuer of the client's SSL/TLS certificate to remote servers.
- `forward-keysize` instructs the proxy server to forward the size of the client's SSL/TLS key to remote servers.
- `forward-proxy-auth` instructs the proxy server to forward the client's proxy authentication credentials
- `forward-secret-keysize` instructs the proxy server to forward the size of the client's SSL/TLS secret key to remote servers.
- `forward-ssl-id` instructs the proxy server to forward the client's SSL/TLS session ID to remote servers.
- `forward-user-dn` instructs the proxy server to forward the distinguished name of the subject of the client's SSL/TLS certificate to remote servers.
- `http-client-config` configures the proxy server's HTTP client.
- `java-ip-check` allows clients to query the proxy server for the IP address used to reroute a resource.

- set-basic-auth matches specific strings in the User-Agent string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.
- set-basic-auth sets the HTTP basic authentication credentials used by the proxy server when it sends an HTTP request.
- set-default-type allows you to define a default charset, content-encoding, and content-language for the response being sent back to the client.
- set-variable 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 requests that .htm and .html files are parsed for server-parsed HTML commands.
- ssl-client-config configures options used when the proxy server connects to a remote server using SSL / TLS.
- suppress-request-headers configures the proxy server to remove outgoing headers from the request.
- type-by-exp sets the Content-Type header for the response based on the requested path.
- type-by-extension sets the Content-Type header for the response based on the file's extension and the MIME types database.

# block-auth-cert

Applicable in ObjectType-class directives.

The block-auth-cert function instructs the proxy server not to forward the client's SSL / TLS certificate to remote servers.

### Parameters

None.

# block-cache-info

Applicable in ObjectType-class directives.

The block-cache-info function instructs the proxy server not to forward information about local cache hits to remote servers.

### Parameters

None.

## block-cipher

Applicable in `ObjectType`-class directives.

The `block-cipher` function instructs the proxy server to forward the name of the client's SSL / TLS cipher suite to remote servers.

### Parameters

None.

## block-ip

Applicable in `ObjectType`-class directives.

The block-ip function instructs the proxy server not to forward the client's IP address to remote servers.

### Parameters

None.

## block-issuer-dn

Applicable in `ObjectType`-class directives.

The `block-issuer-dn` function instructs the proxy server not to forward the distinguished name of the issuer of the client's SSL / TLS certificate to remote servers.

### Parameter

None.

## block-keysize

Applicable in `ObjectType`-class directives.

The `block-keysize` function instructs the proxy server not to forward the size of the client's SSL / TLS key to remote servers.

### Parameters

None.

## block-proxy-auth

Applicable in `ObjectType`-class directives.

The `block-proxy-auth` function instructs the proxy server not to forward the client's proxy authentication credentials (that is, the client's Proxy-authorization HTTP request header) to remote servers.

### Parameter

None.

## block-secret-keysize

Applicable in `ObjectType`-class directives.

The `block-secret-keysize` function instructs the proxy server not to forward the size of the client's SSL / TLS secret key to remote servers.

### Parameters

None.

## block-ssl-id

Applicable in `ObjectType`-class directives.

The `block-ssl-id` function instructs the proxy server not to forward the client's SSL / TLS session ID to remote servers.

### Parameters

None.

## block-user-dn

Applicable in `ObjectType`-class directives.

The `block-user-dn` function instructs the proxy server not to forward the distinguished name of the subject of the client's SSL/TLS certificate to remote servers.

### Parameters

None.

## cache-enable

Applicable in `ObjectType`-class directives.

The `cache_enable` function tells the proxy that an object is cacheable, based on specific criteria. As an example, if it appears in the object `<Object ppath="http://.*">`, then all the HTTP documents are considered cacheable, as long as other conditions for an object to be cacheable are met.

### Syntax

```
ObjectType fn=cache-enable
   cache-auth=0|1
   query-maxlen=number
   min-size=number
   max-size=number
   log-report=feature
   cache-local=0|1
```

### Parameters

The following table describes parameters for the `cache-enable` function.

**Table 5-67** cache-enable parameters

| Parameter | Description |
|---|---|
| `cache-enable` | Tells the proxy that an object is cacheable. As an example, if it appears in the object <Object ppath="http://.*">, then all HTTP documents are considered cacheable (as long as other conditions for an object to be cacheable are met). |
| `cache-auth` | Specifies whether to cache items that require authentication. If set to 1, pages that require authentication can be cached also. If not specified, defaults to 0. |
| `query-maxlen` | Specifies the number of characters in the query string (the "?string" part at the end of the URL) that are still cacheable. The same queries are rarely repeated exactly in the same form by more than one user, and so caching them is often not desirable. That's why the default is 0. |

**Table 5-67** cache-enable parameters

| Parameter | Description |
|---|---|
| `min-size` | The minimum size, in kilobytes, of any document to be cached. The benefits of caching are greatest with the largest documents. For this reason, some people prefer to cache only larger documents. |
| `max-size` | The maximum size in kilobytes of any document to be cached. This allows users to limit the maximum size of cached documents, so no single document can take up too much space. |
| `log-report` | Used to control the feature that reports local cache accesses back to the origin server so that content providers get their true access logs. |
| `cache-local` | Used to enable local host caching, that is, URLs without fully qualified domain names, in the proxy. If set to 1, local hosts are cached. If not specified, it defaults to 0, and local hosts are not cached. |

### Example

The following example of `cache-enable` allows you to enable caching of objects matching the current resource. This applies to normal, non-query, non-authenticated documents of any size. The proxy requires that the document carries either last-modified or expires headers or both, and that the content-type reported by the origin server (if present) is accurate.

```
ObjectType fn=cache-enable
```

The example below is like the first example, but it also caches documents that require user authentication, and it caches queries up to five characters long. The `cache-auth=1` indicates that an up-to-date check is always required for documents that need user authentication (this forces authentication again).

```
ObjectType fn=cache-enable
   cache-auth=1
   query-maxlen=5
```

The example below is also like the first example, except that it limits the size of cache files to a range of 2 KB to 1 MB.

```
ObjectType fn=cache-enable
   min-size=2
   max-size=1000
```

# cache-setting

Applicable in `ObjectType`-class directives.

`cache-setting` is an `ObjectType` function that sets parameters used for cache control.

This function is used to explicitly cache (or not cache) a resource, create an object for that resource, and set the caching parameters for the object.

### Syntax

```
ObjectType fn=cache-setting
   max-uncheck=seconds
   lm-factor=factor
   connect-mode=always|fast-demo|never
   cover-errors=number
```

### Parameters

The following table describes parameters for the `cache-setting` function.

**Table 5-68** cache-setting parameters

| Parameter | Description |
|---|---|
| `max-uncheck` | (Optional) is the maximum time in seconds, allowed between consecutive up-to-date checks. If set to 0 (default), a check is made every time the document is accessed, and the lm-factor has no effect. |
| `lm-factor` | (optional)A floating point number representing the factor used in estimating expiration time (how long a document might be up to date based on the time it was last modified). The time elapsed since the last modification is multiplied by this factor, and the result gives the estimated time the document is likely to remain unchanged. Specifying a value of 0 turns off this function, and then the caching system uses only explicit expiration information (rarely available). Only explicit Expires HTTP headers are used. This value has no effect if max-uncheck is set to 0. |
| `connect-mode` | Specifies network connectivity and can be set to these values:<br>• `always` (default) connects to remote servers when necessary.<br>• `fast-demo` connects only if the item isn't found in the cache.<br>• `never` no connection to a remote server is ever made; returns an error if the document is not found in the cache. |

**Table 5-68** cache-setting parameters

| Parameter | Description |
|---|---|
| `cover-errors` | If present and greater than 0, returns a document from the cache if the remote server is down and an up-to-date check cannot be made. The value specified is the maximum number of seconds since the last up-to-date check; if more time has elapsed, an error is returned. Using this feature involves the risk of getting stale data from the cache while the remote server is down. Setting this value to 0, or not specifying it (default) causes an error to be returned if the remote server is unavailable. |
| `term-percent` | means to keep retrieving if more than the specified percentage of the document has already been retrieved. |

### Example

```
<Object ppath="http://.*">
ObjectType fn=cache-enable
ObjectType fn=cache-setting max-uncheck="7200"
ObjectType fn=cache-setting lm-factor="0.020"
ObjectType fn=cache-setting connect-mode="fast-demo"
ObjectType fn=cache-setting cover-errors="3600"
Service fn=proxy-retrieve
</Object>

# Force check every time
ObjectType fn=cache-setting max-uncheck=0
# Check every 30 minutes, or sooner if changed less than
# 6 hours ago (factor 0.1; last change 1 hour ago would
# give 6-minute maximum check interval).
ObjectType fn=cache-setting max-uncheck=1800 lm-factor=0.1
# Disable caching of the current resource
ObjectType fn=cache-setting cache-mode=nothing
```

# 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 5-69** 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). |
| 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, type-by-exp

# forward-auth-cert

Applicable in ObjectType-class directives.

