

Contents

This book was last modified on 8/12/99

About This Book	3
Chapter 1 Overview	5
API Changes in Enterprise Server 4.0	6
Configuration Files	6
Enterprise Server 4.0 APIs	7
Server-Parsed HTML Tags	8
Server-Side JavaScript	9
CGI	12
Java Servlets and JavaServerPages (JSP)	14
NSAPI	17
Web Application Interface (WAI) API	19
Access Control API	20
Web Publishing API	21
API Summary	22
Chapter 2 Configuration Files	25
Introduction	25
magnus.conf	26
obj.conf	27
mime.types	29
Chapter 3 Server-Parsed HTML Tags	33
Using Server-Side HTML Commands	33
config	34
include	35
echo	35
fsize	35
flastmod	36

exec	36
Environment Variables in Server-Side HTML Commands	36
Defining Customized Server-Parsed HTML Tags	37
The Mechanics	38
Chapter 4 NSAPI Changes	41
Privatization of Some Data Structures	41
Logging Changes	42
Cookie Support	43
New SAF for Security	43
New SAFs for Adding Headers and Footers	43
Minor Changes to Init-class SAFs	44
Chapter 5 WAI Release Notes	45
WAI Compatibility Issues	45
Index	49

About This Book

This book was last updated 8/12/99.

This book is a starting point for developers who need information about using the various APIs and programming technologies that are supported by Enterprise Server 4.0.

This book summarizes each of the APIs and programming technologies, and tells you where to find more information. In general, each API or programming technology is documented in a separate Programmer's Guide, with the exception of the API for defining customized server-parsed tags, which is discussed in Chapter 3, "Server-Parsed HTML Tags" in this book.

This book has the following chapters:

- Chapter 1, "Overview"
This chapter discusses the changes in the APIs that are provided with the server from Enterprise Server 3.x to 4.0. It also summarizes the various APIs and programming technologies supported by the server and tells you where to look for more information.
- Chapter 2, "Configuration Files"
This chapter summarizes the configuration files that the Enterprise Server uses.
- Chapter 3, "Server-Parsed HTML Tags"
This chapter discusses how to use server-parsed tags, lists the standard ones, and explains how to define your own.
- Chapter 4, "NSAPI Changes"
This chapter discusses the changes to NSAPI in Enterprise Server 4.0.
- Chapter 5, "WAI Release Notes"
This chapter discusses how to use WAI in Enterprise Server 4.0.

Overview

Enterprise Server 4.0 supports a variety of application programming interfaces (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

This chapter discusses the changes in the APIs that are provided with the server from Enterprise Server 3.x to 4.0. It also summarizes the various APIs and programming technologies supported by the server. More information on each API or programming technology is provided either in a chapter in this book, or in a separate book.

The sections in this chapter are:

- API Changes in Enterprise Server 4.0
- Configuration Files
- Enterprise Server 4.0 APIs
- API Summary

API Changes in Enterprise Server 4.0

- New API for defining customized server-parsed tags as NSAPI plugins has been added. For more information, see "Chapter 3, "Server-Parsed HTML Tags."
- Java Servlets version 2.1 and JavaServerPages 0.92 are supported.
- Server-side JavaScript includes support for JavaScript 1.4 and the JavaScript Application Manager has some cosmetic interface changes.
JavaScript 1.4 is discussed in the "New Features in this Release" section of the "About this book" chapter of the "Core JavaScript Reference" at:
<http://developer.netscape.com/docs/manuals/js/core/jsref/index.htm>
- Server side Java applets (HttpApplets) are not supported. Use Java servlets instead.
- NSAPI has some additional functions, as discussed in Chapter 4, "NSAPI Changes."
- Agents API is not supported.
- Enterprise Server 4.0 does not contain the Visibroker object request broker. If you want to run WAI or other CORBA/IIOP applications, you must first install Visibroker 3.3 or higher from Inprise. For information about Visibroker, see:
<http://www.inprise.com/visibroker/products/>
- WAI is provided in Enterprise Server 4.0, but is not guaranteed to be supported in future releases. We recommend that you do not develop new WAI applications. Before installing and using WAI you need to separately install Visibroker 3.3 or higher from Inprise. For information about WAI update compatibility issues in this release, see "Chapter 5, "WAI Release Notes"."

Configuration Files

You can configure the Enterprise Server using the Server Manager interface, or by editing configuration files. The configuration files are in the `config` directory in the `https-server_id` directory in the installation directory. For

example, if Enterprise Server is installed on a Windows NT machine in `D:\Netscape\server4\`, the configuration files for the server `boots.mcom.com` are in:

```
D:\Netscape\server4\https-boots.mcom.com\config
```

The main configuration files are `magnus.conf`, `obj.conf`, and `mime.types`, but there are other configuration files as well. See Chapter 2, "Configuration Files," for an overview of these configuration files.

For more detailed information about the files `magnus.conf`, `obj.conf`, and `mime.types`, see the "*NSAPI Programmer's Guide*" at

```
http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm
```

The "*NSAPI Programmer's Guide*" is also shipped in the Enterprise Server 4.0 build in the `manuals/nsapi` directory.

Enterprise Server 4.0 APIs

This section summarizes the various APIs and programming technologies supported by Enterprise Server 4.0, discusses how to enable the functionality in Enterprise Server 4.0, and mentions where to get more information about them.

The main categories of extensions and modifications you can make to the Enterprise Server are:

- Dynamically generating responses (or parts of responses) to requests. The APIs and programming approaches that fall in this category are:
 - Server-Parsed HTML Tags
 - Server-Side JavaScript
 - CGI
 - Java Servlets and JavaServerPages (JSP)
- Modifying the behavior of the server itself by implementing server plugins. Most server plugins are written using Netscape Server API (NSAPI). There are also specialized APIs for writing server plugins, such as the Access Control List API (ACLAPI) which is used for controlling access to server resources.

The APIs for modifying server behavior are:

- NSAPI
- Web Application Interface (WAI) API
- Access Control API
- Modifying the content of the server, by adding, removing, or modifying resources and directories. To do this, either use remote file manipulation or the Web Publishing API.

Server-Parsed HTML Tags

Enterprise Server 4.0 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.

Enabling Server-Parsed Tags

To activate and deactivate the parsing of server-side tags, use the Parse HTML page in the Programs tab of the Server Manager. This page enables you to switch off parsing of server-side HTML tags, or enable it with or without also enabling the `exec` tag. The page also allows you to specify whether to parse all files or just those with a `.shtml` extension.

The directive in `obj.conf` that enables the parsing of server-side tags is:

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

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 will be parsed for server-side tags.

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

After making changes to `mime.types`, restart the Enterprise Server to update its table of MIME type mappings, since the `mime.types` file is only loaded when the server is initialized.

