Sun Java System Web Server 6.1 SP6 Programmer's Guide

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

Part No: 819–6517

060829@15490

Contents

Preface 5

1 **Technology Overview** 11

Sun Java System Web Server Architecture 11

  Content Engines 12

  Server Extensions 12

  Runtime Environments 13

  Application Services 13

Configuration Files 13

Sun Java System Web Server 6.1 APIs 13

  Server-parsed HTML Tags 14

  CGI 14

  Java Servlets and JavaServer Pages (JSP) 14

  NSAPI 15

  Access Control API 16

  Certificate-Mapping API 17

API Summary 18

Changes from Previous Versions 19

  API Changes Since iPlanet Web Server 3.x 19

  API Changes Since iPlanet Web Server 4.0 19

  API Changes Since iPlanet Web Server 4.1 19

  API Changes Since Sun Java System Web Server 6.0 20

2 **Server-parsed HTML Tags** 21

Enabling Server-side HTML 21

▼ To enable server-side parsing using the Administration interface 21

Using Server-side HTML Commands 23

  config 23

include ... 24
echo ... 24
fsize ... 24
flastmod ... 25
exec ... 25
Environment Variables in Server-side HTML Commands ... 25
Embedding Servlets ... 26
Defining Customized Server-parsed HTML Tags ... 27
▼ To define a customized server-parsed tag ... 27
Define the Functions that Implement the Tag ... 28
Write an Initialization Function to Register the New Tag ... 31
Load the New Tag into the Server ... 32
Time Formats ... 32

3 Using CGI ... 35
Enabling CGI ... 35
Specifying CGI Directories ... 36
Specifying CGI File Extensions ... 36
Creating Custom Execution Environments for CGI Programs (UNIX only) ... 37
▼ To create custom execution environments for CGI programs ... 37
Specifying a Unique CGI Directory and UNIX User and Group for a Virtual Server ... 38
▼ To create a `cgi-bin` directory for the virtual server ... 39
Specifying a Chroot Directory for a Virtual Server ... 39
▼ To specify a Chroot Directory for a Virtual Server ... 39
Adding CGI Programs to the Server ... 41
Setting the Priority of a CGI Program ... 41
▼ To configure CGI variables for a virtual server ... 42
Windows CGI and Shell CGI Programs ... 42
Perl CGI Programs ... 42
Global CGI Settings ... 42
▼ To change global CGI settings ... 42
CGI Variables ... 43

Index ... 45

Preface

This guide is a starting point for developers who need information about using the various APIs and programming technologies that are supported in Sun Java™ System Web Server 6.1. The guide summarizes the APIs, and provides information about configuring your server to work with server-side HTML tags and CGI programs.

This preface contains information about the following topics:

- "Who Should Use This Guide" on page 5
- "Using the Documentation" on page 6
- "How This Guide Is Organized" on page 8
- "Documentation Conventions" on page 8
- "Product Support" on page 9

Who Should Use This Guide

The intended audience for this guide is the person who develops, assembles, and deploys Java™ 2 Platform, Standard Edition (J2SE)-supported web applications in a corporate enterprise.

This guide assumes you are familiar with the following topics:

- The J2EE specification
- HTML
- The Java™ programming language
- Java APIs as defined in the Java™ Servlet, JavaServer Pages™ (JSP™), and Java™ DataBase Connectivity (JDBC™) specifications
- Structured database query languages such as SQL
- Relational database concepts
- Software development processes

Using the Documentation

The Sun Java System Web Server manuals are available as online files in PDF and HTML formats from the following location:

`http://docs.sun.com/app/docs/coll/1308.2`

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

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

For Information About	See the Following
Late-breaking information about the software and documentation	*Release Notes*
Getting started with Sun Java System Web Server, including hands-on exercises that introduce server basics and features (recommended for first-time users)	*Getting Started Guide*
Performing installation and migration tasks: ■ Installing Sun Java System Web Server and its various components, supported platforms, and environments ■ Migrating from Sun Java System Web Server 4.1 or 6.0 to Sun Java System Web Server 6.1	*Installation and Migration Guide*

