Sun Java System Web Server 7.0 Developer's Guide

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

Part No: 819–2633
January 2007

070111@15821

Contents

Tables

Examples

Preface

This guide is a starting point for developers who need information about using the various APIs and programming technologies that are supported by Sun Java™ System Web Server 7.0. The guide summarizes the APIs, and provides information about configuring your server to work with server-side HTML tags and CGI programs.

Who Should Use This Book

The intended audience for this guide is the person who develops, assembles, and deploys web applications in a corporate enterprise.

This guide assumes you are familiar with the following topics:

- HTML
- Java programming language
- Structured database query languages such as SQL
- Relational database concepts
- Software development processes

Before You Read This Book

Web Server can be installed as a standalone product or as a component of Sun Java Enterprise System (Java ES), a software infrastructure that supports enterprise applications distributed across a network or Internet environment. If you are installing Web Server as a component of Java ES, you should be familiar with the system documentation at `http://docs.sun.com/coll/1286.2`.

Web Server Documentation Set

The Web Server documentation set describes how to install and administer the Web Server. The URL for Web Server documentation is `http://docs.sun.com/coll/1308.3`. For an introduction to Web Server, refer to the books in the order in which they are listed in the following table.

TABLE P–1 Books in the Web Server Documentation Set

Documentation Title	Contents
Sun Java System Web Server 7.0 Documentation Center	Web Server documentation topics organized by tasks and subject
Sun Java System Web Server 7.0 Release Notes	■ Late-breaking information about the software and documentation ■ Supported platforms and patch requirements for installing Web Server
Sun Java System Web Server 7.0 Installation and Migration Guide	Performing installation and migration tasks: ■ Installing Web Server and its various components ■ Migrating data from Sun ONE Web Server 6.0 or 6.1 to Sun Java System Web Server 7.0
Sun Java System Web Server 7.0 Administrator's Guide	Performing the following administration tasks: ■ Using the Administration and command-line interfaces ■ Configuring server preferences ■ Using server instances ■ Monitoring and logging server activity ■ Using certificates and public key cryptography to secure the server ■ Configuring access control to secure the server ■ Using JavaPlatform Enterprise Edition (Java EE) security features ■ Deploying applications ■ Managing virtual servers ■ Defining server workload and sizing the system to meet performance needs ■ Searching the contents and attributes of server documents, and creating a text search interface ■ Configuring the server for content compression ■ Configuring the server for web publishing and content authoring using WebDAV

TABLE P–1 Books in the Web Server Documentation Set (Continued)

Documentation Title	Contents
Sun Java System Web Server 7.0 Developer's Guide	Using programming technologies and APIs to do the following: ■ Extend and modify Sun Java System Web Server ■ Dynamically generate content in response to client requests and modify the content of the server
Sun Java System Web Server 7.0 NSAPI Developer's Guide	Creating custom Netscape Server Application Programmer's Interface (NSAPI) plug-ins
Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications	Implementing Java Servlets and JavaServer Pages™ (JSP™) technology in Sun Java System Web Server
Sun Java System Web Server 7.0 Administrator's Configuration File Reference	Editing configuration files
Sun Java System Web Server 7.0 Performance Tuning, Sizing, and Scaling Guide	Tuning Sun Java System Web Server to optimize performance
Sun Java System Web Server 7.0 Troubleshooting Guide	Troubleshooting Web Server

Related Books

The URL for all documentation about Sun Java Enterprise System (Java ES) and its components is `http://docs.sun.com/app/docs/prod/entsys.06q4`.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

TABLE P–2 Default Paths and File Names

Placeholder	Description	Default Value

install_dir	Represents the base installation directory for Web Server.	Sun Java Enterprise System (Java ES) installations on the Solaris™ platform:
		`/opt/SUNWwbsvr7`
		Java ES installations on the Linux and HP-UX platform:
		`/opt/sun/webserver/`
		Java ES installations on the Windows platform:
		System Drive`:\Program Files\Sun\JavaES5\WebServer7`
		Other Solaris, Linux, and HP-UX installations, non-root user:
		user's home directory`/sun/webserver7`
		Other Solaris, Linux, and HP-UX installations, root user:
		`/sun/webserver7`
		Windows, all installations:
		System Drive`:\Program Files\Sun\WebServer7`

instance_dir	Directory that contains the instance-specific subdirectories.	For Java ES installations, the default location for instances on Solaris:
		`/var/opt/SUNWwbsvr7`
		For Java ES installations, the default location for instances on Linux and HP-UX:
		`/var/opt/sun/webserver7`
		For Java ES installations, the default location for instance on Windows:
		System Drive`:\Program Files\Sun\JavaES5\WebServer7`
		For stand-alone installations, the default location for instance on Solaris, Linux, and HP-UX:
		<install_dir>
		For stand-alone installations, the default location for instance on Windows:
		System Drive`:\Program Files\sun\WebServer7`

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–3 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–4 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.

TABLE P–4 Symbol Conventions (Continued)

Symbol	Description	Example	Meaning
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Accessing Sun Resources Online

The `http://docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. Books are available as online files in PDF and HTML formats. Both formats are readable by assistive technologies for users with disabilities.

To access the following Sun resources, go to `http://www.sun.com`:

- Downloads of Sun products
- Services and solutions
- Support (including patches and updates)
- Training
- Research
- Communities (for example, Sun Developer Network)