The forward-auth-cert function instructs the proxy server to forward the client’s SSL / TLS certificate to remote servers.

### Parameters

The following table describes parameters for the forward-auth-cert function.

**Table 5-70** forward-auth-cert parameters

| Parameter | Description |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate the client’s DER-encoded SSL/TLS certificate in Base64 encoding. The default is "Proxy-auth-cert". |

# forward-cache-info

Applicable in ObjectType-class directives.

The forward-cache-info function instructs the proxy server to forward information about local cache hits to remote servers.

### Parameter

The following table describes parameters for the forward-cache-info function.

**Table 5-71** forward-cache-info parameters

| Parameter | Description |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate information about local cache hits. The default is "Cache-info". |

# forward-cipher

Applicable in ObjectType-class directives.

The forward-cipher function instructs the proxy server to forward the name of the client’s SSL / TLS cipher suite to remote servers.

### Parameters

The following table describes parameters for the `forward-cipher` function.

**Table 5-72** forward-cipher parameters

| Parameter | Description |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate the name of the client's SSL/TLS cipher suite. The default is `Proxy-cipher`. |

## forward-ip

Applicable in `ObjectType`-class directives.

The `forward-ip` function instructs the proxy server to forward the client's IP address to remote servers.

### Parameters

The following table describes parameters for the `forward-ip` function.

**Table 5-73** forward-ip parameters

| Parameter | Description |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate the client's IP address. The default is "Client-ip". |

## forward-issuer-dn

Applicable in `ObjectType`-class directives.

The `forward-issuer-dn` function instructs the proxy server to forward the distinguished name of the issuer of the client's SSL/TLS certificate to remote servers.

### Parameters

The following table describes parameters for the `forward-issuer-dn` function.

**Table 5-74** forward-issuer-dn

| Parameter | Description |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate the distinguished name of the issuer of the client's SSL/TLS certificate. The default is "Proxy-issuer-dn". |

# forward-keysize

Applicable in `ObjectType`-class directives.

The `forward-keysize` function instructs the proxy server to forward the size of the client's SSL / TLS key to remote servers.

### Parameters

The following table describes parameters for the `forward-keysize` function.

**Table 5-75** forward-keysize

| Parameter | Description |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate the size of the client's SSL/TLS key. The default is "Proxy-keysize". |

# suppress-request-headers

See "suppress-request-headers" on page 126.

# forward-proxy-auth

Applicable in `ObjectType`-class directives.

The `forward-proxy-auth` instructs the proxy server to forward the client's proxy authentication credentials (that is, the client's Proxy-authorization HTTP request header) to remote servers.

### Parameters

None.

# forward-secret-keysize

Applicable in `ObjectType`-class directives.

The `forward-secret-keysize` function instructs the proxy server to forward the size of the client's SSL / TLS secret key to remote servers.

### Parameters

The following table describes parameters for the `forward-secret-keysize` function.

**Table 5-76** forward-secret-keysize parameters

| Parameter | Description |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate the size of the client's SSL/TLS secret key. The default is "Proxy-secret-keysize". |

# forward-ssl-id

Applicable in `ObjectType`-class directives.

The `forward-ssl-id` function instructs the proxy server to forward the client's SSL / TLS session ID to remote servers.

### Parameter

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

**Table 5-77** forward-ssl-id parameters

| Parameter | Description |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate the client's SSL/TLS session ID. The default is "Proxy-ssl-id". |

# forward-user-dn

Applicable in `ObjectType`-class directives.

The forward-user-dn function instructs the proxy server to forward the distinguished name of the subject of the client's SSL / TLS certificate to remote servers.

### Parameters

The following table describes parameters for the `forward-user-dn` function.

**Table 5-78** forward-user-dn parameters

| Parameter | Descrioption |
|---|---|
| hdr | (Optional) Name of the HTTP request header used to communicate the distinguished name of the subject of the client's SSL/TLS certificate. The default is "Proxy-user-dn". |

## http-client-config

Applicable in `ObjectType`-class directives.

The `http-client-config` function configures the proxy server's HTTP client.

### Parameters

The following table describes parameters for the `http-client-config` function.

**Table 5-79** http-client-config parameters

| Parameter | Description |
|---|---|
| keep-alive | (Optional) Boolean that indicates whether the HTTP client should attempt to use persistent connections. The default is "true". |
| keep-alive-timeout | (Optional) The maximum number of seconds to keep a persistent connection open. The default is "29". |
| always-use-keep-alive | (Optional) Boolean that indicates whether the HTTP client can reuse existing persistent connections for all types of requests. The default is "false", meaning persistent connections will not be reused for non-GET requests nor for requests with a body. |
| protocol | (Optional) HTTP protocol version string. By default, the HTTP client uses either "HTTP/1.0" or "HTTP/1.1" based on the contents of the HTTP request. In general, you should not use the protocol parameter unless you encounter specific protocol interoperability problems. |
| proxy-agent | (Optional) Value of the Proxy-agent HTTP request header. The default is a string that contains the proxy server product name and version. |

# java-ip-check

Applicable in ObjectType-class directives.

The java-ip-check function allows clients to query the proxy server for the IP address used to reroute a resource. Because DNS spoofing often occurs with Java Applets, this feature enables clients to see the true IP address of the origin server. When this feature is enabled, the proxy server attaches a header containing the IP address that was used for connecting to the destination origin server.

### Syntax

```
ObjectType fn=java-ip-check
   status=on|off
```

### Parameters

The following table describes parameters for the java-ip-check function.

**Table 5-80** java-ip-check parameters

| Parameter | Description |
|---|---|
| status | Specifies whether Java IP address checking is enabled or not. Possible values are:<br>• on means that Java IP address checking is enabled and that IP addresses will be forwarded to the client in the form of a document header. On is the default setting.<br>• off means that Java IP address checking is disabled. |

# match-browser

See "match-browser" on page 140.

# set-basic-auth

Applicable in ObjectType-class directives.

The set-basic-auth function sets the HTTP basic authentication credentials used by the proxy server when it sends an HTTP request. set-basic-auth can be used to authenticate to a remote origin server or proxy server.

### Parameters

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

**Table 5-81** set-basic-auth parameters

| Parameter | Description |
|---|---|
| `user` | To authenticate user. |
| `password` | The user's password. |
| `hdr` | (Optional) Name of the HTTP request header used to communicate the credentials. |

# 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 5-82** 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

See “set-variable” on page 143.

## 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 5-83** 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
```

## ssl-client-config

Applicable in `ObjectType`-class directives.

The ssl-client-config function configures options used when the proxy server connects to a remote server using SSL / TLS.

### Parameter

The following table describes parameters for the ssl-client-config function.

**Table 5-84** ssl-client-config parameters

| Parameter | Description |
|---|---|
| client-cert-nickname | (Optional) Nickname of the client certificate to present to the remote server. The default is not to present a client certificate. |
| validate-server-cert | (Optional) Boolean that indicates whether the proxy server validates the certificate presented by the remote server. The default is "false", meaning the proxy server will accept any certificate. |

## 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, 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 5-85** 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 5-85** type-by-exp parameters

| 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, force-type

# 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 , see Chapter 2, "Server Configuration Elements in server.xml."

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

```
type=text/html    exts=htm,html
type=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 5-86** 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, force-type

## Input

All Input directives are executed when the server or a plugin 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 Proxy Server 4 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` adds a filter to the filter stack to process incoming data.
- `match-browser` matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.
- `remove-filter` removes a filter from the filter stack.
- `set-variable` 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 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 5-87** 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

See "match-browser" on page 140.

## 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 5-88** 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 143.

# Output

All `Output` directives are executed when the server or a plugin 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 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.

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

- `content-rewrite` rewrites the string in the document that is being sent to the client.
- `insert-filter` adds a filter to the filter stack to process outgoing data.
- `match-browser` matches specific strings in the `User-Agent` string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.
- `remove-filter` removes a filter from the filter stack.
- `set-variable` enables you to change server settings based upon conditional information in a request, and to manipulate variables in parameter blocks by using specific commands.

# content-rewrite

The `content-rewrite` function rewrites the string in the document that is being sent to the client.

When a document is sent by the proxy server, the `content-rewrite` function is invoked if it has been configured and would replace the `from` string / url to `destination` string / url before sending the response to the client.

The patterns are nothing but strings which would be replaced in the outgoing document. The pattern can be either an url with absolute or relative links, or any text string like server name and so on.

### Syntax

```
Output fn="insert-filter" filter="content-rewrite" type="text/html"
from="<sourcepattern>" to="<destpattern>"
```

### Parameters

The following table describes parameters for the `content-rewrite` function.