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

For Information About	See the Following
Performing the following administration tasks: ■ Using the Administration and command-line interfaces ■ Configuring server preferences ■ Using server instances ■ Monitoring and logging server activity ■ Using certificates and public key cryptography to secure the server ■ Configuring access control to secure the server ■ Using Java™ 2 Platform, Standard Edition (J2SE) security features ■ Deploying applications ■ Managing virtual servers ■ Defining server workload and sizing the system to meet performance needs ■ Searching the contents and attributes of server documents, and creating a text search interface ■ Configuring the server for content compression ■ Configuring the server for web publishing and content authoring using WebDAV	*Administrator's Guide*
Using programming technologies and APIs to do the following: ■ Extend and modify Sun Java System Web Server ■ Dynamically generate content in response to client requests ■ Modify the content of the server	*Programmer's Guide*
Creating custom Netscape Server Application Programmer's Interface (NSAPI) plugins	*NSAPI Programmer's Guide*
Implementing servlets and JavaServer Pages™ (JSP™) technology in Sun Java System Web Server	*Programmer's Guide to Web Applications*
Editing configuration files	*Administrator's Configuration File Reference*
Tuning Sun Java System Web Server to optimize performance	*Performance Tuning, Sizing, and Scaling Guide*

How This Guide Is Organized

This guide provides a basic Sun Java System Web Server environment overview for designing programs.

The guide has the following chapters:

- Chapter 1, Technology Overview

  This chapter summarizes the various APIs and programming technologies supported by Sun Java System Web Server 6.1.
- Chapter 2, Server-parsed HTML Tags

  This chapter describes how to configure your server to work with server-side HTML tags.
- Chapter 3, Using CGI

  This chapter describes how to configure your server to work with Common Gateway Interface (CGI) programs.

Documentation Conventions

This section describes the conventions used throughout this guide.

- **File and directory paths**

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

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

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

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

  By default, the location of *install_dir* is as follows:

- On UNIX-based platforms: `/opt/SUNWwbsvr/`
- On Windows: `C:\Sun\WebServer6.1`

Product Support

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

- The online support web site at:

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

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

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

CHAPTER 1

Technology Overview

This chapter summarizes the various APIs and programming technologies supported by Sun Java System Web Server 6.1. More information about each API or programming technology is provided either in a chapter in this guide, or in a separate guide (reference information is provided).

This chapter has the following sections:

- "Sun Java System Web Server Architecture" on page 11
- "Configuration Files" on page 13
- "Sun Java System Web Server 6.1 APIs" on page 13
- "API Summary" on page 18
- "Changes from Previous Versions" on page 19

Sun Java System Web Server Architecture

Sun Java System Web Server incorporates a modular architecture that integrates seamlessly with all products in the Sun Java System family of servers. In addition, Sun Java System Web Server 6.1 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 stored in the server

Sun Java System Web Server includes a number of software modules. These are discussed in the following topics:

- "Content Engines" on page 12
- "Server Extensions" on page 12
- "Runtime Environments" on page 13
- "Application Services" on page 13

Content Engines

Sun Java System 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 Management
- Search

The HTTP engine represents the core of Sun Java System Web Server. From a functional perspective, the rest of the Sun Java System Web Server architecture resides on top of this engine for performance and integration functionality.

The Content Management engine enables you to manage the your content of the server. You create and store HTML pages, JavaServer Pages™ (JSP™) pages, 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 Sun Java System Web Server users to search the contents and attributes of documents on the server. As the server administrator you can create a customized text search interface that works with various types of document formats. Sun Java System 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

Sun Java System Web Server extensions enable you to extend or replace the function of the server. The following server extensions are part of the core Sun Java System 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 stand-alone application development interface that enables you to create programs that process your client requests dynamically.

Netscape Server Application Programming Interface (NSAPI) is used to implement the functions the server calls when processing a request (Server Application Functions or SAFs), which provide the core and extended functionality of Sun Java System Web Server. It allows the server's processing of requests to be divided into small steps that may be arranged in a variety of ways for speed and flexible configuration.

Java Servlets and Java Server Pages extensions enable all servlet and JSP metafunctions, 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

In addition to the various server extensions, Sun Java System 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

Finally, the Sun Java System 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 Sun Java System Web Server using the Administration user interfaces (UI), or by editing configuration files. Most of the configuration files are in the directory in the *server_root*`/https-`*server_id*`/config` directory. For example, if Sun Java System Web Server is installed on a Windows machine in `C:\SunONE\Servers\`, the configuration files for the server `myserver.com` are in:

`C:\SunONE\Servers\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 6.1 SP6 Administrator's Configuration File Reference*.

Sun Java System Web Server 6.1 APIs

This section summarizes the various APIs and programming technologies supported by Sun Java System Web Server 6.1, and describes how to get more information about them.

The main categories of extensions and modifications you can make to Sun Java System Web Server are listed below, and are used to do the following:

- 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 14
- “CGI” on page 14
- “Java Servlets and JavaServer Pages (JSP)” on page 14

  Modify the behavior of the server itself by implementing server plugins. Most server plugins are written using Netscape Server Application Programming Interface (NSAPI). Specialized APIs also exist for writing server plugins, such as the Access Control List API (ACLAPI), which is used to control access to server resources. The APIs for modifying server behavior are:
- “NSAPI” on page 15
- “Access Control API” on page 16
- “Certificate-Mapping API” on page 17

  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

Sun Java System Web Server 6.1 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.