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com (docs.sun.com^{SM}) web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for “Web Server,” type the following:

```
Web Server site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use “`sun.com`” in place of “`docs.sun.com`” in the search field.

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – 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.

Sun Welcomes Your Comments

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. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-2633.

CHAPTER 1

Technology Overview

This chapter summarizes the various APIs and programming technologies supported by Web Server.

This chapter has the following sections:

- "Sun Java System Web Server Architecture" on page 19
- "Configuration Files" on page 21
- "Sun Java System Web Server 7.0 APIs" on page 21
- "API Summary" on page 26
- "Changes from Previous Versions" on page 27

Sun Java System Web Server Architecture

Web Server incorporates a modular architecture that integrates seamlessly with all products in the Sun Java System family of servers. In addition,Web Server supports a variety of APIs and programming technologies that enable you to do the following:

- Generate dynamic content in response to client requests
- Modify and extend the behavior of the server
- Modify the content that is stored in the server

Sun Java System Web Server includes a number of software modules, which are discussed in the following topics in this section:

- "Content Engines" on page 20
- "Server Extensions" on page 20
- "Runtime Environments" on page 21
- "Application Services" on page 21

Content Engines

Web Server content engines are designed for manipulating customer data. The following three content engines make up the Web Publishing layer of the Sun Java System Web Server architecture:

- HTTP (Web Server)
- Content Handling
- Search

The HTTP engine represents the core of Sun Java System Web Server. The Web Server architecture resides on top of this engine for performance and integration functionality.

The Content Handling engine enables you to manage your server's content. You can create and store HTML pages, JavaServer Pages (JSP™), and other files such as graphics, text, sound, or video on your server. When clients connect to your server they can view your files (provided they have access to them).

The Search engine enables Web Server users to search the contents and attributes of documents on the server. As a server administrator, you can create a customized text search interface that works with various types of document formats. Web Server converts many types of non-HTML documents into HTML as it indexes them, so users can use a web browser to view the documents that are found for their search.

Server Extensions

Web Server extensions enable you to extend or replace the function of the server to better suit your business operations. The following server extensions are part of the core Web Server architecture:

- Common Gateway Interface (CGI)
- Netscape Server Application Programming Interface (NSAPI)
- Java™ Servlets and JavaServer Pages (JSP)

Common Gateway Interface (CGI) is a standalone application development interface that enables you to create programs that process your client requests dynamically.

Netscape Server Application Programming Interface (NSAPI) implements the functions and server calls when processing a request (Server Application Functions or SAFs), which provide the core and extended functionality of Web Server. It allows the server to process requests and divide into small steps that can be arranged in a variety of ways for speed and flexible configuration.

Java Servlets and JavaServer Pages extensions enable all servlet and JSP meta functions, including instantiation, initialization, destruction, access from other components, and configuration management. Servlets and JSPs are reusable Java applications that run on a Web Server rather than in a web browser.

Runtime Environments

Web Server includes a set of runtime environments that support the server extensions. These runtime environments include the following:

- CGI Processor
- NSAPI Engine
- Java Virtual Machine (JVM)

Application Services

The Web Server architecture includes a set of application services for various application-specific functions. These application services include the following:

- Security and Access Control
- Session Management Service
- File System Service
- Mail Service

Configuration Files

You can configure the Web Server using the Administration user interfaces (UI) or through CLI. Most of the configuration files are in the directory *install_dir*`/https-`*instance*`/config` directory. For example, if Web Server is installed on a Windows machine in `C:\Program Files\Sun\WebServer7`, the configuration files for the server `myserver.com` are in:

`C:\Program Files\Sun\WebServer7\https-`*myserver*`.com\config`

The main configuration files are `magnus.conf`, `server.xml`, `obj.conf`, and `mime.types`. For more information about configuration files, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

Sun Java System Web Server 7.0 APIs

This section summarizes the various APIs and programming technologies supported by Web Server, and describes how to get more information about them. The main categories of extensions and modifications you can make to Web Server are listed below:

Dynamically generate responses (or parts of responses) to requests. The APIs and programming approaches that fall into this category are:

- "Server-parsed HTML Tags" on page 22
- "CGI" on page 22
- "Java Servlets and JavaServer Pages (JSP)" on page 22
- "NSAPI" on page 23
- "Access Control API" on page 24
- "Certificate-Mapping API" on page 25

Modify the content of the server by adding, removing, or modifying resources and directories. To do this, use remote file manipulation.

Server-parsed HTML Tags

Web Server provides a C API for defining your own server-side tags. These tags can be used in addition to the standard server-side tags (such as `config`, `include,` and so on) in HTML files.

For more information about defining and using server-parsed tags, see Chapter 2.

CGI

Common Gateway Interface (CGI) programs run on the server and generate a response to return to the requesting client. CGI programs can be written in various languages, including C, C++, Java, and Perl, and as shell scripts. CGI programs are invoked through URL invocation.

Web Server complies with the version 1.1 CGI specification.

For more information about using CGI with Web Server, see Chapter 2

Java Servlets and JavaServer Pages (JSP)

Web Server supports the Java™ Servlet 2.4 specification, including web application and the JavaServer Pages™ (JSP™) 2.0 specification.

Java servlets are server-side Java programs that can be used to generate dynamic content in response to client requests in much the same way as CGI programs. Servlets are accessed through URL invocation.

You create servlets using the Java Servlets API. Web Server includes all of the files necessary for developing and running Java servlets.

For information about using the Java Servlet API, see the documentation from Sun at: `http://java.sun.com/products/servlet/index.jsp`

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

For more information about creating web applications that use servlets and JSPs on Web Server, see the *Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications*.

For more information about using JavaServer Pages, see the documentation from Sun at: `http://java.sun.com/products/jsp/index.jsp`

NSAPI

Netscape Server Application Programming Interface (NSAPI) is a set of C functions for implementing extensions to the server. These extensions are known as server plug-ins.

Using NSAPI, you can write plug-ins to extend the functionality of Web Server. An NSAPI plug-in defines one or more Server Application Functions (SAFs). You can develop SAFs for implementing custom authorization, custom logging, and for other ways of modifying how Sun Java System Web Server handles requests. For more information, see the *Sun Java System Web Server 7.0 NSAPI Developer's Guide*.

The file `obj.conf` contains instructions (known as directives) that tell the server how to process requests received from clients. Each instruction is enacted either during server initialization or during a particular stage of the request-handling process. Each instruction invokes a SAF.

For example, the following instruction is invoked when the request method is GET and the requested resource is of type `text/html`. This instruction calls the `append-trailer` function with a trailer argument of `<H4><font color=green>Served by 7.0</font></H4>`. The `append-trailer` function simply returns the requested resource to the client, in this case an HTML file, and appends the given trailer to it.

```
Service method=GET type="text/html" fn=append-trailer trailer=
                             "<H4><font color=green>Served by 7.0</font></H4>"
```

Web Server has a set of predefined SAFs. It also has a library of NSAPI functions for developing your own SAFs to modify the way the server handles requests. For more information about predefined SAFs, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference* . For more information about custom SAFs, see the *Sun Java System Web Server 7.0 NSAPI Developer's Guide*.

Note – The file `obj.conf` is essential for the operation of the server. If it does not exist, the server cannot work, because it has nowhere to look for instructions on how to handle requests.

Note – When defining new SAFs, include the header function `nsapi.h` (which is in *install_dir*/`include`) to have access to all NSAPI functions.

Installing NSAPI Plug-ins

To load new NSAPI plug-ins containing customized SAFs into the server, add an `Init` directive to `magnus.conf` to load the shared library file that defines the new SAFs. This directive must call the `load-modules` function, which takes the following arguments:

- `shlib`: The shared library to load.
- `funcs`: The functions to be made available to the server.

  See the *Sun Java System Web Server 7.0 NSAPI Developer's Guide* for more information about the following topics:

- Directives in `obj.conf` and how they determine how the server handles requests
- NSAPI functions available for writing custom SAFs
- Writing custom SAFs
- Loading custom SAFs into Sun Java System Web Server by adding an `Init` directive to `magnus.conf` that calls `load-modules`

  For more information about the predefined SAFs that are included with Web Server, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference.*

Access Control API

The Access Control API is a C API that allows you to programmatically control the access privileges on Web Server.

Access control lists (ACLs) determines the access privileges to the resources on the server. Each ACL contains a list of access control entries. The following access control entry, for example, specifies that all access is denied to everyone for any resource having a URI that starts with `/private`.