For More Information

See Chapter 3, “Server-Parsed HTML Tags,” for more information about defining and using server-parsed tags.

Server-Side JavaScript

Enterprise Server 4.0 supports JavaScript version 1.4.

Using JavaScript, you can create dynamic HTML pages that process user input and maintain persistent data using special objects, files, and relational databases. Through JavaScript's LiveConnect functionality, your applications can access Java and CORBA distributed-object applications.

LiveConnect allows you to access Java objects from a JavaScript application. With LiveConnect, you can create an instance of a Java class from within a JavaScript script. You can also access JavaScript objects from within Java.

Some developers choose to use JavaScript solely on the client (such as Netscape browsers). Larger-scale applications frequently have more complex needs, such as communicating with a relational database, providing continuity of information from one invocation to another of the application, or performing file manipulations on a server. For these more demanding situations, Netscape web servers contain server-side JavaScript, which has extra JavaScript objects to support server-side capabilities.

Some aspects of the core language act differently when run on a server. In addition, to supporting the increased performance demands in these situations, server-side JavaScript is compiled before installation, whereas the runtime engine compiles each client-side JavaScript script at runtime.

For information about server-side JavaScript, see the book "Writing Server-Side JavaScript Applications" at:

<http://developer.netscape.com/docs/manuals/enterprise/4.0/wrijsap/index.htm>

Enabling Server-side JavaScript

To enable or disable server-side JavaScript, use the Server-Side JavaScript page in the Programs tab in the Server Manager interface.

When server-side JavaScript is enabled, the `obj.conf` file has the following directives:

- In the Init section:

```
Init objects="d:/netscape/server4/https-boots.mcom.com/
config/jsa.conf" fn="livewireInit"
```

- In the default object:

```
NameTrans fn="livewireNameTrans" name="ServerSideJS"
```

- In a separate named object:

```
<Object name="ServerSideJS">
  Service fn="livewireService"
</Object>
```

If an administration server password is required to access the Server Side Javascript Application Manager, the `ServerSideJS` object will have more directives.

Compiling Server-side JavaScript Applications

Before you can deploy a server-side JavaScript application, you must compile it into a `.web` file. You will find several sample JavaScript applications, including the source files, the `.web` files, and the make files in the directory `server-root/plugins/samples/js`. One of the easiest ways to compile a new application is to copy a make file for an existing application and modify it to suit your needs.

For information about compiling server-side JavaScript Applications, see the section "Compiling an application" in "Chapter 3, Mechanics of Developing JavaScript Applications" in the book "Writing Server-Side JavaScript Applications" at:

```
http://developer.netscape.com/docs/manuals/enterprise/4.0/wrijsap/
index.htm
```

Installing Server-side JavaScript Applications

After compiling the application, use the JavaScript Application Manager to register it with the Enterprise Server.

To access the JavaScript Manager in a browser, open the URL `http://server_name/appmgr/`, for example `http://poppy.mcom.com/appmgr/`.

To access the JavaScript Application Manager from the Server Manager interface, go to the Server-side JavaScript page in the Programs tab. At the top of the page, you see a line saying something like:

```
The Server side Javascript Application Manager is at http://
poppy.mcom.com/appmgr/
```

Click on the location for the application manager. The application manager opens in another browser window. You can use the application manager to add JavaScript applications, run them, stop them, start them, and debug them.

For information about using the JavaScript Application Manager, see:

- The Programs chapter of the *Enterprise Server 4.0 Administrator's Guide* (which is shipped in the build in the `manuals/ag` directory.)
- "Chapter 3, Mechanics of Developing JavaScript Applications" in the book "Writing Server-Side JavaScript Applications" at:

<http://developer.netscape.com/docs/manuals/enterprise/4.0/wrijsap/index.htm>

For More Information

For information about JavaScript, you can view the following online books on DevEdge:

- JavaScript 1.4 is discussed in the "New Features in this Release" section of the "About this book" chapter of the "Core JavaScript Reference" at:

<http://developer.netscape.com/docs/manuals/js/core/jsref/index.htm>

- JavaScript Reference

This book is a reference manual for the JavaScript language, including objects in the core language and both client-side and server-side extensions. (Do not use client-side extensions in server-side programs!)

<http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm>

- Writing Server-Side JavaScript Applications

This book provides information about JavaScript's server-side capabilities, what has been added to core JavaScript to run on the server, and how the language differs between the client and the server. It also describes the mechanics of creating a server-side JavaScript application and adding it to the server.

<http://developer.netscape.com/docs/manuals/enterprise/4.0/wrijsap/index.htm>

- Accessing External Databases

For a detailed explanation of interacting with external databases, see chapters 8, 9, and 10 of "Writing Server-Side JavaScript Applications" at:

<http://developer.netscape.com/docs/manuals/enterprise/4.0/wrijsap/index.htm>

Also see "Chapter 10, LiveWire Database Service" of the JavaScript Reference at:

<http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm>

- Additional JavaScript Information

<http://www.mozilla.org/js/>

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 as shell scripts. CGI programs are invoked through URL invocation.

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

Enabling CGI

Enterprise Server provides two ways to identify CGI programs:

- Specifying CGI Directories. The server treats all files in CGI directories as CGI programs.
- Specifying CGI File Extensions. The server treats all files with the specified extensions as CGI programs.

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 Server 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 Server 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 `D:/Netscape/Server4/docs/mycgi`.

```
NameTrans fn="pfx2dir" from="/cgi-local"
dir="D:/Netscape/Server4/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 never recognize CGI directories, regardless of whether you specify them in the Server 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 Server Manager to instruct the 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 `mime.types` 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 an appropriate extensions as CGI programs, the `obj.conf` file contains the following `Service` directive:

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

Adding CGI Programs to the Server

To add CGI programs to the Enterprise Server, simply 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 More Information

A myriad of information about writing CGI programs is available. A good starting point is "The Common Gateway Interface" at:

<http://hoohoo.ncsa.uiuc.edu/cgi/overview.html>

Java Servlets and JavaServerPages (JSP)

Enterprise Server 4.0 supports Java servlets and JavaServer Pages (JSP). The server supports Java Servlets API 2.1, and JSP API Level 0.92.

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 do. Servlets are invoked through URL invocation.

You create servlets using the Servlets API, which was created by Sun Microsystems. Enterprise Server 4.0 includes all the files necessary for developing and running Java Servlets. You can compile servlets using any Java compiler you like, so long as the `servlets.jar` file is accessible to your Java compiler. The `servlets.jar` file is in the server installation directory at:

`/bin/https/jar`

For information on using the Servlet API see the "Java Servlet API" documentation from Sun Microsystems at:

<http://www.javasoft.com/products/servlet/index.html>

A JavaServer Page (JSP) is a page much like an HTML page that can be viewed in a web browser. However, as well as containing HTML tags, it can include a set of JSP tags that extend the ability of the web page designer to incorporate dynamic content in a page. These tags provide functionality such as displaying property values and using simple conditionals.

Enabling Java Servlets and JavaServer Pages

When you install Enterprise Server 4.0, you can choose to install the Java Runtime Environment (JRE) or you can specify a path to the Java Development Kit (JDK).

The server can run servlets using the JRE, but it needs the JDK to run JSP. The JDK is not bundled with the Enterprise Server, but you can download it for free from Sun Microsystems at:

<http://www.javasoft.com/products/jdk/1.2/>

Enterprise Server 4.0+ requires you to use an official version of JDK1.2 on Solaris and NT. On HP, AIX and IRIX use JDK 1.1.

Regardless of whether you choose to install the JRE or specify a path to the JDK during installation, you can tell the Enterprise Server to switch to using either the JRE or JDK at any time, by using the “Configure JRE/JDK Paths” page in the Servlets tab of the Server Manager.

Before the server can serve servlets and JSP, the servlet engine must be enabled. To enable servlets and JSP, use the Enable/Disable Servlets page in the Servlets tab of the Server Manager interface. If servlets are enabled, JSP can be enabled or disabled. If servlets are disabled, JSP is also disabled.

When servlets are enabled, the `obj.conf` file contains the following `Init` directives. The first one loads the servlets library and makes the servlet-related functions available to the Enterprise Server. The other two directives initialize the servlet engine.