See Chapter 2, Server-parsed HTML Tags for more information about defining and using server-parsed tags.

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.

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

For more information about using CGI with Sun Java System Web Server 6.1, see Chapter 2, Server-parsed HTML Tags.

Java Servlets and JavaServer Pages (JSP)

Sun Java System Web Server 6.1 supports the Java™ Servlet 2.3 specification, including web application and WAR file (Web ARchive file) support, and the JavaServer Pages™ (JSP™) 1.2 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. Sun Java System Web Server 6.1 includes all of the files necessary for developing and running Java servlets. You can compile servlets using any Java compiler you like, as long as the `webserv-ext.jar` file is accessible to your Java compiler. The `webserv-ext.jar` file is in the server installation directory at:

```
/bin/https/jar
```

For information about using the Java Servlet API, see

`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 Sun Java System Web Server 6.1, see the *Sun Java System Web Server 6.1 SP6 Programmer's Guide to Web Applications*.

For more information about using JavaServer Pages, see

`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 plugins.

Using NSAPI you can write plugins to extend the functionality of Sun Java System Web Server. An NSAPI plugin 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 6.1 SP6 NSAPI Programmer's Guide*.

The `obj.conf` file contains instructions known as directives that instructs the server how to process requests received from clients. Each instruction is processed either during server initialization or during a particular stage of the request-handling process. Each instruction invokes an 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 6.1</font></H4>`. The `append-trailer` function simply returns the requested resource to the client and appends the given trailer to it.

```
Service method=GET type="text/html"
fn=append-trailer trailer="<H4>
<font color=green>Served by 6.1</font></H4>"
```

Sun Java System Web Server 6.1 comes with a set of predefined SAFs. It also comes with 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 6.1 SP6 Administrator's Configuration File Reference*. For more information about custom SAFs, see the *Sun Java System Web Server 6.1 SP6 NSAPI Programmer'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 no instructions to handle requests.

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

Installing NSAPI Plugins (SAFs)

To load new NSAPI plugins 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. This function 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 6.1 SP6 NSAPI Programmer's Guide* for more information about the following topics:

- Directives in `obj.conf` and 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 Sun Java System Web Server, see the *Sun Java System Web Server 6.1 SP6 Administrator's Configuration File Reference*.

Access Control API

The Access Control API is a C API that allows you to programmatically control accessibility on Sun Java System Web Server.

Access control lists (ACLs) determine 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 access control lists, use the Restrict Access page in the Preferences tab of the Server Manager interface. You can also edit the files that contain the ACLs used by the server.

The default access control list resides in the directory *server_root*/`httpacl`. The default ACL file is `generated.https-`*server_id*`.acl`. There is also a file called `genwork.https-`*server_id*`.acl` that is a working copy the server uses until you save and apply your changes when working with the user interface. When editing the ACL file, you might want to work in the `genwork` file and then use the Server Manager to load and apply the changes.

With the Sun Java System Web Server 6.1, 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 6.1 SP6 Administrator's Guide*.

With the Access Control API you can modify 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 can 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

To tell the server to use your attributes for authentication, you must define your own Loadable Authentication Service (LAS), which is an NSAPI plugin. You 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 Sun Java System Web Server 6.1, see the comments in the *server_root*/`plugins/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 a Sun Java System server by sending a client certificate to the server, the server uses information in the certificate to search the user directory for the entry of the user.

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

- How the server searches the directory for the entry of the user.
- 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 6.1 SP6 Administrator's Configuration File Reference*.

  You can also modify this process programmatically. Sun Java System servers include 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`

  For information about using the Certificate-Mapping API, see the *Certificate-Mapping Programmer's Guide* (http://developer.netscape.com/docs/manuals/certificate/contents.htm).

API Summary

The following table lists the APIs available in Sun Java System Web Server 6.1.