```
acl "uri=/private/*";
deny (all)
(user = "anyone");
```

To create ACL:

1. From the Common Task screen select the configuration from the drop-down list and click Edit Configurations tab.
2. Click Access Control tab in the configuration screen.
3. Click Access Control Lists (ACL) tab in the Authentication Databases screen.
4. Click New to create an ACL.

The default access control list resides in the directory *install_dir*`/admin-server/config.` The default ACL file is `default.acl`.

With Web Server you can configure and reference multiple ACL files. For more information about configuring ACL files for virtual servers, see the *Sun Java System Web Server 7.0 Administrator's Guide*.

With the Access Control API you can manipulate ACLs, read and write ACL files, and evaluate and test access to resources on the server.

You can also define your own attributes for authentication. For example, you might want to authenticate users based on e-mail address or on the URL that referred them to the resource:

```
allow (read) referer="*www.acme.com*"
```

You can also authenticate the client based on your own authentication methods and databases.

Registering New Authentication Services

You must Define your own Loadable Authentication Service (LAS), which is an NSAPI plug-in. For the server to use your attributes for authentication. Load it into the server in the usual manner by adding the following directives to `magnus.conf`:

- An `Init` directive that invokes the `load-modules` function to load the shared library.
- An `Init` directive that calls the initialization function.

  For information about changes to the Access Control API in Web Server, see the comments in the *install_dir*`/include/nsacl/aclapi.h` file.

Certificate-Mapping API

The Certificate-Mapping API consists of data structures and functions used to manage certificate mapping.

When a user authenticates to the Web Server by sending a client certificate, the server uses information in the certificate to search the user directory for the user's entry.

You can configure some parts of this process by editing the file `certdb.conf`. This file specifies the following:

- Server searches the directory for the user's entry.
- Whether the server goes through an additional step of verifying that the user's certificate matches the certificate presented to the server.

  For more information about `certmap.conf`, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

  You can also modify this process programmatically. Web Server includes a set of API functions (referred to here as the Certificate-Mapping API functions) that allow you to control this process. You can write your own functions to customize how certificate subject entries are found in the directory.

  To use this API, you must have a copy of the Directory SDK. You can download a copy of this SDK from the following location: `http://developers.sun.com/index.html`.

API Summary

The following table lists the APIs available in Web Server.

TABLE 1–1 APIs Available in Web Server

API/Interface/Protocol	Language	Documentation
Interfaces for Generating Dynamic Content		
Custom Server-parsed HTML Tags	C	Chapter 2
Java Servlets	Java	"About Servlets" in *Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications*
JavaServer Pages	HTML with additional JSP tags	"Introducing JSPs" in *Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications*
CGI (one process per request)	C, C++, Perl, shell, and other languages	*The Common Gateway Interface:* `http://hoohoo.ncsa.uiuc.edu/cgi/overview.html`
APIs for Writing ServerPlug-ins		
NSAPI (in-process shared object/DLL)	C, C++	*Sun Java System Web Server 7.0 NSAPI Developer's Guide*

Changes from Previous Versions

Changes from previous versions of Web Server are summarized in the following sections:

- "API Changes Since iPlanet Web Server 3.0" on page 27
- "API Changes Since iPlanet Web Server 4.0" on page 27
- "API Changes Since iPlanet Web Server 4.1" on page 27
- "API Changes Since Sun ONE Web Server 6.0" on page 28

API Changes Since iPlanet Web Server 3.0

- A new API for defining customized server-parsed tags as NSAPI plug-ins has been added. For more information, see Chapter 2.
- Server-side Java applets (`HttpApplets`) are not supported; use Java servlets instead.
- The Agents API is not supported.
- NSAPI has new features.

API Changes Since iPlanet Web Server 4.0

- Java Servlets 2.2.1 and JavaServer Pages 1.1 are supported.
- HTTP/1.1 cookies are supported.
- Descriptions of CGI variables have been added to "CGI Variables" on page 56.
- You can invoke servlets as SSI in HTML pages by using the `<SERVLET\>` tag, as discussed in Chapter 2.
- NSAPI has new features.

API Changes Since iPlanet Web Server 4.1

- Programs such as servlets modify a virtual server instead of the server as a whole. (To add programs as in iPlanet Web Server 4.1, you can configure only one virtual server.)
- Web applications are now supported as described in the Java Servlet 2.2 API specification.
- NSAPI has new features. For details, see the *NSAPI Programmer's Guide for Sun ONE Web Server*.
- Some configuration files have changed. For details, see the *iPlanet Web Server 6.0 Programmer's Guide* (http://docs.sun.com/source/816-5687-10/index.html).
- The Access Control API has changed. For details, see the comments in the *server_root*`/plugins/include/nsacl/aclapi.h` file.

API Changes Since Sun ONE Web Server 6.0

- Java Servlets 2.3 and JavaServer Pages 1.2are supported.
- HTTP extensions for the WebDAV protocol in compliance with RFC 2518 are supported.
- NSAPI filters that enable the custom processing of HTTP request and response streams are supported.
- HTTP compression through the use of native HTTP request and response stream filters is supported.
- Legacy servlets (servlets configured through the `servlets.properties`, `contexts.properties,` and `rules.properties` files) are not supported.

  For information about migrating legacy servlets to web applications, see the *Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications*.

API Changes Since Sun Java System Web Server 6.1

- Java Servlets 2.4 and JavaServer Pages 2.0 are supported.
-

CHAPTER 2

Server-parsed HTML Tags

HTML files can contain tags that are executed on the server. In addition to supporting the standard server-side tags, Web Server allows you to embed servlets and define your own server-side tags.

This chapter has the following sections:

Enabling Server-side HTML

The server parses server-side tags only if server-side parsing has been enabled.

To enable server-side parsing using the Administration interface, perform the following steps:

▼ To Enable Server-side HTML

1 **Access the Edit Virtual Server , and click the Content Handling tab.**

2 **Click the General tab in the Content Handling screen.**

3 **Select the check boxes to enable the server-parsed HTML and with exec tag.**

4 **Specify a resource for which the server will parse HTML, use the drop-down list.**

Choose the virtual server or a specific directory within the virtual server. If you choose a directory, the server will parse HTML only when the server receives a URL for that directory or any file in that directory.

5 **Choose the files to parse. The choices are:**

- **Files with the extension .shtml.** The server parses only files with the extension `.shtml`. In this case, all files you want to parse must have the `.shtml` extension. This is the most common (and default) choice.
- **Files with the execute bit and the extension .html.** (Unix and Linux only) The server parses files whose UNIX and Linux permissions specify that the execute bit is on. Using the execute permissions can be unreliable because in some cases the bit is set on files that are not executable.
- **All HTML files.** The server parses all HTML files. Choosing this option can slow server performance.

6 **Click Save.**

When you activate parsing, make sure that the following directives are added to the `magnus.conf` file:

Note – You must set `NativeThread="no"` for Web Server. In addition, these functions now originate from `Shtml.dll` (or `libShtml.so` on UNIX), which is located in *install_dir*`C:\Program Files\Sun\WebServer7` for Windows, and *install_dir*`/sun/webserver7` for UNIX.

To enable parsing of server-side tags for files with extensions other than `.shtml`, add the extension to the appropriate line in the `mime.types` file. For example, the following line in `mime.types` indicates that files with either a `.shtml` or `.jbhtml` extension are parsed for server-side tags:

```
type=magnus-internal/parsed-html exts=shtml,jbhtml
```

Using Server-side HTML Commands

This section describes the HTML commands for including server-parsed tags in HTML files. These commands are embedded into HTML files, which are processed by the built-in SAF `parse-html`.

The server replaces each command with data determined by the command and its attributes.

The format for a command is:

```
<!--#command attribute1 attribute2 [Please define the ellipsis text entity] -->
```

The format for each `attribute` is a name-value pair such as:

```
name="value"
```

Commands and attribute names should be in lower case.

The commands are hidden within HTML comments so they are ignored if not parsed by the server. The standard server-side commands are listed below, and described in this section:

- “config” on page 31
- “include” on page 32
- “echo” on page 32
- “fsize” on page 32
- “flastmod” on page 32
- “exec” on page 33

config

The `config` command initializes the format for other commands.

- The `errmsg` attribute defines a message sent to the client when an error occurs while parsing the file. This error is also logged in the error log file.
- The `timefmt` attribute determines the format of the date for the `flastmod` command. It uses the same format characters as the `util_strftime` function. The default time format is: `"%A, %d-%b-%y %T"`. For more information about time formats, see the “Time Formats” on page 41.
- The `sizefmt` attribute determines the format of the file size for the `fsize` command. It can have one of these values:
  - `bytes` to report file size as a whole number in the format 12,345,678.
  - `abbrev` (the default) to report file size as a number of KB or MB.

Example

```
<!--#config timefmt="%r %a %b %e, %Y" sizefmt="abbrev"-->
```

This sets the date format to a value such as 10:45:35 AM Wed Apr 21, 2006, and the file size format to the number of KB or MB of characters used by the file.

include

The `include` command inserts a file into the parsed file. You can nest files by including another parsed file, which then includes another file, and so on. The client requesting the parsed document must also have access to the included file if your server uses access control for the directories in which they reside.

In Web Server, you can use the `include` command with the `virtual` attribute to include a CGI program file. You must also use an `exec` command to execute the CGI program.

- The `virtual` attribute is the URI of a file on the server.
- The `file` attribute is a relative path name from the current directory. It cannot contain elements such as `../` and it cannot be an absolute path.

Example

```
<!--#include file="bottle.gif"-->
```

echo

The `echo` command inserts the value of an environment variable. The `var` attribute specifies the environment variable to insert. If the variable is not found, "(none)" is inserted. For a list of environment variables, see "Environment Variables in Server-side HTML Commands" on page 33.

Example

```
<!--#echo var="DATE_GMT"-->
```

fsize

The `fsize` command sends the size of a file. The attributes are the same as those for the `include` command (`virtual` and `file`). The file size format is determined by the `sizefmt` attribute in the `config` command.

Example

```
<!--#fsize file="bottle.gif"-->
```

flastmod

The `flastmod` command prints the date a file was last modified. The attributes are the same as those for the `include` command (`virtual` and `file`). The date format is determined by the `timefmt` attribute in the `config` command.

Example

```
<!--#flastmod file="bottle.gif"-->
```

exec

The `exec` command runs a shell command or CGI program.

- The `cmd` attribute (UNIX only) runs a command using `/bin/sh`. You may include any special environment variables in the command.
- The `cgi` attribute runs a CGI program and includes its output in the parsed file.

Example

```
<!--#exec cgi="workit.pl"-->
```

Environment Variables in Server-side HTML Commands

In addition to the standard set of environment variables used in CGI, you can include the following variables in your parsed commands:

- `DOCUMENT_NAME`

  File name of the parsed file.
- `DOCUMENT_URI`

  Virtual path to the parsed file (for example, `/shtml/test.shtml`).
- `QUERY_STRING_UNESCAPED`

  Unescaped version of any search query the client sent with all shell-special characters escaped with the slash character.
- `DATE_LOCAL`

  Current date and local time.
- `DATE_GMT`

  Current date and time expressed in GMT (Greenwich Mean Time).
- `LAST_MODIFIED`

  Date the file was last modified.

Embedding Servlets

Web Server supports the `<servlet>` tag. This tag allows you to embed servlet output in an SHTML file. No configuration changes are necessary to enable this behavior. If SSI and servlets are both enabled, the `<servlet>` tag is enabled.

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

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

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

- The value of the `url-pattern` element defined in the `web.xml` file for the web application. For more information about `web.xml`, see the Java Servlet 2.4 specification (chapter SRV .13, "Deployment Descriptor"). You can find the specification at: `http://java.sun.com/products/servlet/download.html`
- The value of the `uri` attribute defined in the `web-apps.xml` file for the web application.

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

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

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

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

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

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

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

For legacy servlets, the `code` parameter specifies the `.class` file for the servlet and is required. The `codebase` parameter is required if the servlet is *not* defined in the `servlets.properties` file and the `.class` file is *not* in the same directory as the HTML file containing the `<servlet>` tag. Legacy servlets must be configured in the default virtual server and do not require a `web.xml` file.

For more information about creating servlets, see the "Creating Servlets" in *Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications*.

Defining Customized Server-parsed HTML Tags

In Web Server, users can define their own server-side tags. For example, you could define the tag `HELLO` to invoke a function that prints "Hello World!" You could have the following code in your `hello.shtml` file:

```
<html>
  <head>
    <title>shtml custom tag example</title>
  </head>
  <body>
    <!--#HELLO-->
  </body>
</html>
```

When the browser displays this code, each occurrence of the `HELLO` tag calls the function.

▼ To Define Customized Server-parsed Tag

1. **"Define the Functions that Implement the Tag" on page 36.**

   You must define the tag execution function. You must also define other functions that are called on tag loading and unloading and on page loading and unloading.

2 "Write an Initialization Function to Register the New Tag" on page 40.

Write an initialization function that registers the tag using the `shtml_add_tag` function.

3 "Load the New Tag into the Server" on page 40.

Define the Functions that Implement the Tag

Define the functions that implement the tags in C, using NSAPI.

- Include the header `shtml_public.h`, which is in the directory *install_dir*`/libinclude/shtml`.
- Link against the `shtml` shared library. On Windows, `shtml.dll` is in *install_dir*`/lib` On UNIX platforms, `libShtml.so` or `.sl` is in *install_dir*`/lib`

  `ShtmlTagExecuteFunc` is the actual tag handler. It gets called with the usual NSAPI `pblock`, `Session`, and `Request` variables. In addition, it also gets passed the `TagUserData` created from the result of executing the tag loading and page loading functions (if defined) for that tag.

  The signature for the tag execution function is:

```
typedef int (*ShtmlTagExecuteFunc)(pblock*, Session*, Request*, TagUserData, TagUserData);
```

Write the body of the tag execution function to generate the output to replace the tag in the `.shtml` page. Do this by using the `net_write` NSAPI function. This function writes a specified number of bytes to a specified socket from a specified buffer.

For more information about NSAPI plug-ins and functions, see the *Sun Java System Web Server 7.0 NSAPI Developer's Guide*.

The tag execution function must return an `int` that indicates whether the server should proceed to the next instruction in `obj.conf`:

- `REQ_PROCEED`: Execution was successful
- `REQ_NOACTION`: Nothing happened
- `REQ_ABORTED`: An error occurred
- `REQ_EXIT`: The connection was lost

  The other functions you must define for your tag are:

- `ShtmlTagInstanceLoad`

  This is called when a page containing the tag is parsed. It is not called if the page is retrieved from the browser's cache. It basically serves as a constructor, the result of which is cached and is passed into `ShtmlTagExecuteFunc` whenever the execution function is called.
- `ShtmlTagInstanceUnload`

This is basically a destructor for cleaning up whatever was created in the `ShtmlTagInstanceLoad` function. It gets passed the result that was originally returned from the `ShtmlTagInstanceLoad` function.

- `ShtmlTagPageLoadFunc`

  This is called when a page containing the tag is executed, regardless of whether the page is still in the browser's cache. This provides a way to make information persistent between occurrences of the same tag on the same page.
- `ShtmlTagPageUnLoadFn`

  This is called after a page containing the tag has executed. It provides a way to clean up any allocations done in a `ShtmlTagPageLoadFunc` and thus gets passed the result returned from the `ShtmlTagPageLoadFunc`.

  The signatures for these functions are:

```
#define TagUserData void*
typedef TagUserData (*ShtmlTagInstanceLoad)(
    const char* tag, pblock*, const char*, size_t);
typedef void (*ShtmlTagInstanceUnload)(TagUserData);
typedef int (*ShtmlTagExecuteFunc)(
    pblock*, Session*, Request*, TagUserData, TagUserData);
typedef TagUserData (*ShtmlTagPageLoadFunc)(
    pblock* pb, Session*, Request*);
typedef void (*ShtmlTagPageUnLoadFunc)(TagUserData);
```

Here is the code that implements the `HELLO` tag:

```
/*
 * mytag.c: NSAPI functions to implement #HELLO SSI calls
 *
 *
 */

#include "nsapi.h"
#include "shtml/shtml_public.h"

/* FUNCTION : mytag_con
 *
 * DESCRIPTION: ShtmlTagInstanceLoad function
 */
#ifdef __cplusplus
extern "C"
#endif
TagUserData
mytag_con(const char* tag, pblock* pb, const char* c1, size_t t1)
```

```
{
    return NULL;
}

/* FUNCTION : mytag_des
 *
 * DESCRIPTION: ShtmlTagInstanceUnload
 */
#ifdef __cplusplus
extern "C"
#endif
void
mytag_des(TagUserData v1)
{

}

/* FUNCTION : mytag_load
 *
 * DESCRIPTION: ShtmlTagPageLoadFunc
 */
#ifdef __cplusplus
extern "C"
#endif
TagUserData
mytag_load(pblock *pb, Session *sn, Request *rq)
{
    return NULL;
}

/* FUNCTION : mytag_unload
 *
 * DESCRIPTION: ShtmlTagPageUnloadFunc
 */
#
#ifdef __cplusplus
extern "C"
#endif
void
mytag_unload(TagUserData v2)
{

}

/* FUNCTION : mytag
 *
 * DESCRIPTION: ShtmlTagExecuteFunc
 */
```

```
#ifdef __cplusplus
extern "C"
#endif
int
mytag(pblock* pb, Session* sn, Request* rq, TagUserData t1, TagUserData t2)
{
    char* buf;
    int length;
    char* client;
    buf = (char *) MALLOC(100*sizeof(char));
    length = util_sprintf(buf, "<h1>Hello World! </h1>", client);
    if (net_write(sn->csd, buf, length) == IO_ERROR)
    {
        FREE(buf);
        return REQ_ABORTED;
    }
    FREE(buf);
    return REQ_PROCEED;
}

/* FUNCTION : mytag_init
*
* DESCRIPTION: initialization function, calls shtml_add_tag() to
 * load new tag
*/
#
#ifdef __cplusplus
extern "C"
#endif
int
mytag_init(pblock* pb, Session* sn, Request* rq)
{
    int retVal = 0;
// NOTE: ALL arguments are required in the shtml_add_tag() function
    retVal = shtml_add_tag("HELLO", mytag_con, mytag_des, mytag, mytag_load, mytag_unload);

    return retVal;
}
/* end mytag.c */
```

Write an Initialization Function to Register the New Tag

In the initialization function for the shared library that defines the new tag, register the tag using the function `shtml_add_tag`. The signature is:

```
NSAPI_PUBLIC int shtml_add_tag (
    const char* tag,
    ShtmlTagInstanceLoad ctor,
    ShtmlTagInstanceUnload dtor,
    ShtmlTagExecuteFunc execFn,
    ShtmlTagPageLoadFunc pageLoadFn,
    ShtmlTagPageUnLoadFunc pageUnLoadFn);
```

Any of these arguments can return `NULL` except for `tag` and `execFn`.

Load the New Tag into the Server

After creating the shared library that defines the new tag, you load the library into Web Server in the usual way for NSAPI plug-ins. That is, add the following directives to the configuration file `magnus.conf`:

▼ To Add a New Tag into the Server

1. **Add an `Init` directive whose `fn` parameter is `load-modules` and whose `shlib` parameter is the shared library to load. For example, if you compiled your tag into the shared object *install_dir* `/hello.so`, it would be:**

   ```
   Init funcs="mytag,mytag_init" shlib="install_dir/hello.so" fn="load-modules"
   ```

2. **Add another `Init` directive whose `fn` parameter is the initialization function in the shared library that uses `shtml_add_tag` to register the tag. For example:**

   ```
   Init fn="mytag_init"
   ```

Time Formats

The following table describes the format strings for dates and times used by server-parsed HTML.

TABLE 2–1 Time Formats

Symbol	Meaning
%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"
%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"

TABLE 2–1 Time Formats *(Continued)*

Symbol	Meaning
%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

CHAPTER 3

Using Common Gateway Interface

Common Gateway Interface (CGI) programs run on the server and generate a response to return to the requesting client. CGI programs can be written in various languages, including C, C++, Java, Perl, and as shell scripts. CGI programs are executed through URL invocation.

For information about writing CGI programs, see:

`http://hoohoo.ncsa.uiuc.edu/cgi/overview.html`

Sun Java System Web Server 7.0 complies with the CGI specification version 1.1.

This chapter has the following sections:

Configuring CGI Settings

You can configure the Web Server to recognize and execute CGI programs.

This section discusses the following topics:

▼ Enabling CGI

◗ **Using the Admin Console:**

a. **Select Common Tasks tab > CGI Directories from the Virtual Server Tasks list.**

   The CGI Settings page is displayed.

b. **Select the CGI as file type Enabled box.**

▼ Specifying CGI Directories

You can specify a directory that contains only CGI programs. All files are run as programs regardless of the file extensions.

1 **Using the Admin Console:**

a. **Select Common Tasks tab > CGI Directories from the Virtual Server Tasks list.**

   The CGI Directory window is displayed.

b. **Click New.**

   The Add CGI Directory window is displayed.

c. **Specify the URI and path where the CGI programs are stored.**

   For example, if you enter `cgi-bin` as the URL prefix, then all URLs to these CGI programs have the following structure:

   `http://`*yourserver.domain.com*`/cgi-bin/`

2 **Using the Command-Line Interface (CLI):**

a. **You can also specify the CGI directory using the `create-cgi-dir` command.**

   The syntax of the command is as follows:

```
create-cgi-dir [--user|-u admin-user] [--password-file|-w admin-pswd-file]
[--host|-h admin-host] [--port|-p admin-port] [--no-ssl|-N] [--echo|-e]
[--rcfile|-R rcfile] [--no-prompt|-Q] [--verbose|-v] --config|-c name --vs|-s
name [--shell-cgi=boolean] --uri-prefix|-r prefix --directory|-d dir
```

   Example:

```
./wadm create-cgi-dir --user=admin --password-file=admin.pwd
--host=serverhost --port=8989 --config=config1 --vs=config1_vs_1
--uri-prefix=/config1_urlprefix --directory=/cgi-dir
```

More Information

Changes to the `obj.conf` File

For each CGI directory, the file `obj.conf` contains a `NameTrans` directive that associates the name `cgi` with each request for a resource in that directory. These directives are automatically added to `obj.conf` when you specify CGI directories.

For example, the following instruction interprets all requests for resources in `http://`*server-name*`/cgi-local` as requests to invoke CGI programs in the directory `C:/Sun/Servers/docs/mycgi`:

```
NameTrans fn="pfx2dir" from="/cgi-local" dir="C:/Sun/Servers/docs/mycgi"
name="cgi"
```

The `obj.conf` file must contain the following named object:

```
<Object name="cgi">

ObjectType fn="force-type" type="magnus-internal/cgi"

Service fn="send-cgi"

</Object>
```

Note – Do not remove this object from `obj.conf`. If you do, the server will not recognize CGI directories, regardless of whether you specify them in the Administration Console or manually add more `NameTrans` directives to `obj.conf`.

▼ Specifying CGI File Extensions

You can associate the file types for CGI programs. That is, you can configure the CGI programs to have the file extensions `.cgi`, `.exe`, or `.bat`. The programs can be located in any directory in or under the document root directory of the virtual server.

By default, the file extensions for CGI programs are `.cgi`, `.exe` and `.bat`. However, you can change which extensions indicate CGI programs by modifying the MIME types file.

1 **Modify the following line in the `mime.types` file to specify the desired extensions.**

```
type=magnus-internal/cgi exts=cgi,exe,bat
```

2 **Restart the server after editing `mime.types`.**

When the server is enabled to treat all files with an appropriate extensions as CGI programs, the `obj.conf` file contains the following `Service` directive:

```
Service fn="send-cgi" type="magnus-internal/cgi"
```

See Also For more information on configuring CGI for your server, see the “Configuring CGI Subsystem for Your Server” in *Sun Java System Web Server 7.0 Administrator’s Guide.*

Creating Custom Execution Environments for CGI Programs

Creating custom execution environment includes the following steps:

1. Installing `suid` and `Cgistub`
2. Creating a `cgi-bin` directory
3. Specifying a `chroot` directory for virtual server

▼ To Install `suid` and `Cgistub` Directories

1 **Login as a superuser.**

2 **Create the `private` directory for `Cgistub`:**

`cd` *install_dir*`/https-`*instance*

```
mkdir private
```

Note – Installing `Cgistub` in the *install_dir*`/https-`*instance*`/private` directory is recommended. If you install it anywhere else, you must specify the path to `Cgistub` in the `init-cgi` function in `magnus.conf`. For details, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

Note – It is not possible to install the `suid` `Cgistub` program on an NFS mount. If you wish to use an `suid` `Cgistub`, you must install your server instance to a local file system.

3 **Copy `Cgistub` to the `private` directory:**

```
cd private
```

```
cp ../../lib/Cgistub.
```

4 **Set the owner of `private` to the server user:**

`chown` *user* `.`

5 **Set the permissions on `private`:**

```
chmod 500.
```

6 **Set the owner of `Cgistub` to `root`:**

```
chown root Cgistub
```

7 **Set the permissions on `Cgistub`:**

```
chmod 4711 Cgistub
```

8 **You can give each reference to the `send-cgi` SAF in `obj.conf` a `user` parameter.**

For example: `Service fn="send-cgi" user="`*user*`"`

You can use variable substitution. For example, in `server.xml`, give a `virtual-server` element the following `variable` subelement:

`<variable user="`*user*`"\>`

This modification to the `server.xml` allows you to write the `send-cgi` SAF line in the `obj.conf` as follows:

```
Service fn="send-cgi" user="$user"
```

For more information about `server.xml` and `send-cgi` in the `obj.conf` file and , see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

9 **Restart the server to apply the changes.**

`Cgistub` enforces the following security restrictions:

- The user the CGI program executes as must have a `uid` of 100 or greater. This prevents anyone from using `Cgistub` to obtain root access.
- The CGI program must be owned by the user it is executed as and must not be writable by anyone other than its owner. This makes it difficult for anyone to covertly inject and then remotely execute programs.
- `Cgistub` creates its UNIX listen socket with 0700 permissions.

Note – Socket permissions are not respected on a number of UNIX variants, including current versions of Sun operating systems/Solaris. To prevent a malicious user from exploiting `Cgistub`, ensure that the server's temporary directory is set (using the `server.xml` `temp-path` element) to a directory accessible only to the server user.

▼ To Create a `cgi-bin` Directory and Define User and Group

To prevent a virtual server's CGI programs from interfering with other users, the CGI programs must be stored in a unique directory and executed with the permissions of a unique UNIX user and group.

Before You Begin Create the UNIX user and group. The exact steps required to create a user and group vary by operating system. For instructions, refer your operating systems' documentation.

1. **Login as a superuser.**

2. **Change to the document root directory for the virtual server.**

   `cd` *document-root*

3. **Create the `cgi-bin` directory and set appropriate permissions.**

   a. `mkdir cgi-bin`

   b. `chown` *user*`:`*group* `cgi-bin`

   c. `chmod 755 cgi-bin`

4. **Set the virtual server's CGI directory, user, and group in one of the following ways:**

   - **Modify the `obj.conf` file.**

     Use the `user` and `group` parameters of the `send-cgi Service` SAF in the `obj.conf` file. For detailed instructions, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

   - **Use the Admin Console.**

     a. **Select Common Tasks tab > CGI Directories from the Virtual Server Tasks list.**

        The CGI Settings window is displayed.

     b. **Enter the user and the group who can execute CGI programs.**

        For more information on CGI directory, see the *Sun Java System Web Server 7.0 Administrator's Guide*.

   - **Use the `set-cgi-prop` command to define a unique CGI directory, UNIX user and group for a virtual server.**

     The syntax of the command is:

     ```
     ./wadm set-cgi-prop [--user|-u admin-user] [--password-file|-w
     admin-pswd-file] [--host|-h admin-host] [--port|-p admin-port] [--no-ssl|-N]
     [--echo|-e] [--rcfile|-R rcfile] [--no-prompt|-Q] [--verbose|-v] --config|-c
     name [--vs|-s name] (propertyname=value)
     ```

     For example:

     ```
     /wadm set-cgi-prop --user=admin --password-file=admin.pwd --host=serverhost
     --port=8989 --config=config1 user=admin group=group
     ```

▼ Specifying a `Chroot` Directory for a Virtual Server

To further improve security, the CGI scripts must be prevented from accessing data above and outside of the *document-root* directory.

Before You Begin Set up the `chroot` environment. The exact steps required to set up the `chroot` environment vary by operating system. For instructions, refer your operating system's documentation. The `man` pages for `ftpd` and `chroot` are often a good place to start.

Steps required for Solaris versions 2.8 through 10 are described in the following procedure:

1 **Login as a superuser.**

2 **Change to the `chroot` directory.**

`chroot` is typically the *document-root* directory of the virtual server.

`cd` *chroot*

3 **Create `tmp` in the `chroot` directory and set appropriate permissions.**