```
Init shlib="d:/netscape/server4/bin/https/bin/NSServletPlugin.dll"
func="NSServletEarlyInit,NSServletLateInit,NSServletNameTrans,
NSServletService" shlib_flags="(global|now)" fn="load-modules"

Init EarlyInit="yes" fn="NSServletEarlyInit"

Init LateInit="yes" fn="NSServletLateInit"
```

The file `obj.conf` also has other directives that relate to servlets, and defines several additional objects for processing requests for servlets.

Adding Servlets and JavaServerPages to the Server

There are two ways to make a servlet accessible to clients once servlet activation has been enabled:

- Put the servlet class file in a directory that has been registered with the Enterprise Server as a servlet directory.

Servlets in registered servlet directories are dynamically loaded when needed. The server monitors the servlet files and automatically reloads them on the fly as they change. Initially, the Enterprise Server has a single servlet directory, which is `server_id/docs/servlet/`.

For details of how to register servlet directories, see the book "Programmer's Guide to Servlets in ES 4.0."

- Define a servlet virtual path translation for the servlet. In this case, the servlet class file can be located anywhere in the file system or even reside on a remote machine.

For details of how to define servlet virtual path translations, see the book "Programmer's Guide to Servlets in ES 4.0."

To add a JSP file to the server, simply give the file a `.jsp` extension, and put it on the server in a directory at or below the document root. So long as JSP is enabled, the Enterprise Server treats all files with a `.jsp` extension as JavaServer Pages.

Note Do not put JSP files in a registered servlets directory, since the Enterprise Server expects all files in a registered servlet directory to be servlets.)

For More Information

For more information about using servlets in Enterprise Server 4.0, see the book "Programmer's Guide to Servlets in ES 4.0."

For more information about using the Servlets API to create servlets, see the "Java Servlet API" documentation from Sun Microsystems at:

<http://www.javasoft.com/products/servlet/index.html>

For information about creating JavaServer Pages, see Sun Microsystem's JavaServer Pages 0.92 spec in the build in the manuals/https/servlets/jsp092. subdirectory.

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 the Enterprise Server. an NSAPI plugin defines one or more Server Application Functions (SAFs). You can develop SAFs for implementing custom authorization, custom logging, or for other ways of modifying how the Enterprise Server handles requests.

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 server application function (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>Served by 4.0</H4>`. (The `append-trailer` function simply returns the requested resource -- in this case an HTML file -- 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 4.0</font></H4>"
```

Enterprise Server 4.0 comes with a set of pre-defined SAFs. It also comes with a library of NSAPI functions for developing your own SAFs to modify the way that the server handles requests.

Enabling NSAPI

You don't enable NSAPI as such. You use it to develop server application functions (SAFs) to use in the file `obj.conf`. The file `obj.conf` is essential for the operation of the server -- if it does not exist, the server cannot work, since it has nowhere to look for instructions on how to handle requests.

When defining new SAFs, include the header function `nsapi.h` (which is in `server-root/plugins/include`) to get access to all the NSAPI functions.

Installing NSAPI Plugins (SAFs)

To load new NSAPI plugins containing customized SAFs into the server, add an `Init` directive to `obj.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.

For example, the following directive loads the shared library `d:/netscape/server4/bin/https/bin/httpdlw.dll`, (which enables server-side Javascript) and makes the functions `livewireInit`, `livewireNameTrans`, and `livewireService` available to the server.

```
Init fn="load-modules"
shlib="d:/netscape/server4/bin/https/bin/httpdlw.dll"
funcs="livewireInit,livewireNameTrans,livewireService"
```

For More Information

For information about changes to NSAPI in Enterprise Server 4.0, see Chapter 4, "NSAPI Changes."

For information about

- the directives in `obj.conf` and how they determine how the server handles requests
- the pre-defined SAFs that ship with Enterprise Server 4.0
- the NSAPI functions available for writing custom SAFs
- how to write custom SAFs
- how to load custom SAFs into the Enterprise Server by adding an `Init` directive to `obj.conf` that calls `load-modules`

see the "*NSAPI Programmer's Guide*" at

```
http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm
```

The "*NSAPI Programmer's Guide*" is also shipped in the Enterprise Server 4.0 build in the `manuals/nsapi` directory.

Web Application Interface (WAI) API

WAI is supported in Enterprise Server 4.0, but is not guaranteed to be supported in future releases.

Using the Web Application Interface (WAI) API, you can write C, C++, and Java applications that process HTTP requests sent to the server. A WAI application runs within its own process. The Enterprise Server interacts with your application over Internet Inter-ORB Protocol (IIOP).

WAI is a CORBA-based programming interface that defines object interfaces to the HTTP request/response data and server information. Using WAI, you can write a web application in C, C++, or Java that accepts an HTTP request from a client, processes it, and returns a response to the client. You can also write your own server plugins for processing HTTP requests.

For more information on writing applications in WAI, see the online manual *Writing Web Applications with WAI* at:

<http://developer.netscape.com/docs/manuals/enterprise/3.0/wai/index.htm>

Enabling WAI

Before installing the WAI component of your Enterprise Server 4.0, you need to install Visibroker 3.3 or higher from Inprise. For information about Visibroker, see:

<http://www.inprise.com/visibroker/products/>

After installing Visibroker, install the WAI component of Enterprise Server 4.0. After WAI is installed, you then need to enable WAI. Do this by using the WAI Management page in the Programs tab of the Server Manager. (If WAI is not installed, this button does not appear.)