TABLE 1–1 APIs Available in Sun Java System Web Server 6.1

API/Interface/Protocol	Language	Documentation
Interfaces for GeneratingDynamic Content		
Custom Server-parsed HTML Tags	C	Chapter 2, Server-parsed HTML Tags
Java Servlets	Java	Sun Java System Web Server 6.1 *Programmer's Guide to Web Applications*
JavaServer Pages	HTML with additional JSP tags	Sun Java System Web Server 6.1 *Programmer's Guide to 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 ServerPlugins		
NSAPI (in-process shared object/DLL)	C, C++	Sun Java System Web Server 6.1 *NSAPI Programmer's Guide*

TABLE 1–1 APIs Available in Sun Java System Web Server 6.1 *(Continued)*

API/Interface/Protocol	Language	Documentation
Access Control API	C, C++	*Access Control Programmer's Guide*
Certificate-Mapping API	C, C++	*Certificate-Mapping Programmer's Guide*

Changes from Previous Versions

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

- "API Changes Since iPlanet Web Server 3.x" on page 19
- "API Changes Since iPlanet Web Server 4.0" on page 19
- "API Changes Since iPlanet Web Server 4.1" on page 19
- "API Changes Since Sun Java System Web Server 6.0" on page 20

API Changes Since iPlanet Web Server 3.x

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

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 43.
- You can invoke servlets as SSI in HTML pages by using the `<SERVLET>` tag, as discussed in Chapter 2, Server-parsed HTML Tags.
- New features NSAPI

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 *Sun Java System Web Server 6.1 SP6 NSAPI Programmer's Guide*.
- 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 Java System Web Server 6.0

- Java Servlets 2.3 and JavaServer Pages 1.2 are 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 6.1 SP6 Programmer's Guide to Web Applications*.

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, Sun Java System Web Server 6.1 allows you to embed servlets and define your own server-side tags.

This chapter has the following sections:

- "Enabling Server-side HTML" on page 21
- "Using Server-side HTML Commands" on page 23
- "Environment Variables in Server-side HTML Commands" on page 25
- "Embedding Servlets" on page 26
- "Defining Customized Server-parsed HTML Tags" on page 27
- "Time Formats" on page 32

Enabling Server-side HTML

The server parses server-side tags only if server-side parsing is enabled.

▼ To enable server-side parsing using the Administration interface

1. **Access the Class Manager, and then click on the Content Management tab.**

2. **Click the Parse HTML link.**

3. **Use the drop-down list to specify a resource for which the server will parse HTML.**

   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.

4 Choose whether to activate server-parsed HTML. The choices are:

- **No.** The server does not parse HTML.
  - **Yes, with exec tag.** The server parses HTML and allows HTML files to execute arbitrary programs on the server.
  - **Yes, without exec tag.** The server parses HTML but does not allow HTML files to execute arbitrary programs on the server. You might not want to allow the `exec` tag for security or performance reasons.

5 Choose which 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 .shtml.** (Unix/Linux only) The server parses files whose UNIX/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 down server performance.

6 Click OK, and then apply your changes.

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

Note – native threads are turned off

```
Init funcs="shtml_init,shtml_send"
shlib="install_dir/bin/https/bin/Shtml.dll" NativeThreads="no"
fn="load-modules"Note that you must set NativeThread="no" for Sun Java System Web Server
version 6.1 and above. In addition, these functions now originate from Shtml.dll (or
libShtml.so on UNIX), which is located in install_dir/bin/https/bin for Windows, and
install_dir/bin/https/lib for UNIX.
```

In addition, make sure the following directive is added to the `obj.conf` file:

```
<Object name="default">
...
...
Service fn="shtml_send" type="magnus-internal/parsed-html"
method="(GET|HEAD)"
...
</Object>
```

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 an `.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* `<Body>... -->`

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 23
- "include" on page 24
- "echo" on page 24
- "fsize" on page 24
- "flastmod" on page 25
- "exec" on page 25

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" section in the *Sun Java System Web Server 6.1 SP6 NSAPI Programmer's Guide*.
- 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` to report file size as a number of KB or MB. This is the default value.

Example

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

This sets the date format to a value such as 08:23:15 AM Wed Apr 15, 1996, 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 Sun Java System Web Server 6.1, 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 25.

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 can 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 `\` 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

Sun Java System Web Server 6.1 supports the `<SERVLET>` tag as introduced by Java Web Server. This tag allows you to embed servlet output in an SHTML file. No configuration changes are necessary to enable this behavior. If SSI and servlets are both enabled, the `<SERVLET>` tag is enabled.

The `<SERVLET>` tag syntax is slightly different from that of other SSI commands 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 2.3 specification (chapter SRV .13, “Deployment Descriptor”). You can find the specification here:

  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 more information about `web-apps.xml`, see the *Programmer’s Guide to Servlets* (http://docs.sun.com/source/816-5689-10/).