**Table 5-89** content-rewrite parameters

| Parameter | Description |
|---|---|
| `filter` | Specifies the name of the filter to be executed. |
| `type` | Indicates on what content-type this filter is applied. For example, text , html, and so on. |

### Example

```
Output fn="insert-filter" type="text/*" filter="content-rewrite"
from="iPlanet"  to="Sun ONE (now called) Sun Java System Web Server"
```

# 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 5-90** 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

See "match-browser" on page 140.

## 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 5-91** 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 143.

# 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.

- `timeout`

  (Optional) Used by the ftp and connect proxy to determine the value of connection timeout.

If there is more than one `Service`-class function, the first one matching the optional wildcard parameters (`type`, `method`, and `query`) is executed.

By default, the server sends the requested file to the client by calling the `send-file` 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 Proxy Server 4.0.1 *NSAPI Developer's Guide*.

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

- add-footer appends a footer specified by a file name or URL to an HTML file.
- add-header prepends a header specified by a file name or URL to an HTML file.
- append-trailer appends text to the end of an HTML file.
- deny-service prevents access to the requested resource.
- imagemap handles server-side image maps.
- index-common generates a fancy list of the files and directories in a requested directory.
- index-simple generates a simple list of files and directories in a requested directory.
- key-toosmall indicates to the client that the provided certificate key size is too small to accept.
- list-dir lists the contents of a directory.
- make-dir creates a directory.
- match-browser matches specific strings in the User-Agent string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.
- proxy-retrieve retrieves a document from a remote server and returns it to the client. It manages caching if it is enabled.
- query-handler handles the HTML ISINDEX tag.
- remove-dir deletes an empty directory.
- remove-file deletes a file.
- remove-filter removes a refilter from the filter stack.
- rename-file renames a file.
- send-error sends an HTML file to the client in place of a specific HTTP response status.

- `send-file` sends a local file to the client.
- `send-range` sends a range of bytes of a file to the client.
- `send-shellcgi` sets up environment variables, launches a shell CGI program, and sends the response to the client.
- `send-wincgi` sets up environment variables, launches a WinCGI program, and sends the response to the client.
- `service-dump` creates a performance report based on collected performance bucket data.
- `service-j2ee` services requests made to Java web applications. This is applicable only to the Administration Server.
- `service-trace` services TRACE requests.
- `set-variable` 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` parses an HTML file for server-parsed HTML commands.
- `stats-xml` creates a performance report in XML format.
- `upload-file` 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` function.

### Parameters

The following table describes parameters for the `add-footer` function.

**Table 5-92** add-footer parameters

| Parameter | Description |
|---|---|
| `file` | (Optional) Path name to the file containing the footer. Specify either `file` or `uri`.<br>By default, the path name is relative. If the path name is absolute, pass the `NSIntAbsFilePath` parameter as yes. |

**Table 5-92** add-footer parameters

| Parameter | Description |
|---|---|
| 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. |
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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/Server1/server1/footers/footer1.html" NSIntAbsFilePath="yes"
```

### See Also

append-trailer, add-header

## 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 5-93** add-header parameters

| Parameter | Description |
|---|---|
| file | (Optional) Path name to the file containing the header. Specify either file or uri.<br>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 | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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/Server61/server1/headers/header1.html" NSIntAbsFilePath="yes"
```

### See Also

add-footer, append-trailer

# 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 5-94** 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.<br>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 | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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, add-header

## deny-service

See "deny-service" on page 165.

## 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 5-95** imagemap parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |

**Table 5-95** imagemap parameters

| Parameter | Description |
|---|---|
| ChunkedRequestTimeout | (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, 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 or home-page.

The icons displayed are .gif files dependent on the content-type of the file, as listed in the following table:

**Table 5-96** content-type icons

| Content-type | Icon |
|---|---|
| "text/*" | text.gif |
| "image/*" | image.gif |
| "audio/*" | sound.gif |

**Table 5-96** content-type icons

| Content-type | Icon |
|---|---|
| "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 5-97** 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 | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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, find-index, home-page

## index-simple

Applicable in Service-class directives.

The index-simple function generates a simple index of the files in the requested directory. It 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 or home-page.

### Parameters

The following table describes parameters for the index-simple function.

**Table 5-98** index-simple parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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

## key-toosmall

Applicable in Service-class directives.

**NOTE** This function is replaced by the PathCheck-class SAF ssl-check.

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 5-99** key-toosmall parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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 5-100** list-dir parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |

**Table 5-100** list-dir parameters

| Parameter | Description |
|---|---|
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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 5-101** make-dir parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (Optional) Common to all Service-class functions. |
| bucket | (Optional) Common to all obj.conf functions. |

### Example

```
Service fn="make-dir" method="MKDIR"
```

## match-browser

See “match-browser” on page 140.

## proxy-retrieve

The `proxy-retrieve` function retrieves a document from a remote server and returns it to the client. It manages caching if it is enabled. The `proxy-retrieve` function also lets you configure the proxy to allow or block arbitrary methods.

### Syntax

```
Service fn=proxy-retrieve
   method=GET|HEAD|POST|INDEX|CONNECT...

   allow|block=<List-of-comma-separated-methods>
```

### Parameters

**method** lets you specify a retrieval method.

**allow** configures the proxy to allow specified arbitrary methods.

**block** configures the proxy to block specified arbitrary methods.

**NOTE** **allow** takes precedence over **block**.

### Examples

```
# Normal proxy retrieve
Service fn=proxy-retrieve
# Proxy retrieve with POST method disabled
Service fn=proxy-retrieve
    method=(POST)
# Proxy retrieve allows methods FOO and BAR to pass through
Service fn=proxy-retrieve
    allow="FOO,BAR"
# Proxy retrieve blocks methods MKCOL,DELETE,LOCK,UNLOCK
Service fn=proxy-retrieve
    block="MKCOL,DELETE,LOCK,UNLOCK"
```

# 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 5-102** query-handler parameters

| Parameter | Description |
|---|---|
| path | Full path and file name of the CGI program to run. |
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |

**Table 5-102** query-handler parameters

| Parameter | Description |
|---|---|
| ChunkedRequestTimeout | (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 5-103** remove-dir parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (Optional) Common to all Service-class functions. |

**Table 5-103** remove-dir parameters

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

### Example

```
Service fn="remove-dir" method="RMDIR"
```

## remove-file

Applicable in Service-class directives.

The remove-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 remove-file when the request method is DELETE.

### Parameters

The following table describes parameters for the remove-file function.

**Table 5-104** remove-file parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (Optional) Common to all Service-class functions. |
| bucket | (Optional) Common to all obj.conf functions. |

### Example

```
Service fn="remove-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 5-105** remove-filter parameters

| Parameter | Description |
|---|---|
| filter | Specifies the name of the filter to remove. |
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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. It 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 5-106** rename-file parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (Optional) Common to all Service-class functions. |
| bucket | (Optional) Common to all obj.conf functions. |

### Example