Installing WAI Applications

You install a WAI application in the same way that you install other NSAPI plugins. The application must contain an initialization function that registers the application. You load it into the server in the usual manner, by adding the following directives to `obj.conf`:

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

- An `init` directive that calls the initialization function.

Start your application on the host machine that runs the Enterprise server. Make sure that when the initialization function registers the application, it specifies the host name and port of the Enterprise server.

Note that it is possible (but not recommended) to run WAI applications on other machines in the local network. For a complete explanation of the security concerns and instructions for configuring the server to recognize WAI applications on other machines, see Chapter 8, "Security Guidelines for Using WAI" online manual *Writing Web Applications with WAI*.

For More Information

For more information about Visibroker from Inprise see:

<http://www.inprise.com/visibroker/products/>

For more information about writing WAI applications, see:

<http://developer.netscape.com/docs/manuals/enterprise/3.0/wai/index.htm>

Access Control API

The Access Control API is a C API that lets you programmatically control who has access to what on the Enterprise Server.

Access control lists (ACLs) determine who has what kind of access privileges to which resources on the server. Each ACL contains a list of access control entries. The following access control entry, for example, says that all access is denied to everyone for any resource that contains the substring `private`.

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

To create access control lists, use the Restrict Access page in the Programs tab of the Server Manager interface. You can also edit the files that contain the ACLs used by the server.

Access control lists reside in the directory `server-installation-dir/httpacl`. The server uses the default settings in the file `server_root/httpacl/generated.https-serverid.acl`. There is also a file called `genwork.https-serverid.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 Access Control API, you can manipulate access control lists (ACLs), read and write ACL files, and evaluate and test access control to resources on the server. You can also define your own attributes for authentication. For example, you might want to authenticate users based on email address or on the URL that referred them to the resource. 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 need to 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 `obj.conf`:

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

For More Information

For information about using the ACL API, see the "Access Control Programmer's Guide" at:

<http://developer.netscape.com/docs/manuals/enterprise/3.0/accessapi/index.htm>

For information about the syntax for editing ACL files, see Appendix A in the same book.

Web Publishing API

The Web Publishing API provides a set of Java classes that allow client Java applications and applets to manipulate the resources, such as files and directories, on the server. With these classes, the client application can perform standard file system actions such as creating, deleting, and copying resources.

In addition, you can associate meta-information (attributes) with a resource to track arbitrary information about the resource, such as associating a project or a list of reviewers with the resource. You can use the locking facility to ensure that two users don't edit the same resource at the same time. You can also choose to track the history of a component resource by maintaining separate versions of it -- this is known as version control.

Enabling Web Publishing

To enable Web Publishing, use the Web Publishing State page in the Web Publishing tab of the Server Manager interface.

For More Information

For more information on using the Web Publishing Client API, see the "*Web Publishing Client API Guide*" at:

<http://developer.netscape.com/docs/manuals/enterprise/3.0/webpub/index.htm>

API Summary

The following table lists the APIs available in Enterprise Server 4.0.

Table 1.1 APIs available in Netscape Enterprise Server 3.0

API/Interface/Protocol	Language	Documentation
Interfaces for Generating Dynamic Content		
Server-Parsed HTML Tags	C	<i>Chapter 3, "Server-Parsed HTML Tags."</i>
Server-Side JavaScript (LiveWire) and LiveConnect	JavaScript	<i>JavaScript Reference Writing Server-Side JavaScript Applications</i>
Java Servlets	Java	<i>Programmer's Guide to Servlets in Enterprise Server 4.0</i>
JavaServerPages	HTML with additional JSP tags	<i>Programmer's Guide to Servlets in Enterprise Server 4.0</i>

Table 1.1 APIs available in Netscape Enterprise Server 3.0

API/Interface/Protocol	Language	Documentation
CGI (one process per request)	C, C++, Perl, shell, and other languages	<i>The Common Gateway Interface</i>
API for Writing Server Plugins		
NSAPI (in-process shared object/DLL)	C, C++	<i>NSAPI Programmer's Guide</i>
WAI (separate process)	C, C++, Java	<i>Writing Web Applications with WAI</i>
Access Control API	C, C++	<i>Access Control Programmer's Guide</i>
API For Modifying Server Resources		
Web Publishing Interface	Java	<i>Web Publishing Client API Guide</i>

Configuration Files

This chapter gives an overview of the Enterprise Server's three main configuration files. The sections are:

- Introduction
- `magnus.conf`
- `obj.conf`
- `mime.types`

For more detailed information about the files `magnus.conf`, `obj.conf`, and `mime.types`, see the "*NSAPI Programmer's Guide*" at

<http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm>

The "*NSAPI Programmer's Guide*" is also shipped in the Enterprise Server 4.0 build in the `manuals/nsapi` directory.

Introduction

You can configure the Enterprise Server using the Server Manager interface, or by editing configuration files. The configuration files live in the `config` directory in the `https-server_id` directory in the installation directory. For example, if Enterprise Server is installed on a Windows NT machine in `D:\Netscape\server4\`, the configuration files for the server `boots.mcom.com` are in:

D:\Netscape\server4\https-boots.mcom.com\config

The configuration files are:

- `magnus.conf`
Defines global settings for the server, such as the server name. When the server is initialized, it executes the directives in `magnus.conf`.
- `obj.conf`
Provides instructions to the server about how to handle requests from clients such as browser. Whenever you make changes to the server through the Server Manager interface, the system automatically edits the `obj.conf` file. You can also manually edit `obj.conf` to modify the server behavior.
- `mime.types`
Defines the MIME types supported by the server. When the server starts up, it loads this file and creates a table that maps file extensions to MIME types, as defined in the file. For example, the extension `.html` is always mapped to the content type `text/HTML`.
- Other configuration files.
Depending on which features are enabled in the server, the `config` directory contains other configuration files, such as:
`servlets.properties` -- defines the servlet properties.
`rules.properties` -- defines virtual paths for servlets.
`jvm.conf` -- defines Java settings, such as class paths.
`webpub.conf` -- defines settings for Web Publishing.
`acl.conf` -- defines access control lists.
`jsa.conf` -- defines server side JavaScript configuration.

The rest of this section discusses the three main configuration files:

magnus.conf

The file `magnus.conf` defines settings that the server uses for initialization. After the server starts up, it does not look in `magnus.conf` again. This file contains directives that each consist of a variable name and the setting for that variable.