  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-nam>
      <servlet-class> PrintPackage.PrintParams </servlet-class>
  ```

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

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

For legacy (iPlanet Web Server 4.*x*) servlets, the `code` parameter specifies the `.class` file for the servlet and is required. The `codebase` parameter is required if the servlet is *not* defined in the `servlets.properties` file and the `.class` file is *not* in the same directory as the HTML file containing the `<SERVLET>` tag. Legacy servlets must be configured in the default virtual server and do not require a `web.xml` file.

For more information about creating servlets, see the *Sun Java System Web Server 6.1 SP6 Programmer's Guide to Web Applications*.

Defining Customized Server-parsed HTML Tags

In Sun Java System Web Server 6.1, 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 a customized server-parsed tag

1. **"Define the Functions that Implement the Tag" on page 28.**

   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 31.**

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

3. **"Load the New Tag into the Server" on page 32.**

Define the Functions that Implement the Tag

Define the functions that implements the tag in C, using NSAPI.

- Include the header `shtml_public.h`, which is in the directory *install_dir*`/plugins/include/shtml`.
- Link against the `shtml` shared library.
  - On Windows, `shtml.dll` is in *install_dir*`/bin/https/bin.`
  - On UNIX platforms, `libshtml.so` or `.sl` is in *install_dir*`/bin/https/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 to 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 in the usual NSAPI way, using the `net_write` NSAPI function, which writes a specified number of bytes to a specified socket from a specified buffer.

  For more information about NSAPI plugins and functions, see the *Sun Java System Web Server 6.1 SP6 NSAPI Programmer's Guide*.

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

- `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 what 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 Sun Java System Web Server in the usual way for NSAPI plugins. That is, add the following directives to the configuration file `magnus.conf`

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. The left column lists time format symbols, and the right column explains the meanings of the symbols.

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

TABLE 2–1 Time Formats *(Continued)*

Symbol	Meaning
%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"
%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 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, Perl, and shell scripts. CGI programs are invoked through URL.

For more information about CGI, see

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

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

Since the server starts up a process each time the CGI script or program runs, this is an expensive method of programming the server.

This chapter has the following sections:

- "Enabling CGI" on page 35
- "Creating Custom Execution Environments for CGI Programs (UNIX only)" on page 37
- "Adding CGI Programs to the Server" on page 41
- "Setting the Priority of a CGI Program" on page 41
- "Windows CGI and Shell CGI Programs" on page 42
- "Perl CGI Programs" on page 42
- "Global CGI Settings" on page 42
- "CGI Variables" on page 43

Enabling CGI

Sun Java System Web Server provides two ways to identify CGI programs, which are listed below and described in this section:

- "Specifying CGI Directories" on page 36
- "Specifying CGI File Extensions" on page 36

Specifying CGI Directories

To specify directories that contain CGI programs (and only CGI programs) use the CGI Directory page in the Programs tab of the Class Manager. The server treats all files in these directories as CGI programs.

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 in the Class Manager interface, or you can manually add them to `obj.conf` if desired.

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>
```

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 Class Manager interface or manually add more `NameTrans` directives to `obj.conf`.

Specifying CGI File Extensions

Use the CGI File Type page in the Programs tab of the Class Manager to instruct the server to treat all files with certain extensions as CGI programs, regardless of which directory they reside in. The default CGI extensions are `.cgi`, `.bat`, and `.exe`.

To change which extensions indicate CGI programs, modify the following line in the `mime.types` file to specify the desired extensions. Be sure to restart the server after editing `mime.types`.

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

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

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

Creating Custom Execution Environments for CGI Programs (UNIX only)

Before you can create a custom execution environment, you must install the `suid Cgistub` and run it as root:

▼ To create custom execution environments for CGI programs

1. **Log in as the superuser.**

   `su`

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

   `cd` *server_root*`/https-`*instance*

   `mkdir private`

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

   `cd private`

   `cp ../../bin/https/bin/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` add a VARS subelement to VS (virtual server) element.

   `<VARS user="`*user*`"/>`

This lets you write the `send-cgi` SAF line in `obj.conf` as follows:

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

For more information about `send-cgi` in the `obj.conf` file and `server.xml`, see the *Sun Java System Web Server 6.1 SP6 Administrator's Configuration File Reference*.

9 **Restart the server to update the changes into effect.**

Note – Installing `Cgistub` in the *server_root*/`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 6.1 SP6 NSAPI Programmer's Guide*

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

`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 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 SunOS/Solaris. To prevent a malicious user from exploiting `Cgistub`, change the server's temporary directory (using the `magnus.conf` `TempDir` directive) to a directory accessible only to the server user.

After you have installed `Cgistub` you can create custom execution environments by doing the following, as described in this section:

- "Specifying a Unique CGI Directory and UNIX User and Group for a Virtual Server" on page 38
- "Specifying a Chroot Directory for a Virtual Server" on page 39

Specifying a Unique CGI Directory and UNIX User and Group for a Virtual Server

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

First, create the UNIX user and group. The exact steps required to create a user and group vary by operating system. For instructions, consult your operating system's documentation.