```
mkdir tmp
```

```
chmod 1777 tmp
```

4 **Create `dev` in the `chroot` directory and set appropriate permissions.**

```
mkdir dev
```

```
chmod 755 dev
```

5 **List `/dev/tcp`, and note the major and minor numbers of the resulting output.**

```
ls -lL /dev/tcp
```

```
crw-rw-rw- 1 root sys 11, 42 Apr 9 1998 /dev/tcp
```

In this example, the major number is 11 and the minor number is 42.

6 **Create the `tcp` device using the major and minor numbers.**

```
mknod dev/tcp c 11 42
```

```
chmod 666 dev/tcp
```

7 **Repeat steps 4, 5 and 6 for each of the following devices.**

Note – Each device will have a different major and minor number combination.

```
/dev/udp /dev/ip /dev/kmem /dev/kstat /dev/ksyms /dev/mem /dev/null /dev/stderr
/dev/stdin /dev/stdout /dev/ticotsord /dev/zero
```

8 **Set permissions on the devices in `dev` in the `chroot` directory:**

```
chmod 666 dev/*
```

9 **Create and populate `lib` and `usr/lib` in the `chroot` directory:**

a. `mkdir usr`

b. `mkdir usr/lib`

c. `ln -s /usr/lib`

d. `ln /usr/lib/* usr/lib`

   You can ignore the messages this command generates.

   - **If the `/usr/lib` directory is on a different file system, use the following command:**

     ```
     cp -rf /usr/lib/* usr/lib
     ```

10 **Create and populate `bin` and `usr/bin` in the `chroot` directory:**

a. `mkdir usr/bin`

b. `ln -s /usr/bin`

c. `ln /usr/bin/* usr/bin`

   You can ignore the messages this command generates.

   - **If the `/usr/bin` directory is on a different file system, use the following command:**

     ```
     cp -rf /usr/bin/* usr/bin
     ```

11 **Create and populate `etc` in the `chroot` directory.**

```
mkdir etc
```

```
ln /etc/passwd /etc/group /etc/netconfig etc
```

12 **Test the `chroot` environment.**

`chroot` *chroot* `bin/ls -l`

The output should look something like this:

```
total 14
 lrwxrwxrwx                1 root          other          8 Jan 13 03:32 bin -\> /usr/bin
 drwxr-xr-x                2 user         group         512 Jan 13 03:42 cgi-bin
 drwxr-xr-x                2 root          other        512 Jan 13 03:28 dev
 drwxr-xr-x                2 user         group         512 Jan 13 03:26 docs
 drwxr-xr-x                2 root          other        512 Jan 13 03:33 etc
 lrwxrwxrwx                1 root          other          8 Jan 13 03:30 lib -\> /usr/lib
 drwxr-xr-x                4 root          other        512 Jan 13 03:32 usr
```

13 Set the virtual server's `chroot` directory in one of the following ways:

- **Modify the `obj.conf` file.**

  Use the `chroot` parameter of the `send-cgi` `Service` SAF in the `obj.conf` file. For detailed instructions, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

- **Use the Admin Console.**

  a. **Select Common Tasks tab > CGI Directories from the Virtual Server Tasks list.**

     The CGI Settings window is displayed.

  b. **Enter the Chroot directory.**

     For detailed instructions, see the *Sun Java System Web Server 7.0 Administrator's Guide*.

- **Use the `set-cgi-prop` command.**

  Example: `./wadm set-cgi-prop --user=admin --password-file=admin.pwd --host=serverhost --port=8989 --config=config1 chroot=`*vs_dir*

  For information on the `set-cgi-prop` command, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

Adding CGI Programs to the Server

To add CGI programs to the server, do one of the following:

- Store the program file in a CGI directory (if there are any).
- Give it a file name that the server recognizes as a CGI program and put it in any directory at or below the document root (if CGI file type recognition has been activated).

Note – On UNIX platforms, make sure the program file has `execute` permissions set.

Setting the Priority of a CGI Program

The priority of a CGI program can be set using the `nice` parameter of the `send-cgi` function. This is a UNIX-only parameter and accepts an increment that determines the CGI program's priority relative to the priority of the server process. Typically, the server is run with a `nice` value of 0 and the `nice` increment can be between 0 and 19.

0	The CGI program runs at the same priority as server
19	The CGI program runs at a much lower priority than the server

As with all other CGI variables, the `nice` value can be configured per virtual server.

Note – While it is possible to increase the priority of the CGI program above that of the server by specifying a nice increment of -1, Sun does not recommended you to do so.

For more information about `send-cgi`, see the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

▼ To Configure CGI Properties for a Virtual Server

You can configure CGI variables for a virtual server either using the Admin Console or using the CLI. The following procedure describes the steps:

◗ **Configure CGI variables.**

- **Using the Admin Console:**

  a. **Select Common Tasks tab > CGI Directories from the Virtual Server Tasks list.**

     The CGI Settings window is displayed.

  b. **Specify the `nice` value as desired.**

  c. **Click Save to apply your changes.**

- **Using the `set-cgi-prop` command.**

  Example:

```
./wadm set-cgi-prop --user=admin --password-file=admin.pwd --host=serverhost
--port=8989 --config=config1 nice=5
```

Shell CGI Programs

Shell CGI is a text file that contains commands for the Bourne Shell command interpreter. You can use any text editor to create a shell CGI program. Shell CGI programs are typically given the suffix `.sh`.

For information about installing CGI and shell CGI programs on Windows using the Admin Console, see the *Sun Java System Web Server 7.0 Administrator's Guide*.

Perl CGI Programs

Web Server 7.0 bundles a Perl interpreter to run CGI programs that are written in the Perl programming language. The Perl interpreter is located at *install_dir*`/lib/perl`.

On Windows platform, file associations need to be in place for CGI programs to run. Files with a `.pl` filename extension need to be associated with the Perl interpreter executable before they can be executed as CGI programs. To create an association for Perl programs to run as CGI programs, perform the following steps:

- Open a command-line window from start-> run -> cmd.
- Create an association for Perl programs.
  - `C:\>assoc .pl=perl`
  - `C:\>ftype perl="c:\WS7\lib\perl\perl.exe" "%1"`

Note – You cannot run CGIs using Perl 5.6.*x* with the `-w` flag. Instead, include the following code in the Perl script:

```
use warnings;
```

This section describes the steps to configure the server to execute Perl programs as CGI programs.

▼ To Configure Perl Programs to Execute as CGI Programs

1. **Create a simple Perl program called `hello.pl`.**

   a. **Use a text editor to create a file called `hello.pl` and copy the following lines into it:**

   ```
   #!install_dir/lib/perl/perl
   print "Content-type:text/html\n\n";
   print "Hello World!";
   ```

2. **Create a directory for this program. By convention, this directory is named as the `cgi-bin` directory though you can name it whatever you like.**

   It is recommended that you create the `cgi-bin` directory under the instance configuration directory. For example:

   ```
   mkdir /var/opt/SUNWwbsvr7/cgi-bin
   ```

3. **Copy the `hello.pl` file into the `/var/opt/SUNWwbsvr7/cgi-bin` directory.**

4. **Select Common Tasks tab > CGI Directories from the Virtual Servers Tasks list.**

   The CGI Settings window is displayed.

5. **Click the New button in the CGI Directories section.**

   The Add CGI Directories window is displayed.

6. **Enter the Prefix and the CGI Directory path. Choose whether the program is a CGI or a ShellCGI program.**

   For example (On UNIX platforms):

   Prefix: `/cgi-bin`

   CGI Directory: `/var/opt/SUNWwbsvr7/cgi-bin`

   On Windows platforms:

   Prefix: `\cgi-bin`

   CGI Directory: `c:\sun\webserver7\cgi-bin`

   Note – The CGI directory you specified must have execute permissions.

   ---

7 **Click Save to save the changes. Access the program from a browser window.**

Open a browser window. If your server is called `acme`, running on port `2222`), type `http://acme:2222/cgi-bin/hello.pl`.

Displays the message “Hello World” on the browser window.

Global CGI Settings

You can change the CGI settings globally for all available virtual servers. You can either use the Administration Console or the CLI.

▼ To Change Global CGI Settings

◗ **You can use either the Admin Console or the CLI to change the global CGI settings:**

- **Using the Admin Console:**

  a. **Click the Configuration tab.**

     List of available configurations is displayed.

  b. **Select a configuration from the list.**

  c. **Select Performance tab > CGI subtab.**

     The CGI Settings window is displayed.

  d. **Specify the values for the following settings:**

     CGI Timeout Settings:

     - **Timeout** (*CGIExpirationTimeout*) Specifies the time in seconds after which the server terminates a long-running CGI program.
     - **Idle Timeout**(*CGIStubIdleTimeout*) 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.

CGI Stub Process Settings:

- **Timeout** Time in seconds after which an unused CGI stub process is terminated.
- **CGI Stub Excecutable** Controls the number of processes that are started by default. The first `CGIStub` process is not started until a CGI program has been accessed. The default value is 2. If you have a `init-cgi` directive in the `magnus.conf` file, the minimum number of `CGIStub` processes are spawned at startup. Controls the number of processes that are started by default.
- **Minimum Stub Size** Controls the maximum number of `CGIStub` processes the server can spawn. This is the maximum concurrent `CGIStub` processes in execution, not the maximum number of pending requests. The value must be less than the Maximum Stub Size.
- **Maximum Stub Size** 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.

For more information about these global CGI settings, see the description of the `magnus.conf` file in the *Sun Java System Web Server 7.0 Administrator's Configuration File Reference*.

e. **Click Save to apply your changes.**

- **Using the `set-cgi-prop` command.**

  Example:

  ```
  ./wadm set-cgi-prop --user=admin --password-file=admin.pwd --host=serverhost
  --port=8989 --config=config1 timeout=10 max-cgistubs=20 idle-timeout=600
  min-cgistubs=10 cgistub-idle-timeout=60 cgistub-path=../../lib/Cgistub
  ```

For more information on the `set-cgi-prop` command, see the *Sun Java System Web Server 7.0 CLI Reference Manual*.

CGI Variables

In addition to the standard CGI variables, you can use the Sun Java System Web Server CGI variables in CGI programs to access information about the client certificate if the server is running in secure mode. The `CLIENT_CERT` and `REVOCATION` variables are available only when client certificate-based authentication is enabled.

The following table lists the Sun Java System Web Server CGI variables.

TABLE 3–1 CGI Variables

Variable	Description
`SERVER_URL`	URL of the server that the client requested
`HTTP_xxx`	An incoming HTTP request header, where *xxx* is the name of the header
`HTTPS`	`ON` if the server is in secure mode, otherwise `OFF`
`HTTPS_KEYSIZE`	Keysize of the SSL handshake (available if the server is in secure mode)
`HTTPS_SECRETKEYSIZE`	Keysize of the secret part of the SSL handshake (available if the server is in secure mode)
`HTTPS_SESSIONID`	Session ID for the connection (available if the server is in secure mode)
`CLIENT_CERT`	Certificate the client provided (binary DER format)
`CLIENT_CERT_SUBJECT_DN`	Distinguished Name of the subject of the client certificate
`CLIENT_CERT_SUBJECT_OU`	Organization Unit of the subject of the client certificate
`CLIENT_CERT_SUBJECT_O`	Organization of the subject of the client certificate
`CLIENT_CERT_SUBJECT_C`	Country of the subject of the client certificate
`CLIENT_CERT_SUBJECT_L`	Location of the subject of the client certificate
`CLIENT_CERT_SUBJECT_ST`	State of the subject of the client certificate
`CLIENT_CERT_SUBJECT_E`	E-mail of the subject of the client certificate
`CLIENT_CERT_SUBJECT_UID`	UID part of the CN of the subject of the client certificate
`CLIENT_CERT_ISSUER_DN`	Distinguished Name of the issuer of the client certificate
`CLIENT_CERT_ISSUER_OU`	Organization Unit of the issuer of the client certificate
`CLIENT_CERT_ISSUER_O`	Organization of the issuer of the client certificate
`CLIENT_CERT_ISSUER_C`	Country of the issuer of the client certificate
`CLIENT_CERT_ISSUER_L`	Location of the issuer of the client certificate
`CLIENT_CERT_ISSUER_ST`	State of the issuer of the client certificate
`CLIENT_CERT_ISSUER_E`	E-mail of the issuer of the client certificate
`CLIENT_CERT_ISSUER_UID`	UID part of the CN of the issuer of the client certificate

TABLE 3–1 CGI Variables *(Continued)*

Variable	Description
`CLIENT_CERT_VALIDITY_START`	Start date of the certificate
`CLIENT_CERT_VALIDITY_EXIRES`	Expiration date of the certificate
`CLIENT_CERT_EXTENSION_xxx`	Certificate extension, where *xxx* is the name of the extension
`REVOCATION_METHOD`	Name of the certificate revocation method if it exists
`REVOCATION_STATUS`	Status of certificate revocation if it exists

FastCGI

FastCGI is an enhancement to the CGI, which is a standard for interfacing external applications with Web Servers. Like CGI, FastCGI applications run in separate, isolated processes. FastCGI plug-in allows Web Server to safely work with popular third-party dynamic content generation technologies (such as Perl and Python) in a scalable way.

For information on creating FastCGI applications, see the FastCGI Plug-in chapter in the *Sun Java System Web Server 7.0 Administrator's Guide*.

Index