An example of `magnus.conf` is:

```
#ServerRoot D:/Netscape/Server4/https-boots.mcom.com
ServerID https-boots.mcom.com
ServerName boots.mcom.com
Port 80
ExtraPath D:/Netscape/Server4/bin/https/jdk/jre/bin; D:/Netscape/
Server4/bin/https/jdk/jre/bin/classic;D:/Netscape/Server4/wai/bin
LoadObjects obj.conf
RootObject default
ErrorLog D:/Netscape/Server4/https-boots.mcom.com/logs/errors
MtaHost name-of-mail-server
DNS off
Security off
Ciphers +rc4,+rc4export,+rc2,+rc2export,+des,+desede3
SSL3Ciphers +rsa_rc4_128_md5,+rsa_3des_sha,+rsa_des_sha,
+rsa_rc4_40_md5,+rsa_rc2_40_md5,-rsa_null_md5
ACLFile D:/Netscape/Server4/httpacl/generated.https-boots.mcom.com.acl
ClientLanguage en
AdminLanguage en
DefaultLanguage en
AcceptLanguage off
RqThrottle 512
```

For a complete list of the directives in `magnus.conf`, see the `magnus.conf` appendix in the *"NSAPI Programmer's Guide"* at

<http://developer.netscape.com/docs/manuals/enterprise/nsapi/index.htm>

The *"NSAPI Programmer's Guide"* is also shipped in the Enterprise Server 4.0 build in the `manuals/nsapi` directory.

obj.conf

The `obj.conf` file contains additional initialization instructions as well as instructions for the server about how to process requests from clients.

The normal procedure for modifying the Enterprise Server is to use the Server Manager interface. When you use the Server Manager interface to make changes to the Enterprise Server, the system automatically updates the `obj.conf` file.

You can also manually edit `obj.conf` if desired to add, remove, or modify directives. But if you do so, be sure to load `obj.conf` into the Server Manager before using the Server Manager to make further changes, otherwise Server Manager over-writes your manual changes.

Each instruction, or directive, in `obj.conf`, applies during initialization or during a particular stage of the request handling process. The stages are:

1. `Init` -- instructions for initialization. These are performed after the server has set the variables defined in `magnus.conf`.
2. `AuthTrans` -- authorization translation.
3. `NameTrans` -- translates the logical URI into a local file system path.
4. `PathCheck` -- checks the local file system for validity and access permissions.
5. `ObjectType` -- determines the MIME type of the requested resource
6. `Service` -- Generates the response and returns it to the client.
7. `AddLog` -- adds entries to the log file if appropriate.
8. `Error` -- updates the error log if an error occurred.

Each directive in `obj.conf` invokes a server application function (SAF) and passes arguments to it. For example, the following directive is invoked during the Service stage if the request method is GET and the requested content is of the type `text/html`. This directive sets the value of the `trailer` argument to "`<H4>Served by 4.0</H4>`" and passes it to the `append-trailer` SAF.

```
Service fn=append-trailer method=GET type="text/html"
trailer="<H4><FONT COLOR=green>Served by 4.0</FONT></H4>"
```

For more details about `obj.conf`, about the different stages in the request handling process, and for a list of the pre-defined SAFs you can use in directives, see the *NSAPI Programmer's Guide* at:

```
http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm
```

The "*NSAPI Programmer's Guide*" is also shipped in the Enterprise Server 4.0 build in the `manuals/nsapi` directory.

mime.types

When a client, such as a browser, sends a request to the Enterprise Server, the MIME type determines the kind of content being requested. The MIME type is usually indicated by the extension of the requested resource, for example, `.gif` implies the client wants a GIF image; and `.html` implies the client wants an HTML file.

MIME stands for Multipurpose Internet Mail Extension (or Encoding).

The file `mime.types` maps extensions to MIME types. When the Enterprise Server starts up, it loads `mime.types` and uses it to create a table of mappings between MIME types and extensions.

The `ObjectType` directives in the file `obj.conf` give the server instructions on how to determine the MIME type. The catch-all `ObjectType` directive is:

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

The `type-by-extension` function looks up the MIME type according to the requested resource's extension.

The `ObjectType` directives set the `type` parameter. This parameter helps the server determine which `Service` directive to use to generate the response to send back to the client.

For example, if the request is `http://boots/docs/servlet/jos.jsp`, this is how the server decides which `Service` directive to use:

- In `obj.conf`, the directive:

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

tells the server to look up the type in its MIME types table, which is based on the file `mime.types`.
- In the MIME types table (which is based on `mime.types`), the server finds:

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

This tells the server that the type is "magnus-internal/jsp" because the extension is `jsp`.
- Back in `obj.conf`, the directive

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

tells the server to use the function `NSServletService` to generate the response because the type is `Service fn="NSServletService"`.

The server also puts the MIME type in the header information to return to the client so that the client knows what kind of content to receive.

An example of `mime.types` is:

```

type=text/html                exts=htm,html
type=text/plain               exts=txt
type=text/richtext            exts=rtx
type=text/tab-separated-values exts=ts
type=text/x-setext            exts=etx
type=text/x-speech            exts=talk

type=video/isivideo           exts=fvi
type=video/mpeg                exts=mpeg,mpg,mpe,mpv,vbs,mpgvr
type=video/x-mpeg2             exts=mpv2,mp2v
type=video/msvideo             exts=avi
type=video/quicktime           exts=qt,mov,moov
type=video/vivo                exts=viv,vivo
type=video/wavelet             exts=wav

#type=video/x-msvideo          exts=avi
type=video/x-sgi-movie         exts=movie
type=x-world/x-svr             exts=svr
type=x-world/x-vrml            exts=wrl
type=x-world/x-vrt             exts=VRT
type=x-conference/x-cooltalk   exts=ice

enc=x-gzip                    exts=gz
enc=x-compress                exts=z
enc=x-uencode                  exts=uu,uue

type=magnus-internal/imapmap   exts=map
type=magnus-internal/parsed-html exts=shtml
type=magnus-internal/cgi       exts=cgi,exe,bat

type=magnus-internal/jsp       exts=jsp
type=application/x-x509-ca-cert exts=cacert
type=application/x-x509-server-cert exts=scert
type=application/x-x509-user-cert exts=ucert

```

```
type=application/x-x509-email-cert      exts=ecert
```

For more details about the MIME types file and how the server uses it, see the MIME types appendix in the *NSAPI Programmer's Guide* at:

```
http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/  
index.htm
```

The "*NSAPI Programmer's Guide*" is also shipped in the Enterprise Server 4.0 build in the `manuals/nsapi` directory.

mime.types

Server-Parsed HTML Tags

HTML files can contain tags that are executed on the server. In addition to supporting the standard server-side tags, Enterprise Server 4.0 allows you to define your own server-side tags.

This chapter has the following sections:

- Using Server-Side HTML Commands
- Defining Customized Server-Parsed HTML Tags

Note: The server parses server-side tags only if server-side parsing has been activated. Use the "Parse HTML" page in the Content Management tab of the Server Manager interface to enable or disable the parsing of server-side tags.