▼ To create a `cgi-bin` directory for the virtual server

1. **Log in as the superuser.**

   ```
   su
   ```

2. **Change to the virtual server directory.**

   `cd` *vs_dir*

3. **Create the `cgi-bin` directory.**

   ```
   mkdir cgi-bin
   ```

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

   ```
   chmod 755 cgi-bin
   ```

   Now you can set the virtual server's CGI directory, user, and group in one of these ways:

   - Use the `dir`, `user`, and `group` parameters of the `send-cgi` `Service` SAF in the `obj.conf` file, see the *Sun Java System Web Server 6.1 SP6 Administrator's Configuration File Reference*.
   - Enter this information using the Settings page in the Preferences tab of the Virtual Server Manager, see the *Sun Java System Web Server 6.1 SP6 Administrator's Guide*.

Specifying a Chroot Directory for a Virtual Server

To further improve security, these CGI scripts should be prevented from accessing data outside the virtual server directory.

First, set up the chroot environment. The exact steps required to set up the chroot environment vary by operating system. For instructions, consult your operating system's documentation. The `man` pages for `ftpd` and `chroot` are good place to start.

These are the steps required for Solaris versions 2.6 through 8

▼ To specify a Chroot Directory for a Virtual Server

1. **Log in as the superuser.**

   ```
   su
   ```

2. **Change to the chroot directory. This is typically the *vs_dir* directory mentioned in the previous section.**

   `cd` *chroot*

3. **Create `tmp` in the chroot directory:**

   ```
   mkdir tmp
   ```

```
chmod 1777 tmp
```

4. **Create `dev` in the chroot directory:**

   ```
   mkdir dev
   ```

   ```
   chmod 755 dev
   ```

5. **List `/dev/tcp`, and note the major and minor numbers of the resulting output. In this example, the major number is 11 and the minor number is 42:**

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

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

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

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

   ```
   chmod 666 dev/tcp
   ```

7. **Repeat steps 5 and 6 for each of the following devices each device will have a different major and minor 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:**

   ```
   mkdir usr
   ```

   ```
   mkdir usr/lib
   ```

   ```
   ln -s /usr/lib
   ```

   ```
   ln /usr/lib/* usr/lib
   ```

   You can ignore the messages this command generates.

   If the `/usr/lib` directory is on a different file system, replace the last command with the following:

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

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

    ```
    mkdir usr/bin
    ```

    ```
    ln -s /usr/bin
    ```

    ```
    ln /usr/bin/* usr/bin
    ```

    You can ignore the messages this command generates.

If the `/usr/bin` directory is on a different file system, replace the last command with the following:

```
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:

```
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
```

Now you can set the chroot directory of the virtual server in one of these ways:

- Use the `chroot` parameter of the `send-cgi Service` SAF in the `obj.conf` file, see the *Sun Java System Web Server 6.1 SP6 Administrator's Configuration File Reference*.
- Enter this information using the Settings page in the Preferences tab of the Virtual Server Manager, see the *Sun Java System Web Server 6.1 SP6 Administrator's Guide*.

Adding CGI Programs to the Server

To add CGI programs to Sun Java System Web Server, do one of the following:

- Drop 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).

  For UNIX, 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 priority of the CGI program relative to the server. The server is run with a `nice` value of 0 and the `nice` increment would be between 0 (the CGI program runs at same priority as server) and 19 (the CGI program runs at much lower priority than server). As with all other CGI variables, this can be configured per class or per virtual server.

For more information about `send-cgi`, see the *Sun Java System Web Server 6.1 SP6 Administrator's Configuration File Reference*.

To configure CGI variables for a class:

- Access the Class Manager, click CGI Settings on the Virtual Server tab, and then specify the `nice` value as desired before clicking OK and applying your changes.
- Access the Server Manager, click the Virtual Server Class tab, and then click Edit Classes. Click Advanced, and specify the `nice` value as desired before clicking OK and applying your changes.

  All virtual servers in the specified class will inherit the value from the class. You can override this by setting the value for an individual virtual server, as described below.

▼ To configure CGI variables for a virtual server

1. **Access the Class Manager, and then click Manage Virtual Servers on the Virtual Servers tab.**
2. **Select a virtual server, click Manage, and then click Settings.**
3. **Specify the `nice` value as desired, and then click OK to apply your changes.**

   The value overrides any value set at the class level.

Windows CGI and Shell CGI Programs

For information about installing CGI and shell CGI programs on Windows using the Class Manager interface, see the *Sun Java System Web Server 6.1 SP6 Administrator's Guide*.

Perl CGI Programs

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