```
Service fn="rename-file" method="MOVE"
```

# 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 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 5-107** 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. |
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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. It 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."

### Parameters

The following table describes parameters for the send-file function.

**Table 5-108** 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.<br>The value you assign to this parameter is ignored. If you do not wish to use this parameter, leave it out. |
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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 that portion.

### Parameters

The following table describes parameters for the send-range function.

**Table 5-109** send-range parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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, consult the Sun Java System Web Proxy Server 4.0.1 *Administration Guide.*

### Parameters

The following table describes parameters for the send-shellcgi function.

**Table 5-110** send-shellcgi parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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, consult the Sun Java System Web Proxy Server 4.0.1 *Administration Guide*.

### Parameters

The following table describes parameters for the send-wincgi function.

**Table 5-111** send-wincgi parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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-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 100).

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 5-112** service-dump parameters

| Parameter | Description |
|---|---|
| type | Must be perf for this function. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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

# service-j2ee

This is applicable only to the Administration Server.

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 5-113** service-j2ee parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (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, error-j2ee

## service-trace

Applicable in Service-class directives.

The service-trace function services TRACE requests. TRACE requests are typically 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 5-114** service-trace parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (Optional) Common to all Service-class functions. |
| query | (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 143.

# shtml_send

Applicable in Service-class directives.

The shtml_send function parses an HTML document, scanning 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 plugin (libShtml.so on UNIX libShtml.dll on Windows) is loaded.

### Parameters

The following table describes parameters for the shtml_send function.

**Table 5-115** 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 | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |
| ChunkedRequestTimeout | (Optional) Common to all Service-class functions. |
| query | (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 a `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`

### Parameters

The following table describes parameters for the `stats-xml` function.

**Table 5-116** stats-xml parameters

| Parameter | Description |
|---|---|
| `type` | (Optional) Common to all `Service`-class functions. |
| `method` | (Optional) Common to all `Service`-class functions. |
| `query` | (Optional) Common to all `Service`-class functions. |
| `UseOutputStreamSize` | (Optional) Common to all `Service`-class functions. |
| `flushTimer` | (Optional) Common to all `Service`-class functions. |
| `ChunkedRequestBufferSize` | (Optional) Common to all `Service`-class functions. |
| `ChunkedRequestTimeout` | (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

## 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 5-117** upload-file parameters

| Parameter | Description |
|---|---|
| type | (Optional) Common to all Service-class functions. |
| method | (Optional) Common to all Service-class functions. |
| query | (Optional) Common to all Service-class functions. |
| UseOutputStreamSize | (Optional) Common to all Service-class functions. |
| flushTimer | (Optional) Common to all Service-class functions. |
| ChunkedRequestBufferSize | (Optional) Common to all Service-class functions. |

**Table 5-117** upload-file parameters

| Parameter | Description |
|---|---|
| ChunkedRequestTimeout | (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 records information about the request in the common log format.
- flex-log records information about the request in a flexible, configurable format.
- match-browser matches specific strings in the User-Agent string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.
- record-useragent records the client's IP address and User-Agent header.
- set-variable 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 Proxy Server.

The common log must have been initialized previously by the init-clf function. For information about rotating logs, see flex-rotate-init in the Sun Java System Web Proxy Server 4.0.1 *NSAPI Developer'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 5-118** 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, flex-log

# 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 Proxy Server.

There are also a number of free statistics generators for the common log format.

The log format is specified by the flex-init function call. For information about rotating logs, see flex-rotate-init in the Sun Java System Web Proxy Server 4.0.1 *NSAPI Developer's Guide*.

### Parameters

The following table describes parameters for the flex-log function.

**Table 5-119** 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.<br>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, record-useragent

## match-browser

See "match-browser" on page 140.

## 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 5-120** 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, flex-log

## 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 143.

# 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 handles errors that occur during execution of Java™ 2 Platform, Enterprise Edition (J2EE™ platform) applications and modules deployed to the Sun Java System Web Proxy Server. This is applicable only to the Administration Server.
- match-browser matches specific strings in the User-Agent string supplied by the browser, and then modifies the behavior of Sun Java System Web Proxy Server based upon the results by setting values for specified variables.
- query-handler runs a CGI program instead of referencing the path requested.
- remove-filter removes a filter from the filter stack.

- `send-error` sends an HTML file to the client in place of a specific HTTP response status.
- `set-variable` 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

This is applicable only to the Administration Server.

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 Proxy Server individually or as part of full J2EE applications.file name

### Parameters

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

**Table 5-121** error-j2ee Parameters

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

### See Also

`ntrans-j2ee`, `service-j2ee`

# match-browser

See "match-browser" on page 140.

# 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 5-122** 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`.<br>This can be any HTTP response status code or reason phrase according to the HTTP specification.<br>The following is a list of common HTTP response status codes and reason strings:<br>• `401 Unauthorized`<br>• `403 Forbidden`<br>• `404 Not Found`<br>• `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 5-123** 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 5-124** 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. |
| 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.<br>This can be any HTTP response status code or reason phrase according to the HTTP specification.<br>The following is a list of common HTTP response status codes and reason strings:<br>• 401 Unauthorized<br>• 403 Forbidden<br>• 404 Not Found<br>• 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 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 143.

# Connect

The Connect directive calls the connect function you specify.

### Syntax

`Connect fn=`*your-connect-function*

Only the first applicable `Connect` function is called, starting from the most restrictive object. Occasionally it is desirable to call multiple functions (until a connection is established). The function returns REQ_NOACTION if the next function should be called. If it fails to connect, the return value is REQ_ABORT. If it connects successfully, the connected socket descriptor will be returned.

The `Connect` function must have this prototype:

```
int your_connect_function(pblock *pb, Session *sn, Request *rq);
```

`Connect` gets its destination host name and port number from:

```
pblock_findval (“connect-host”, rq->vars)
atoi (pblock_findval (“connect-port”, rq->vars))
```

The host can be in a numeric IP address format.

To use the NSAPI custom DNS class functions to resolve the host name, make a call to this function:

```
struct hostent *servact_gethostbyname(char *host name, Session *sn, Request
*rq);
```

### Example

This example uses the native connect mechanism to establish the connection:

```
#include "base/session.h"
#include "frame/req.h"
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
int my_connect_func(pblock *pb, Session *sn, Request *rq)
{

    struct sockaddr_in sa;
    int sd;
    memset(&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port   = htons(atoi (pblock_findval (“connect-port”,
rq->vars)));
    /* host name resolution */
    if (isdigit(*pblock_findval (“connect-host”, rq->vars)))
    sa.sin_addr.s_addr = inet_addr(rq->host);
    else
    {
    struct hostent *hp = servact_gethostbyname(pblock_findval
(“connect-host”, rq->vars), sn, rq));
    if (!hp)
        return REQ_ABORTED; /* can't resolv */
    memcpy(&sa.sin_addr, hp->h_addr, hp->h_lenght);
    }
    /* create the socket and connect */
sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sd == -1)
    return REQ_ABORTED; /* can't create socket */
    if (connect(sd, (struct sockaddr *)&sa, sizeof(sa)) == -1) {
    close(sd);
        return REQ_ABORTED; /* can't connect */
    }
    return sd;           /* ok */
}
```

# DNS

The DNS directive calls either the dns-config built-in function or a DNS function that you specify.

# dns-config

### Syntax

```
DNS fn=dns-config local-domain-levels=<n>
```

`local-domain-levels` specifies the number of levels of subdomains that the local network has. The default is 1.

Web Proxy Server optimizes DNS lookups by reducing the times of trying to resolve hosts that are apparently fully qualified domain names but which DNS would otherwise by default still try to resolve relative to the local domain.

For example, suppose you're in the netscape.com domain, and you try to access the host www.xyzzy.com. At first, DNS will try to resolve:

```
www.xyzzy.com.netscape.com
```

and only after that the real fully-qualified domain name:

```
www.xyzzy.com
```

If the local domain has subdomains, such as `corp.netscape.com`, it would do the two additional lookups:

```
www.xyzzy.com.corp.netscape.com
www.xyzzy.com.netscape.com
```

To avoid these extra DNS lookups, you can suggest to the proxy that it treat host names that are apparently not local as remote, and it should tell DNS immediately not to try to resolve the name relative to the current domain.

If the local network has no subdomains, you set the value to 0. This means that only if the host name has no domain part at all (no dots in the host name) will it be resolved relative to the local domain. Otherwise, DNS should always resolve it as an absolute, fully qualified domain name.

If the local network has one level of subdomains, you set the value to 1. This means that host names that include two or more dots will be treated as fully qualified domain names, and so on.

An example of one level of subdomains would be the netscape.com domain, with subdomains:

```
corp.netscape.com
engr.netscape.com
mktg.netscape.com
```

This means that hosts without a dot, such as step would be resolved with respect to the current domain, such as engr.netscape.com, and so the `dns-config` function would try this:

```
step.engr.netscape.com
```

If you are on corp.netscape.com but the destination host `step` is on the `engr` subdomain, you could say just:

```
step.engr
```

instead of having to specify the fully qualified domain name:

```
step.engr.netscape.com
```

## your-dns-function

This is a `DNS`-class function that you define.

**Syntax**

`DNS fn=`*your-dns-function*

Only the first applicable DNS function is called, starting from the most restrictive object. In the rare case that it is desirable to call multiple DNS functions, the function can return REQ_NOACTION.

The DNS function must have this prototype:

```
int your_dns_function(pblock *pb, Session *sn, Request *rq);
```

To get the host name use

```
pblock_findval("dns-host", rq->vars)
```

and set the host entry using the new NSAPI function

```
dns_set_hostent
```

The struct `hostent *` will not be freed by the caller but will be treated as a pointer to a static area, as with the gethostbyname call. It is a good idea to keep a pointer in a static variable in the custom DNS function and on the next call either use the same struct hostent or free it before allocating a new one.

The DNS function returns REQ_PROCEED if it is successful, and REQ_NOACTION if the next DNS function (or gethostbyname, if no other applicable DNS class functions exist) should be called instead. Any other return value is treated as failure to resolve the host name.

### Example

This example uses the normal gethostbyname call to resolve the host name:

```
#include <nsapi.h>
int my_dns_func(pblock *pb, Session *sn, Request *rq)
{
    char *host = pblock_findval("dns-host", rq->vars);
    struct hostent *hostent;
hostent = gethostbyname(host); // replace with custom DNS implementation
    dns_set_hostent(hostent, sn, rq);
return REQ_PROCEED;
}
```

# Filter

The `Filter` directive runs an external command and then pipes the data through the external command before processing that data in the proxy. This is accomplished using the `pre-filter` function. The format of the `Filter` directive is as follows:

### Syntax

`Filter fn="pre-filter" path="`*/your/filter/prog*`"`

The `Filter` directive performs these tasks:

1. It runs the program `/your/filter/prog` as a separate process.
2. It establishes pipes between the proxy and the external program.
3. It writes the response data from the remote server to the stdin of the external program.
4. It reads the stdout of the program as if it were the response generated by the server.

This is equivalent to this command:

```
Filter fn="pre-filter"
       path="/your/filter/prog"
       headers="stdin"
```

The following `Filter` functions are described in detail in this section:

- `filter-ct`

- filter-html
- pre-filter

# filter-ct

Applicable in Filter-class directives.

filter-ct can be used to block response content that matches a certain MIME type.

### Parameters

The following table describes parameters for the filter-ct function.

**Table 5-125** filter-ct parameters

| Parameter | Description |
|---|---|
| regexp | Regular expression of the mime type to be filtered. |

### Example

```
Filter fn="filter-ct" regexp="(application/octet-stream)"
```

# filter-html

Applicable in Filter-class directives.

filter-html can be used to filter out HTML tags from the response content before sending it to the client.

### Parameters

The following table describes parameters for the filter-html function.

**Table 5-126** filter-html parameters

| Parameter | Description |
|---|---|
| start | HTML start tag |
| end | HTML end tag |

### Example

```
Filter fn="filter-html" start="APPLET" end="APPLET"
```

## pre-filter

Applicable in Filter-class directives.

pre-filter is used to run external filter programs before returning response content to the client.

### Parameters

The following table describes parameters for the pre-filter function.

**Table 5-127** pre-filter parameters

| Parameter | Description |
|---|---|
| path | absolute path to external filter program. |

### Example

```
Filter fn="pre-filter" path="/your/filter/prog"
```

# Route

The Route directive specifies information about where the proxy server should route requests.

## icp-route

Applicable in Route-class directives.

The icp-route function tells the proxy server to use ICP to determine the best source for a requested object whenever the local proxy does not have the object.

### Syntax

```
Route fn=icp-route
   redirect=yes|no
```

### Parameters

The following table describes parameters for the `icp-route` function.

**Table 5-128** icp-route parameters

| Parameter | Description |
|---|---|
| `redirect` | Specifies whether the proxy server will send a redirect message back to the client telling it where to get the object.<br>• yes means the proxy will send a redirect message back to the client to tell it where to retrieve the requested object.<br>• `no` means the proxy will not send a redirect message to the client. Instead it will use the information from ICP to get the object. |

## pa-enforce-internal-routing

Applicable in `Route`-class directives.

The `pa-enforce-internal-routing` function enables internal routing through a proxy array. Internal routing occurs when a non PAC-enabled client routes requests through a proxy array.

### Syntax

```
Route fn="pa_enforce_internal_routing"
   redirect="yes|no"
```

### Parameters

The following table describes parameters for the `pa-enforce-internal-routing` function.

**Table 5-129** pa-enforce-internal-routing parameters

| Parameter | Description |
|---|---|
| `redirect` | Specifies whether or not client's requests will be redirected. Redirecting means that if a member of a proxy array receives a request that it should not service, it tells the client which proxy to contact for that request. |

## pa-set-parent-route

Applicable in `Route`-class directives.

The `pa-set-parent-route` function sets a route to a parent array.

### Syntax

```
Route fn="pa_set_parent_route"
```

## set-proxy-server

Applicable in `Route`-class directives.

The `set-proxy-server` function directs the proxy server to connect to another proxy for retrieving the current resource. It also sets the address and port number of the proxy server to be used.

### Syntax

```
Route fn=set-proxy-server
    server=URL of other proxy server
    host name=otherhost name
    port=number
```

### Parameters

The following table describes parameters for the `set-proxy-server` function.

**Table 5-130** set-proxy-server parameters

| Parameter | Description |
|---|---|
| `server` | URL of the other proxy server. If multiple server parameters are given, the proxy server will distribute load among the specified proxy servers. (For compatibility with earlier releases, hostname and port may be specified instead of server.) |
| `host name` | The name of the host on which the other proxy server is running. |
| `port` | The port number of the remote proxy server. |

### Example

```
Route fn=set-proxy-server
    host name=proxy.netscape.com
    port=8080
```

## set-origin-server

Applicable in `Route`-class directives.

The `set-origin-server` function allows load to be distributed across a set of homogeneous HTTP origin servers by controlling which origin server the proxy server sends a request to.

### Parameters

The following table describes parameters for the `set-origin-server` function.

**Table 5-131** set-origin-server parameters

| Parameter | Description |
|---|---|
| server | URL of an origin server. If multiple server parameters are given, the proxy server will distribute load among the specified origin servers. |
| sticky-cookie | (Optional) Name of a cookie that, when present in a response, will cause subsequent requests to "stick" to that origin server. The default is "JSESSIONID". |
| sticky-param | (Optional) Name of a URI parameter to inspect for route information. When the URI parameter is present in a request URI and its value contains a colon, ':', followed by a route ID, the request will "stick" to the origin server identified by that route ID. The default is "jsessionid". |
| route-hdr | (Optional) Name of the HTTP request header used to communicate route IDs to origin servers. set-origin-server associates each origin server named by a server parameter with a unique route ID. Origin servers may encode this route ID in the URI parameter named by the sticky-param parameter to cause subsequent requests to "stick" to them. The default is "Proxy-jroute". |
| route-cookie | (Optional) Name of the cookie generated by the proxy server when it encounters a sticky-cookie cookie in a response. The route-cookie cookie stores a route ID that enables the proxy server to direct subsequent requests back to the same origin server. The default is "JROUTE". |

**Table 5-131** set-origin-server parameters

| Parameter | Description |
|---|---|
| rewrite-host | (Optional) Boolean that indicates whether the Host HTTP request header is rewritten to match the host specified by the server parameter. The default is "false", meaning the Host header is not rewritten. |
| rewrite-location | (Optional) Boolean that indicates whether Location HTTP response headers that match the server parameter should be rewritten. The default is "true", meaning matching Location headers are rewritten. |
| rewrite-content-location | (Optional) Boolean that indicates whether Content-location HTTP response headers that match the server parameter should be rewritten. The default is "true", meaning matching Content-location headers are rewritten. |
| rewrite-headername | (Optional) Boolean that indicates whether headername HTTP response headers that match the server parameter should be rewritten, where headername is a user-defined header name. With the exception of the Location and Content-location headers, the default is "false", meaning the headername header is not rewritten. |

# set-socks-server

Applicable in `Route`-class directives.

The `set-socks-server` directs the proxy server to connect to a SOCKS server for retrieving the current resource. It also sets the address and port number of the SOCKS server to be used.

### Syntax

```
Route fn=set-socks-server
    host name=sockshost name
    port=number
```

### Parameters

The following table describes parameters for the `set-socks-server` function.

**Table 5-132** set-socks-server parameters

| Parameter | Description |
|---|---|
| `host name` | The name of the host on which the SOCKS server runs. |
| `port` | The port on which the SOCKS server listens. |

### Example

```
ObjectType fn=set-socks-server
    host name=socks.netscape.com
    port=1080
```

## unset-proxy-server

Applicable in Route-class directives.

The unset-proxy-server function tells the proxy server not to connect to another proxy server to retrieve the current resource. This function nullifies the settings of any less specific set-proxy-server functions.

### Syntax

```
Route fn=unset-proxy-server
```

## unset-socks-server

Applicable in Route-class directives.

The unset-socks-server function tells the proxy server not to connect to a SOCKS server to retrieve the current resource. This function nullifies the settings of any less specific set-socks-server functions.

### Syntax

```
Route fn=unset-socks-server
```

Chapter 6

# MIME Types

This chapter discusses the MIME types file.

The sections are:

- Introduction
- Determining the MIME Type
- How the Type Affects the Response
- What Does the Client Do with the MIME Type?
- Syntax of the MIME Types File
- Sample MIME Types File

## 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 Proxy 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 Proxy Server 4.0.1 NSAPI Developer'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 whichever `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 whichever 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, the usual thing is to 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 7

# 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
- dbswitch.conf
- Deployment Descriptors
- generated.instance.acl
- password.conf
- *.clfilter
- bu.conf
- icp.conf
- socks5.conf
- parray.pat
- parent.pat

## certmap.conf

### Purpose

Configures how a certificate, designated by *name*, is mapped to an LDAP entry, designated by *issuerDN*.

### Location

<*Install_Root*>`/bin/https/install/misc`

<*Install_Root*>`/userdb`

### Syntax

```
certmap name issuerDN
name: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 Proxy Server 4.0.1 Administration 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 7-1** 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:<br>• Commented out: takes the user's DN from the certificate as is.<br>• Empty: searches the entire LDAP tree (DN == suffix).<br>• Comma-separated attributes: forms the DN. |

**Table 7-1** certmap.conf properties

| Attribute | Allowed Values | Default Value | Description |
|---|---|---|---|
| `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:<br>• Commented out or empty: sets the filter to `"objectclass=*"`.<br>• Comma-separated attributes: forms the filter. |
| `verifycert` | `on` or `off` | `off` (commented out) | Specifies whether certificates are verified. |
| `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 Proxy Server uses.