When you activate parsing, you need to be sure that the following directives are added to your `obj.conf` file (Note that native threads are turned off.):

```
Init funcs="shtml_init,shtml_send" shlib="<install_dir>/bin/https/bin/  
Shtml.dll" NativeThreads="no" fn="load-modules"  
  
Init LateInit = "yes" fn="shtml_init"
```

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 ... -->
```

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:

- `config`
- `include`
- `echo`
- `fsize`
- `flastmod`
- `exec`

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`.

Refer to the Time Formats appendix in the *"NSAPI Programmer's Guide"* for details about time formats. You can find this book online at :

```
http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm
```

- 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.

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 (it can't be a CGI program). 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 where they reside.

- 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 may not contain elements such as `../` and it may not 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 the section "Environment Variables in Server-Side HTML Commands."

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 normal set of environment variables used in CGI, you may include the following variables in your parsed commands:

- `DOCUMENT_NAME`
is the file name of the parsed file.
- `DOCUMENT_URI`
is the virtual path to the parsed file (for example, `/shtml/test.shtml`).
- `QUERY_STRING_UNESCAPED`
is the unescaped version of any search query the client sent with all shell-special characters escaped with the `\` character.
- `DATE_LOCAL`
is the current date and local time.
- `DATE_GMT`
is the current date and time expressed in Greenwich Mean Time.

- `LAST_MODIFIED`
is the date the file was last modified.

Defining Customized Server-Parsed HTML Tags

The parsing of server-side tags in `.shtml` files in Enterprise Server 4.0 has been substantially improved over previous releases of Enterprise Server. First, the performance of handling server-side tags has been significantly speeded up. Secondly, users can now define their own server-side tags.

For example, you could define the tag `<PRICE>` to invoke a function that calculates and displays the price of a product. Then in your `.shtml` file you could have code such as:

```
<H2>Product Prices</H2>
<UL>
<LI>Oak Table: <PRICE product="oaktable">
<LI>Pine Bench: <PRICE product="pinebench">
<LI>Patio Chair: <PRICE product="patiochair">
</UL>
```

When the browser displays this code, each occurrence of the `<PRICE>` tag calls the function that is associated with that tag, and returns the price of the relevant product. The result in the browser might look like:

Product Prices

- Oak Table: \$600
- Pine Bench: \$400
- Patio Chair: \$115

The Mechanics

The steps for defining a customized server-parsed tag are:

1. Define the Functions that Implement the Tag.

You must define the tag execution function, and you can optionally 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.

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

3. Load the New Tag into the Server.

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/plugins/include/shtml`.
- Link against the `shtml` shared library. On Windows NT, `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 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 writing NSAPI plugins see Chapter 4, "Writing Custom SAFs" in the *NSAPI Programmer's Guide* at:

<http://developer.netscape.com/docs/manuals/enterprise/nsapi/4.0/index.htm>

For more information about `net_write()` and other NSAPI functions, see Chapter 5, "NSAPI Function Reference" of the *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` or not, which is one of:

- `REQ_PROCEED` -- the execution was successful.
- `REQ_NOACTION` -- nothing happened.
- `REQ_ABORTED` -- an error occurred.
- `REQ_EXIT` -- the connection was lost.

The other functions you can 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 or not. 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 hence 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);
```

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 be `NULL` except for the `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 the Enterprise Server in the usual way for NSAPI plugins. That is, add the following directives to the configuration file `obj.conf`:

1. Add an `Init` directive whose `fn` parameter is `load-modules` and whose `shlib` parameter is the shared library to load.
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.

NSAPI Changes

This chapter lists the changes to NSAPI in Enterprise Server 4.0. These changes are:

- Privatization of Some Data Structures
- Logging Changes
- Cookie Support
- New SAF for Security
- New SAFs for Adding Headers and Footers
- Minor Changes to Init-class SAFs

Privatization of Some Data Structures

In Enterprise Server 4.0, some data structures have been moved from `nsapi.h` to `nsapi_pvt.h`. The data structures in `nsapi_pvt.h` are now considered to be private data structures, and you should not write code that accesses them directly. Instead, use accessor functions. We expect that very few people have written plugins that access these data structures directly, so this change should have very little impact on existing customer-defined plugins. Look in `nsapi_pvt.h` to see which data structures have been removed from the public domain and to see the accessor functions you can use to access them from now on.

Plugins written for Enterprise Server 3.x that access contents of data structures defined in `nsapi_pvt.h` will not be source compatible with ES 4.0, that is, it will be necessary to `#include "nsapi_pvt.h"` in order to build such plugins from source. There is also a small chance that these programs will not be binary compatible with Enterprise Server 4.0, because some of the data structures in `nsapi_pvt.h` have changed size. In particular, the `directive` structure is larger, which means that a plugin that indexes through the directives in a `dtable` will not work without being rebuilt (with `nsapi_pvt.h` included).

We hope that the majority of plugins do not reference the internals of data structures in `nsapi_pvt.h`, and therefore that most plugins will be both binary and source compatible with Enterprise Server 4.0.

Logging Changes

The following API changes support the logging modifications in Enterprise Server 4.0:

- New Init-class SAF, `flex-rotate-init`, lets you initialize log rotation for logs that use the flexible format.

For more information, see the discussion of `flex-rotate-init` in Chapter 3, "Directives and Their Predefined SAFs" in the *"NSAPI Programmer's Guide"* at

<http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm>

- New relaxed parameter to the Init-class SAF `flex-init` allows you to specify whether logging uses relaxed mode or not. Also, the `format` parameter for `flex-init` now lets you log the values of named cookies.

When logging uses relaxed mode, it skips the logging of any variable that would normally block cache acceleration when processing requests for static files.

For more information, see the discussion of `flex-init` in Chapter 3, "Directives and Their Predefined SAFs" in the *"NSAPI Programmer's Guide"* at

<http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm>

Cookie Support

- Two new NSAPI function for use with cookies.

The `util_cookie_find` function finds a specific cookie in a cookie string and return its value.

The `util_cookie_next` function enumerates all the cookie name-value pairs in a cookie string.

For more information, see the discussion of `util_cookie_find` and `util_cookie_next` in Chapter 5, "NSAPI Function Reference" in the *"NSAPI Programmer's Guide"* at

<http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm>

New SAF for Security

- New PathCheck-class function `ssl-check` helps enforce keysize restriction for cipher settings.

For more information, see the discussion of `ssl-check` in Chapter 3, "Directives and Their Predefined SAFs" in the *"NSAPI Programmer's Guide"* at

<http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm>

New SAFs for Adding Headers and Footers

- New Service-class SAFs, `add-header` and `add-footer`, allow you to specify file names or URLs that provide a header or footer for a page being returned to the requesting client.