```
use warnings;
```

Global CGI Settings

▼ To change global CGI settings

1. **Access the Server Manager, and click the Magnus Editor page.**
2. **Select CGI Settings from the drop-down list, and then click Manage.**

3 In the Magnus Editor, specify values for the following settings:

- *MinCGIStubs.* Controls the number of processes that are started by default. Note that if you have an `init-cgi` directive in the `magnus.conf` file, the minimum number of CGIStub processes are spawned at startup. The value must be less than the MaxCGIStubs value.
- *CGIExpirationTimeout.* Specifies the maximum time in seconds that CGI processes are allowed to run before being killed.
- *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.
- *MaxCGIStubs.* Controls the maximum number of CGIStub processes the server can spawn. This is the maximum concurrent CGIStub processes in execution and not the maximum number of pending requests.

4 Click OK and apply your changes.

For more information about these global CGI settings, see the description of `magnus.conf` file in the *Sun Java System Web Server 6.1 SP6 Administrator's Configuration File Reference*.

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. The left column lists the variable, and the right column provides a description.

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)

TABLE 3–1 CGI Variables *(Continued)*

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

Index

A

abbrev, value of sizefmt attribute, 24
about this guide
  audience, 5
  contents, 8
  conventions, 8-9
Access Control API, 16-17
  changes to, 17
  documentation, 19
  loading new authentication services, 17
access control in Sun Java System Web Server, 16
access control lists, creating, 17
ACL files
  configuring for virtual servers, 17
  default, 17
ACLs, about, 16
Administration interface, using, 7
Agents API, 19
APIs
  Access Control, 16-17
  Certificate-Mapping, 17-18
  CGI, 14
  changes in Sun Java System Web Server, 19
  documentation, 18-19
  for server-parsed HTML tags, 14
  in Sun Java System Web Server, 18-19
  Java servlets and JavaServer Pages, 14-15
  NSAPI, 15-16
  summary, 18-19
  supported, 13-18
applets, server-side, 19
application services, 13
architecture, overview, 11

B

bytes, value of sizefmt attribute, 24

C

Certificate-Mapping API, 17-18
Certificate-Mapping Programmer', 18
certmap.conf, 18
CGI
  adding CGI programs to the server, 41
  architecture overview, 12
  enabling, 35-36
  overview, 14
  specifying CGI directories, 36
  specifying file extensions, 36
  variables, 43-44
  web site, 35
cgi attribute of the exec command, 25
CGI Processor, runtime environment, 13
CGI programs, priority of, 41-42
CGI settings, global, 42-43
CGI variables
  configuring, 41
  Sun Java System Web Server, 43-44
CGIExpirationTimeout, 43
Cgistub, 37
CGIStubIdleTimeout, 43
changes, API, 19
changing global CGI settings, 42-43
CLIENT_CERT CGI variable, 43, 44
cmd attribute of the exec command, 25
compiling Java servlets, 15
compression, HTTP, 20

config command, 23
config directory, 13
configuration files, 13
  location, 13
  overview, 13
  reference, 13
configuring
  CGI variables, 41
  Sun Java System Web Server, 13
content
  changing on server, 14
  dynamically generating, 13
Content Management engine, 12
creating
  access control lists, 17
  custom SAFs, 15
  custom server-side tags, 27-32
  NSAPI plugins, 7
  servlets, 27
  web applications, 15
custom server-side HTML tags
  initialization functions for, 31-32
  loading, 32

D

DATE_GMT, 25
DATE_LOCAL, 25
day of month, 32
defining, server-side tags, 27-32
directives, 15
directories for CGI, 36
DOCUMENT_NAME, 25
DOCUMENT_URI, 25
documentation, Sun Java System Web Server, 6-7
dynamically generating content, 13

E

echo command, 24
editing
  configuration files, 7, 13
embedding servlets, 26-27
enabling
  CGI, 35-36
  server-side HTML, 21-23
environment variables, server-side HTML
  commands, 25-26
errmsg attribute of config command, 23
exec command, 25
extending
  server functionality, 14
  the server, 15
extensions, server, 12

F

file attribute of include command, 24
file extensions, for CGI, 36
File System Service, 13