### Location

<*Install_Root*>`/userdb`

### Syntax

`directory` *name LDAP_URL*
*name*`:`*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
```

The following table describes properties in the `dbswitch.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 7-2** 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.<br>If not present, the default value of the base DN is the `base DN` value in the LDAP URL.<br>The `basedn` attribute of a `USERDB` element in the `server.xml` file overrides this value. |

**Table 7-2** dbswitch.conf properties

| Property | Allowed Values | Default Value | Description |
|---|---|---|---|
| `digestauth` | `off`, `on` | `off` | Specifies whether the database can perform digest authentication. If `on`, a special Directory Server plugin is required. For information about how to install this plugin, see the Sun Java System Web Proxy Server 4.0.1 *Administration Guide.* |
| `syntax` | `keyfile`,<br>`digest`,<br>`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 Proxy Server 4.0.1 *Administration 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 Proxy Server 4.0.1 *Administration Guide.* |

# Deployment Descriptors

### Purpose

Configures features specific to the Sun Java System Web Proxy Server for deployed web applications.

### Location

The `META-INF` or `WEB-INF` directory of a module or application.

# 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

*`Install_Root`*`/httpacl`

### See Also

*Sun Java System Web Proxy Server 4.01 Administration Guide*

# password.conf

### Purpose

By default, the Sun Java System Web Proxy 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

<*Instance_Directory*>`/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 Proxy Server 4.0.1 *Administration Guide*

# *.clfilter

**Purpose**

The files `obj.conf.clfilter`, `magnus.conf.clfilter`, and `server.xml.clfilter` contain filter specifications for cluster management operations.

**Location**

*`Instance_Directory`*`/config`

# bu.conf

The optional `bu.conf` file contains batch update directives. You can use these directives to update many documents at once. You can time these updates to occur during off-peak hours to minimize the effect on the efficiency of the server. The format of this file is described in this section.

## Accept

A valid URL `Accept` filter consists of any POSIX regular expression. It is used as a filter to test URLs for retrieval in the case of internal updates, and determines whether branching occurs for external updates.

This directive may occur any number of times, as separate `Accept` lines or as comma or white space delimited entries on a single `Accept` line and is applied sequentially. Default behavior is .*, letting all URLs pass.

### Syntax

`Accept` *regular expression*

## Connections

For the `Connections` directive, *n* is the number of simultaneous connections to be used while retrieving. This is a general method for limiting the load on your machine and, more importantly, the remote servers being contacted.

This directive can occur multiple times in a valid configuration, but only the smallest value is used.

### Syntax

`Connections` *n*

## Count

The argument *n* of the `Count` directive specifies the total maximum number of URLs to be updated via this process. This is a simple safeguard for limiting the process and defaults to a value of 300. This directive can occur multiple times in a valid configuration, but only the smallest value is used.

### Syntax

`Count` *n*

## Depth

The `Depth` directive lets you ensure that, while enumerating, all collected objects are no more than a specified number of links away from the initial URL. The default is 1.

### Syntax

`Depth` *depth*

## Object boundaries

The `Object` wrapper signifies the boundaries between individual configurations in the `bupdate.conf` file. It can occur any number of times, though each occurrence requires a unique name.

All other directives are only valid when inside `Object` boundaries.

**Syntax**

```
<Object name=name>
...
</Object>
```

# Reject

A valid URL `Reject` filter consists of any POSIX regular expression. It is used as a filter to test URLs for retrieval in the case of internal updates, and determines whether branching occurs for external updates.

This directive may occur any number of times, as separate `Reject` lines or as comma or white space delimited entries on a single `Reject` line, and is applied sequentially. Default behavior is no reject for internal updates and .* (no branching, get single URL) for recursive updates.

**Syntax**

```
Reject regular expression
```

# Source

In the `Source` directive, if the argument is the keyword `internal`, it specifies batch updates are to be done only on objects currently in the cache (and a directive of `Depth 1` is assumed); otherwise, you specify the name of a URL for recursive enumeration.

This directive can occur only once in a valid configuration.

**Syntax**

```
Source internal
```

```
Source URL
```

# Type

This function lets you control the updating of mime types that the proxy caches. This directive can occur any number of times, in any order.

### Syntax

```
Type ignore

Type inline

Type mime_type
```

### Parameters

`ignore` means that updates will act on all MIME types that the proxy currently caches. This is the default behavior and supersedes all other `Type` directives if specified.

`inline` means that in-lined data is updated as a special type, regardless of any later MIME type exclusions, and are meaningful only when doing recursive updates.

`mime-type` is assumed to be a valid entry from the system `mime-types` file, and is included in the list of MIME types to be updated. If the proxy doesn't currently cache the given MIME type, the object may be retrieved but is not cached.

# icp.conf

This file is used to configure the Internet Cache Protocol (ICP) feature of your server. There are three functions in the `icp.conf` file, and each can be called as many times as necessary. Each function should be on a separate line. The three functions are `add_parent`, `add_sibling`, and `server`.

## add_parent

The `add_parent` function identifies and configures a parent server in an ICP neighborhood.

### Syntax

```
add_parent name=name icp_port=port number
proxy_port=port number mcast_address=IP address ttl=number round=1|2
```

| | |
|---|---|
| **NOTE** | The above text should all be on one line in the `icp.conf` file. |

### Parameters

`name` specifies the name of the parent server. It can be a dns name or an IP address.

`icp_port` specifies the port on which the parent listens for ICP messages.

`proxy_port` specifies the port for the proxy on the parent.

`mcast_address` specifies the multicast address the parent listens to. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors listening to that multicast address can receive, therefore eliminating the need to send a query to each neighbor separately.

`ttl` specifies the time to live for a message sent to the multicast address. ttl controls the number of subnets a multicast message will be forwarded to. If the ttl is set to 1, the multicast message will only be forwarded to the local subnet. If the ttl is 2, the message will go to all subnets that are one hop away.

`round` specifies in which polling round the parent will be queried. A polling round is an ICP query cycle. Possible values are:

- `1` means that the parent will be queried in the first query cycle with all other round one neighbors.
- `2` means that the parent will only be queried if none of the neighbors in polling round one return a "HIT."

### Example

```
add_parent name=proxy1 icp_port=5151 proxy_port=3333
mcast_address=189.98.3.33 ttl=3 round=2
```

# add_sibling

The `add_sibling` function identifies and configures a sibling server in an ICP neighborhood.

### Syntax

`add_sibling name=`*name* `icp_port=`*port number* `proxy_port=`*port number*
`mcast_address=`*IP address* `ttl=`*number* `round=1|2`

**NOTE** The above text will all be on one line in the `icp.conf` file.

### Parameters

`name` specifies the name of the sibling server. It can be a dns name or an IP address.

`icp_port` specifies the port on which the sibling listens for ICP messages.

`proxy_port` specifies the port for the proxy on the sibling.

`mcast_address` specifies the multicast address the sibling listens to. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors listening to that multicast address can receive, therefore eliminating the need to send a query to each neighbor separately.

`ttl` specifies the time to live for a message sent to the multicast address. ttl controls the number of subnets a multicast message will be forwarded to. If the ttl is set to 1, the multicast message will only be forwarded to the local subnet. If the ttl is 2, the message will go to all subnets that are one hop away.

`round` specifies in which polling round the sibling will be queried. A polling round is an ICP query cycle. Possible values are:

- `1` means that the sibling will be queried in the first query cycle with all other round one neighbors. This is the default polling round value.
- `2` means that the sibling will only be queried if none of the neighbors in polling round one return a “HIT.”

### Example

```
add_sibling name=proxy2 icp_port=5151 proxy_port=3333
mcast_address=190.99.2.11 ttl=2 round=1
```

**NOTE** The above text will all be on one line in the `icp.conf` file.

## server

The `server` function identifies and configures the local proxy in an ICP neighborhood.

### Syntax

```
server bind_address=IP address mcast=IP address num_servers=number
icp_port=port number default_route=name default_route_port=port number
no_hit_behavior=fastest_parent|default timeout=seconds
```

**NOTE** The above text will all be on one line in the `icp.conf` file.

### Parameters

`bind_address` specifies the IP address to which the server will bind. For machines with more than one IP address, this parameter can be used to determine which address the ICP server will bind to.

`mcast` the multicast address to which the neighbor listens. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors who are listening to that multicast address can see, therefore eliminating the need to send a query to each neighbor separately.

If both a multicast address and bind address are specified for the neighbor, the neighbor uses the bind address to communicate with other neighbors. If neither a bind address nor a multicast address is specified, the communication subsystem will decide which address to use to send the data.

`num_servers` specifies the number of processes that will service ICP requests.

`icp_port` specifies the port number to which the server will listen.

`default_route` tells the proxy server where to route a request when none of the neighboring caches respond. If default_route and default_route_port are set to "origin," the proxy server will route defaulted requests to the origin server. The meaning of default_route is different depending on the value of no_hit_behavior. If no_hit_behavior is set to default, the default_route is used when none of the proxy array members return a hit. If no_hit behavior is set to fastest_parent, the default_route value is used only if no parent responds.

`default_route_port` specifies the port number of the machine specified as the default_route. If default_route and default_route_port are set to "origin," the proxy server will route defaulted requests to the origin server.

`no_hit_behavior` specifies the proxy's behavior whenever none of the neighbors returns a "HIT" for the requested document. Possible values are:

- `fastest_parent` means the request is routed through the first parent that returned a "MISS."
- `defaul`**t** means the request is routed to the machine specified as the default route.

`timeout` specifies the maximum number of milliseconds the proxy will wait for an ICP response.

### Example