For more information, see the discussion of `add-header` and `add-footer` in Chapter 3, "Directives and Their Predefined SAFs" in the *"NSAPI Programmer's Guide"* at

<http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm>

Minor Changes to Init-class SAFs

- Minor changes to the parameters for the Init-class SAF `cache-init`.
For more information, see the discussion of `cache-init` in Chapter 3, "Directives and Their Predefined SAFs" in the *"NSAPI Programmer's Guide"* at
<http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm>
- Minor changes to the parameters for the Init-class SAF `cindex-init`.
For more information, see the discussion of `cache-init` in Chapter 3, "Directives and Their Predefined SAFs" in the *"NSAPI Programmer's Guide"* at
<http://developer.netscape.com/docs/manuals/enterprise/4.0/nsapi/index.htm>

WAI Release Notes

Web Application Interface (WAI) is available in Enterprise Server 4.0, but is not guaranteed to be available in future releases. We recommend that you do not develop new WAI applications, instead use servlets.

For more information about using the WAI API, see "*Writing Web Applications with WAI*" at:

<http://devedge/docs/manuals/enterprise/3.0/wai/index.htm>

Before installing the WAI component of your Enterprise Server 4.0, you need to install Visibroker 3.3 or higher from Inprise. For information about Visibroker, see:

<http://www.inprise.com/visibroker/products/>

After installing Visibroker, install the WAI component of Enterprise Server 4.0. After WAI is installed, you then need to enable WAI. Do this by using the WAI Management page in the Programs tab of the Server Manager. (If WAI is not installed, this button does not appear.)

WAI Compatibility Issues

The main points to be aware of are:

- OSAGENT Registration is Disabled

- Backward Compatibility Issues
- Different Signature for ORB.init()
- Other Java Compatibility Issues

OSAGENT Registration is Disabled

OSAGENT is not started by default in Enterprise Server 4.0 since the server does not need it. Therefore, WAI applications need to specify a command line option to disable OSAGENT registration. If a WAI application starts without using the command-line option, it tries to register itself with the OSAGENT (which is not running) and fails.

You can always start OSAGENT manually, in which case you do not need to specify the command-line option to disable registration. You can find more information about OSAGENT in the Visigenic documents.

For Java WAI applications, use the following command to disable OSAGENT registration:

```
-ORBdisableLocator=true
```

For example:

```
java -DORBdisableLocator=true WASP
```

For C++ WAI applications, use:

```
-ORBagent 0
```

For example:

```
WASP.EXE -ORBagent 0
```

Backward Compatibility Issues

WAI applications built using the previous versions of Enterprise Server need to pass a command line command to work correctly.

For Java WAI applications under JDK 1.1, the command is:

```
-ORBbackCompat=true
```

For example:

```
java -DORBdisableLocator=true -ORBbackCompat=true WASP
```

For C++ WAI applications, the command is:

```
-ORBbackCompat 1
```

For example:

```
WASP.EXE -ORBagent 0 -ORBbackCompat 1
```

Different Signature for ORB.init()

Java WAI applications that use the `ORB.init ()` call have to be modified to use the call with a different signature in order to work under Java 2. Replace the call `ORB.init ()` with the call `ORB.init(String [] args, Properties props)`.

Other Java Compatibility Issues

Please refer to Visibroker for Java 3.4 release notes for more information on compatibility with Java 2 platform at

<ftp://ftp.visigenic.com/private/vbj/vbj34/vbjrel.html>

Index

A

- abbrev, value of `sizefmt` attribute 34
- about this book 3
- Access Control API 20
 - for more info 21
 - loading new authentication services 21
- Access Control Programmer's Guide 21
- `acl.conf` 26
- ACL files
 - editing 21
- add-footer SAF 43
- add-header SAF 43
- AddLog
 - step in request handling process 28
- Agents API 6
- API reference
 - JSP 16
- APIs
 - Access Control 20
 - CGI 12
 - changes in Enterprise Server 4.0 6
 - for server-parsed HTML tags 8
 - in Enterprise Server 4.0 7
 - Java servlets and JSP 14
 - NSAPI 17
 - server-side JavaScript
 - summary 22
 - WAI 19
 - Web Publishing 21
- Application Manager
 - server-side JavaScript 10
- AuthTrans
 - step in request handling process 28

B

- bytes, value of `sizefmt` attribute 34

C

- cache-init SAF
 - minor changes to 44
- CGI 12
 - adding CGI programs to the server 13
 - enabling 12
 - for more info 14
 - specifying CGI directories 12
 - specifying file extensions 13
 - website 14
- cgi attribute of the `exec` command 36
- changes
 - API 6
 - logging changes 42
 - NSAPI 41
- cindex-init SAF
 - minor changes to 44
- cmd attribute of the `exec` command 36
- Common Gateway Interface web site 14
- compiling
 - Java servlets 14
 - server-side JavaScript applications 10
- config
 - server-side HTML command 34
- config directory 6
 - location 6
- configuration files 6, 25
 - location 6, 25
 - `magnus.conf` 26
 - `mime.types` 29
 - `obj.conf` 27

- configuring
 - Enterprise Server 25
 - JDK 15
 - JRE 15
- content
 - changing on server 8
 - dynamically generating 7
- cookies
 - logging 42
 - NSAPI functions 43
- CORBA 19
- CORBA/IIOP 6
- Core JavaScript Reference 6
- creating
 - custom server-side tags 37
 - servlets 16
- custom server-side HTML tags
 - defining 38
 - initialization functions for 40
 - loading 40

D

- databases
 - accessing from server-side JavaScript 11
- DATE_GMT
 - variable in server-side HTML command 36
- DATE_LOCAL
 - variable in server-side HTML command 36
- defining
 - custom server-side HTML tags 37
 - servlets 16
- directives 17, 28
- directories
 - for CGI 12
 - for servlets 16
- DOCUMENT_NAME
 - variable in server-side HTML command 36
- DOCUMENT_URI
 - variable in server-side HTML command 36
- dynamically generating content 7

E

- echo
 - server-side HTML command 35
- editing
 - ACL files 21
- enabling
 - CGI 12
 - JSP 15
 - NSAPI 17
 - server-side JavaScript 9
 - server-side tags 8
 - servlets 15
 - WAI 19
 - web publishing 22
- environment variables
 - in server-side HTML commands 36
- errmsg attribute of config command 34
- Error
 - step in request handling process 28
- exec
 - server-side HTML command 8, 36
- extending
 - server functionality 7
 - the server 17
- external databases
 - accessing from server-side JavaScript 11

F

- file attribute of include command 35
- file extensions
 - .jsp 16
 - .shtml 37
 - for CGI 13
- flastmod
 - affected by timefmt attribute 34
 - server-side HTML command 36
- flex-init SAF 42
- flex-rotate-init SAF 42
- footers
 - adding 43

fsize
server-side HTML command 35

G

generating
dynamic content 7

H

header files
nsapi.h 17
shtml_public.h 38
headers
adding 43
HTML tags
server-parsed 8
server-parsed commands 33
HttpApplets 6