flastmod
  affected by timefmt attribute, 23
  server-side HTML command, 25
fsize command, 24

G

generating dynamic content, 13
global CGI settings, 42-43

H

header files
  nsapi.h, 16
  shtml_public.h, 28
HTML tags
  server-parsed, 14
  server-parsed commands, 21-33, 35-44
HTTP compression, 20
HTTP engine, architecture overview, 12
HttpApplets, 19

I

include command, 24

installing, 12
  CGI and shell CGI programs, 42
  plugins (SAFs), 16
  Sun Java System Web Server, 6

J

Java
  applications, 12
  compiler, 15
  server-side applets, 19
  Servlet API, 15
  Servlet specification, 14
  servlets (see also servlets), 14-15
Java Virtual Machine (JVM), 13
JavaServer Pages (see also JSP), 14-15
JSP
  about, 14-15
  architecture overview, 12
  more information, 15
  specification, 14

L

LAS, defining, 17
LAST_MODIFIED, 26
legacy servlets, 20
load-modules SAF, 16, 32
Loadable Authentication Service (LAS), 17
loading
  custom SAFs, 16
  custom server-side HTML tag, 32
  new authentication services, 17
  NSAPI plugins, 16

M

magnus.conf, 13
  adding new authentication services, 17
  installing NSAPI plugins, 16
  more information about, 7, 13
Mail Service, 13
MaxCGIStubs, 43
mime.types, 13
  server-side parsing, 23
  specifying CGI extensions, 36
MinCGIStubs, 43
modifying server behavior, 14
modules, software, 11
month name, 33

N

NativeThreads, 22
net_write NSAPI function, 28
nice parameter, 41
NSAPI, 14, 15-16
  architecture overview, 12
  filters, 20
  header file, 16
  installing plugins, 16
  more information about, 7
  server extension, overview of, 12
  using to write plugins, 15
NSAPI Engine, runtime environment, 13
nsapi.h, 16

O

obj.conf, 13
  directives, 15
  more information about, 7, 13
  specifying CGI directories, 36
  specifying CGI extensions, 36

P

parse-html SAF, 23
parsing, server-side, 21-23
plugins
  creating, 15
  installing, 16
product support, 9

Q

QUERY_STRING_UNESCAPED, 25

R

return codes
  REQ_ABORTED, 28
  REQ_EXIT, 28
  REQ_NOACTION, 28
  REQ_PROCEED, 28
REVOCATION variable, 43
runtime environments, Web Server, 13

S

s Guide, 18
SAFs, 15
  creating custom, 15
  installing, 16
  predefined, 16
Search engine, architecture overview, 12
security and access control, 13
send-cgi, 41
Server Application Functions, see also SAFs, 15
server behavior, modifying, 14
server extensions, 12, 15
server-parsed HTML tags, 14
  defining customized, 27-32
server plugins, 15
server-side applets, 19
server-side HTML, enabling, 21-23
server-side HTML commands, 21-33, 35-44
  config, 23
  creating new server-side tags, 27-32
  echo, 24
  environment variables in, 23, 25-26
  exec, 25
  flastmod, 25
  format, 23
  fsize, 24
  include, 24
  using, 23-25
server-side HTML tags, creating, 27-32
server-side tags, 14
server.xml, 13
SERVLET tag, 26-27
servlets
  about, 14-15
  API, 15
  architecture overview, 12
  compiling, 15
  embedding in HTML files, 26-27
  legacy, 20
  specification, 14
Session Management Service, 13
SHTML, 26
shtml_add_tag, 27, 32
shtml.dll, 28
shtml_init, 22
shtml_public.h, 28
shtml_send, 22
shtml.so, 28
ShtmlTagExecuteFunc, 28
ShtmlTagInstanceLoad, 28
ShtmlTagInstanceUnload, 28
ShtmlTagPageLoadFunc, 28
ShtmlTagPageUnLoadFn, 29
sizefmt attribute of config command, 23
software modules, Web Server, 11
specifications, supported, 14
SSI, 26
summary, APIs, 18-19
Support, 9
supported specifications, 14

T

tag execution function, 27
tags, server-parsed HTML, 21-33, 35-44
TagUserData, 28, 29
technology overview, 11-20
time formats, 23, 32-33
timefmt tag, 23

U

using
  Administration interface, 7

using *(Continued)*
    CGI, 35-44
    NSAPI, 15
    server-side HTML commands, 23-25

V

variables
    CGI, 43-44
    in server-side HTML commands, 25-26
virtual attribute of the include command, 24
virtual server, configuring ACL files for, 17

W

WAR file, 14
Web Publishing layer, architecture overview, 12
Web Server
    application services, 13
    architecture overview, 11
    configuration files, 13
    content engines, 12
    documentation, 6-7
    runtime environments, 13
    server extensions, 12
    software modules, 11
WebDAV support, 20
webserv-ext.jar, 15
weekday, 32