```
server bind_address=198.4.66.78 mcast=no num_servers=5 icp_port=5151
default_route=proxy1 default_route_port=8080
no_hit_behavior=fastest_parent timeout=2000
```

**NOTE** The above text will all be on one line in the `icp.conf` file.

# socks5.conf

The proxy uses <*Install_Root*>/<*Instance_Directory*>`/config/socks5.conf` to control access to the SOCKS proxy server SOCKD and its services. Each line defines what the proxy does when it gets a request that matches the line.

When SOCKD receives a request, it checks the request against the lines in <*Install_Root*>/<*Instance_Directory*>`/config/socks5.conf`. When it finds a line that matches the request, the request is permitted or denied based on the first word in the line (permit or deny). Once it finds a matching line, the daemon ignores the remaining lines in the file. If there are no matching lines, the request is denied. You can also specify actions to take if the client's identd or user ID is incorrect by using #NO_IDENTD: or #BAD_ID as the first word of the line. Each line can be up to 1023 characters long.

There are five sections in the `socks5.conf` file. These sections do not have to appear in the following order. However, because the daemon uses only the first line that matches a request, the order of the lines within each section is extremely important. The five sections of the `socks5.conf` file are:

- ban host/authentication - identifies the hosts from which the SOCKS daemon should not accept connections and which types of authentication the SOCKS daemon should use to authenticate these hosts
- routing - identifies which interface the SOCKS daemon should use for particular IP addresses
- variables and flags - identifies which logging and informational messages the SOCKS daemon should use
- proxies - identifies the IP addresses that are accessible through another SOCKS server and whether that SOCKS server connects directly to the host
- access control - specifies whether the SOCKS daemon should permit or deny a request

When the SOCKS daemon receives a request, it sequentially reads the lines in each of these five sections to check for a match to the request. When it finds a line that matches the request, it reads the line to determine whether to permit or deny the request. If there are no matching lines, the request is denied.

## Authentication/Ban Host Entries

There are two lines in authentication/ban host entries. The first is the authentication line.

**Syntax**

`auth` *source-hostmask source-portrange auth-methods*

**Parameters**

*source-hostmask* identifies which hosts the SOCKS server will authenticate.

*source-portrange* identifies which ports the SOCKS server will authenticate.

*auth-methods* are the methods to be used for authentication. You can list multiple authentication methods in order of your preference. In other words, if the client does not support the first authentication method listed, the second method will be used instead. If the client does not support any of the authentication methods listed, the SOCKS server will disconnect without accepting a request. If you have more than one authentication method listed, they should be separated by commas with no spaces in between. Possible authentication methods are:

- `n` (no authentication required)
- `u` (user name and password required)
- `-` (any type of authentication)

The second line in the authentication/ban host entry is the ban host line.

**Syntax**

`ban` *source-hostmask source-portrange*

**Parameters**

*source-hostmask* identifies which hosts are banned from the SOCKS server.

*source-portrange* identifies from which ports the SOCKS server will not accept requests.

**Example**

```
auth 127.27.27.127 1024 u,-
ban 127.27.27.127 1024
```

# Routing Entries

**Syntax**

`route` *dest-hostmask dest-portrange interface/address*

**Parameters**

*dest-hostmask* indicates the hosts for which incoming and outgoing connections must go through the specified interface.

*dest-portrange* indicates the ports for which incoming and outgoing connections must go through the specified interface.

*interface/address* indicates the IP address or name of the interface through which incoming and outgoing connections must pass. IP addresses are preferred.

**Example**

```
route 127.27.27.127 1024 le0
```

# Variables and Flags

**Syntax**

`set` *variable value*

**Parameters**

`variable` indicates the name of the variable to be initialized.

`value` is the value to set the variable to.

**Example**

```
set SOCKS5_BINDPORT 1080
```

## Available Settings

The following settings are those that can be inserted into the variables and flags section of the `socks5.conf` file. These settings will be taken from the administration forms, but they can be added, changed, or removed manually as well.

### *SOCKS5_BINDPORT*

The `SOCKS5_BINDPORT` setting sets the port at which the SOCKS server will listen. This setting cannot be changed during rehash.

**Syntax**

`set SOCKS5_BINDPORT` *port-number*

**Parameters**

*port-number* is the port at which the SOCKS server will listen.

**Example**

```
set SOCKS5_BINDPORT 1080
```

## *SOCKS5_PWDFILE*

The `SOCKS5_PWDFILE` setting is used to look up user name / password pairs for user name / password authentication.

### Syntax

`set SOCKS5_PWDFILE` *full-pathname*

### Parameters

*full-pathname* is the location and name of the user name / password file.

### Example

```
set SOCKS5_PWDFILE /etc/socks5.passwd
```

## *SOCKS5_LOGFILE*

The `SOCKS5_LOGFILE` setting is used to determine where to write log entries.

### Syntax

`set SOCKS5_LOGFILE` *full-pathname*

### Parameters

*full-pathname* is the location and name of the SOCKS logfile.

### Example

```
set SOCKS-5_LOGFILE /var/log/socks5.log
```

## *SOCKS5_NOIDENT*

THe `SOCKS5_NOIDENT` setting disables Ident so that SOCKS does not try to determine the user name of clients. Most servers should use this setting unless they will be acting mostly as a SOCKS4 server. (SOCKS4 used ident as authentication.)

### Syntax

```
set SOCKS5_NOIDENT
```

### Parameters

None.

## *SOCSK5_DEMAND_IDENT*

The `SOCKS5_DEMAND_IDENT` setting sets the Ident level to “require an ident response for every request”. Using Ident in this way will dramatically slow down your SOCKS server. If neither SOCKS5_NOIDENT or SOCKS5_DEMAND_IDENT is set, then the SOCKS server will make an Ident check for each request, but it will fulfill requests regardless of whether an Ident response is received.

#### Syntax

```
set SOCSK5_DEMAND_IDENT
```

#### Parameters

None.

### SOCKS5_DEBUG

The `SOCKS5_DEBUG` setting causes the SOCKS server to log debug messages. You can specify the type of logging your SOCKS server will use.

If it's not a debug build of the SOCKS server, only number 1 will work.

#### Syntax

`set SOCSK5_DEBUG` *number*

#### Parameters

*number* determines the number of the type of logging your server will use. Possible values are:

- `1` - log normal debugging messages. This is the default.
- `2` - log extensive debugging (especially related to configuration file settings).
- `3` - log all network traffic.

#### Example

```
set SOCKS5_DEBUG 2
```

### SOCKS5_USER

The `SOCKS5_USER` setting sets the user name to use when authenticating to another SOCKS server. This is used when SOCKS server is routed through another down stream SOCKS server which requires authentication.

#### Syntax

`set SOCKS5_USER` *user-name*

#### Parameters

*user-name* is the user name the SOCKS server will use when authenticating to another SOCKS server.

#### Example

```
set SOCKS5_USER mozilla
```

## *SOCKS5_PASSWD*

The SOCKS5_PASSWD setting sets the password to use when authenticating to another SOCKS server. It is possible for a SOCKS server to go through another SOCKS server on its way to the Internet. In this case, if you define SOCKS5_USER, sockd will advertise to other SOCKS servers that it can authenticate itself with a user name and password.

### Syntax

set SOCKS5_PASSWD *password*

### Parameters

*password* is the password the SOCKS server will use when authenticating to another SOCKS server.

### Example

```
set SOCKS5_PASSWD m!2@
```

## *SOCKS5_NOREVERSEMAP*

The SOCKS5_NOREVERSEMAP setting tells sockd not to use reverse DNS. Reverse DNS translates IP addresses into host names. Using this setting can increase the speed of the SOCKS server.

If you use domain masks in the configuration file, the SOCKS server will have to use reverse DNS, so this setting will have no effect.

### Syntax

```
set SOCKS5_NOREVERSEMAP
```

### Parameters

None.

## *SOCKS5_HONORBINDPORT*

The SOCKS5_HONORBINDPORT setting allows the client to specify the port in a BIND request. If this setting is not specified, the SOCKS server ignores the client's requested port and assigns a random port.

### Syntax