I

IIOP 6, 19
include
server-side HTML command 35
Init
step in request handling process 28
Inprise 6, 19, 45
installing
JRE or JDK 15
JSP 15
plugins (SAFs) 18
server-side JavaScript applications 10
servlets 15
WAI applications 19
WAI component 19, 45
introduction 5

J

Java
JDK versions 15
JSP 14

server-side applets 6
servlets 14
servlets.jar 14

Java Development Kit
see JDK

Java Runtime Environment
see JRE

JavaScript
changes in server-side 6
Core Reference 6
see also Server-Side JavaScript
server-side 9
Writing Server-Side JavaScript Applications 9

JavaScript Application Manager 10
for more info 11

JavaScript Reference 11

JavaServerPages
see JSP

Java Servlet API documentation 16

JDK
configuring 15
installing 15
switching to 15
versions 15

JRE
configuring 15
installing 15
switching to 15

jsa.conf 26

JSP 14
API reference 16
enabling 15
for more info 16
installing 15

jvm.conf 26

L

LAST_MODIFIED
variable in server-side HTML command 37
LiveConnect 9
loading

- custom server-side HTML tag 40
- new authentication services 21
- NSAPI plugins 18
- load-modules SAF 18, 40
- logging
 - changes in NSAPI functions 42
 - cookie values 42
 - relaxed mode 42

M

- magnus.conf 26
 - list of directives 27
- mime.types 29
 - specifying CGI extensions 13
- MIME types 29
 - for more info 31
- modifying
 - server behavior 7

N

- NameTrans
 - step in request handling process 28
- NativeThreads 33
- net_write()
 - NSAPI function 38
- Netscape Server Application Programming Interface
 - see NSAPI
- NSAPI 7, 17
 - changes in Enterprise Server 4.0 41
 - enabling 17
 - for more info 18
 - header file 17
 - installing plugins 18
- nsapi.h 17
- NSAPI Programmer's Guide 7

O

- obj.conf 27

- adding new authentication services 21
- directives 17
- enabling server-parsed tags 8
- enabling server-side JavaScript 9
- enabling servlets 15
- for more info 28
- installing NSAPI plugins 18
- installing WAI plugins 19
- see NSAPI Programmer's Guide for more info 18
- specifying CGI directories 13
- specifying CGI extensions 13
- using NSAPI to write SAFs 17

- ObjectType
 - directive 29
 - step in request handling process 28
- ORB.init() 47
- OSAGENT 46
- overview 5

P

- parse-html SAF 33
- PathCheck
 - step in request handling process 28
- plugins
 - creating 17
 - for more info 18
 - installing 18
- preface 3

Q

- QUERY_STRING_UNESCAPED
 - variable in server-side HTML command 36

R

- relaxed
 - logging mode 42
 - parameter to flex-init SAF 42
- request-handling process
 - steps 28

- return codes
 - REQ_ABORTED 39
 - REQ_EXIT 39
 - REQ_NOACTION 39
 - REQ_PROCEED 39

- rules.properties 26

S

- SAFs

- for more info 18
 - installing 18

- security

- functions 43

- Server Application Functions

- see SAFs

- server behavior

- modifying 7

- server extensions 17

- server-parsed HTML tags 8

- server plugins 17

- server-side applets 6

- server-side HTML commands 33

- config 34
 - creating new server-side tags 37
 - echo 35
 - environment variables in 36
 - exec 36
 - flastmod 36
 - format 34
 - fsize 35
 - include 35
 - using 33

- server-side HTML tags

- creating 37

- server-side JavaScript

- accessing external databases 11
 - Application Manager 10
 - changes in 4.0 6
 - compiling 10
 - enabling 9
 - installing applications 10

- Writing SSJS Applications Applications 9

- server-side tags 8

- enabling parsing of 8
 - exec 8

- Service

- step in request handling process 28

- servlets

- API documentation from Sun
 - Microsystems 14

- compiling 14

- enabling 15

- for more info 16

- installing 15

- more info on creating 16

- registered directories 16

- virtual paths 16

- servlets.jar 14

- servlets.properties 26

- shtml.dll 38

- shtml.so 38

- shtml_add_tag 38

- function for registering custom server-side tags 40

- shtml_init 33

- shtml_public.h 38

- shtml_send 33

- shtml files 37

- ShtmlTagExecuteFunc

- function for defining server-side tags 38

- ShtmlTagInstanceLoad

- function for defining server-side tags 39

- ShtmlTagInstanceUnload

- function for defining server-side tags 39

- ShtmlTagPageLoadFunc

- function for defining server-side tags 39

- ShtmlTagPageUnLoadFn

- function for defining server-side tags 39

- sizeof attribute of config command 34

- SSJS

- see server-side JavaScript

ssl-check SAF 43

summary

ES 4.0 APIs 22

T

tag execution function

for customized server-side tag 38

tags

server-parsed HTML 33

TagUserData

data structure for custom server-side tags 38,
39

timefmt tag 34

type-by-extension SAF 29

U

util_cookie_find NSAPI function 43

util_cookie_next NSAPI function 43

V

variables

in server-side HTML commands 36

version control 22

virtual attribute of the include command 35

virtual paths

for servlets 16

Visibroker 6, 19, 45

W

WAI 45

compatibility issues 45

enabling 19

for more info 20

installing applications 19

installing component 19, 45

need for Visibroker 6

ORB.init() 47

OSAGENT 46

Web Application Interface

see WAI

webpub.conf 26

web publishing 21

enabling 22

version control 22

Web Publishing API 21

for more info 22

Writing Server-Side JavaScript Applications 9,
11

Writing Web Applications with WAI 19

Programmer's Guide to Enterprise Server 4.0

Contents

About This Book

1. Overview

API Changes in Enterprise Server 4.0

Configuration Files

Enterprise Server 4.0 APIs

Server-Parsed HTML Tags

Server-Side JavaScript

CGI

Java Servlets and JavaServerPages (JSP)

NSAPI

Web Application Interface (WAI) API

Access Control API

Web Publishing API

API Summary

2. Configuration Files

Introduction

magnus.conf

obj.conf

mime.types

3. Server-Parsed HTML Tags

Using Server-Side HTML Commands

config

include

echo

fsize

flastmod

exec

Environment Variables in Server-Side HTML Commands

Defining Customized Server-Parsed HTML Tags

The Mechanics

4. NSAPI Changes

Privatization of Some Data Structures

Logging Changes

Cookie Support

New SAF for Security

New SAFs for Adding Headers and Footers

Minor Changes to Init-class SAFs

5. WAI Release Notes

WAI Compatibility Issues

Index