```
set SOCKS5_HONORBINDPORT
```

### Parameters

None.

### *SOCKS5_ALLOWBLANKETBIND*

The SOCKS5_ALLOWBLANKETBIND setting allows the client to specify an IP address of all zeros (0.0.0.0) in a BIND request. If this setting is not specified, the client must specify the IP address that will be connecting to the bind port, and an IP of all zeros is interpreted to mean that any IP address can connect.

#### Syntax

```
set SOCKS5_ALLOWBLANKETBIND
```

#### Parameters

None.

### *SOCKS5_WORKERS*

The SOCKS5_WORKERS setting tunes the performance of the SOCKS server by adjusting the number of worker threads. Worker threads perform authentication and access control for new SOCKS connections. If the SOCKS server is too slow, you should increase the number of worker threads. If it is unstable, decrease the number of worker threads.

The default number of worker threads is 40, and the typical number of worker threads falls between 10 and 150.

#### Syntax

```
set SOCKS5_WORKERS number
```

#### Parameters

*number* is the number of worker threads the SOCKS server will use.

#### Example

```
set SOCKS5_WORKERS 40
```

### *SOCKS5_ACCEPTS*

The SOCKS5_ACCEPTS setting tunes the performance of the SOCKS server by adjusting the number of accept threads. Accept threads sit on the SOCKS port listening for new SOCKS requests. If the SOCKS server is dropping connections, you should increase the number of accept threads. If it is unstable, decrease the number of accept threads.

The default number of accept threads is 1, and the typical number of accept threads falls between 1 and 10.

#### Syntax

```
set SOCKS5_ACCEPTS number
```

#### Parameters

*number* is the number of accepts threads the SOCKS server will use.

#### Example

```
set SOCKS5_ACCEPTS 1
```

### LDAP_URL

The `LDAP-URL` setting sets the URL for the LDAP server.

#### Syntax

set LDAP-URL *URL*

#### Parameters

*URL* is the URL for the LDAP server used by SOCKS.

#### Example

```
set LDAP-URL ldap://name:8180/0=Netscape,c=US
```

### LDAP_USER

The `LDAP-USER` setting sets the user name that the SOCKS server will use when accessing the LDAP server.

#### Syntax

set LDAP-USER *user-name*

#### Parameters

*user-name* is the user name SOCKS will use when accessing the LDAP server.

#### Example

```
set LDAP-USER uid=admin
```

### LDAP_PASSWD

The `LDAP-PASSWD` setting sets the password that the SOCKS server will use when accessing the LDAP server.

#### Syntax

set LDAP-PASSWD *password*

#### Parameters

*password* is the password SOCKS will use when accessing the LDAP server.

#### Example

```
set LDAP-PASSWD T$09
```

### *SOCKS5_TIMEOUT*

The `SOCKS5-TIMEOUT` setting specifies the idle period that the SOCKS server will keep a connection alive between a client and a remote server before dropping the connection.

**Syntax**

`set SOCKS5_TIMEOUT` *time*

**Parameters**

*time* is the time, in minutes, SOCKS will wait before timing out. The default value is 10. The value can range from 10 to 60, including both these values.

**Example**

```
set SOCKS5_TIMEOUT 30
```

## Proxy Entries

**Syntax**

*proxy-type dest-hostmask dest-portrange proxy-list*

**Parameters**

*proxy-type* indicates the type of proxy server. This value can be:

- socks5 - SOCKS version 5
- socks4 - SOCKS version 4
- noproxy - a direct connection

*dest-hostmask* indicates the hosts for which the proxy entry applies.

*dest-portrange* indicates the ports for which the proxy entry applies.

*proxy-list* contains the names of the proxy servers to use.

**Example**

```
socks5 127.27.27.127 1080 proxy1
```

# Access Control Entries

### Syntax

`permit|deny` *auth-type connection-type source-hostmask dest-hostmask source-portrange dest-portrange* `[`*LDAP-group*`]`

### Parameters

*auth-type* indicates the authentication method for which this access control line applies.

*connection-type* indicates the type of command the line matches. Possible command types are:

- c (connect)
- b (bind; open a listen socket)
- u (UDP relay)
- - (any command)

*source-hostmask* - indicates the hosts for which the access control entry applies.

*dest-hostmask* indicates the hosts for which the access control entry applies.

*source-portrange* indicates the ports for which the access control entry applies.

*dest-portrange* is the port number of the destination.

*LDAP-group* is the group to deny or permit access to. This value is optional. If no LDAP group is identified, the access control entry applies to everyone.

### Example

```
permit u c - - - [0-1023] group1
```

## Specifying Ports

You will need to specify ports for many entries in your `socks5.conf` file. Ports can be identified by a name, number, or range. Ranges that are inclusive should be surrounded by brackets (i.e. [ ]). Ranges that are not inclusive should be in parentheses.

# parray.pat

The `parray.pat` (PAT) file describes each member in the proxy array of which the proxy you are administering is a member. The PAT file is an ASCII file used in proxy to proxy routing. It contains proxy array members' machine names, IP addresses, ports, load factors, cache sizes, etc.

### Syntax

```
Proxy Array Information/1.0
ArrayEnabled: number
ConfigID: ID number
ArrayName: name
ListTTL: minutes
```

*name IPaddress proxyport URLforPAT infostring state time status loadfactor cachesize*

### Parameters

`Proxy Array Information` is version information.

`ArrayEnabled` specifies whether the proxy array is enabled or disabled. Possible values are:

- `0` means the array is disabled.
- `1` means the array is enabled.

`ConfigID` is the identification number for the current version of the PAT file. The proxy server uses this number to determine whether the PAT file has changed.

`ArrayName` is the name of the proxy array.

`ListTTL` specifies how often the proxy should check the PAT file to see if it has changed. This value is specified in minutes.

`name` is the name of a specific member of the proxy array.

`IPaddress` is the IP address of the member.

`proxyport` is the port at which the master proxy accepts HTTP requests.

`URLforPAT` is the URL of the PAT file that the member will poll the master proxy for.

`infostring` is version information.

`statetime` is the amount of time the member has been in its current state.

`status` specifies whether the member is enabled or disabled.

- `on` means that the member is on.
- `off` means that the member is off. If the member is off, its requests will be routed through another member of the array.

`loadfactor` is an integer that reflects the number of requests that should be routed through the member.

`cachesize` is the size of the member's cache.

### Example

```
Proxy Array Information/1.0
ArrayEnabled: 1
ConfigID: 1
ArrayName: parray
ListTTL: 10

proxy1 200.29.186.77 8080 http://pat SunJavaSystemWebProxy/4 0 on 100 512
proxy2 187.21.165.22 8080 http://pat SunJavaSystemWebProxy/4 0 on 100 512
```

# parent.pat

The `parent.pat` file is the Proxy Array Table file that contains information about an upstream proxy array. This file has the same syntax as the `parray.pat` file.

Appendix A

# Configuration Changes Between iPlanet Web Proxy Server 3.6 and Sun Java System Web Proxy Server 4

See Sun Java System Web Proxy Server 4.0.1 *Installation and Migration Guide.*

Appendix B

# 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 B-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" |
| %c | Date & time "%m/%d/%y %H:%M:%S" |
| %D | Date "%m/%d/%y" |

**Table B-1** Format Strings

| Attribute | Allowed Values |
|---|---|
| %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 C

# Alphabetical List of Server Configuration Elements

## A

ACLFILE

## C

CACHE

## D

DESCRIPTION

## E

EVENTTIME

EVENTACTION

## F

FILECACHE

Appendix D

# Alphabetical List of Predefined SAFs

This appendix provides an alphabetical list for the easy lookup of predefined SAFs.

## A

add-footer

add-header

append-trailer

assign-name

## B

basic-auth

basic-ncsa

block-auth-cert

block-cache-info

block-cipher

block-ip

block-issuer-dn

block-keysize

block-multipart-posts

block-proxy-auth

find-links

find-pathinfo

flex-log

force-type

forward-auth-cert

forward-cache-info

forward-cipher

forward-ip

forward-issuer-dn

forward-keysize

forward-proxy-auth

forward-secret-keysize

forward-ssl-id

forward-user-dn

## G

get-client-cert

get-sslid

## H

home-page

host-dns-cache-init

http-client-config

## I

icp-init

icp-route

map

## N

ntcgicheck

ntrans-j2ee

nt-uri-clean

## P

pac-map

pat-map

pa-enforce-internal-routing

pa-init-parent-array

pa-init-proxy-array

pa-set-parent-route

perf-init

pfx2dir

pool-init

proxy-auth

proxy-retrieve

## Q

query-handler

## R

record-useragent

redirect

ssl-check

ssl-client-config

ssl-logout

stats-init

stats-xml

strip-params

suppress-request-headers

## T

thread-pool-init

tune-cache

tune-proxy

type-by-exp

type-by-extension

## U

unix-home

unix-uri-clean

unset-proxy-server

unset-socks-server

upload-file

url-check

url-filter

user-agent-check

## Y

your-dns-function

# Index